УДК 004.738.5(07)
ББК 32.973.202я7
С32

Сергеенко С. В.

Разработка и проектирование Web-приложений в Oracle Developer: учебное пособие [Электронный ресурс] / С. В. Сергеенко. — 2-е изд. (эл.) — Электрон. дан. и прогр. (3 Мб.) — М: Интернет-Университет Информационных Технологий; Саратов: Вузовское образование, 2017.

ISBN 978-5-4487-0091-0

Книга «Разработка и проектирование Web-приложений в Oracle Developer» будет полезна всем, кто хочет получить от Oracle Forms Developer стабильную и эффективную работу, кто хочет знать возможности данной среды в организации систем, удовлетворяющих всем требованиям внедрения и эксплуатации. Работая с книгой, читатели смогут создавать, запускать и отлаживать формы в среде Oracle Forms, научатся создавать клиент-серверные и Web-ориентированные приложения. Используя примеры, приведенные в книге, читатель научится не только проекти-ровать экранные формы, но и добавлять в форму и ее элементы различныефунк-циональныевозможности,используятриггерысобытий,создавать многооконные и многомодульные приложения. Читатель научится создавать приложения, взаимодействующие с Microsoft Office, а также интегрировать в форму различные приложения — такие, как Adobe Acrobat, Windows Media Player, Explorer и др.

Издание осуществленно при финансовой и технической поддержке издательства «Открытие системы», «РМ Телеком» и Kraftway Comouters.

Для создания электронного издания использовано:
Приложение pdf2swf из ПО Swftools, ПО IPRbooks Reader,
разработано на основе Adobe Air.
Подписано к использованию 22.06.2017.

О проекте

Интернет-Университет Информационных Технологий – это первое в России высшее учебное заведение, которое предоставляет возможность получить дополнительное образование во Всемирной сети. Web-сайт университета находится по адресу www.intuit.ru.

Мы рады, что вы решили расширить свои знания в области компьютерных технологий. Современный мир – это мир компьютеров и информации. Компьютерная индустрия – самый быстрорастущий сектор экономики, и ее рост будет продолжаться еще долгое время. Во времена жесткой конкуренции от уровня развития информационных технологий, достижений научной мысли и перспективных инженерных решений зависит успех не только отдельных людей и компаний, но и целых стран. Вы выбрали самое подходящее время для изучения компьютерных дисциплин. Профессионалы в области информационных технологий сейчас востребованы везде: в науке, экономике, образовании, медицине и других областях, в государственных и частных компаниях, в России и за рубежом. Анализ данных, прогнозы, организация связи, создание программного обеспечения, построение моделей процессов – вот далеко не полный список областей применения знаний для компьютерных специалистов.

Обучение в университете ведется по собственным учебным планам, разработанным ведущими российскими специалистами на основе международных образовательных стандартов Computer Curricula 2001 Computer Science. Изучать учебные курсы можно самостоятельно по учебникам или на сайте Интернет-Университета, задания выполняются только на сайте. Для обучения необходимо зарегистрироваться на сайте университета. Удостоверение об окончании учебного курса или специальности выдается при условии выполнения всех заданий к лекциям и успешной сдачи итогового экзамена.

Книга, которую вы держите в руках, – очередная в многотомной серии «Основы информационных технологий», выпускаемой Интернет-Университетом Информационных Технологий. В этой серии будут выпущены учебники по всем базовым областям знаний, связанным с компьютерными дисциплинами.

Добро пожаловать в
Интернет-Университет Информационных Технологий!

Анатолий Шкред
anatoli@shkred.ru

Об авторе

Сергеенко Сергей Васильевич

Родился в городе Мариуполе в 1985 году. Закончил Механико-металлургический техникум по специальности «Программирование ЭВМ и автоматизированных систем» и университет ПГТУ по специальности «Информатика и прикладная математика». С 2005 года работает на комбинате ОАО «ММК им. Ильича» ведущим инженером-программистом в «Бюро разработки и внедрения программного обеспечения». С СУБД Oracle работает, начиная с версии 8i. Помимо разработки приложений в Oracle Developer и Business Intelligence занимается оптимизацией и настройкой запросов. Кроме Oracle использует СУБД My SQL, но предпочтение отдает первому варианту. Также изучает языки PHP, JAVA и интегрирует их со своими разработками в Oracle. Хобби автора – дизайн и графика, поэтому каждую программу он старается сделать не только функциональной, но и красивой.

Это одна из первых русскоязычных книг в СНГ по данной технологии. В книге описано то, чем Сергеенко С.В. занимается ежедневно. Предлагаемый в ней материал посвящен темам и вопросам, с которыми пользователи сталкиваются постоянно. Данная книга – итог многолетнего использования СУБД Oracle.

Благодарности

Я бы хотел поблагодарить многих людей, помогавших мне создать эту книгу. На комбинате ОАО «ММК им. Ильича» я работаю с лучшими и наиболее яркими людьми из тех, кого мне удалось узнать, и они все так или иначе помогли мне. Выражаю благодарность всему коллективу ОАСУП и особенно своему наставнику – Бадановой Валентине Алексеевне, за поддержку в испытании писательским трудом.

Наконец, самое главное, я благодарен за неизменную поддержку своей семье. Я просто не представляю, как бы я закончил данную книгу без постоянной поддержки моих родителей и жены Анастасии.

Оглавление

Лекция 1. Введение в Oracle Forms Developer

В этой лекции слушатель ознакомится с ключевым продуктом для разработчиков – сервером приложений Oracle AS. В лекции будут рассмотрены все имеющиеся компоненты, технологии и стандарты, поддерживаемые сервером приложений.

Ключевые слова: Oracle Application Server, Oracle Forms Services, Oracle Developer Suite.

Цель лекции: ознакомить слушателя с основными продуктами Oracle и составляющими компонентами Oracle AS.

Корпорация **Oracle** является крупнейшим в мире поставщиком корпоративного программного обеспечения и предлагает полный комплекс технологий для построения ИТ-инфраструктуры и управления современным предприятием: семейство базовых программных технологий Oracle10*g*, готовое решение для коллективной работы Oracle Collaboration Suite, полнофункциональный комплекс бизнес-приложений Oracle E-Business Suite и интеграционное решение для управления данными Oracle Data Hub. Корпорация предоставляет свои продукты и услуги в области консалтинга, обучения и технической поддержки более чем в 150 странах мира. Официальный сайт корпорации – www.oracle.com.

На сегодняшний день СУБД Oracle – это интегрированное программное обеспечение для сети распределенных вычислений Grid. Для реализации сложных комплексных решений имеется широкий набор программных продуктов, который можно разбить на несколько разделов по их функциональному назначению.

Oracle Application Server

Oracle Application Server (сервер приложений) – это первая в мире основанная на стандартах GRID, интегрированная программно-прикладная платформа, которая позволяет упростить управление приложениями, выполняемыми в распределенной вычислительной среде.

Оптимизированная для Grid-вычислений платформа Oracle AS позволяет снизить расходы на приобретение техники. Сервер приложений Oracle AS дает возможность организациям и предприятиям любого масштаба возможность оперативно реагировать на меняющиеся требования рынка. **Oracle Application Server** обеспечивает полную поддержку следующих технологий и возможностей в одном продукте:

- J2EE;
- Распределенные вычисления;
- Корпоративные порталы (Portal);
- Web-Cache – высокоскоростное веб-кеширование;
- Разработка и интеграция бизнес-приложений (Business Intelligence);
- Rapid Development – быстрая разработка и внедрение приложений;
- Wireless – поддержка беспроводных технологий;
- Веб-сервисы.

Помимо перечисленных возможностей в состав Oracle Application Server входит полный набор инструментариев и инфраструктуры для обеспечения безопасности на всех уровнях разработки и развертывания приложения. Oracle Application Server позволяет создавать приложения, используя различные языки и технологии (табл. 1.1).

Таблица 1.1. Технологии и языки, поддерживаемые Oracle AS

Технология/Язык	
Java and J2EE	Java Server Pages (JSP) v. 1.2 Java Servlets v. 2.3 Enterprise Java Beans (EJB) v.2.0 Java Database Connectivity (JDBC) v. 2.0 Extensions Java Transaction API (JTA) v. 1.0 Java Naming and Directory Interface (JNDI) v. 1.2 Java Message Service (JMS) v.1.0.2b Java Authentication and Authorization Service (JAAS) v.1.0 J2EE Connector Architecture v. 1.0 Java API for XML Parsing (JAXP) v.1.1 Java Mail v. 1.0
XML	XML v. 1.0 XML Namespaces v. 1.0 Document Object Model (DOM) v.1.0/2.0 Extensible Stylesheet Language Transformations (XSLT) v. 1.0 XML Schemas v.1.0 Simple API for XML (SAX) v.1.0/2.0 + Extensions XML Path Language (XPath) v. 1.0 XSQL Internet Data Access Presentation

PL/SQL	PL/SQL Server Pages v. 9.0.4 PL/SQL Web Toolkit v. 9.0.4 Oracle Application Server Forms Services v. 9.0.4
Web services	Web Services Description Language (WSDL) v. 1.1 Universal Description, Discovery, and Integration (UDDI) v. 2.0 Simple Object Access Protocol (SOAP) v. 1.1

Применение оптимизированного под Grid-вычисления сервера приложений существенно снижает затраты на аппаратные средства и упрощает администрирование, позволяя разворачивать и управлять приложениями, разработанными в Oracle Developer Suite, на одном прикладном сервере. Модуль Oracle Forms Services, который мы будем рассматривать чуть позже, также является частью Oracle AS и позволяет развертывать приложения, написанные в Forms Developer в Web. Ниже перечислены основные сервисы сервера приложений.

- Коммуникационные сервисы (Communication services).
- Сервисы выполнения приложения (Application Runtime Services).
- Системные сервисы (System Services).
- Управление сервисами (Management Services).
- Сервисы соединения (Connectivity services).
- Решения (Solutions).

В таблице 1.2 представлены основные компоненты и ассоциированные с ними решения (Solutions) Oracle Application Server.

Таблица 1.2. Компоненты Oracle AS

Решение	**Компонент**
J2EE и интернет-приложения	Oracle HTTP Server Oracle AS Containers for J2EE Oracle AS TopLink Oracle Business Components for Java Oracle Application Server Web Services Oracle JDeveloper Oracle Application Server Forms Services Oracle XML Developer Kit Oracle PL/SQL Oracle Content Management SDK Oracle Application Server MapViewer

Portals	Oracle Application Server Portal Oracle Application Server Portal Developer Kit Oracle Ultra Search Oracle Application Server Syndication Services
Wireless	Oracle Application Server Wireless Oracle Application Server Wireless Developer Kit
Business Intelligence	Oracle Application Server Reports Services Oracle Application Server Discoverer Oracle Application Server Personalization
E-Business Integration	Oracle Application Server InterConnect Oracle Application Server ProcessConnect
Caching	Oracle Application Server Web Cache
System Management	Oracle Enterprise Manager
Identity Management и Security	Oracle Application Server Single Sign-On Oracle Application Server Certificate Authority Java Authentication and Authorization Service Oracle Internet Directory

Как вы уже успели заметить, платформа Oracle Application Server объединяет в себе множество технологий и инструментов для создания полного цикла разработки и развертывания приложения. Мы рассматриваем версию Oracle Application не только потому, что она является актуальной и сочетает в себе опыт и возможности предыдущих версий, но еще и потому, что она стала первой Grid-ориентированной платформой, открывающей много новых возможностей. Среди нововведений отметим:

- **Управляемость** – в состав Oracle Application Server входит набор средств управления рабочей нагрузкой, который за счет перераспределения имеющихся ресурсов между приложениями упрощает оптимизацию вычислительной мощности.
- **Надежность** – по заявлению разработчиков корпорации Oracle, новая платформа Oracle Application Server обладает новыми средствами повышения безопасности и надежности корпоративных решений, выполняемых на кластерах и в сети распределенных вычислений предприятия. Среди новых возможностей прежде всего следует отметить функции:

  - Fast-Start Fault Recovery Architecture – это усовершенствованная архитектура быстрого запуска при устранении отказа;
  - Failure Notification (FaN) – функция оповещения об ошибках.
- **Интеграция приложений** – новые возможности интеграции позволяют использовать один программный продукт для различных типов интеграции приложений и создать единую модель данных, выступающую информационным ядром для поддержки реализации будущих интеграционных проектов.
- **Новые возможности для веб-сервисов** – это прежде всего усовершенствование функциональности веб-сервисов. Grid Computing Model (GCM) оптимизирует процесс наращивания и распределения вычислительной мощности, а веб-сервисы, в свою очередь, облегчают повторное использование приложений.

Oracle Forms Services

Oracle Forms Services (OFS) – это компонент Oracle Application Server, предназначенный для переноса приложения в Web. Когда пользователь запускает приложение Oracle Forms, автоматически загружается и кешируется на клиентской Java-машине Java-апплет из сервера приложений (Oracle AS). Также модуль OFS при запуске приложения Forms Developer преобразует его графический интерфейс в набор Java-апплетов, которые пересылаются на клиентский компьютер. Даже несмотря на то, что Forms использует Java-апплет для отображения формы в клиентском браузере, от пользователя не требуется знание языка JAVA.

Преимущество над другими клиент-серверными версиями Forms

По сравнению с предыдущими версиями Forms, такая архитектура дает следующие преимущества:

- возможность не устанавливать клиентское программное обеспечение на все компьютеры, которые работают с приложением, что существенно снижает затраты на развертывание такой системы;
- администрирование – благодаря использованию модуля OFS достигается высокая масштабируемость, так как администрирование одного сервера приложений намного проще, чем множества клиентских компьютеров;
- автоматическое распределение нагрузки – Oracle AS Forms Services автоматически распределяет нагрузку на все серверы приложений, работающие в системе. Что же касается распределения нагрузки на сервер в критические моменты или переадресации пользовательских

запросов, то в этом вам поможет Oracle Enterprise Manager (OEM), который содержит компоненты для администрирования всех серверов приложений.

Oracle Forms Services использует трехзвенную архитектуру (рис. 1.1).

1. Звено Клиент содержит браузер, в котором отображается форма.
2. Сервер приложений – это промежуточное звено, в котором располагается сервер программного обеспечения и логики приложений.
3. Звено База Данных – это сервер БД, в которой хранятся данные.

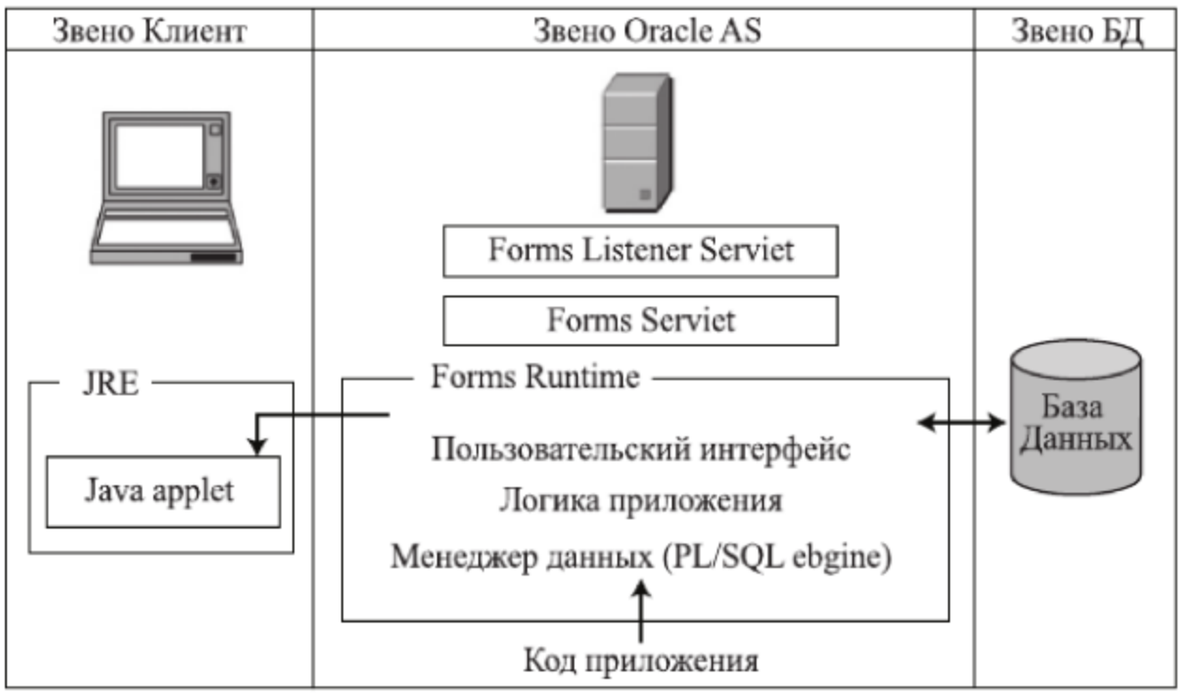

Рис. 1.1. Архитектура Oracle Forms Services

Основные компоненты Oracle Forms Services:

- JAVA Client;
- Forms Listener Servlet;
- Forms Servlet;
- Forms Runtime Engine.

Oracle Developer Suite

Oracle Developer Suite 10g – это полный набор интегрированных средств для разработки интернет-приложений, который включает в себя удобную интегрированную среду разработки со средствами моделирования, программирования на PL/SQL, Java, разработки компонентов, бизнес-анализа, составления отчетов и диаграмм. И, что очень важно, все эти средства используют общие ресурсы, что позволяет совместно работать над одним проектом группе разработчиков. В таблице 1.3 приведены компоненты (продукты) Oracle Developer Suite.

Таблица 1.3. Состав Oracle DS

Компонент	**Описание**
Oracle Designer	Проектирование БД и приложений
Oracle Forms Developer	Разработка экранных форм
Oracle Reports Developer	Разработка отчетов
Oracle JDeveloper	Разработка приложений на Java
Business Components for Java	Бизнес-компоненты на Java
Oracle Discoverer	Разработка аналитических приложений
Oracle Warehouse Builder	Проектирование хранилищ данных

В состав Oracle Developer Suite также входит XML Developer's Kit (XDK) – набор различных компонентов, утилит и интерфейсов для организации работы с XML-документами. Различают пять различных XDK:

- XDK for Java;
- XDK for JavaBeans;
- XDK for C;
- XDK for C++;
- XDK for PL/SQL.

Oracle Developer Suite поддерживает стандарт UML-моделирования объектных приложений, который позволяет моделировать классы и рабочие процессы с последующей генерацией кода для среды BC4J. Хранение моделей предусмотрено в общем репозитории Oracle, где удобно контролировать версии объектов.

Если у вас большой коллектив разработчиков, то для поддержки их работы в Oracle Developer Suite предусмотрен репозиторий, который централизованно хранит структурированные данные как объекты базы данных, а неструктурированные – как файлы и XML-документы. Репозиторий – это своего рода единое хранилище метаданных приложений, обеспечивающее параллельную работу разработчиков. Использование репозитория дает разработчикам возможность анализировать зависимости между объектами, контролировать использование общих компонентов. Для удобства работы с репозиторием предусмотрены такие функции, как просмотр архива версий, управление процессом обновления версий объектов, а также функции контроля зависимостей и управление конфигурациями.

Oracle Developer Suite содержит набор переносимых JavaBean-компонентов пользовательского интерфейса, разработку и поддержку веб-приложений. Для реализации этих возможностей разработчики могут использовать API-интерфейсы Java или User Interface XML (UIX).

Oracle Designer

Oracle Designer – это интегрированная CASE-среда для автоматизации процессов всех этапов полного жизненного цикла сложной прикладной системы, включая детальный анализ предметной области, проектирование, программирование, тестирование, оценку, сопровождение, обеспечение качества, управление конфигурацией, управление проектом, документирование системы, формулировку и анализ требований.

Oracle JDeveloper 10g

Oracle JDeveloper 10g – это интегрированная среда разработки для моделирования, разработки и отладки J2EE-приложений и веб-сервисов. Как и в случае с Oracle Forms, разработка приложений в JDeveloper существенно упрощена за счет использования мастеров, редакторов, инструментов моделирования и дружественного интерфейса. Также среди прочего следует отметить поддержку возможности интерактивной привязки данных к пользовательскому интерфейсу и автоматического развертывания в сервере приложений.

Oracle Reports

Oracle Reports – это мощный инструмент для разработки отчетов любой степени сложности на основе информации, хранящейся в базах данных или других источниках. Oracle Reports позволяет не только отображать информацию из БД, но и управлять внешним видом документа, а также сформировать различные типы документов с помощью мастеров. Вы также можете генерировать выходной файл в другом формате, таком как HTML, PDF, RTF, SPREADSHEETS или XML.

Oracle Warehouse Builder

Oracle Warehouse Builder (OWB) – это расширяемая многофункциональная CASE-среда, предназначенная для разработки и развертывания корпоративных хранилищ и витрин данных.

Oracle Discoverer

Oracle Discoverer – это инструмент для получения произвольных отчетов, формирования нерегламентированных запросов и анализа данных. Он обеспечивает быстрый и удобный доступ к информации, содержащейся в реляционных и многомерных хранилищах и витринах данных,

а также в транзакционных системах, в том числе не обязательно работающих под управлением СУБД Oracle. С помощью Oracle Discoverer пользователь может получать необходимые ему данные в виде различных таблиц, графиков и диаграмм.

Oracle Business Intelligence Beans

Oracle Business Intelligence Beans – это набор компонентов JavaBeans, предназначенных для создания приложений бизнес-анализа. Взаимодействие с JDeveloper – «бесшовная» интеграция компонентов BI Beans. Компоненты Oracle Business Intelligence Beans позволяют создавать приложения бизнес-аналитики и использовать преимущества функциональности OLAP.

Oracle Forms Developer

Oracle Forms Developer – это мощное средство для быстрой разработки приложений, которые основаны на информации, хранящейся в базах данных или других источниках. Forms Developer содержит исчерпывающий набор инструментов для создания полнофункционального прикладного программного обеспечения, состоящего из форм, отчетов и деловой графики. Вы можете программировать вызов отчета, построенного с помощью Oracle Reports, по нажатии пункта меню или кнопки, или в форму может быть вставлен вывод диаграммы, сгенерированной с помощью Oracle Graphics. Такой модульный подход предоставляет максимальную гибкость при проектировании и разработке новых прикладных программ, а также для поддержания и улучшения существующих. Forms Developer имеет большое количество различных мастеров для быстрого создания объектов. Если вы разрабатываете приложение в Forms, то в первую очередь вы избавляете себя от написания большого количества кода, так как все основные операции взаимодействия с БД уже автоматизированы.

Oracle Forms и другие инструменты Developer оптимизированы так, что имеют и используют множество новых и мощных свойств текущего сервера Oracle, а также средства разработки приложений для ввода, доступа, изменения или удаления данных из БД Oracle в реальном времени. Ниже приведены другие особенности Forms Developer.

- Деление приложения – в зависимости от ситуации вы можете хранить ваши PL/SQL-модули на сервере или в приложении. Вы можете перетаскивать объекты как между модулями, так и в пределах модуля.
- Инструменты быстрой разработки – Forms Developer имеет большое количество различных мастеров для быстрого создания объектов.

Когда вы разрабатываете приложение с помощью мастера, количество кода сводится к минимуму или его не требуется вообще, поэтому создание простейшего приложения может занимать считанные минуты.

- Гибкость и контроль исходного кода – менеджер конфигурации программного обеспечения (Software Configuration Manager – SCM), интегрированный с Forms Developer, предназначен для управления структурированными и неструктурированными данными и всеми типами файлов в рамках жизненного цикла разработки программного продукта.
- Масштабируемость – используя многозвенную архитектуру, вы можете масштабировать приложение от одного до десятков тысяч пользователей без надобности вносить в приложение какие-либо изменения. Вы также можете повысить масштабируемость приложения за счет использования функциональности сервера, связываемых переменных, курсоров, хранимых процедур, пакетов, DML- и DDL-операций.
- Поддержка Java, SQL, PL/SQL – Forms Developer поддерживает разработку на всех перечисленных языках.
- Повторное использование объектов – Forms Developer имеет очень удобную модель наследования, а также средства ее реализации, которые облегчают наследование различных признаков (атрибутов) от одного объекта другому, от одного приложения к другому, используя объектные библиотеки (Object Library) и подклассы.
- Набор встроенных пакетов – Forms Developer содержит множество пакетов, которые существенно упрощают разработку приложений, предоставляя такие возможности, как файловый ввод-вывод, подключение внешних библиотек.

Что касается различных версий Forms, то в этой книге будут упомянуты различные версии, одна из которых Oracle Forms 6i. Это единственная версия, поддерживающая три режима вывода формы на экран: терминальный, графический (GUI) и в браузере. Особенность такого приложения состоит в том, что в любом из перечисленных режимов оно может быть запущено без перекомпиляции. Каждая новая версия предоставляет много других новых и важных возможностей по сравнению с предыдущими версиями; так, например, если брать новшества, появившиеся в Oracle Forms 6i, то в отличие от более ранних версий здесь есть возможность интегрировать сформированные отчеты с офисными приложениями (Microsoft Word, Excel). Также вам предоставляется возможность использовать мощные средства программирования, предоставляемые OLE-сервером приложений.

Однажды созданное вами приложение в Oracle Forms на платформе Windows может работать на таких платформах, как Linux, Mac Os, AIX и многих других. Другими словами, приложение, разработанное в Forms, – кросс-платформенное и может работать под управлением любой операционной системы единственное требование – это перекомпиляция модуля. Все версии до Forms 6i включительно являются клиент-серверными и требуют установки клиентского программного обеспечения на всех компьютерах, работающих с необходимым приложением.

Приложения, созданные в Oracle Forms Developer, могут быть развернуты на любом уровне предприятия, также вы можете создавать решения для малого и крупного бизнеса, поддерживающие от одного до сотни одновременно работающих пользователей. Учитывая возможность расширения инструментария пользователем, с помощью Forms можно создать практически любое требуемое окружение базы данных Oracle.

Лекция 2. Структура Oracle Forms

В этой лекции слушатель будет ознакомлен со структурой Oracle Forms: компонентами, модулями, объектами. Слушатель научится запускать формы, генерировать модули форм, меню и библиотек, как в графической среде, так и в командной строке.

Ключевые слова: Oracle Application Server, Oracle Forms Services, Oracle Developer Suite.

Цель лекции: понимание структуры Forms Developer, объектной зависимости и основных компонентов интерфейса.

Несмотря на то что все версии Forms отличаются друг от друга различными возможностями, нововведениями и улучшенными методами разработки, концептуально подход к разработке приложения не изменился. Если еще вчера на вашем компьютере стоял Forms 6i, а сегодня вы поставили Forms 10g, то при первом знакомстве с этим продуктом вы не найдете существенных отличий до тех пор, пока не запустите форму или детально не обследуете каждый пункт меню. Независимо от версии, Oracle Forms можно разбить на три части:

- **Oracle Forms Builder** – это основная среда для разработки приложения, в которой вы работаете с тремя типами модулей: форма, меню и библиотека. Oracle Designer – набор инструментов визуального моделирования, мастеров и прочего инструментария, позволяющий создавать объекты, устанавливать их свойства и писать программные модули для прикладных программ;
- **Oracle Forms Compiler** – используется для генерации файлов программ, чтобы создавать исполняемые файлы. Генерация модуля формы компилирует все ее программные объекты и создает исполняемый файл с расширениями .FMX, .MMX и .PLX;
- **Oracle Forms Runform** – используется для запуска уже скомпилированной прикладной программы Oracle Forms.

Приятно удивляет и то, что с появлением новых веб-ориентированных версий Forms Developer – Forms 9i и Forms 10g – типы поддерживаемых модулей не изменились, поэтому в Oracle Forms вы по-прежнему можете работать с тремя типами модулей.

- **Библиотеки** – это совокупность процедур, функций и пакетов, которые могут вызываться из других модулей прикладной программы.
- **Формы** – это совокупность объектов и подпрограмм. Примерами объектов, которые вы можете определять в модуле формы, могут

быть окна, элементы текста (поля), переключатели, кнопки, сигналы, списки значений и программные единицы.

- **Меню** – это совокупность объекта главного меню и любого количества объектов подменю, а также команд элементов меню, которые вместе образуют меню приложения.

В таблице 2.1 приведены спецификации используемых в Oracle Forms модулей.

Таблица 2.1. Расширения модулей Oracle Forms

Обозначение	Расширение
.FMB	Form Module Binary – двоичный модуль формы
.FMT	Form Module Text – текстовый модуль формы
.FMX	Form Module Executable – исполняемый модуль формы
.MMB	Menu Module Binary – двоичный модуль меню
.MMT	Menu Module Text – текстовый модуль меню
.MMX	Menu Module Executable – исполняемый модуль меню
.PLL	PL/SQL Library Module Binary – двоичный модуль библиотеки PL/SQL
.PLD	PL/SQL Library Module Text – текстовый модуль библиотеки PL/SQL
.PLX	PL/SQL Library Executable – исполняемая библиотека PL/SQL (не содержит исходного текста)

Примечание: если вы создаете прикладную программу с несколькими модулями, то вы должны предоставлять все файлы .FMX, .MMX и .PLX, необходимые для развертывания во время выполнения.

Компоненты интерфейса Forms Builder

К главным компонентам интерфейса построителя форм относят инструменты проектирования приложения (см. Лекцию 6. Инструменты проектирования) и мастера:

- Объектный Навигатор;
- Редактор Разметки;
- Палитра свойств;
- Редактор PL/SQL;
- мастер создания блоков данных (Data Block Wizard);

- мастер разметки (Layout Wizard);
- мастер создания списков значений (LOV Wizard).

Весь представленный набор инструментов позволяет быстро разрабатывать приложения, избавляя разработчика от написания большого количества кода и базовой разметки. Инструменты проектирования используются для создания и управления свойствами объектов, PL/SQL-программ, навигации и поиска.

Структура меню и панели инструментов построителя форм

Главное меню (рис. 2.1) построителя форм дает вам возможность создавать и модифицировать новые модули, объекты и управлять приложением с помощью встроенных команд. Меню разбито на пункты, каждый из которых содержит ассоциированные с ним команды:

- **File** – набор стандартных общепринятых файловых команд, таких как создание (Create), сохранение (Save, Save As), открытие (Open) и команды управления модулем;
- **Edit** – команды копирования, вставки, редактирования, настройки и так далее;
- **View** – команды управления отображением объектов, панелей инструментов и видов;
- **Layout** – набор команд, используемых в Редакторе Разметки;
- **Program** – команды компиляции и любые связанные с кодированием операции, включая вызов инструментов для работы с PL/SQL-редактором;
- **Debug** – включение поддержки команд отладчика;
- **Tools** – набор команд для вызова компонентов интерфейса Forms;
- **Window** – набор команд для управления окнами;
- **Help** – вызов справочной информации и сведений о продукте.

Панель инструментов – это горизонтальная панель (см. рис. 2.1), которая содержит набор «горячих» клавиш для вызова наиболее часто используемых команд. Вы можете убрать главную панель инструментов, сняв флаг меню View | Main Toolbar.

Рис. 2.1. Главная панель инструментов (Main Toolbar)

Объектная модель Oracle Forms

Приложение Oracle Forms Developer может включать в себя от одного до n модулей, которые, в свою очередь, состоят из других компонентов – объектов. Несмотря на довольно солидное многообразие объектов в Forms, можно выделить три основных объекта (рис. 2.2):

- Блок Данных;
- Элемент;
- Холст.

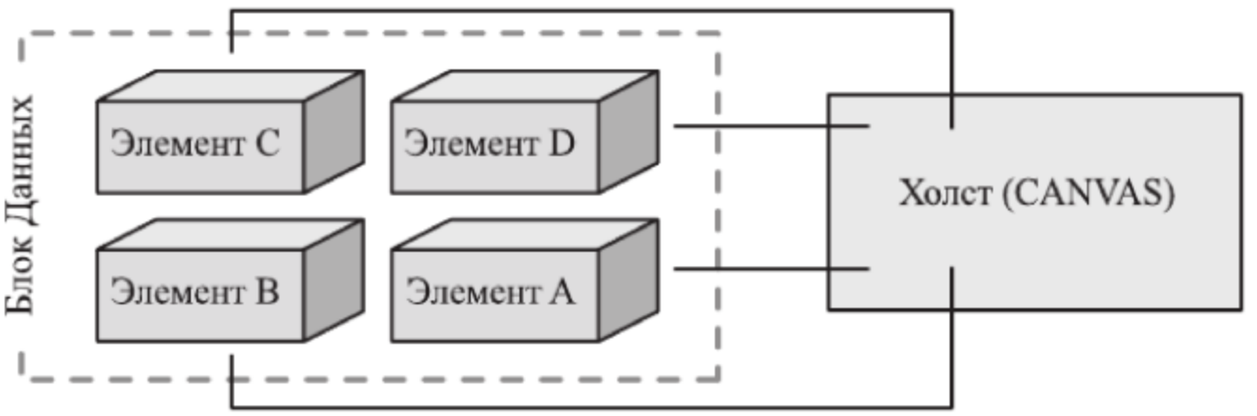

Рис. 2.2. Схема взаимосвязи между основными объектами

На рис. 2.3 представлена иерархическая объектная модель модуля Forms, которая отображает объекты модуля и зависимость между ними.

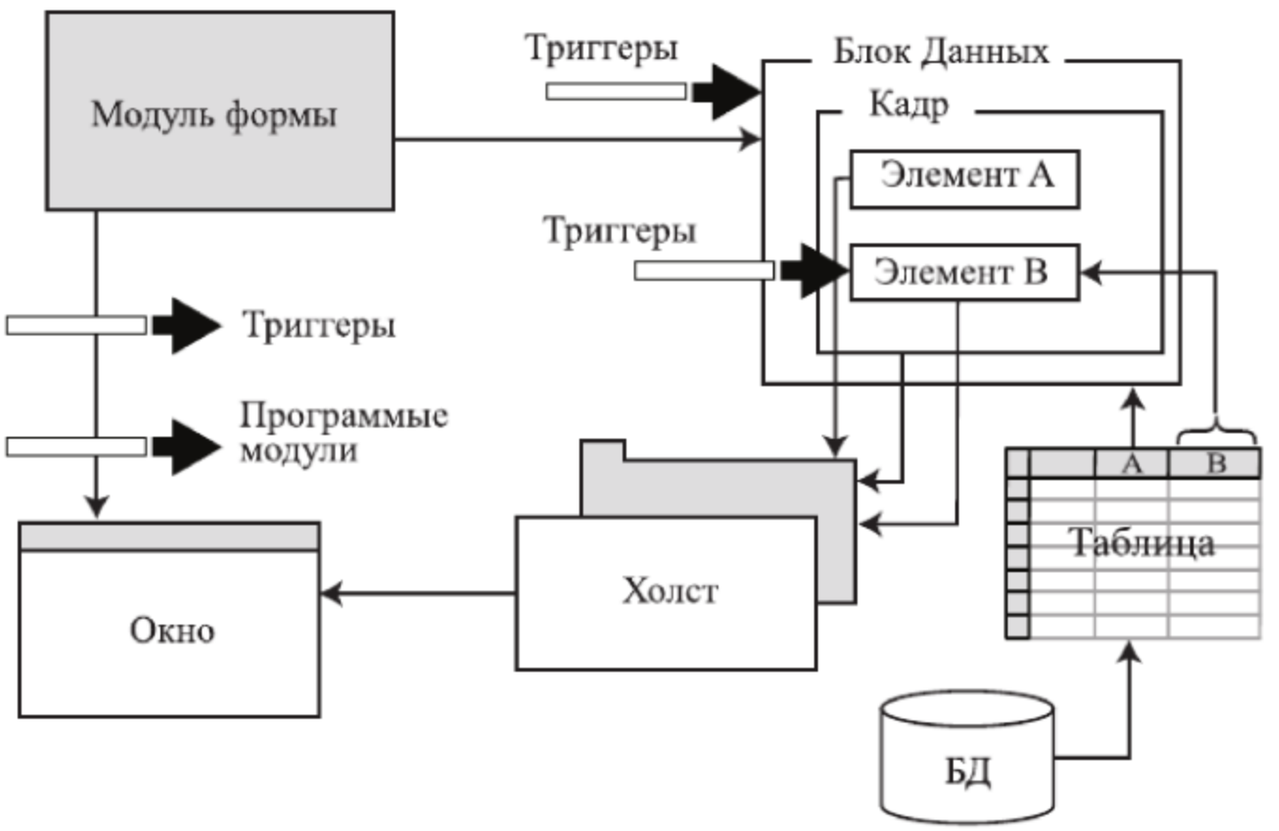

Рис. 2.3. Объектная модель модуля Forms

Рассмотрим рис. 2.3 более подробно. Модуль Oracle Forms может состоять из множества блоков, которые могут быть базовыми, то есть базироваться на объектах БД (View, Table, Synonym), или небазовыми. Блоки, в свою очередь, состоят из одного или более элементов, которые также могут быть базовыми или небазовыми. Элементы блоков данных, а также элементы графики, такие как линии, кадры и фигуры, располагаются на холсте. Холсты размещаются в окнах, причем с одним окном может быть ассоциировано несколько холстов. В пределах одного модуля существует три уровня триггеров: триггер уровня формы, триггер уровня блока и триггер уровня элемента. В форме может быть сколько угодно холстов, окон, элементов, блоков и программных единиц.

Генерация модулей

Генерация модулей – это процесс превращения двоичного файла (.FMB, .MMB, .PLL) в исполняемый файл – .FMX, .MMX, .PLX. В Oracle Forms существует несколько способов генерации исполняемых файлов:

- Генерация с помощью утилиты Forms Compiler.
- Генерация модуля из Form Builder.
- Генерация из командной строки.

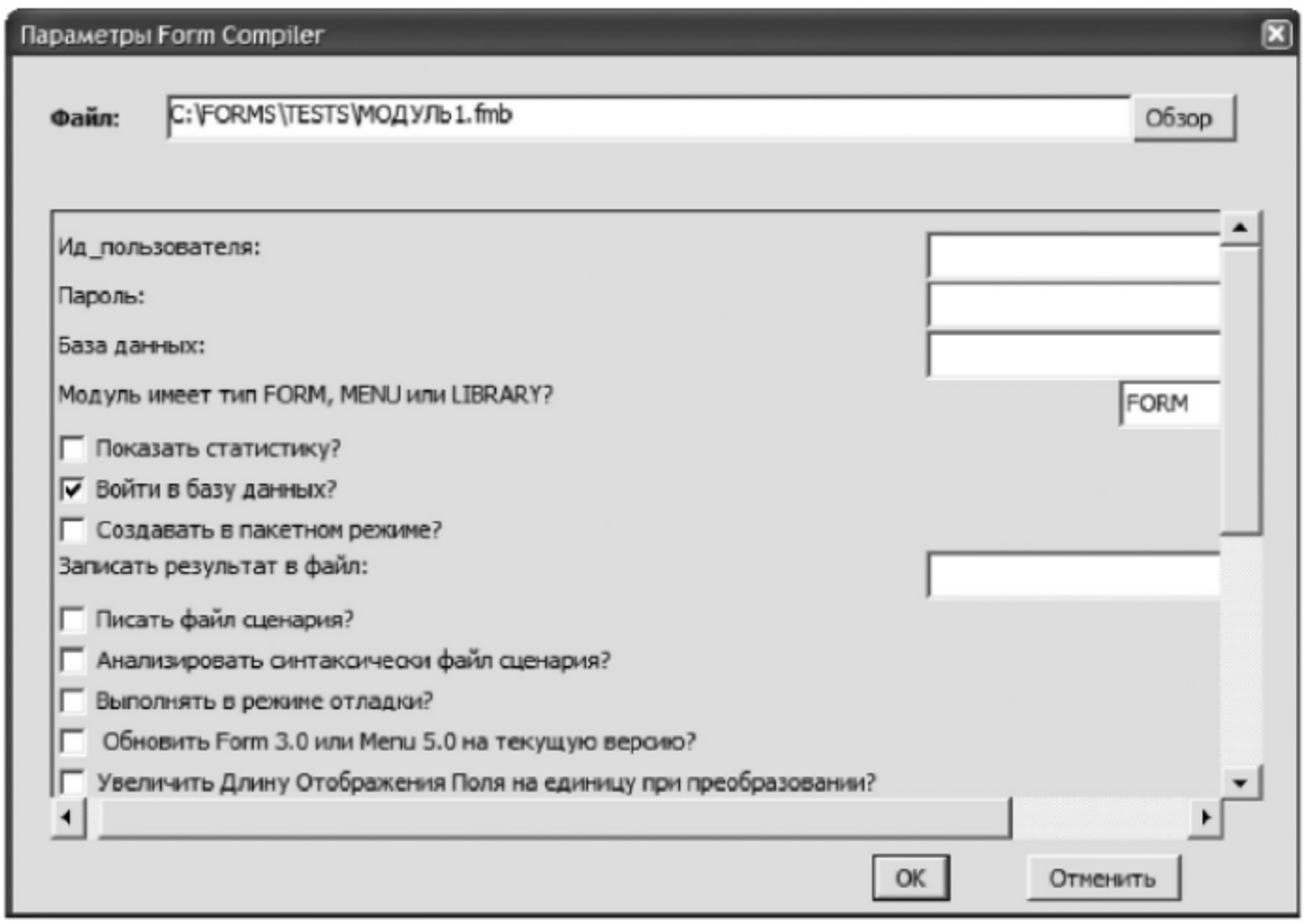

Рис. 2.4. Oracle Forms Compiler

Генерация модулей с помощью утилиты Forms Compiler

Утилиту Forms Compiler можно найти по следующему пути – DevSuiteHome_1\BIN\frmcmp.exe. На рис. 2.4 показан внешний вид Forms Compiler.

Утилита имеет довольно простой вид и логически разделена на три части: имя генерируемого модуля, параметры соединения и компиляции. В Forms Compiler вы можете скомпилировать модули форм, меню и библиотек. Для того чтобы скомпилировать форму с помощью Forms Compiler, достаточно выполнить следующее:

- В поле «Файл» укажите имя выполняемого модуля, с указанием полного пути. Это поле является обязательным.
- Заполните поля авторизации: «Ид_пользователя», «Пароль» и «База данных».
- В поле «Модуль имеет тип FORM, MENU или LIBRARY?» требуется указать один из перечисленных типов модуля. Этот параметр необязателен.

Генерация модулей из Form Builder

Генерация формы из Form Builder:

1. Форма, которую вы хотите компилировать, должна быть открыта в Forms Builder. Если в Forms Builder открыт более чем один модуль, то сделайте необходимый модуль текущим (активным), выбрав любой содержащийся в нем объект.
2. Если вы работаете в Forms 6i, то выберите File | Administration | Compile Module, если же вы используете более поздние версии, то выполните команду меню Program | Compile Module.
3. Если генерация прошла успешно, то в статусной строке выводится сообщение «Module Generated Successfully». Oracle Forms записывает файл .FMX в директорию по умолчанию (текущую), давая ему такое же имя, как у формы в Forms Builder.

Чтобы сгенерировать форму автоматически при запуске ее из Forms Builder, выполните следующие действия:

- Для вызова диалогового окна Options выберите Tools | Options.
- Установите предварительную установку Generate Before Run на On (установка по умолчанию).
- Сделайте в Forms Builder форму, которую хотите запустить, активным модулем, затем выберите File | Run.

Oracle Forms безоговорочно генерирует и затем выполняет указанную форму.

Примечание: *запуск формы при Generate Before Run, установленной на On, не генерирует никаких меню или библиотек, подсоединенных к этой форме. Генерировать меню и библиотеки вы должны явно командой File | Administration | Generate.*

Внимание: *очень важно не забывать о таком параметре, как Generate Before Run. Представьте себе случай: вы разработали демонстрационную форму, которую могут запускать один или несколько сотен пользователей. Если параметр включен, то при каждом новом запуске формы все формы, которые были запущены ранее, станут инвалидными.*

Генерация модулей меню

Когда вы создаете модуль пользовательского меню, вы подсоединяете его к форме установкой свойства формы Menu Module. Потом, когда вы запустите форму, Oracle Forms автоматически загрузит подсоединенное меню.

Прежде чем вы сможете успешно запустить форму, имеющую подсоединенное меню, вы должны явно сгенерировать исполняемый файл меню .MMX из конструкторского модуля меню .MMB.

Чтобы сгенерировать меню из Forms Builder:

1. Меню, которое вы хотите сгенерировать, должно быть открыто в Forms Builder. Если в Designer открыто более одного модуля, то сделайте нужное меню текущим, выбрав любой объект, который оно содержит.
2. Выберите File | Administration | Compile Module.

Oracle Forms записывает файл .MMX в директорию по умолчанию, давая ему то же имя, что и меню в Forms Builder.

Компиляция библиотек

В отличие от модулей форм и меню модули библиотек не должны явно генерироваться для создания отдельного исполняемого файла. Файл библиотеки .PLL, с которым вы работаете в Forms Builder, включает как исходную программу, необходимую во время проектирования, так и скомпилированную исполняемую программу, необходимую во время выполнения. Применение библиотеки .PLL требуется в случае, если вы захотите использовать Runform Debugger для просмотра исходной программы.

Когда вы в Forms Builder подсоединяете библиотеку к форме, меню или к другой библиотеке, вы подсоединяете версию .PLL этой библиотеки. Чтобы запустить прикладную программу из Forms Builder, вам только нужно убедиться, что подпрограммы и пакеты в библиотеке скомпилированы.

Для того чтобы скомпилировать библиотеку .PLL, выберите эту библиотеку в навигаторе, затем выберите пункт меню File | Compile Selection или File | Compile All.

Чтобы скомпилировать файл библиотеки .PLL с помощью одной только компоненты Generate, используйте опции `module_type` и `compile_all`, как показано ниже:

```
f60gen mylib.pll scott/tiger module_type=library compile_all=yes
```

Генерация библиотек для исполнения

Когда вы будете готовы развернуть свою прикладную программу, вы можете сгенерировать файл библиотеки .PLX из файла библиотеки .PLL. Файл .PLX не включает компоненты исходной программы, являющейся частью файла .PLL. Вы можете сгенерировать файл .PLX из Designer или из командной строки.

Когда вы в Designer подсоединяете библиотеку к модулю, вы подсоединяете версию .PLL этой библиотеки. Во время исполнения Oracle Forms ищет это имя по путям поиска по умолчанию файл .PLX. Если файл .PLX отсутствует, то Oracle Forms ищет файл .PLL с этим именем.

Чтобы сгенерировать файл библиотеки .PLX из Designer:

1. Библиотека, которую вы хотите сгенерировать, должна быть открыта в Designer. Если в Designer открыто более одного модуля, сделайте нужную библиотеку текущим модулем, выбрав любой программный модуль, который она содержит.
2. Выберите File | Administration | Compile All.

Oracle Forms записывает файл .PLX в директорию по умолчанию, давая ему то же имя, что и у библиотеки .PLL в Forms Builder.

Чтобы сгенерировать файл библиотеки .PLX из командной строки:

Вызовите компоненту Forms Compiler с опцией module_type:

```
f60gen mylib.pll scott/tiger module_type=library
```

Эта команда компилирует все программные модули, находящиеся в нескомпилированном состоянии. Она не перекомпилирует программные модули, успешно скомпилированные в Forms Builder.

Лекция 3. Создание форм. Настройка и запуск

В этой лекции рассмотрено пошаговое создание формы, особенности и способы запуска веб-приложений в Forms Builder. Также в лекции рассмотрены варианты работы приложения как с базой данных, так и самостоятельно.

Ключевые слова: Module, OC4J, J2EE, Login, Browser (браузер), Java Applet, база данных.

Цель лекции: научить ся создавать элементарные формы, соединяться с базой данных и запускать формы в Web.

Создание формы

Любое приложение Oracle Forms Builder состоит из одного или нескольких модулей форм. Форма (модуль) представляет собой совокупность блоков данных, элементов, холстов и программных единиц, которые определяют структуру и логику создаваемого приложения. Форма является самым верхним уровнем для всех объектов.

Построитель форм предоставляет все возможности для создания информационно-функциональной модели предметной области, для которой разрабатывается приложение.

Создание формы

Создать новую форму в Oracle Forms очень просто, для этого необходимо выполнить следующие действия:

1. Запустите файл ...\DevSuiteHome_1\BIN\frmbld.exe, если у вас Forms последней версии (Forms 9i ,10g), или ...\BIN\ifbld60.EXE, если у вас Forms 6i.
2. После того как приложение полностью загрузится, вы увидите на экране построитель Форм (рис. 3.1).
3. По умолчанию Forms Builder при запуске создает пустой модуль с именем MODULE1*. Каждому вновь создаваемому модулю присваивается имя по умолчанию, которое состоит из слова MODULEXX, где MODULE – постоянный префикс, а XX – генерируемый номер.
4. Для того чтобы создать модуль самостоятельно, выберите пункт меню Файл | Новая | Форма. Существует несколько способов создать новую форму:

* Каждому вновь создаваемому модулю присваивается имя по умолчанию, которое состоит из слова MODULEXX, где MODULE – постоянный префикс, а XX – генерируемый номер.

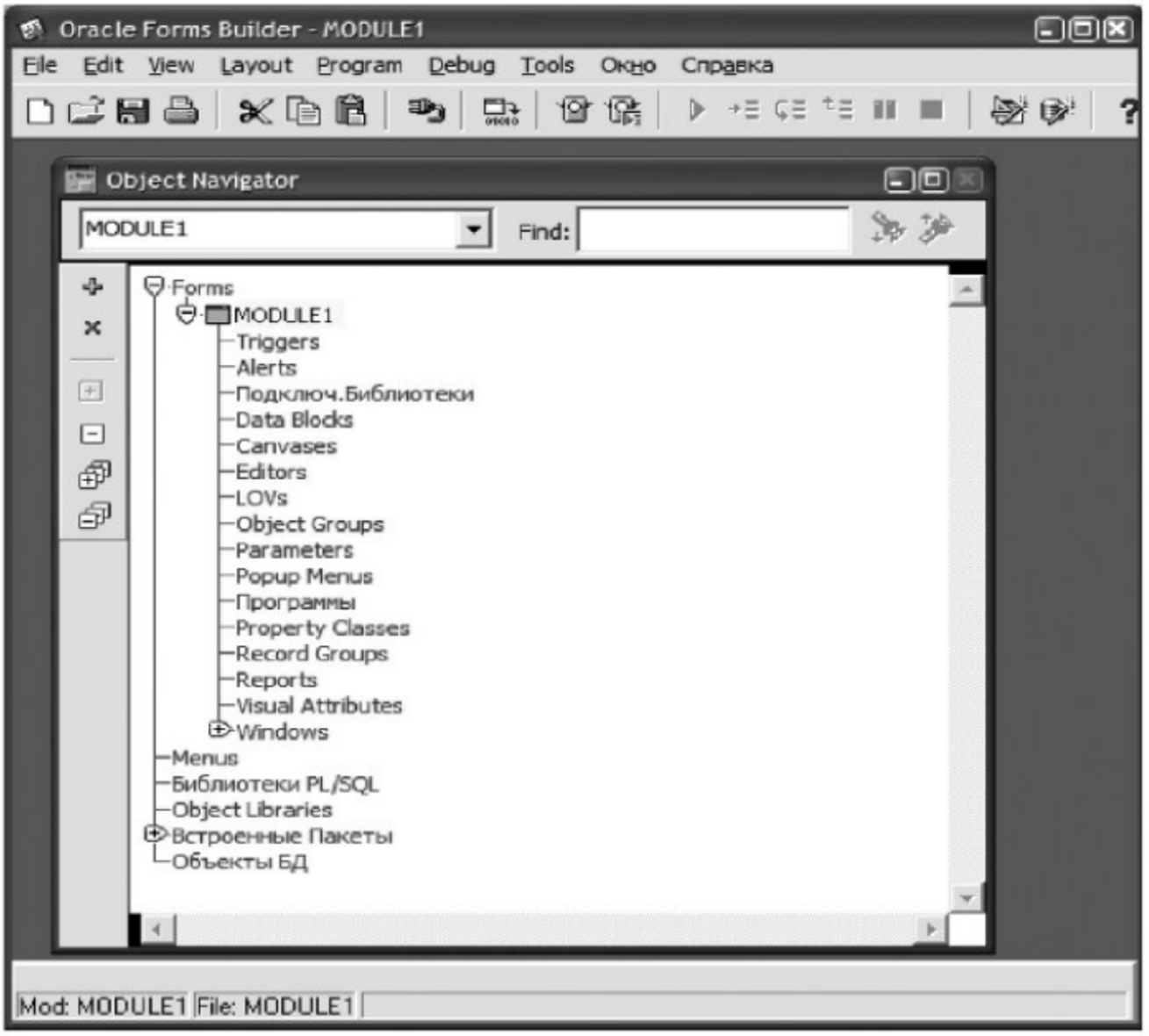

Рис. 3.1. Построитель форм (Form Builder)

- нажатием комбинации клавиш Ctrl+N или Ctrl+Insert;
- двойным щелчком левой кнопкой мыши на узле «Формы»;
- нажатием кнопки «Создать» на панели инструментов навигатора объектов;
- выбрав пункт меню Навигатор | Создать.

Примечание: *когда вы создаете форму вручную, она не содержит никаких объектов.*

Запуск формы

Форма создана и скомпилирована, теперь необходимо ее запустить. В первую очередь обратите внимание на необходимые условия запуска формы:

1. Форма должна содержать минимальный набор объектов:
   - минимум один блок данных;
   - минимум одну канву;
   - минимум один элемент.
2. Соединение с базой данных:
   - фиктивное соединение, с пустой строкой соединения;
   - нормальное соединение, с указанием имени пользователя, пароля и строки соединения.

Для того чтобы создать форму с минимальным набором объектов, выполните следующие действия:

1. Находясь в навигаторе объектов, выберите узел «блоки данных» и вызовите всплывающее меню правым щелчком мыши. Выберите пункт «Редактор разметки», после чего появится инструмент «Редактор разметки», а в навигаторе объектов автоматически сгенерируется объект Канва (Canvas).
2. Находясь в Редакторе разметки, найдите и выберите левым щелчком мыши на панели инструментов элемент «Кнопка» и начертите его на канве. После того как вы начертили кнопку, Forms автоматически сгенерировал блок данных.
3. Сохраните форму как Test.fmb командой меню File | Save as.

Теперь, когда вы обеспечили минимальный набор элементов, форма готова к запуску. В зависимости от версии Forms и от требований к приложению существует три варианта запуска формы:

- запуск формы в браузере;
- запуск формы клиент-сервер в GUI;
- запуск формы клиент-сервер в терминале.

Последние версии Forms веб-ориентированы и предназначены для запуска в веб-навигаторе (браузере); что касается более ранних версий, то они ориентированы на архитектуру клиент-сервер и запускаются в GUI- и терминальном режимах. Исключением являются формы версии Forms 6i, которые могут быть запущены во всех перечисленных режимах.

Запуск формы в графической среде (GUI) и в командной строке

В этом разделе будет описано, как запустить ваше приложение в графическом и терминальном режиме. Также мы рассмотрим возможность настройки параметров запуска с помощью утилиты Forms Runtime и параметров командной строки.

По отношению к СУБД Oracle, Oracle Forms может работать в двух режимах:

- зависимо – Oracle создает новую сессию для пользователя, прошедшего авторизацию, и большинство действий, выполняемых в форме, имеют отражение в СУБД;
- независимо – Oracle не создает новой сессии, и все действия, производимые пользователем явно или неявно, не приведут к каким-либо изменениям в БД.

Далее мы рассмотрим, как правильно соединяться с БД и что для этого необходимо.

Соединение с ORACLE

Для того чтобы выполнять операции над данными БД, для начала вам необходимо получить к ним доступ, то есть соединиться с источником. В Oracle Forms, как и в других программных средствах, для соединения с СУБД Oracle необходимо корректно указать имя пользователя, пароль и идентификатор БД (SID).

Для соединения с Oracle из Forms Builder выполните команду Файл | Соединить (рис. 3.2). Введите строку соединения с базой данных – *Имя пользователя* | *Пароль* | *Строка соединения* – и нажмите кнопку «Соединить».

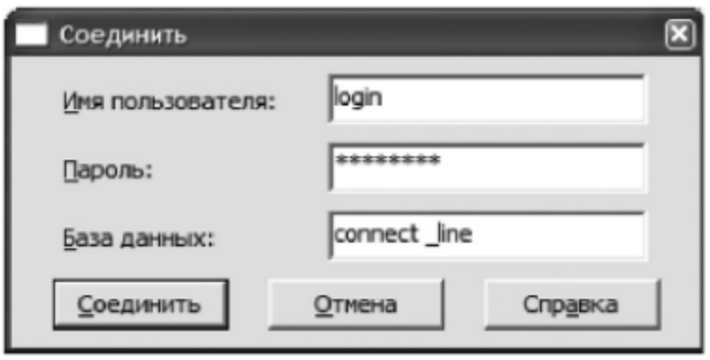

Рис. 3.2. Окно «Соединение»

Для соединения с Oracle в командной строке достаточно указать параметр USERID в команде `f60run`:

```
f60run module_name USERID=login/password@connect _line
```

Вы также можете запустить утилиту Forms Runtime (рис. 3.3), в которой сможете не только соединиться с БД, но и настроить параметры формы перед запуском. Для запуска формы с помощью утилиты Forms Runtime запустите файл \BIN\ifrun60.EXE или выполните команду ifrun60 в командной строке.

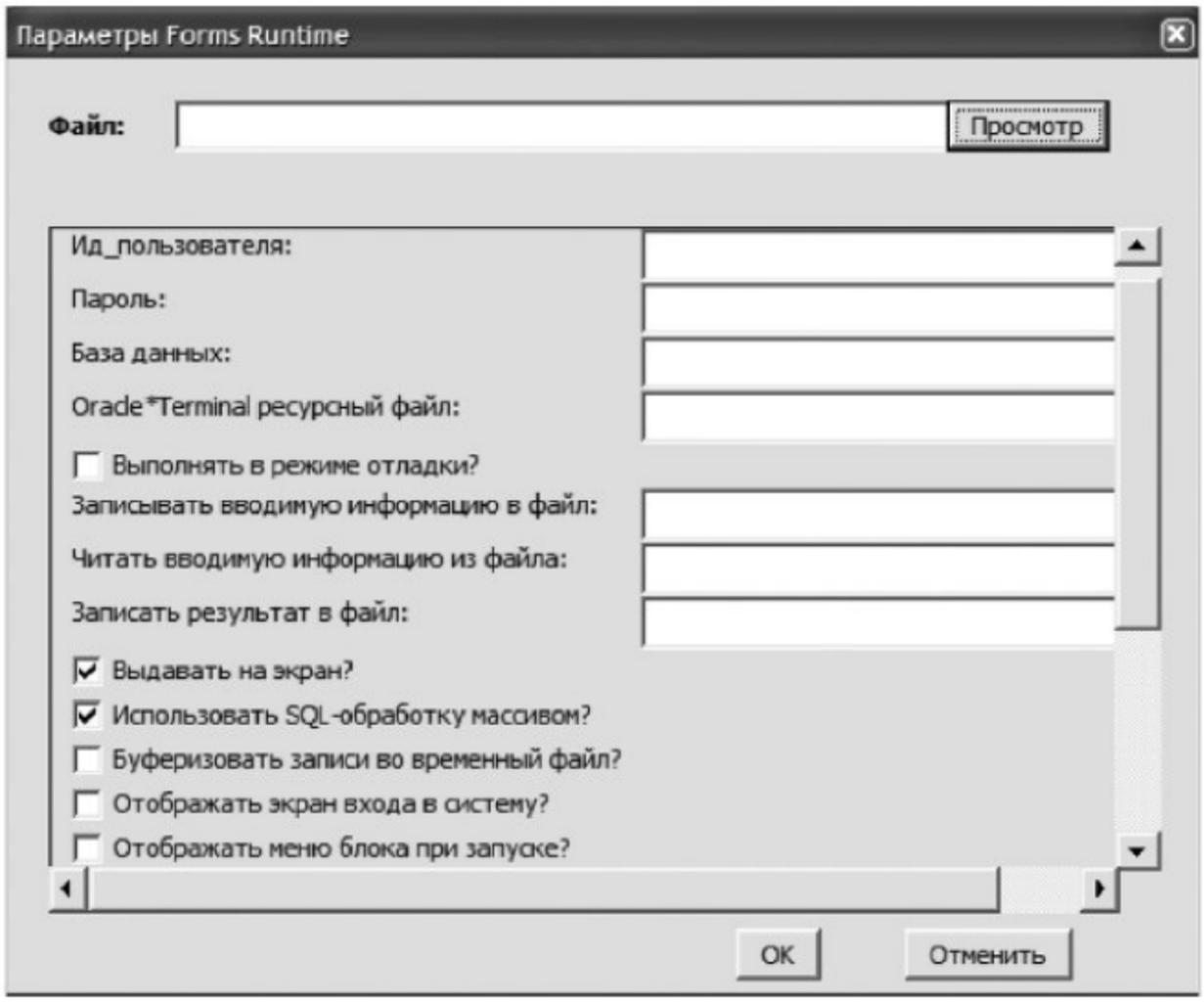

Рис. 3.3. Forms Runtime

В принципе, не обязательно подсоединяться к Oracle сразу, это можно сделать и при создании таблицы либо процедуры, в тот момент, когда вам будет предложено выбрать таблицу для создания блока данных.

> ***Примечание:*** *в Oracle Forms вы не можете соединиться с БД как SYS. Поэтому после создания БД вам необходимо создать еще одного пользователя и предоставить ему соответствующие привилегии.*

Запуск в графической среде (GUI)

Все версии Forms, начиная с Forms 4.5 и заканчивая Forms 6i, могут запускаться в пользовательском интерфейсе (GUI). Для того чтобы запустить форму в GUI, выполните следующие действия:

1. Запустите Oracle Forms командой `f60desm` или `frmbld`. После этого появится окно Form Builder (рис. 3.4, в котором вы можете выбрать параметры запуска:

- **Use the Data Block Wizard** (Создать форму с помощью мастера) – вам будет предложено создать форму с помощью мастера Data Block Wizard и Layout Wizard;
- **Build a new form manually** (Создать форму вручную) – после выбора этого параметра запустится «чистая» среда, в которой не будет блоков данных, а только стандартный набор инструментов и модуль;
- Open an existing form (Открыть существующую форму) – выбрать форму из списка существующих форм, после чего запустится выбранная вами форма;
- **Build a form based on template** – запускает Designer с предложением открыть форму;
- **Run the Quick Tour (concepts)** – запускает интернет-страницу (*.htm) с кратким обзором новых возможностей продукта Developer как в целом, так и отдельных его опций. При выборе этого пункта кроме страницы Интернет запустится Designer с опцией Build a new form manually;
- **Explore the Cue Cards (tasks)** – запускает Forms Designer, с такой же опцией, как в предыдущем случае; также в Designer сразу после запуска отобразится не только Object Navigator, но и окно справки Cue Cards, в котором вы можете найти справку о первичной работе в Forms (запуск форм, создание объектов).

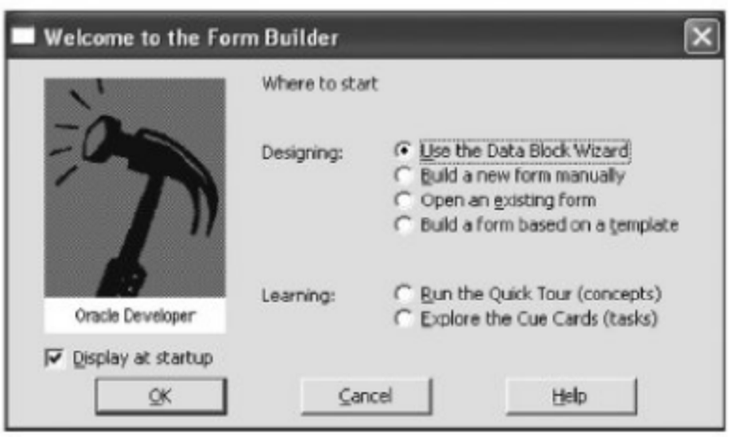

Рис. 3.4. Стартовое окно Form Builder

2. Учитывая, что вы работаете с приложением первый раз, выберите первый параметр (Use the Data Block Wizard), в этом режиме Forms на самом деле запускает последовательно два мастера:
   - мастер блока данных (Data Block Wizard);
   - мастер разметки (Layout Wizard).
3. Находясь в Редакторе разметки, нажмите кнопку «Запустить форму» с изображением светофора, расположенную на верхней панели инструментов, или выберите пункт главного меню Программа | Запуск Формы | Клиент-сервер.

Запуск формы в терминале и в GUI

Все ранние версии Forms до Forms 6i включительно поддерживают терминальный режим. Форма, запускаемая в терминальном режиме, имеет монохромный режим отображения. При выводе на экран терминала форма подавляет и не отображает те элементы, которые не поддерживаются терминалом, а именно: элементы псевдографики, OLE-контейнеры, ActiveX, Chart Item, Image Item, Bean Area. Также ранние версии не поддерживали некоторые события интерфейса, такие как передвижение и клик мыши.

Для того чтобы запустить форму, в командной строке выполните следующие действия:

1. Откройте форму из предыдущего примера и сохраните ее как Form_Term. Скомпилируйте модуль.
2. Запустите командную строку, например, SHELL, и выполните следующую команду:

```
f60run module_name USERID=login/password@connect_line
```

3. После того как команда введена, нажмите ENTER для запуска формы (см. рис. 3.4).

Вы можете использовать в командной строке параметры настройки запуска, включая их в команду `f60run`.

Для запуска формы в командной строке достаточно выполнить команду:

`f60run module_name` – для версии 6i;
`frmbld module_name` – для версии 9i и выше.

Запуск формы в браузере

Процесс запуска веб-формы существенно отличается от тех процессов, которые мы рассматривали ранее, – запуск формы в GUI или в терминале. Если вы запустили Forms Builder и выполнили все необходимые условия, то есть создали и скомпилировали приложение, то при попытке запустить форму у вас появится на экране сообщение об ошибке (рис. 3.5).

Вы можете избавиться от этой ошибки, если перед запуском формы запустите экземпляр OC4J. Запуск экземпляра OC4J производится командой `DevSuiteHome_1\j2ee\DevSuite\startinst.bat`.

Что такое OC4J, и почему без него невозможно запустить форму? OC4J (Oracle Application Server Containers for J2EE) – это компонент

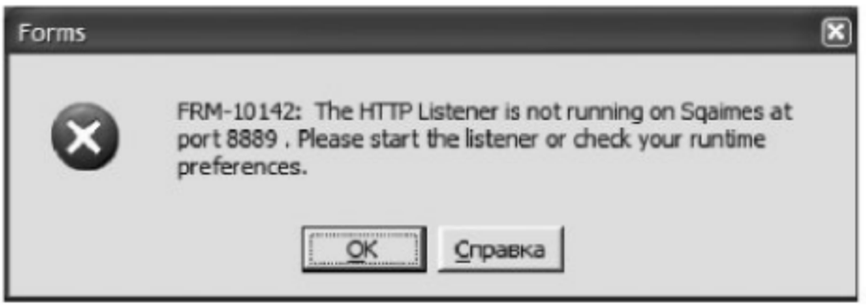

Рис. 3.5. HTTP Listener Error

Oracle AS, который является основой поддержки Java-технологий в соответствии со спецификацией Java 2 Enterprise Edition. Контейнеры OC4J написаны полностью на Java и выполняются на виртуальной машине Java, входящей в состав JDK 1.2 или 1.3. OC4J – это полная поддержка J2EE, включающая транслятор JSP, механизм выполнения Java-сервлетов и контейнер Enterprise Java Beans (EJB).

Итак, для того чтобы запустить первую форму, выполните следующие действия:

1. Запустите OC4J. После запуска OC4J у вас на экране появится Java Console (рис. 3.6).
2. Запустите Forms Builder и откройте форму Test.fmb.
3. Вызовите диалог соединения для подключения к БД командой File | Connect или с помощью кнопки «Соединение» на панели инструментов.
4. Запустите форму командой Run Form по щелчку кнопки на панели инструментов или командой меню Program | Run Form.

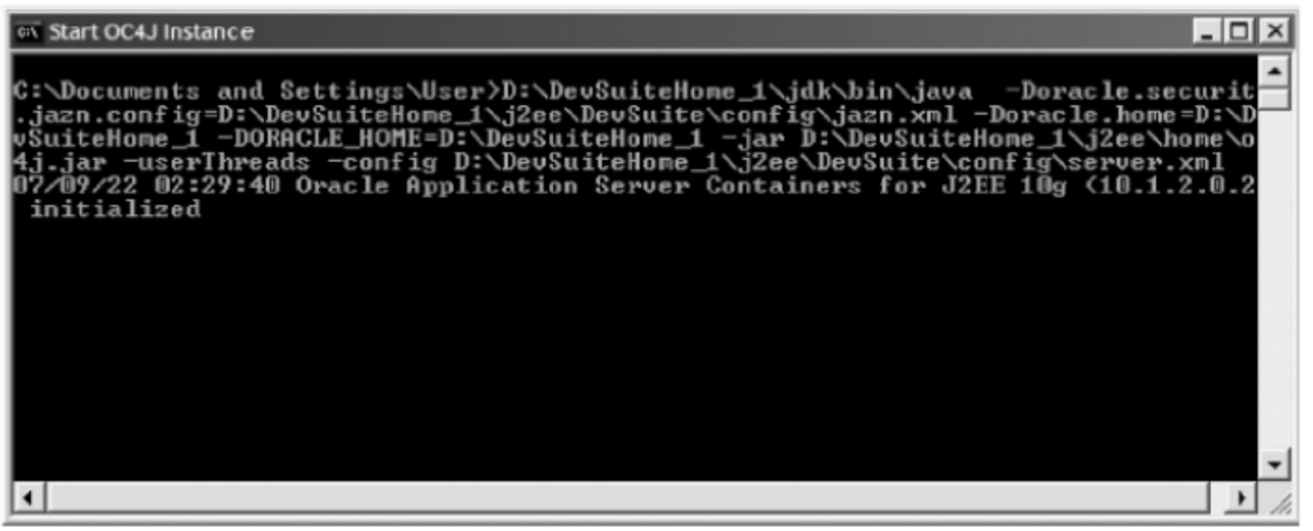

Рис. 3.6. Java Console

После запуска формы на экране появится браузер, в который будет загружен апплет, отображающий форму. В Java Console сразу же после запуска формы появятся новые записи:

```
07/09/23 10:29:21 FormsServlet init():
    configFileName:     D:\DevSuiteHome_1
    testMode:                   false
07/09/23 10:29:28 ListenerServlet init()
```

На рис. 3.7 кружочками показаны основные элементы, которые вы видите при отображении формы:

1. браузер (Browser);
2 Java applet;
3. консоль окна – строка состояния (Console);
4. URL;
5. панель инструментов меню (Menu Toolbar);
6. меню по умолчанию (Default Menu);
7. кнопка – графический элемент формы «Кнопка» (Item Button);
8. окно (Window).

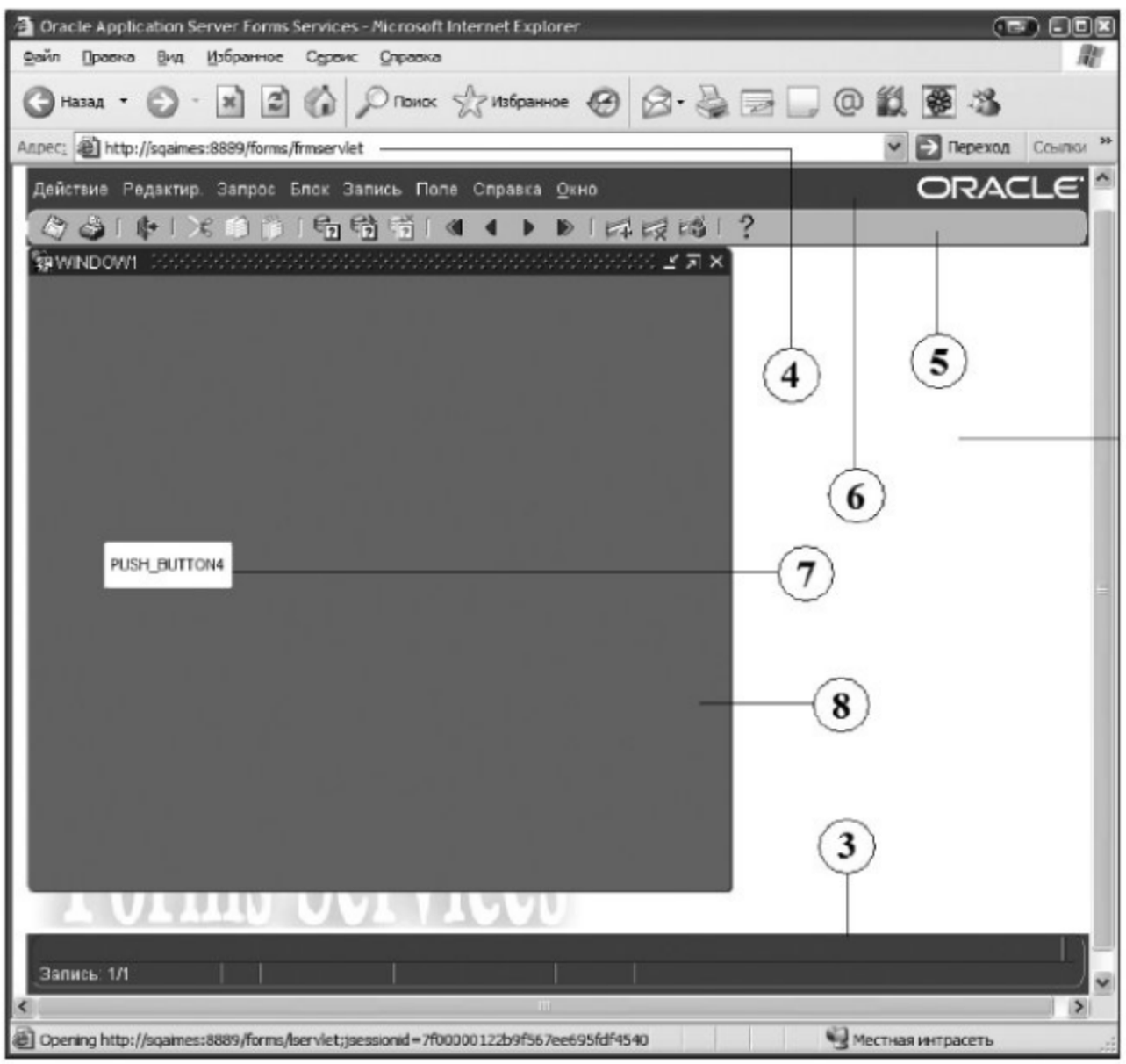

Рис. 3.7. Форма Test.fmx

Важно: если при запуске формы в браузере появилась надпись «ORACLE FORMS», апплет не загрузился и браузер самостоятельно закрылся, то вам необходимо переставить Jiniator.

Закрытие браузера

После того как вы нажимаете кнопку EXIT_FORM на главной панели инструментов, форма закрывается, а браузер остается. Для того чтобы избавиться от этой проблемы, можно воспользоваться одним из самых распространенных способов, а именно создать файл *.html, который выполняет процедуру закрытия браузера. Выполните следующие действия:

1. Создайте пустой файл с расширением *.html и вставьте в него следующий код:

```
Структура файла *.html
<html>
<head>
<script language="javascript">
function closeit()
{
win = top;
win.opener = top;
win.close ();
}
</script>
</head>
<body onload="closeit()">
close window
</body>
</html>
```

2. Сохраните файл в папке «C:\<DevSuiteHome_home>\tools\web\html» и назовите его exit.html.
3. Создайте триггер POST-FORM (form_level) и напишите в нем всего одну строчку:

```
Триггер POST-FORM
Web.Show_Document('/forms/html/exit.html','_self');
```

4. Запустите форму и попробуйте нажать кнопку EXIT_FORM, чтобы инициировать триггер POST-FORM. После нажатия кнопки форма и браузер должны закрыться.

Лекция 4. Создание форм. Режимы и свойства формы

В этой лекции слушатель знакомится с палитрой свойств формы и основными режимами формы Normal, Query и Enter Query Mode. Также в лекции будут рассмотрены настройка файла конфигурации TNSNAMES.ORA и настройка внешнего вида приложения и дизайнера форм.

Ключевые слова: TNSNAMES.ORA, Net Configuration Assistant, Oracle_home, Manifest XP, Normal Mode, Query Mode и Enter-Query Mode.

Цель лекции: обучить слушателя понимать особенности различных режимов формы и назначение свойств формы. Также слушатели научатся конфигурировать файл tnsnames.ora и настраивать внешний вид формы.

Режимы формы

Режим формы – это состояние формы, ассоциированное с определенным действием пользователя. Каждый из режимов накладывает свои ограничения на выполнение различных операций, причем очень важно знать эти ограничения, чтобы избежать возникновения исключительных ситуаций и других непредвиденных ошибок. Форма может находиться в следующих состояниях:

- Режим ввода запроса – Enter Query Mode;
- Нормальный режим – Normal Mode;
- Режим запроса – Query mode.

Для того чтобы узнать, в каком состоянии находится форма в текущий момент, необходимо проверить системную переменную – *:SYSTEM.FORM_STATUS*.

Enter Query Mode (Режим ввода запроса)

Режим ввода запроса предназначен для ввода критерия поиска данных в БД. Вы можете вводить критерии запроса в любые базовые элементы, поддерживающие этот режим. При работе в этом режиме пользователю:

Разрешено:

1. выполнение запросов, как с критерием, так и без критерия ограничения выборки;
2. подсчет записей;
3. использование Query/Where-диалога;

Запрещено:

1. перемещение в другие блоки;
2. выход из приложения;
3. выполнение операций DML – Insert (Вставка), Delete (Удаление) и Update (Обновление).

Также в режиме Enter Query запрещено выполнение некоторых триггеров. Пользователь при создании приложения может управлять поведением формы и элементов в режиме ввода запроса. На рис. 4.1 показан пример ограниченного и неограниченного запроса.

	A	B	C	D
1	Игорь	12	M	null
2	Ваня	13	M	null
3	Коля	16	M	12.01.01
4	Петя	90	M	null

Запрос без ограничения *(select ALL)*

	A	B	C	D
1	Игорь	12	M	null
2	Ваня	13	M	null
3	Коля	16	M	12.01.01
4	Петя	90	M	null

Запрос с критерием *...where D='12.01.01'*

	A	B	C	D
3	Коля	16	M	12.01.01

Рис. 4.1. Ограниченная и неограниченная выборка

Для того чтобы извлечь все данные из таблицы (Unrestricted Query), достаточно выполнить одну из перечисленных операций:

1. Нажать кнопку Execute Query на панели инструментов Menu Toolbar.
2. Перевести форму в режим запроса с помощью команды меню Query|Enter Query или нажатием кнопки на панели инструментов Enter Query и выполнить команду Execute Query.
3. Выполнить команду меню Query|Execute Query.

Для того чтобы выполнить запрос с ограничением, необходимы следующие действия:

1. Переведите форму в режим запроса.
2. Установите курсор в любой базовый элемент и введите критерий поиска.
3. Нажмите кнопку Execute Query на панели инструментов или выполните любое другое действие, ассоциированное с этой командой.

Если в качестве критерия запроса вы введете конкретное значение, то при формировании запроса оно будет интерпретироваться аналогично оператору «AND=критерий». Если же вам необходимо задействовать другие операторы ограничения выборки, такие как Like, IN или Between, то вам необходимо знать специальные символы, которые реализуют эту возможность.

Элемент	Символ/ критерий	Действие
ID	122	Извлекает строку с ID=122, то есть действует аналогично оператору «...and ID=122»
Name	%ван	Символ «% и _» действует аналогично оператору Like, то есть возвращает все имена, в которых используется сочетание «ван»
ID	#BETWEEN 100 AND 122	Символ hash «#» реализует оператор BETWEEN
ID	:I (I – :variable_name)	Вызывает Query/Where-диалог

Normal Mode (Нормальный режим)

Нормальный режим – это режим, в котором пользователь может вставлять и модифицировать данные БД, то есть выполнять операции DML. Любое действие, которое пользователь выполняет над базовым элементом, интерпретируется как вставка новой записи, удаление или изменение существующей. При работе в этом режиме пользователю:

Разрешено:

- выполнять операции DML – Insert (Вставка), Delete (Удаление) и Update (Обновление);
- выполнять запросы без ограничений выборки, то есть извлекать все записи;
- фиксировать (commit) или откатывать (rollback) изменения;
- перемещаться в другие блоки;
- выходить из приложения – завершать текущую run-time сессию.

Запрещено:

- выполнение запросов с критерием выборки;
- вызов Query/Where-диалога.

Нормальный режим, как вы уже, наверно, успели заметить, в отличие от режима Enter Query позволяет пользователю выполнять намного больше функциональных операций. Нормальный режим – это режим по умолчанию. При проектировании формы разработчик может настраивать режимы вручную или программно, переводя ее из одного режима в другой в зависимости от какой-либо ситуации.

Примечание: *если вы сделали изменения и не зафиксировали их, то такая операция, как Execute Query, то есть «выполнить запрос», становится недоступной.*

Display Error (Отображение ошибок)

Если при выполнении какой либо операции возникает ошибка или исключительная ситуация, которая отображается в строке состояния, то она может быть отображена командой `Display Error Help|Display Error`.

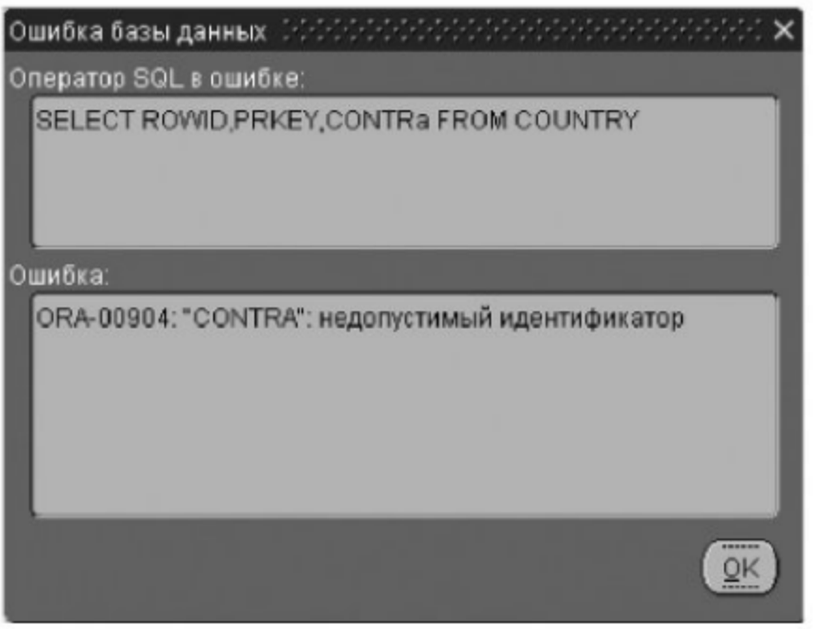

Рис. 4.2. Окно Display Error

Когда надо использовать такой способ отображения ошибки и в чем его преимущество? Самая главная особенность этого диалога в том, что он отображает не только код, но и весь текст ошибки. Когда вы получаете в строке состояния сообщение об ошибке типа «FRM-40505:Ошибка Oracle: не в состоянии выполнить запрос» или просто необработанное исключение (Unhandled exception), которое по большому счету вам ни о чем не говорит, то, вызвав окно «Display Error», вы получаете более конкретизированный ответ, а именно код и описание причины возникновения ошибки.

Свойства формы

Разрабатывая форму, вы можете управлять не только настройками объектов модуля, но и настройками самого модуля. Большинство свойств модуля глобальны и действительны для всего приложения. Изменяя настройки модуля, вы можете подключать или отключать меню, устанавливать уровень проверки БД. Разработчику доступны следующие свойства для управления настройками формы:

- Имя (Name) – внутреннее имя формы. Имя формы необязательно должно совпадать с именем модуля.
- Информация о подклассе (Subclass Information) – в этом свойстве вы можете указать Класс свойств, на котором хотите базировать форму.
- Окно консоли (Console Window) – в этом свойстве вы указываете окно, в котором будет показана консоль Forms Builder. Если вы зададите для этого свойства значение NULL, то все сообщения, которые вы определили в своей форме, не будут отображаться в статусной строке.
- Вызвана отсрочка (Defer Required Enforcement) – если значение этого свойства истинно, то проверка всех элементов, имеющих установленный атрибут «Обязательный», будет отложена до тех пор, пока не наступит проверка записи, – то есть до того момента, пока не будет осуществлена проверка записи, свойство «Обязательный» будет отключено.
- Ограничения на перемещение мышью (Mouse Navigation Limit) – в этом свойстве вы можете задать область перемещения мыши относительно текущего объекта. По умолчанию текущее значение – «Форма». Ниже перечислены объекты ограничения:
  - Форма;
  - Блок Данных;
  - Запись;
  - Элемент.
- Первый Блок Данных при перемещении (First Navigation Data Block) – в этом свойстве вы можете задать блок, к которому будет выполнена навигация при запуске формы или при ее очистке, то есть после выполнения процедуры CLEAR_FORM. Если значение этого свойства установлено в NULL, то Forms выполнит навигацию к блоку, который размещен в списке блоков Данных под номером один или к блоку, навигация к которому задана в триггере WHEN-NEW-FORM-INSTANCE.
- Группа Атрибутов Визуализации Текущей Записи – имя именованного атрибута визуализации, который используется, когда элемент входит в текущую запись.
- Уровень проверки – определяет область проверки формы во время выполнения. Вы можете устанавливать следующие уровни проверки формы во время выполнения:

  - По умолчанию;
  - Форма;
  - Блок Данных;
  - Запись;
  - Элемент.
- Режим Взаимодействия – устанавливает режим блокирования для взаимодействия с Базой Данных.
- Максимальное Время Запроса – в этом свойстве вы можете определить максимальное время, которое форма будет ожидать выполнения запроса, после чего выполнит прерывание.
- Максимум Выбранных Записей – это свойство определяет максимальное число выбранных записей, после чего выполнение запроса будет прервано.
- Режим Изоляции (Isolation Mode) – уровень изоляции транзакции.
- Канва с Горизонтальной/Вертикальной Панелью Инструментов – это свойство характерно только для окна MDI, так как выводит для него горизонтальную/вертикальную панель инструментов.
- Направление – задает направление текста «Слева направо» или «Справа налево», другими словами, определяет позицию курсора ввода.

Выбор системы координат

Используя Палитру Свойств, вы можете определять, в какой системе координат даны размеры и позиции в символьных ячейках. Чтобы определить систему координат для формы, выполните следующие действия:

1. Находясь в Навигаторе Объектов, запустите Палитру свойств модуля формы.
2. Найдите и выберите свойство «Система координат» для вызова окна «Информация о координатах» (рис. 4.3.).

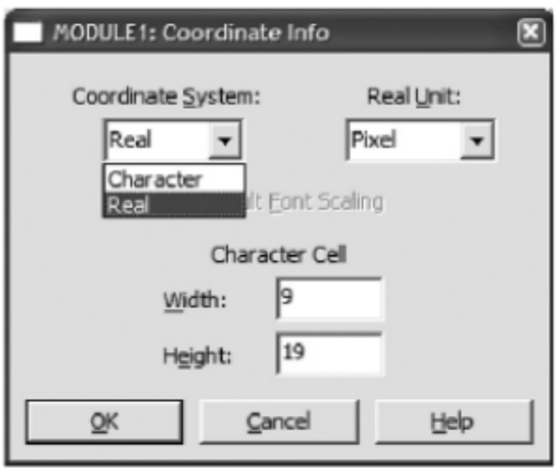

Рис. 4.3. Окно «Информация о координатах»

3. В поле «Система координат» вы можете выбрать тип системы – Абсолютная (Real) или Символьная (Character). Единицы измерения доступны только для абсолютной системы координат, что же касается Символьной системы, то тут вы ограничиваетесь заданием ширины и высоты символьной ячейки. В Символьной системе координат единицы измерения не используются. В абсолютной системе координат возможны следующие единицы измерения:
   - пиксел;
   - сантиметр;
   - дюйм;
   - пункт;
   - десятичная точка.
4. Выберите Абсолютную систему координат, а в качестве единицы измерения – Пиксел.
5. При выборе значения выключателя «Масштабирование шрифта по умолчанию» установки высоты и ширины символьной ячейки становятся недоступными. Если значение выбрано, Forms самостоятельно выполняет масштабирование шрифта.
6. Отмените выбор «Масштабирования шрифта по умолчанию» и установите ширину ячейки, равную 7, а высоту – 14.

Для того чтобы изменения вступили в силу, нажмите кнопку «OK» – или нажмите кнопку «Отмена», чтобы вернуть значения по умолчанию.

Конфигурирование TNSNAMES.ORA

Чтобы соединиться с Базой Данных, вам необходимо настроить соответствующий сервис.

Файл TNSNAMES.ORA – это файл, предназначенный для определения сетевого имени сервиса, с помощью которого можно обращаться к БД.

Net Configuration Assistant установление соединения с локальными БД

Если вы хотите установить соединение между Oracle Forms и БД Oracle, выполните следующие шаги:

1. Проверьте, запущен ли Forms, если да, то закройте его.
2. Если запущен OC4J Instance, то остановите и его.
3. Скопируйте файл TNSNAMES.ORA из ORACLE_HOME\NETWORK\ADMIN и замените им файл, расположенный в DevSuiteHome_1\NETWORK\ADMIN.

4. Файл TNSNAMES.ORA – это файл, предназначенный для определения сетевого имени сервиса, с помощью которого можно обращаться к БД.
5. Выполните команду `\network\tools\netca.cl` или Пуск|Программы|Oracle—DevSuiteHome\Configuration and Migration Tools|Net Configuration.
6. При запуске Net Configuration Assistant на экране появится приветственное окно с предложением выбора последующих действий. Выберите опцию «Local Net Service Name Configuration» для работы с файлом TNSNAMES.ORA и нажмите «Далее» (рис. 4.4).
7. В следующем окне вам будет предложено выполнить следующие действия:

Рис. 4.4. Окно приветствия Net Configuration Assistant

- Add – создать новый сервис;
- Modify – изменить существующий сервис;
- Delete – удалить сервис;
- Rename – переименовать сервис;
- Test – протестировать существующую конфигурацию.

Так как мы создаем новый сервис, то выбираем опцию Add и нажимаем кнопку «Следующий».

8. В следующем окне (рис. 4.5) вам будет предложено выбрать имя сервиса; укажите соответствующий на свое усмотрение и нажмите кнопку «Следующий».

Рис. 4.5. Выбор имени сервиса

9. В следующем окне (рис. 4.6) вам будет предложено выбрать сетевой протокол. Выберите TCP и переходите дальше.

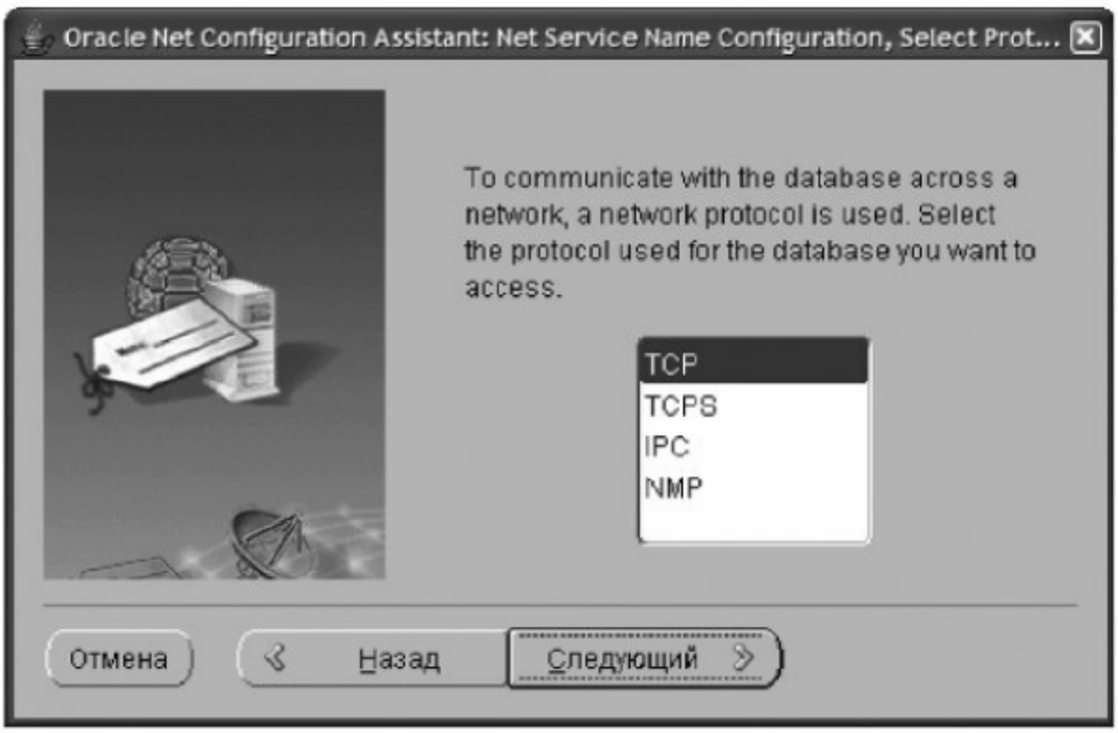

Рис. 4.6. Окно выбора сетевого протокола

10 Далее вам будет предложено выбрать порт и имя сервера, на котором находится ваша БД (рис. 4.7). После того как вы указали необходимые данные, нажмите кнопку «Следующий» для перехода к заключительному шагу.

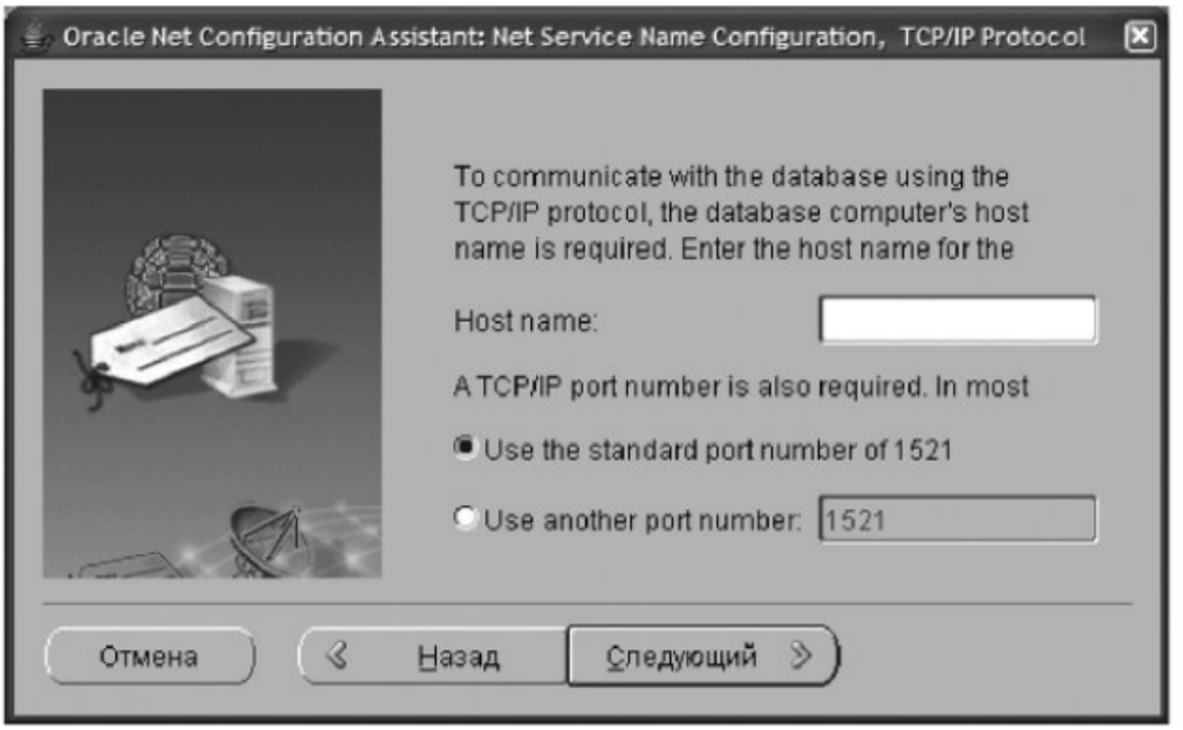

Рис. 4.7. Окно настройки соединения

11. Следующий этап необязателен, он предназначен для тестирования созданного сервиса, поэтому жмите кнопку «Готово» для окончания работы с Net Configuration Assistant.

После того как вы завершите работу с Net Configuration Assistant, он автоматически обновит файл TNSNAMES.ORA.

Формы в стиле XP

Построитель экранных форм, особенно ранних версий, имеет, так скажем, не слишком дружественный интерфейс. В этом нет ничего плохого, так как основная идея, которую преследовали разработчики этого продукта, – это функциональность и быстродействие. Начиная с версии Forms 4.5 вы можете совершенствовать внешний вид формы, используя элементы шаблонной графики, визуальные эффекты и изображения.

Если вы разрабатываете ваше приложение в Windows XP, вы можете применить манифест XP (XP Manifest) к вашим формам и самой среде разработки. Применив стиль XP к своим формам, вы добьетесь такого же эффекта при отображении элементов, как и в самой операционной

системе. Если провести аналогию, то в Delphi для того, чтобы придать приложению стиль XP, достаточно разместить на форме одноименный компонент Manifest XP. В Oracle Forms такого компонента нет, поэтому мы пойдем другим путем.

1. Закройте Oracle Forms.
2. Войдите в корневую папку Windows и запустите окно «Свойства папки». Для этого выберите пункт меню Сервис|Свойства папки.
3. Выберите вкладку «Вид» окна «Свойства папки» и выберите радио-кнопку «Показывать скрытые файлы и папки».
4. Найдите файл WindowsShell.Manifest и скопируйте его в папку BIN корневого каталога Forms.
5. Переименуйте WindowsShell.Manifest в ifbld60.exe.manifest
6. Скопируйте еще один файл WindowsShell.Manifest в папку BIN и назовите его ifdbg60.exe.manifest.
7. Скопируйте третий файл и назовите его ifrun60.exe.manifest.

После того как переименуете все файлы, запустите Forms. Попробуйте создать новые элементы и запустить форму или откройте существующую. Все элементы в вашей форме будут отображены в стиле XP. Файл ifbld60.exe.manifest назначает стиль XP для самого построителя форм, а файлы ifdbg60.exe.manifest и ifrun60.exe.manifest управляют отображением элементов в режиме выполнения. Ниже приведен фрагмент (рис. 4.8) приложения до и после применения стиля.

Чтобы отменить стиль XP для формы, удалите созданные файлы.

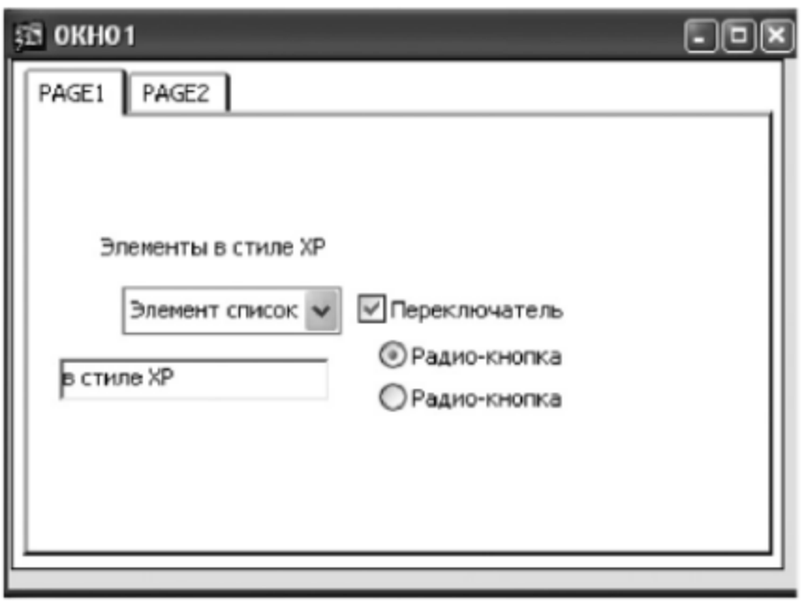

Рис. 4.8. Форма в стиле XP

Запуск формы без соединения с Базой Данных

Не всегда в нашей практике возникает необходимость соединения формы с базой, иногда есть потребность в управляющих приложениях, которые не выполняют взаимодействие с БД.

Каждый раз при открытии построителя форм и попытке запуска приложения Forms отображает на экран предупреждение с предложением соединения с Базой Данных (рис. 4.9).

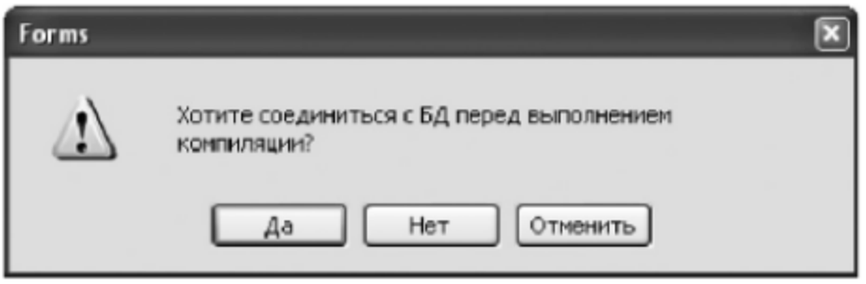

Рис. 4.9. Приглашение соединения с БД

Если вы нажимаете кнопку «Да», то на экране появляется новое окно – «Соединение» (рис. 4.10), где вам предлагается ввести строку соединения с БД. Если вы нажимаете кнопку «Отмена», то Forms возвращает вас обратно в Designer, закрывая окно выполнения. Кнопка «Нет» продолжит выполнение, правда, по сути ситуация останется прежней, Forms вновь запросит строку соединения (рис. 4.11). Если вы нажмете кнопку «Отменить», то Forms, как и в предыдущем случае, закроет окно выполнения и вернет вас в Designer.

Рис. 4.10. Окно «Соединение»

Рис. 4.11. Повторное приглашение соединения с БД

Чтобы избежать этих приглашений, выполните следующий пример:

1. Создайте на уровне формы триггер ON-LOGON. Для этого, находясь в Объектном Навигаторе, выделите название формы и вызовите всплывающее меню правым щелчком мыши. В появившемся меню выберите пункт Универсальные триггеры | Другие.
2. В окне «Триггеры» найдите триггер ON-LOGON и подтвердите выбор (рис. 4.12).

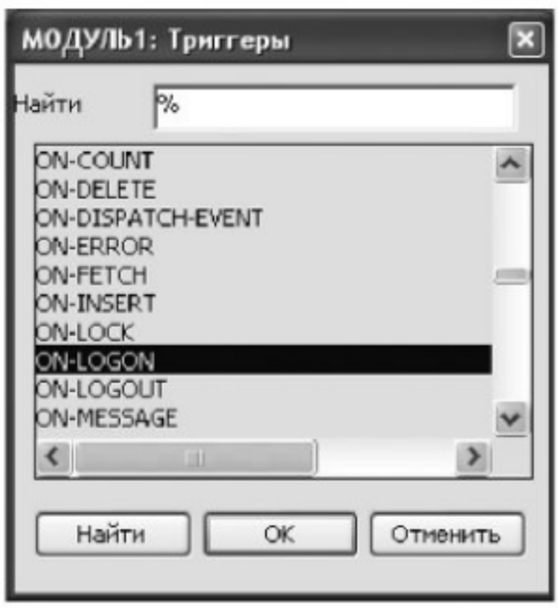

Рис. 4.12. Окно «Триггеры»

3. После того как вы подтвердили выбор, на экране появится PL/SQL-редактор. Наберите в нем следующий код и нажмите кнопку «Компиляция»:

```
logon('','');
```

4. Закройте редактор PL/SQL и вернитесь в Навигатор. Теперь выберите в главном меню Инструменты|Редактор Разметки. На левой панели Редактора Разметки найдите элемент «Кнопка» и нарисуйте ее на канве. Это необходимо сделать для того, чтобы форма запустилась, так как Forms не запустит окно, если в нем не будет канвы и элемента.
5. Запустите форму на выполнение. При появлении окна «Соединение» нажмите кнопку «Нет», и вы увидите окно формы с созданной вами кнопкой.
6. Для выхода из формы выберите меню «Действие|Выход» или нажмите кнопку с изображением дверцы.

Настройка Logon screen. Повторное запрашивание пароля

Несмотря на то что в Forms предусмотрен только один диалог аутентификации, который можно вызывать с помощью встроенной процедуры LOGON_SCREEN, вы все же можете вызвать и другой диалог, а именно PASSWORD EXPIRE. Этот тип диалога отличается от стандартного тем, что запрашивает повторный ввод и подтверждение нового пароля. Особенность такого типа аутентификации в том, что вновь введенному паролю будет присвоен статус устаревшего.

PASSWORD EXPIRE – паролю немедленно будет присвоен статус устаревшего. Пользователю придется сменить пароль перед первым использованием своей учетной записи (аккаунта).

Чтобы реализовать этот тип диалога, вам не придется выполнять в Forms какие-либо действия, вам всего лишь нужно создать Пользователя с опцией PASSWORD EXPIRE. Для создания пользователя с опцией PASSWORD EXPIRE выполните следующую команду:

```
CREATE USER Sqaimes IDENTIFIED BY sqaimes
PASSWORD EXPIRE;
```

Чтобы проверить, как это работает, запустите Forms и попробуйте зарегистрироваться под новым пользователем.

Лекция 5. Инструменты проектирования. Виды и назначения

В этой лекции рассматриваются инструменты, необходимые для разработки приложения: Объектный Навигатор, Палитра свойств, Редактор разметки.

Ключевые слова: Объектный навигатор (Object Navigator), Редактор разметки (Layout Editor).

Цель лекции: ознакомить слушателя с основными инструментами проектирования форм в Forms Developer.

Oracle Forms функционирует в среде графического интерфейса (GUI), например, Microsoft Windows, Linux. Основные инструментальные средства, которые используются для разработки, навигации и изменения свойств формы и элементов, – это Навигатор объектов (Object Navigator), Редактор Разметки (Layout Editor) и Инспектор свойств (Object Properties Sheets). Далее подробнее остановимся на каждом из перечисленных инструментов.

Объектный навигатор (Object Navigator)

Как вы уже смогли убедиться, глядя на рис. 5.1, Объектные Навигаторы концептуально между собой ничем не отличаются. В Объектном Навигаторе отображаются все объекты Oracle Forms: элементы блоков данных, модули, блоки, библиотеки, параметры, меню, объекты БД, стандартные встроенные пакеты Forms. Элементами Объектного Навигатора можно манипулировать при помощи кнопок и элементов главного меню. Навигатор представляет собой иерархическое дерево, поэтому внутри Объектного Навигатора компоненты вашего приложения представлены в виде иерархии, что позволяет разработчику видить сгруппированную и упорядоченную структуру создаваемого приложения.

В Объектном Навигаторе вы можете выполнять следующие действия:

- Навигатор представляет собой иерархическое дерево, поэтому в нем осуществляются такие действия, как: раскрывать и сворачивать элементы для быстрого поиска объектов, отображать соответствующие иконки напротив выбранных объектов.
- Выбирать объекты – выбранный вами объект становится выделенным и в редакторе разметки и таблице свойств (если включен режим синхронизации). При выборе любого элемента в навигаторе главное

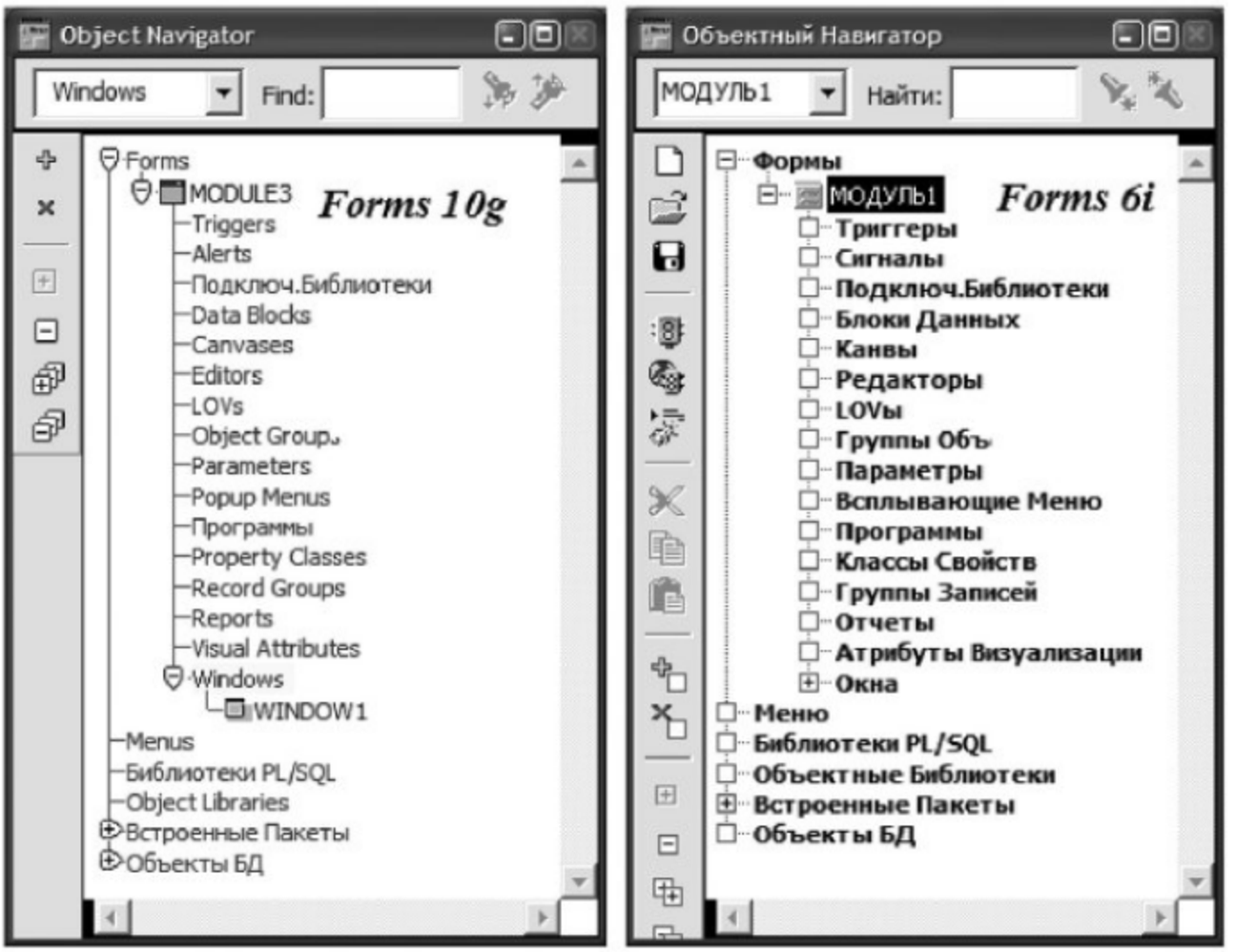

Рис. 5.1. Объектный навигатор

меню Forms Designer будет подсвечивать только те пункты меню, которые будут соответствовать выбранному вами элементу.

- Создавать и удалять объекты, библиотеки или элементы в вашем приложении.
- Возможность перенести один элемент (библиотеку, объект) в другой модуль обычным копированием, то есть возможность перемещать объекты, как в модуле, так и между модулями.
- Быстро находить объекты с помощью «Быстрого поиска», причем поиск ведется без учета регистра и достаточно ввода одной буквы.
- Двойным щелчком по триггеру или программному модулю вы можете вызвать Редактор PL/SQL.
- Быстрый доступ к командам и программным модулям элемента – достаточно нажать правой кнопкой мыши для выпадения всплывающего меню, пункты которого будут соответствовать выбранному элементу.
- Установка меток – используется для пометки элемента в Навигаторе. Для создания метки достаточно выделить объект в Навигаторе, перейти в меню Navigator, а затем выбрать пункт -> Add Bookmark

(добавить метку) или перейти к уже созданной метке командой -> Go to Bookmark.

- Переименование объектов. Для этого нужно выбрать объект для переименования, затем щелкнуть на его имени, после чего появится прямоугольная область с мигающим курсором, в которой вы присваиваете новое имя вашему объекту. По умолчанию при создании объекта ему присваивается имя с последовательным номером, например, BLOCK1, LOV8 и т. д.
- Множественное выделение – выделение группы элементов. Для этого нужно с нажатой клавишей Control щелкать левой кнопкой мыши по элементам, которые вы хотите выделить. Перемещаться как часть множественного выбора могут только объекты одного и того же типа.
- Возможность запуска и генерации приложения. Для этого достаточно выделить любой элемент, принадлежащий тому модулю, который вы хотите запустить или сгенерировать в исполняемый файл меню, библиотеки или формы, а затем выбрать нужную вам команду запуска или генерирования в главном меню Forms'a — «Tools» либо меню «Файл->Администратор...».

По умолчанию в Oracle Formsопция Synchronize включена, а это значит, что все открытые Layout Editor, Table Properties и Menus Editor поддерживают одни и те же выбранные объекты. То есть при выборе какого-либо элемента в Навигаторе или Редакторе Разметки он становится текущим везде. Если элемент выбран в Редакторе Разметки, то он также будет выделен в Навигаторе, и наоборот.

При выключенной синхронизации выбор объекта в Навигаторе или в Редакторе не приведет к выделению этого же элемента в других окнах.

Элементы, которые не принадлежат ни одной вид-картинке (Canvas), отображаются на Null-канве; точно так же и Canvas, не имеющий своего окна, отображается в Null Window.

Правила принадлежности копирования и перемещения объектов в пределах одного модуля или между несколькими модулями:

- При копировании или перемещении объекта вы также перемещаете (копируете) принадлежащие ему объекты. К примеру, при копировании блока копируются также и триггеры, подсоединенные к этому блоку, и все элементы, принадлежащие этому блоку, и триггеры, подсоединенные к этим элементам.
- Принадлежность объекта и простые связи, которые могут существовать между объектами, нужно отличать, т. к. при создании окна ему назначается вид-картинка, но эта вид-картинка не будет однозначно принадлежать этому окну, т. к. и окно, и Canvas принадлежат форме. Отношения между окном и Canvas – это просто связь, которую вы

определяете установкой свойства Canvas Window, можете поменять и переназначить в любой момент и любому окну, и любому блоку.

- Правила принадлежности для объектов в модулях формы могут быть просто установлены следующим образом: элементы и отношения принадлежат блокам; триггеры могут принадлежать элементам, блокам или формам; все остальные объекты принадлежат формам. Причем при копировании нужно соблюдать согласованность элементов – это значит, что элемент, принадлежащий блоку, нельзя переместить в узел, в котором находятся программные модули, или в узел Parameters.
- Правила принадлежности для модулей меню: элементы Menu принадлежат меню; меню принадлежат модулям меню; все остальные объекты принадлежат модулям меню.
- Модули библиотек содержат только подпрограммы и пакеты, которые все принадлежат модулю библиотеки и не могут содержать больше никаких других элементов.

Теперь кратко рассмотрим все элементы Навигатора.

Триггеры (Triggers)

Триггеры Oracle Forms – это функции, программные блоки PL/SQL, которые выполняются при возникновении какого-либо действия или условия в форме. Например, при запуске формы срабатывает триггер WHEN_NEW_FORM_INSTANCE («при запуске формы»), что дает вам возможность выполнять какие-либо действия до запуска формы: изменить размеры формы, извлечь данные и т. д. Поэтому эти действия называются событиями, т. к. возникают в ответ на какое-либо действие. Чаще всего триггеры используют для отключения или запрета (запрет на выполнения DML), модификации или расширения значения по умолчанию. Триггеры определяются для формы, блока, записи или для каждого поля в отдельности, причем триггер, созданный для данного элемента, будет срабатывать только для этого элемента. Триггер в Forms'е могут вызывать друг друга, поэтому нужно учитывать приоритеты срабатывания триггера в зависимости от уровня его определения.

Предупреждения (Alerts)

Предупреждения используются для вывода дополнительной информации или сообщения, написанного разработчиком для обработки исключительной ситуации, а также различных предупреждений перед выполнением какой-нибудь операции, которая требует ответа или подтверждения от

пользователя. Изменять или присваивать какие-либо функции предупреждениям вы можете с помощью таблицы свойств этого компонента, где вам будет предложено выбрать тип предупреждения, количество кнопок в его окне и т. д. Окно свойств этого объекта можно вызвать двойным щелчком мыши, выбрав пункт главного меню Tools->Property Pallete, клавишей F4 или через всплывающее меню. Создается также двойным щелчком по узлу Alerts, через Навигатор или меню Navigator-> Create.

Библиотеки (Attached Libraries)

Attached Libraries (подсоединяемые библиотеки) – это очень удобные специальные модули Oracle Forms, в которые вы можете «засунуть» часто используемые вами процедуры и функции. При двойном щелчке мышью по этому объекту будет вызвано окно Attche Library (рис. 5.2).

Рис. 5.2. Attche Library

В строке Library нужно указать имя и размещение библиотеки, которую вы хотите подсоединить. Для этого вам необходимо выполнить следующие действия:

1. С помощью кнопки Browse вызвать окно PL/SQL Library File (размещение библиотеки PL/SQL) и выбрать нужную вам библиотеку либо самостоятельно набрать путь к библиотеке и ее имя с расширением *.pll.
2. С помощью радиокнопок выбрать, откуда будет подсоединена библиотека: в нашем случае это File System, то есть из вашей файловой системы, соответственно, Database — из вашей БД, Either – из другого источника.
3. После этого нажмите кнопку Attach – выполнить подсоединение, Cancel – отменить присоединения библиотеки и Help – вызов справки по данному объекту.

Блоки (Blocks)

Блоки – этот узел содержит созданные вами «Блоки данных»; по умолчанию Oracle Forms в качестве названия блока берет имя таблицы (пред-

ставления), на основе которой он строился. Причем блоки необязательно должны быть привязаны к объекту базы данных (таблице, представлению), поэтому могут быть созданы и при этом не относиться ни к одной таблице – так, например, объект Tree View всегда требует одиночного размещения в блоке данных, а также такие блоки используются для содержания какой-либо временной информации, вспомогательных элементов, могут быть обычным отображателем информации (например «Об авторе»), различных индикаторов и т. д. **Базовыми** или **основными блоками таблицы** называются блоки, которые относятся к объекту (таблице, снимку, представлению).

Элементы блоков (Items)

Элементы сгруппированы в Навигаторе объектов внутри соответствующих блоков. Элемент соответствует одиночному значению данных или полю. Элементы могут содержать столбцы базы данных или использоваться в качестве контейнера для других связанных данных.

Связи блоков (Relationships)

Связи блоков определяют реляционную взаимосвязь между блоками данных формы. Реляционная связь в Oracle Forms рассматривается как объект и имеет свои настраиваемые свойства. Связи, созданные в Forms, существуют только в пределах данного приложения и не распространяются на другие.

Раскладки объектов (Canvas-Views)

Раскладка объектов (canvas, от англ. «холст») – структура, предназначенная для размещения объектов формы. Существует 4 типа раскладок: содержательный (content), стековый (stacked), горизонтальная и вертикальная панель инструментов (Horizontal/Vertical Toolbar Canvas-Views). В большинстве случаев используется один из двух типов раскладок – либо содержательный, либо стековый.

- Содержательная раскладка не предполагает наличия еще каких-либо раскладок, она занимает все окно формы.
- Стековую раскладку можно размещать поверх других раскладок окна формы.
- Горизонтальная и вертикальная панель инструментов – набор кнопок-иконок с определенными для них пользовательскими командами.
- Есть еще один специальный тип раскладки – пустой (null). Пустая раскладка используется для размещения объектов, не требующих отображения в окне формы.

Редактор (Editor)

Редактор – это окно для просмотра и изменения объемных полей данных. В таких полях могут содержаться комментарии пользователя, справки, советы, большие коментарии и т. п., которые не вмещаются в поле, отведенное для отображения элемента.

Списки значений (LOVs)

Списки значений (**LOVs** – *lists of values*) предназначены для выбора возможных значений некоторого поля. Списки значений представляют собой данные, содержащиеся в именованном объекте. Список значений может использоваться и как проверка правильности ввода данных пользователем.

Группы объектов (Object Groups)

Группы объектов – это специальный механизм упаковки различных объектов формы в так называемый контейнер, который может потом использоваться при создании других форм (важно, что при этом теперь требуется внесение минимума изменений). После создания группы объектов как элемента формы любой из прочих объектов формы может быть скопирован в группу объектов.

Данная возможность Forms представляет собой один из элементов объектно ориентированного подхода, который значительно повышает качество разработки, т. к. позволяет использовать уже созданные части при разработке нового продукта.

Параметры (Parameters)

Параметры можно определить для обеспечения начального ввода данных в форму. Обычно параметры используются для передачи значений из одной формы в другую (в приложениях со многими формами). В ранних версиях Developer/2000 единственным методом передачи значений между формами было применение глобальных переменных, которые очень неэффективно использовали доступную память. Использование глобальных переменных возможно и сейчас, но рекомендуется применять параметры в случаях, когда структура передаваемых данных специфична для конкретных форм (не носит универсальный характер для всех форм приложения).

Программные модули (Program Units)

Программные модули – это PL/SQL-процедуры, пакеты или функции, которые можно вызывать из триггеров формы. Программный модуль – понятие, полностью эквивалентное понятию подпрограммы в языках 3-го поколения. Типичным кандидатом в программные модули является фрагмент кода, который используется более чем в одном триггере. Программные модули также могут применяться для разбиения больших программ на логически независимые части.

Классы свойства (Property Classes)

Класс свойства определяет свойства класса объектов. В случаях, когда множество объектов формы имеют идентичные свойства, классы свойства хороши тем, что гарантируют разработчику непротиворечивость объектов. Классы свойства позволяют увеличить степень стандартизации модуля формы, что сокращает общее время разработки.

Группы записей (Record Groups)

О группе записей можно думать как о виртуальной таблице в памяти. Группы записей — это структурированные наборы данных, которые могут использоваться для передачи данных между модулями прикладных программ либо для заполнения списков значений или других элементов списка.

Визуальные атрибуты (Visual Attributes)

Визуальный атрибут объекта определяет цвет, шрифт и характеристики стиля для каждого элемента. В то время как каждое из этих значений может быть установлено индивидуально для каждого объекта, визуальный атрибут обеспечивает механизм опредения правильной комбинации визуальных характеристик для формы. Высококачественное приложение не должно применять много различных визуальных атрибутов в пределах элементов. Твердо придерживаясь набора визуальных атрибутов, формы могут использовать специфические характеристики элемента, чтобы показать определенное значение.

Окна (Windows)

Окно – это рамка, в пределах которой форма появляется на экране пользователя. Каждый холст привязан к определенному окну в форме, и

несколько холстов могут быть привязаны к одному окну. Простая форма может содержать несколько окон или состоять из одного окна.

Таблица свойств (Object Properties Sheets)

Инспектор объектов или таблица свойств (рис. 5.3) – это окошко в виде таблицы, состоящей из двух колонок: Property Name (название свойства), Property Value (значение свойства), а также:

- верхняя строка состояния – отображает имя текущего объекта, в нашем случае – Data Block: Block1;
- Property Hint (строка состояния) – отображает краткую справку по выбранному вами свойству;
- Find (строка поиска) с двумя опциями поиска – Search Forward (искать следующее), Search Backward (искать предыдущее) – для поиска нужного вам свойства по его имени (достаточно ввода одной заглавной буквы наименования свойства).

В левом верхнем углу располагаются кнопки (перечисляю слева направо):

- Copy Properties – копирует в буфер установки свойств, выведенных в текущий момент в окне Properties;
- Paste Properties – вклеивает установки свойств из буфера в список свойств в окне Properties. Свойства, которые есть в буфере, но отсутствуют в списке свойств, игнорируются;
- Add Property – выводит список значений (LOV), из которого вы можете выбрать свойство для добавления в класс свойств. Эта команда доступна только в том случае, если единственный текущий выбранный объект – класс свойств;
- Delete Property – удалить свойство;
- Property Class (свойство класса) – по нажатию создает класс свойств в Объектном Навигаторе;
- Inherit – устанавливает текущее свойство на определение по умолчанию. Если список свойств базируется на классе свойств, который включает текущее свойство, то установкой по умолчанию является установка, определенная в этом классе. Если текущий список свойств не базируется на классе свойств или базируется на классе, который не включает текущее свойство, то Inherit устанавливает свойство на установку Oracle Forms по умолчанию;
- Intersection/Union – переключает окно Properties между режимами вывода целиком и пересечений. Эта опция определяет, какие свойства будут показываться в случае, если одновременно будет выбрано более одного объекта (множественный выбор). В режиме пересече-

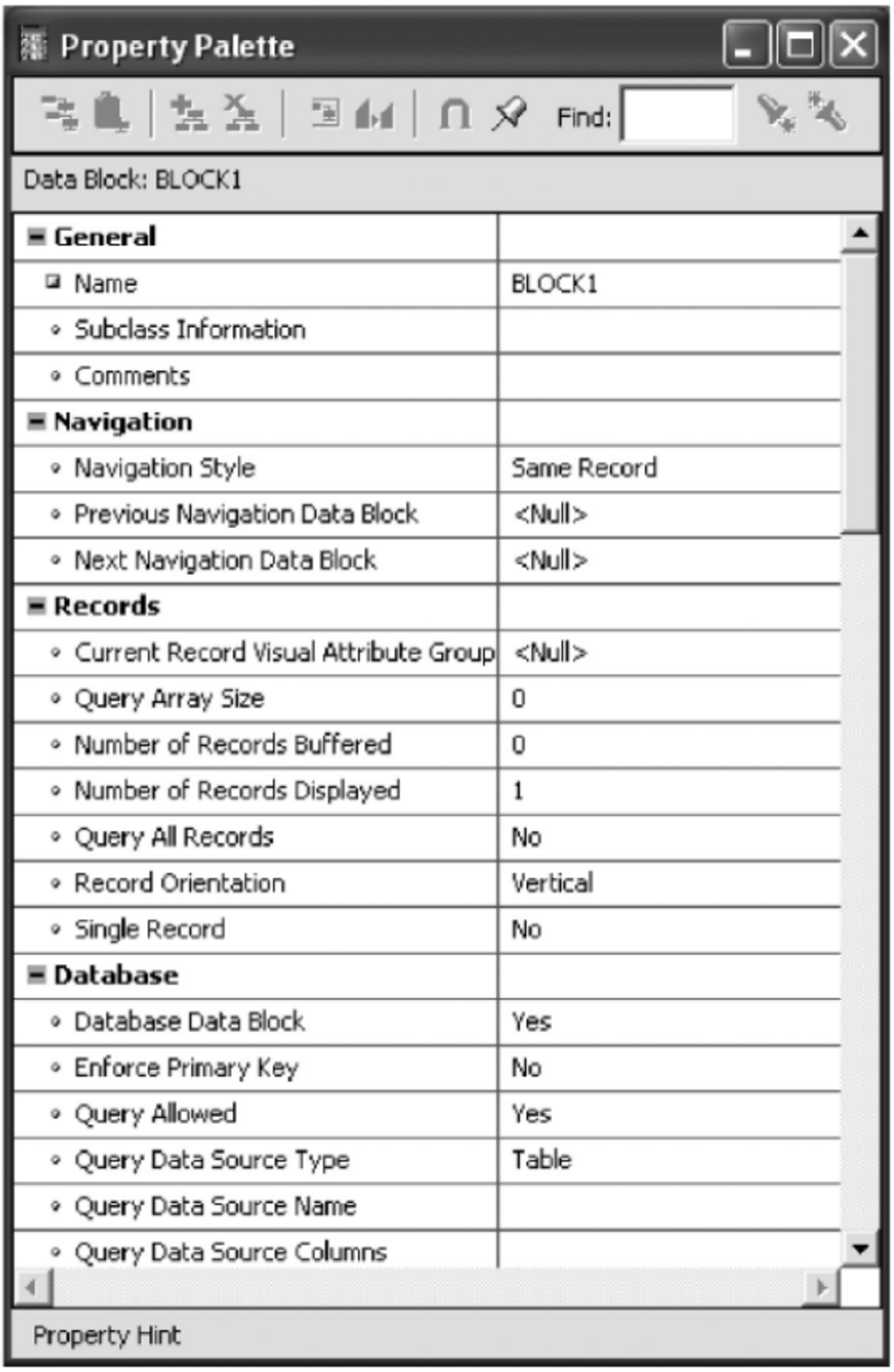

Рис. 5.3. Таблица свойств (Object Properties Sheets)

ния (по умолчанию) выводятся только общие свойства выбранных объектов. В цельном режиме выводятся свойства всех выбранных объектов;

- Freeze/Unfreeze – переключает синхронизацию окна Properties на On и Off. Когда Freeze установлено на Off (по умолчанию), для вывода свойств объектов, выбираемых в Навигаторе и в других окнах, список свойств обновляется. Когда Freeze установлено на On, список

свойств фиксируется и не получает обновлений, позволяя вам сравнивать его с другими списками свойств.

Таблица свойств помогает настраивать, преобразовывать, переименовывать, устанавливать параметры, настраивать на производительность, быстродействие, согласованность и т. д. Когда вы выбираете объект в редакторе (Layout Editor) или Объектном Навигаторе, окно Properties обновляется, чтобы показать свойства этого объекта. По умолчанию редактор свойств находится в опции Synchronize и обновляется каждый раз при выборе другого объекта. Вы можете включать и выключать синхронизацию для определенного набора, щелкая на кнопке Freeze/Unfreeze на линейке инструментов окна Properties – это даст вам возможность в случае надобности запустить дополнительное окно Properties, например, для сравнения свойств двух и более элементов Forms'a. При изменении какого-либо значения свойства объекта изменяется и иконка слева от названия свойства: если вы сделали изменение значения, вместо иконки с изображением шарика появится иконка с изображением зеленого квадратика.

Окно свойств вызывается либо в главном меню «Tools->Properties», либо двойным щелчком по элементу (кроме объектов программ и вид-картинок). А для запуска нескольких окон щелкните на иконке элемента с нажатой клавишей Shift.

Редактор разметки (Layout Editor)

Редактор Разметки (рис. 5.4) – это, собственно, место графического построения формы, служит для создания элементов, а также для их разметки, упорядочения на форме и создания графического интерфейса формы. Визуально Layout Editor можно разбить на несколько частей:

- в левой части Layout Editor находится панель инструментов, на которой размещены кнопки для создания объектов формы, а также элементы управления визуальными эффектами и шрифтами. Далее мы более подробно остановимся на этих компонентах;
- в верхней части находится меню Layout Editor, а также 2 выпадающих списка – список Canvas (вид-картинок) и список Block (блоков данных). С помощью этих списков вы можете быстро назначить выбранному вами блоку данных любой Canvas, причем при выделении какого-либо объекта эти списки будут сразу же отображать принадлежность элемента тому или иному блоку данных или Canvas;
- измерительные линейки и опорные линии. *Горизонтальная и вертикальная измерительные линейки* вверху и слева от рабочего пространства предоставляют справочную информацию для определения размеров и расположения объектов в Layout Editor. Чтобы их скрыть, отмените опцию Rulers в меню View. Единицы измерения этих линеек в Layout

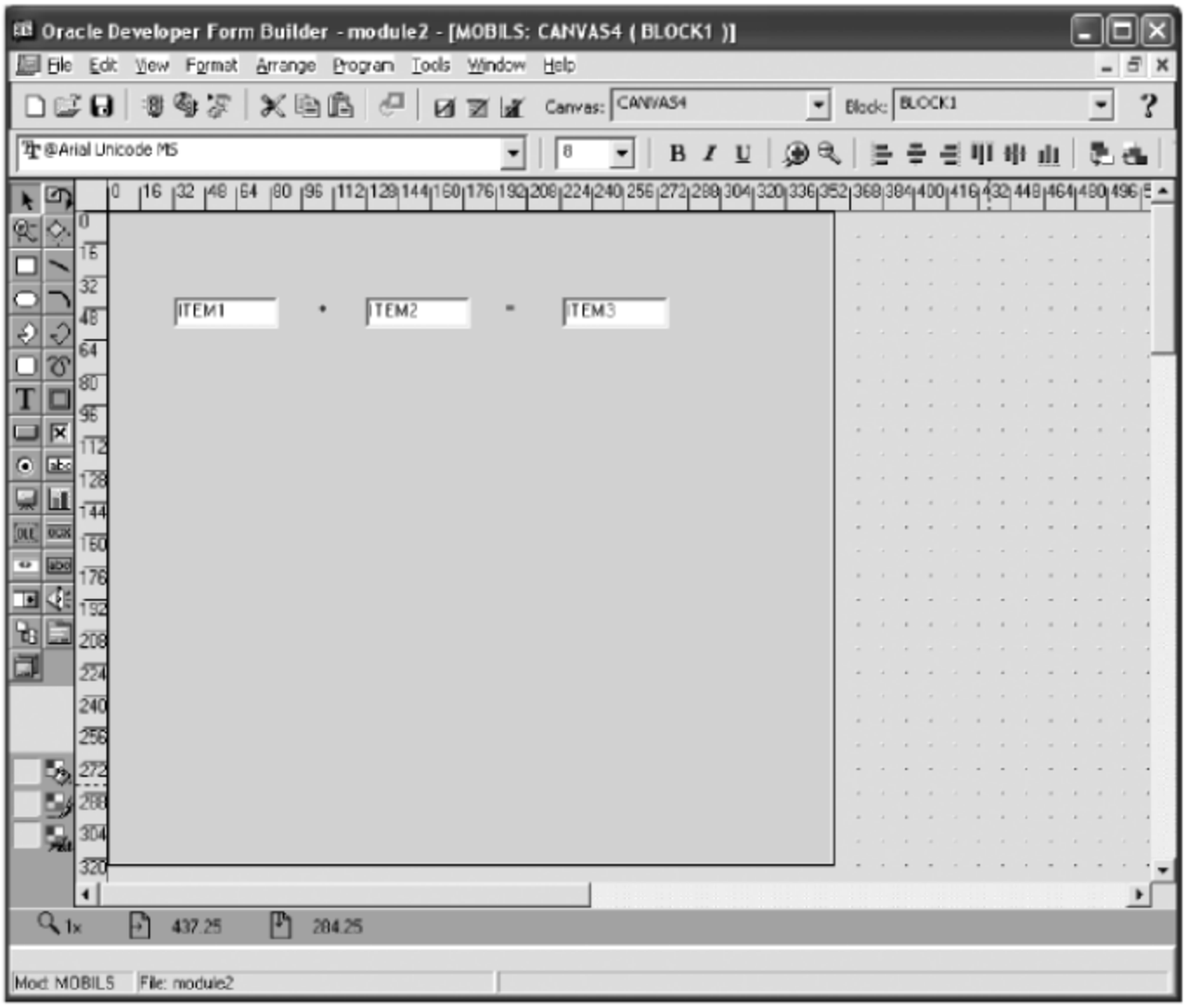

Рис. 5.4. Редактор разметки (Layout Editor)

Editor можно установить на символьные ячейки, дюймы, сантиметры или точки. Для вывода диалогового окна Ruler Settings выберите View->Settings->Ruler Settings. *Опорные линии* – это горизонтальные и вертикальные точечные линии, которыми можно пользоваться для ориентировки при размещении объектов на картинке. Для создания опорной линии поместите указатель на какую-либо измерительную линейку, затем щелкните и перетащите в рабочее пространство. Во время выполнения опорные линии не показываются. Чтобы перенести опорную линию, перетащите ее в другое место. Чтобы удалить опорную линию, вытащите ее из рабочего пространства. Чтобы временно спрятать опорные линии, не удаляя их, отмените выбор опции Ruler Guides в меню Arrange;

- сетка – в пределах рабочего пространства имеется определенная сетка, помогающая вам размещать объекты. Сетка представляется в текущих единицах измерения измерительных линеек. Чтобы скрыть линии сетки, отмените выбор опции Grid в меню View.

Если у вас имеется только один Canvas, то в выпадающем списке вид-картинок будет отображена только одна вид-картинка, если же у вас в форме имеется более одного Canvas, то Oracle Forms выводит окно, где перечислены все вид-картинки вашего модуля в виде списка значений LOV, и вы можете выбрать вид-картинку, которую хотите редактировать.

В Layout Editor вы можете создавать, вырезать, вставлять и удалять объекты, исключениями являются объект картинки (когда включена опция View->Show Canvas), прямоугольник просмотра (когда включена опция View->Show View), линия прокрутки блока.

Способы вызова Редактора Разметки:

- вызов через всплывающее меню – выделите какой-либо объект из узла блока данных, нажмите правую кнопку мыши и в появившемся меню выберете пункт Layout Editor;
- выберите (учтите, что для выполнения этого правила необходимо, чтобы в навигаторе не был выбран объект, поддерживающий разметку, например, библиотека) Tools->Layout Editor, затем укажите вид-картинку (Canvas), с которой вы хотите поработать. И соответственно, если у вас в форме более одной вид-картинки, то появится окно списка вид-картинок, если же вид-картинка одна, то Oracle отобразит ее автоматически, иначе, то есть если она отсутствует, Forms ее автоматически создаст.

Одновременно можно запускать столько Редакторов Разметки, сколько у вас блоков данных.

Лекция 6. Инструменты проектирования. Настройка инструментальных средств Forms

Большая часть лекции посвящена настройке инструментальных средств, параметров генерации и запуска форм, мастеров создания блоков и разметки. В этой лекции рассматриваются различные вспомогательные редакторы, такие как PL/SQL Editor, Find and Replace PL/SQL.

Ключевые слова: объектный навигатор (Object Navigator), редактор разметки (Layout Editor).

Цель лекции: ознакомить слушателя с основными параметрами настройки Oracle Forms, PL/SQL-редактором и другими вспомогательными средствами, позволяющими ускорить процесс разработки и отладки приложения.

Настройка инструментальных средств Forms

Oracle Forms позволяет разработчику изменять настройки инструментальных средств с помощью элемента меню Tools | Preferences (рис. 6.1).

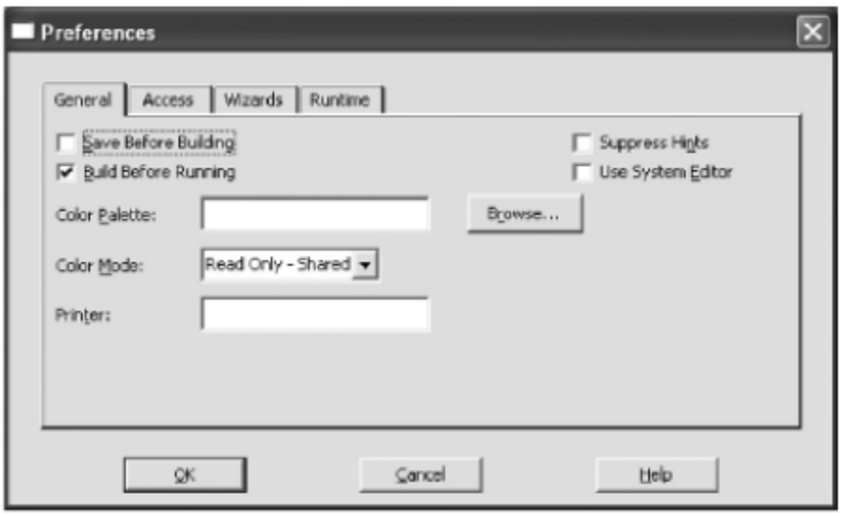

Рис. 6.1. Окно настроек Forms (вкладка General

Девять установок назначают режим работы Oracle Forms Designer:

1. **Save Before Generate** – автоматическое сохранение текущего модуля перед каждым генерированием формы;
2. **Generate Before Run** – генерирование (компиляция) формы перед запуском ее из Oracle Forms;
3. **Supress Hints** – по умолчанию Oracle Forms выдает подсказку в нижнем левом углу в зависимости от контекста. Данные подсказки можно отключить установкой этой галочки;

4. **Run Modules Asynchronously** – определяет возможность одновременной работы в среде Forms Designer и запущенных приложениях;
5. **Use System Editor** – использование текстового редактора операционной системы (например, Vi или Notepad) вместо встроенного редактора Forms;
6. **Color Palette** – определяет цветовую палитру, которая будет использоваться при отображении запущенной формы;
7. **Mode** – определяет режим использования цветовой палитры;
8. **Module Access** – определяет режим доступа к файлам форм, библиотек и меню в базе данных и файловой системе;
9. **Printer** – установка принтера для печати из Forms Designer.

Рассмотрим следующую вкладку – Access (рис. 6.2).

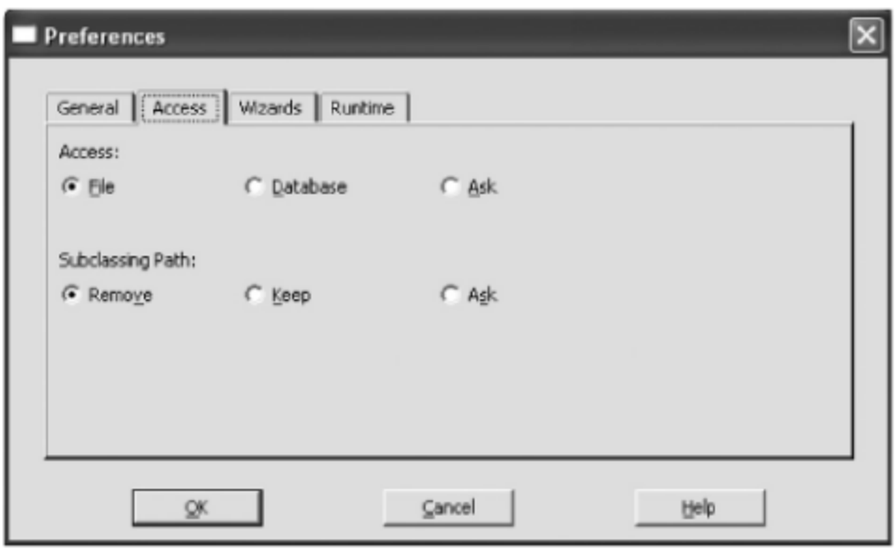

Рис. 6.2. Окно настроек Forms (вкладка Access)

Рассмотрим следующую вкладку окна Preferences – Wizards (рис. 6.3).

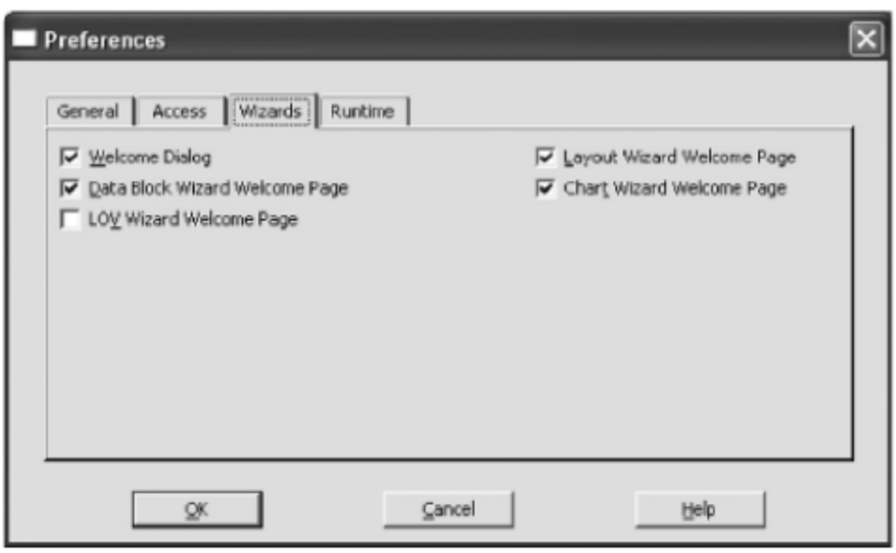

Рис. 6.3. Окно настроек Forms (Wizards)

И в заключение опишем последнюю вкладку – Runtime.

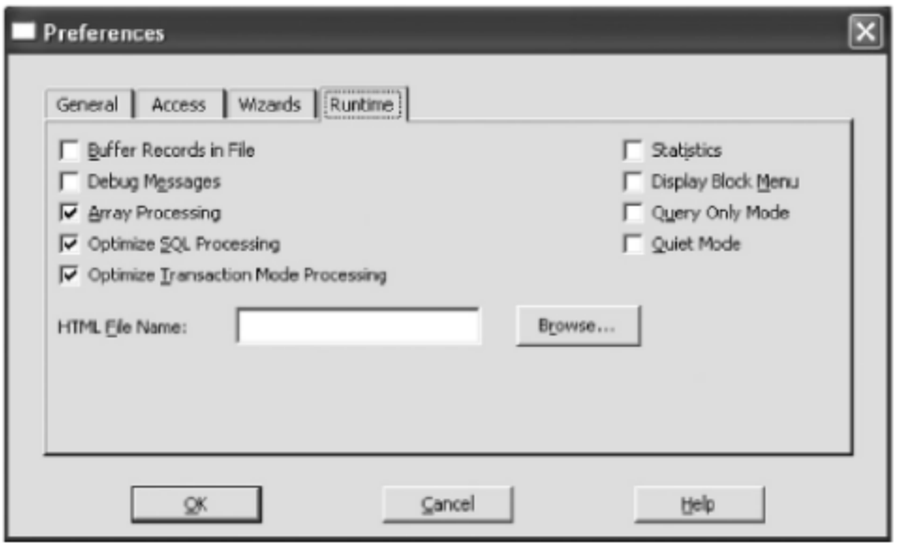

Рис. 6.4. Окно настроек Forms (вкладка Runtime)

Четвертая закладка (рис. 6.4) диалогового окна используется для установки параметров запуска форм из Oracle Forms Designer. Первый параметр Buffer Records устанавливает размер буфера в памяти, равный минимальному числу записей (число отображаемых записей + 3). Все остальные записи будут храниться во временном файле. Следующий параметр Debug Mode включает режим отладки. Данный режим позволяет разработчику устанавливать точки прерывания в программных сегментах PL/SQL с целью просмотра значений элементов форм и последовательности выполнения программного кода.

Следующие четыре параметра связаны с настройкой производительности Oracle Forms. Обработка массивов (Array processing) позволяет форме вводить множество строк выборки из БД вместо одной. Обычно это повышает производительность, однако множественная выборка требует больше памяти для работы формы. Для поддержки совместимости с предыдущими версиями Oracle Forms дает возможность разработчику использовать в формах триггеры, написанные для второй версии. Оптимизированная обработка SQL-запросов (The Optimize SQL Processing) позволяет применять новые возможности PL/SQL. Оптимизация режима транзакций (Transaction Mode Optimization) включает оптимизацию использования курсоров для неявных SQL-запросов (например, вызовы и исполнение триггеров) – возможно применение одних и тех же курсоров в различных функциях формы. Параметр «статистика» (Statistics) позволяет генерировать статистику использования курсоров и прочих ресурсов при запуске формы. Еще одно свойство данного параметра – трассировка SQL-запросов, которую можно проанализировать с помощью TKPROF или иной утилиты анализа производительности.

Следующий параметр – отображение меню блока (вместо собственно меню формы) при запуске формы. Это меню позволяет разработчику

обращаться напрямую к определенному блоку формы вместо запуска блока по умолчанию.

Режим исключительного выполнения запросов (Query-only mode) запрещает форме SQL-запросы на вставку, удаление и обновление.

Наконец, режим **Quiet** (**Тихо**) выключает звуковые сигналы, проигрываемые при выдаче формой сообщения.

Редактор PL/SQL

В Oracle Forms для написания и редактирования любых программных единиц (триггеры, функции, процедуры и т. д.) используется редактор PL/SQL (рис. 6.5). Редактор PL/SQL – это среда для написания, корректирования и компиляции ваших PL/SQL-программ. В отличие от многих сред разработки, где в редакторе кода отображены все программные единицы, в Oracle Forms редактор PL/SQL запускается отдельно для каждого объекта, что очень удобно, так как вам не придется искать по всей программе код нужного объекта – достаточно будет найти этот объект в навигаторе и запустить редактор. Визуально редактор состоит из рабочей области для написания вашего PL/SQL-кода и элементов пользовательского интерфейса. Для того чтобы запустить редактор PL/SQL, создайте новую форму, в объектном навигаторе перейдите в узел «Триггеры» и правым щелчком мыши вызовите всплывающее меню и выберите пункт Универсальные триггеры | WHEN-NEW-FORM-INSTANCE.

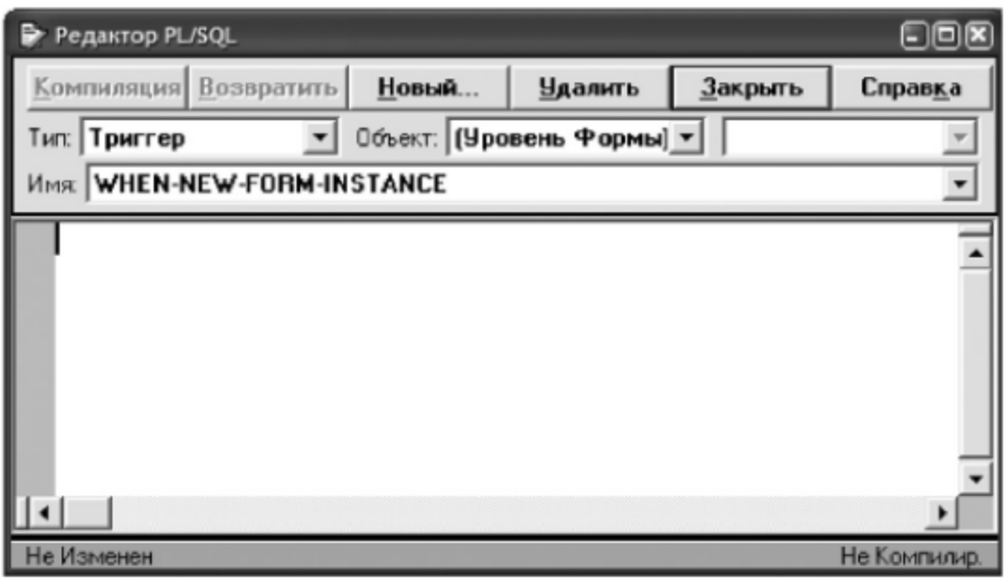

Рис. 6.5. Редактор PL/SQL

Настройка редактора

PL/SQL-редактор можно настроить по своему вкусу, используя «Окно настроек» (рис. 6.6).

Рис. 6.6. Окно свойств

Вызовите всплывающее меню редактора (рис. 6.7) и выберите нижний пункт «Properties» для вызова Окна свойств. В этом окне вы можете установить настройки шрифта, подсветки текста и фона. Параметры вкладки «Keyboard» позволяют переназначить горячие клавиши редактора, установить новые и переопределить старые комбинации.

Рис. 6.7. Всплывающее меню PL/SQL-редактора

Пункты «Find» и «Replace» предназначены для работы с текстом редактора аналогично инструменту «Найти и заменить PL/SQL». Выберите пункт меню «Find» для запуска одноименного окна поиска текста (рис. 6.8).

Рис. 6.8. Окно «Find»

В окне поиска вы можете задать регистр и направления поиска фразы «What». Искать фразу в тексте можно последовательно, нажимая клавишу «Find», тогда поиск будет происходить в соответствии с заданным направлением или одним нажатием на клавишу «Mark All» для подсветки сразу всех совпадений, найденных в тексте. Если вам необходимо найти и заменить какую-либо фразу в редакторе, выберите пункт «Replace» (рис. 6.9).

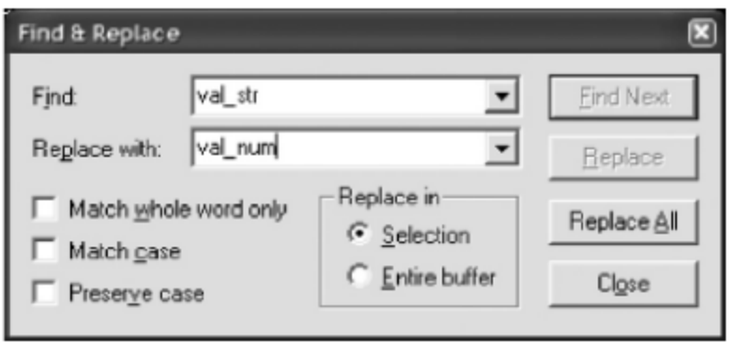

Рис. 6.9. Окно «Replace»

Окно «Find&Replace» позволяет не только найти и заменить искомую фразу, но и управлять ходом замены совпадений. По умолчанию при нажатии кнопки «Replace All» произойдет замена всех совпадений на фразу, указанную в «Replace with». Для того чтобы произвести замену в определенном фрагменте текста, необходимо перед запуском окна «Find&Replace» выделить этот фрагмент.

Элементы управления редактором PL/SQL

На верхней горизонтальной панели редактора находятся элементы управления редактора и его содержимого:

- **«Компиляция»** – по нажатии этой кнопки осуществляется проверка вашего кода на наличие ошибок.

- **«Возвратить»** – по нажатии этой кнопки осуществляется отмена всех действий до последней компиляции или на момент открытия редактора.
- **«Новый...»** – кнопка для создания нового объекта. По нажатии этой кнопки предлагается создать объект на основе выбранного типа.
- **«Имя»** – в этом поле отображается имя PL/SQL-программы или триггера.
- **«Тип»** – в этом выпадающем списке вам предлагается выбрать тип программной единицы. Существует три типа программных единиц:
  – триггер;
  – программа;
  – код элемента меню.
- **«Объект»** – в этом выпадающем списке указывается уровень области действия объекта.
- **«Удалить»** – кнопка для удаления PL/SQL-программы.
- **«Закрыть»** – завершает работу с редактором PL/SQL.
- **«Справка»** – вызов справочной системы Oracle Forms.

Загрузка внешних PL/SQL-программ

Когда мы разрабатываем новое приложение, всегда существует вероятность того, что частично некоторые PL/SQL-программы уже выполнялись вами или же вы знаете о существовании таких программ в вашей системе. Можно, конечно, обернуть эту программу в хранимую процедуру или функцию базы данных и потом использовать ее в Forms, но бывают такие случаи, когда не следует помещать определенный код в базу по различным причинам. Для импорта текста в редактор PL/SQL нужно, находясь в редакторе, нажать комбинацию клавиш Ctrl+M или вызвать окно «Импорт» через меню Файл | Импорт (рис. 6.10).

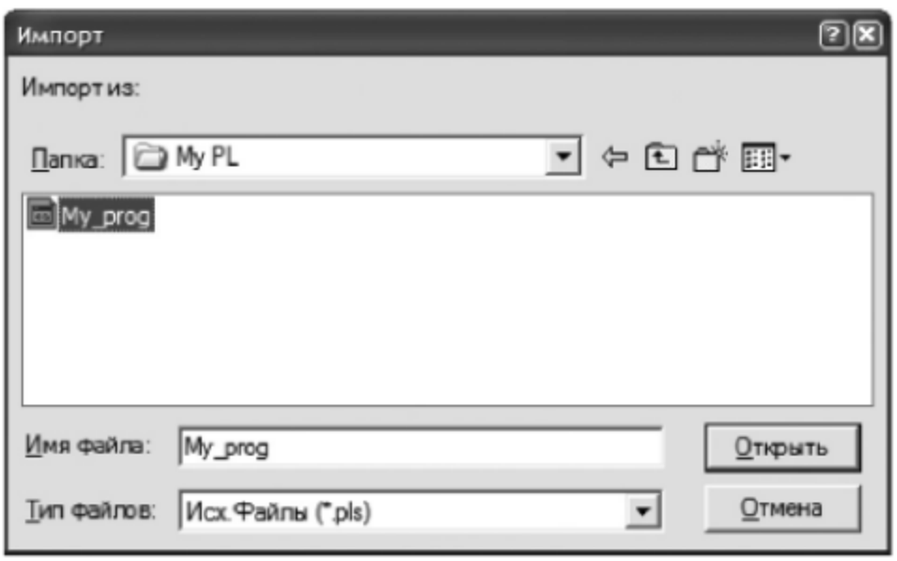

Рис. 6.10. Окно «Импорт»

Компиляция программ

Если ваша PL/SQL-программа набрана и после нажатия кнопки «Компиляция» в нижней строке состояния редактора появилась фраза «компилирован успешно», значит, Oracle Forms не нашел в ней синтаксических ошибок и программа готова к выполнению. Если же в вашем коде имеется ошибка, то после нажатия кнопки «Компиляция» в редакторе появится еще одно окно «Debug» (рис. 6.11), в котором будут отображены допущенные вами ошибки, их код и номера строк, в которых они допущены. Для того чтобы это проверить, нажмите кнопку «Компиляция» в запущенном вами редакторе. Так как любая PL/SQL-программа должна содержать хотя бы один исполняемый оператор, а наш редактор пуст, получим сообщение об ошибке и фразу «компилирован с ошибкой» в строке состояния редактора.

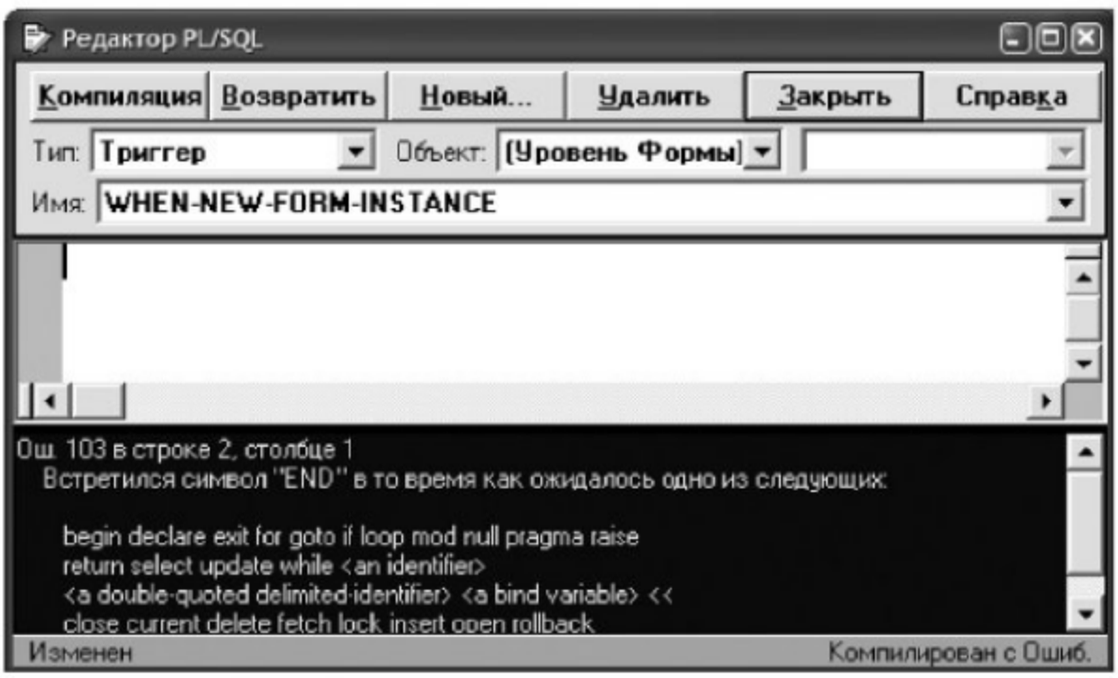

Рис. 6.11. Редактор PL/SQL, окно «Debug»

Редактор PL/SQL – это инструмент, который вы будете чаще всего использовать, разрабатывая приложения. Умение пользоваться редактором и знание его функциональных характеристик поможет вам оперативно работать с текстом, создавать и корректировать программы.

Найти и заменить PL/SQL

В предыдущем разделе мы рассматривали PL/SQL-редактор, в котором ознакомились с возможностями поиска и замены фрагментов кода. В этом разделе мы ознакомимся с инструментом «Найти и заменить PL/SQL» (рис. 6.12). Особенность этого инструмента в том, что он рабо-

тает не с отдельной программой, а с одним или несколькими модулями формы. Мы часто задействуем одни и те же элементы, параметры и глобальные переменные в разных триггерах, именованных блоках и модулях. Поиск и исправление ошибок в таких приложениях может занять много времени, поэтому наличие такого инструментария экономит ваши силы.

Вызовите инструмент «Найти и заменить PL/SQL» через меню Программа | Найти и заменить PL/SQL.

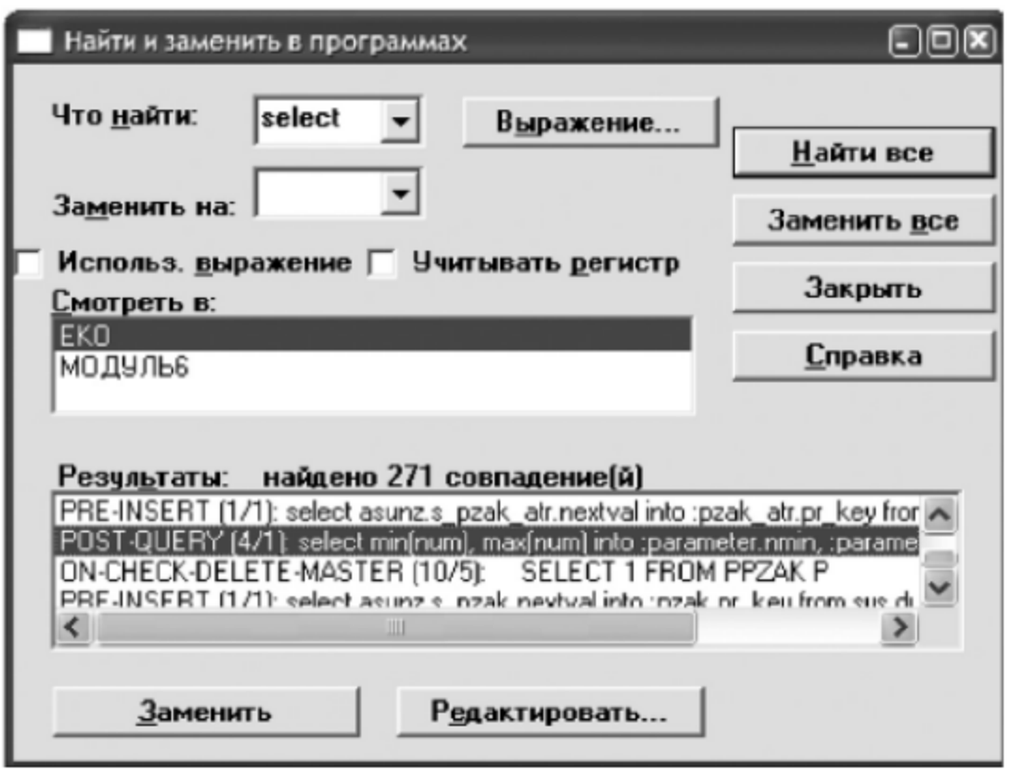

Рис. 6.12. Найти и заменить PL/SQL

Интерфейс окна состоит из элементов управления поиском, заменой и отображением искомой информации. Рассмотрим каждый элемент в отдельности.

- *«Что найти»* – это поле для ввода фразы поиска.
- *«Заменить на»* – это поле для ввода фразы замены.
- *«Использовать выражение»* – это элемент управления выражениями: если элемент включен, значит, выражения будут участвовать в поиске.
- *«Учитывать регистр»* – это элемент управления поведением поиска. Если элемент включен, значит, поиск будет осуществлен с учетом регистра символов.
- *«Смотреть в»* – в этом окне перечисляются все модули, открытые в объектном навигаторе. По умолчанию при запуске окна все модули выделены, а это означает, что все они участвуют в поиске. Для выделения конкретного модуля достаточно щелкнуть по нему левой кнопкой мыши. Чтобы выделить несколько форм, нужно повторить вышеописанное действие с нажатой клавишей Shift.

- *«Результаты»* – это окно отображения результатов поиска. Найденное совпадение отображается в окне результатов в виде:

  ```
  POST-QUERY (4/1): select min(num), max(num)….
  ```

  Здесь говорится о том, что совпадение найдено в четвертой строке и первом символе строки триггера POST QUERY.
- *«Заменить»* – по нажатии этой кнопки осуществляется замена элемента, выделенного в окне результатов совпадения.
- *«Редактировать»* – по нажатии этой кнопки открывается редактор PL/SQL выбранного в окне результата совпадения.
- *«Найти все»* – по нажатии этой кнопки выполняется поиск фразы «Что найти» в выбранных модулях. Найденные совпадения отображаются в окне результатов.
- *«Заменить все»* – по нажатии этой кнопки происходит замена всех найденных совпадений на фразу «Заменить на». Перед тем как выполнить замену, проверьте, что не ошиблись с фразой замены, т. к. операция необратима.
- *«Закрыть»* – завершение работы с инструментом.
- *«Справка»* – вызов справочной системы Oracle Forms.
- *«Выражение»* – по нажатии этой кнопки появляется список выражений, подставляемых во фразу «Что найти». Вы можете самостоятельно попрактиковаться поиском, используя выражения. Чтобы выражение учитывалось в поиске, не забудьте включить «Использовать выражение»; ниже приведен список доступных выражений:
  – любой символ (.);
  – символ в диапазоне ([]);
  – символ не в диапазоне ([~]);
  – группа (());
  – или (|);
  – 1 или более соответствий (*);
  – перенос строки (□);
  – символ табуляции (\t);
  – пустой символ (\b);
  – пробел (\w).

Инструмент «Найти и заменить PL/SQL» поможет вам ориентироваться в вашем приложении и сэкономит время, затрачиваемое на поиск и замену нужного текста.

Лекция 7. Блоки. Виды и структура блоков. Управление свойствами блока

В этой лекции слушатель ознакомится с различными видами блоков и их структурой, а также научится управлять свойствами блоков программно и на этапе проектирования.

Ключевые слова: Data Block (блок данных), Data Block Wizard, Layout Editor.

Цель лекции: ознакомить слушателя с понятием блок данных, изучить его структуру, назначение и научиться управлять свойствами блока.

Блоки

В Oracle Forms существует свой формат представления источника данных – блок данных. Блок данных в Oracle Forms может иметь различные источники данных:

- таблица;
- представление;
- синоним;
- внутреннее представление (from clause);
- транзакционные триггеры.

Несмотря на различные источники данных, структура блока остается неизменной.

Блок (Block) – это логическая структура, которая не является физической копией каких-либо данных, она создает логический структурированный образ на основе выбранной модели данных (таблица, представление, синоним). Другими словами, блок – это упорядоченная, организованная единица представления данных на основе указанного источника данных. При создании блока вам не нужно повторно определять взаимосвязи между таблицами, создавать ограничения целостности, потому что Oracle Forms автоматически определяет наличие взаимосвязей и ограничений. Блоки, как и объекты БД, могут содержать взаимосвязанные элементы, которые выводят записи данных. Поскольку блоки являются лишь логическими единицами, ограничений на их количество нет. Блоки, как и другие объекты, могут создаваться, копироваться, изменяться и имеют свои свойства.

Типы блоков

В зависимости от потребностей пользователя в Oracle Forms предусмотрено два типа блоков.

Базовый блок – это блок, созданный на основе определенной таблицы или представлении базы данных. В таком блоке элементы (один или более) прямо связаны со столбцами базовой таблицы, так как элементы содержат значения столбцов выбранной вами таблицы или представлении.

Небазовый блок, или управляющий блок – это блок, не связанный с таблицей базы данных, и элементы в управляющем блоке не связаны со столбцами базы данных.

Блоки с базовой таблицей автоматически включают в себя функции поддержки запросов, DML операций (обновление, вставка, удаление) с таблицей, с которой этот блок связан, а также встроенные подпрограммы для выполнения этих операций.

Если вы создаете базовой блок с включенной опцией Integrity Constrains, это говорит Oracle Forms о том, что при создании блока нужно автоматически изменять некоторые его свойства, а также сгенерировать триггеры и программные модули (Program units):

- если элемент связан с колонкой первичного ключа в базовой таблице, то автоматически для элемента устанавливается свойство Primary Key («первичный ключ») на True и свойство Update Allowed («разрешено обновление») на False, а для блока свойство Primary Key устанавливается на On;
- для придания силы уникальным значениям в элементах, связанных с колонками первичных ключей, и для включения ограничений CHECK создает триггер When_Validate_Item или When_Validate_Record. При наличии составного первичного ключа создается триггер When_Validate_Record, иначе – When_Validate_Item;
- Oracle Forms создает для блока триггер When_Remove_Record, если его базовая таблица в базе данных создана как имеющая отношение «первичный-внешний ключ» с другой таблицей. Этот триггер предохраняет запись от удаления, если имеются соответствующие подчиненные записи в таблице внешнего ключа.

Позже, говоря о модели главный-подчиненный (master-detail), мы с вами рассмотрим, какие триггеры Oracle Forms автоматически генерирует при создании блоков с зависимостью главный-подчиненный.

Создание блока

В Oracle Forms вы можете создать блок двумя способами:

- мастер блоков данных;
- команда create навигатора объектов.

Рассмотрим первый случай – создание блока с применением мастера блоков данных. Использование мастера позволяет вам быстро создавать блоки данных на основе выбранной вами модели данных: синоним, обзор, таблица и хранимые процедуры.

Рассмотрим создание блока пошагово, но для начала создадим модель данных, на основе которой будем создавать блок. В листинге 7.1 приведен скрипт для создания таблицы, который вы можете выполнить в SQL*Plus или в любой другой утилите, предназначенной для работы с БД Oracle.

Листинг 7.1. Скрипт для создания таблицы PZAK

```
Create table PZAK
       (
pr_key   number PRIMARY KEY,
nlet   varchar2(20),
pdat      date,
nfirm  varchar2(100),
tplus     number,
tmin      number,
yslp      varchar2(300),
srpost   varchar2(100),
rnum     number,
sdat      date
)
```

1. Вызовите мастер блоков данных командой меню Tools | Data Block Wizard (рис. 7.1). Сразу после выполнения команды на экране появится приглашение Data Block Wizard. Вы можете отключить появление этого приглашения, сняв флажок «Display this page next time». Вы также можете запустить окно «Preferences» (Edit | Preference), в котором можете управлять отображением приглашений, изменяя состояние переключателей вкладки Wizard.
2. Нажмите кнопку «Далее» для перехода к следующему этапу. Следующий этап – это выбор модели данных для создания блока данных. Вы можете выбрать между хранимой процедурой и таблицей (рис. 7.2).
3. Нажмите кнопку «Далее» для продолжения. Следующий этап – это выбор таблицы, синонима или представления как источника данных для блока (рис. 7.3).
4. Чтобы выбрать источник данных, нажмите кнопку «Browse», после чего на экране появится окно Tables (рис. 7.4).

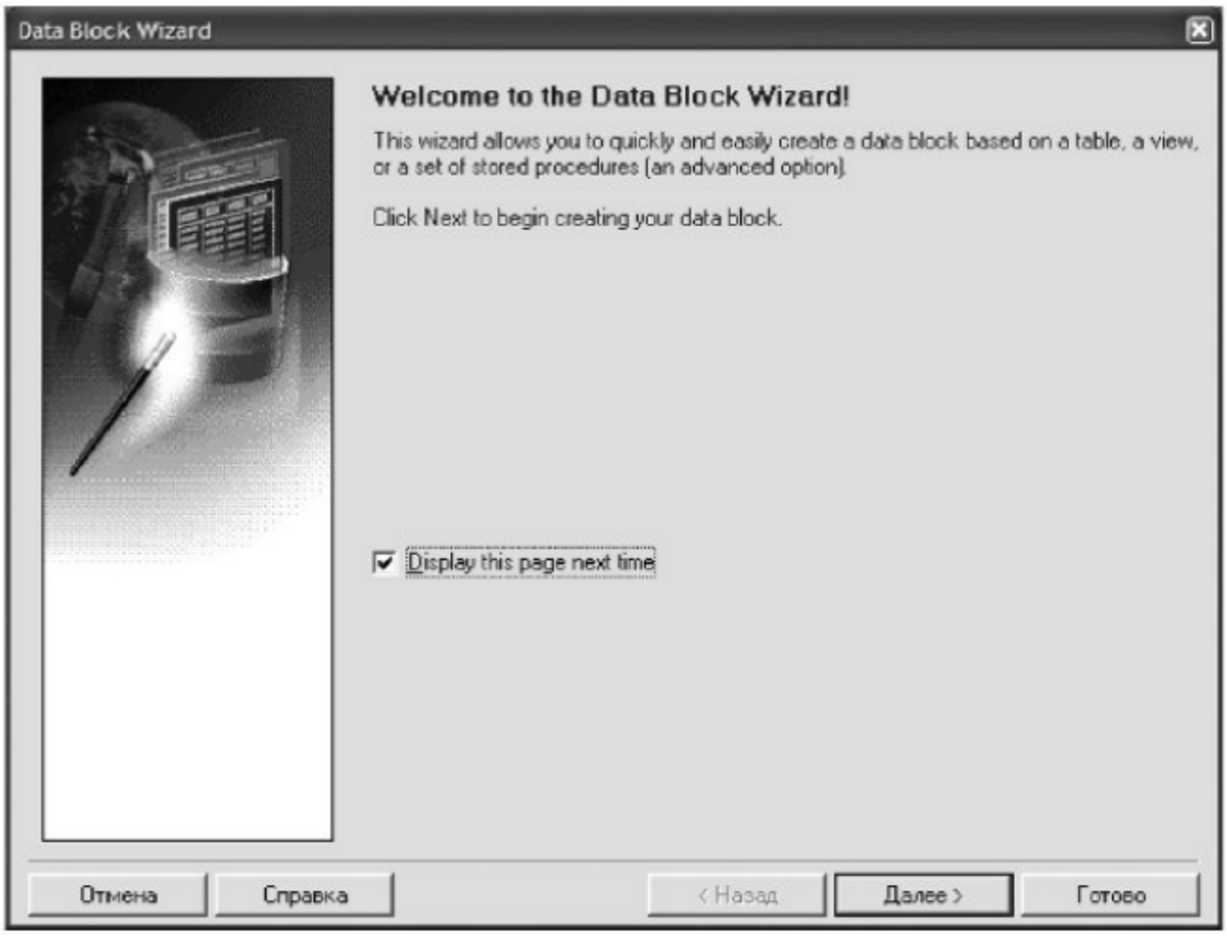

Рис. 7.1. Приглашение Data Block Wizard

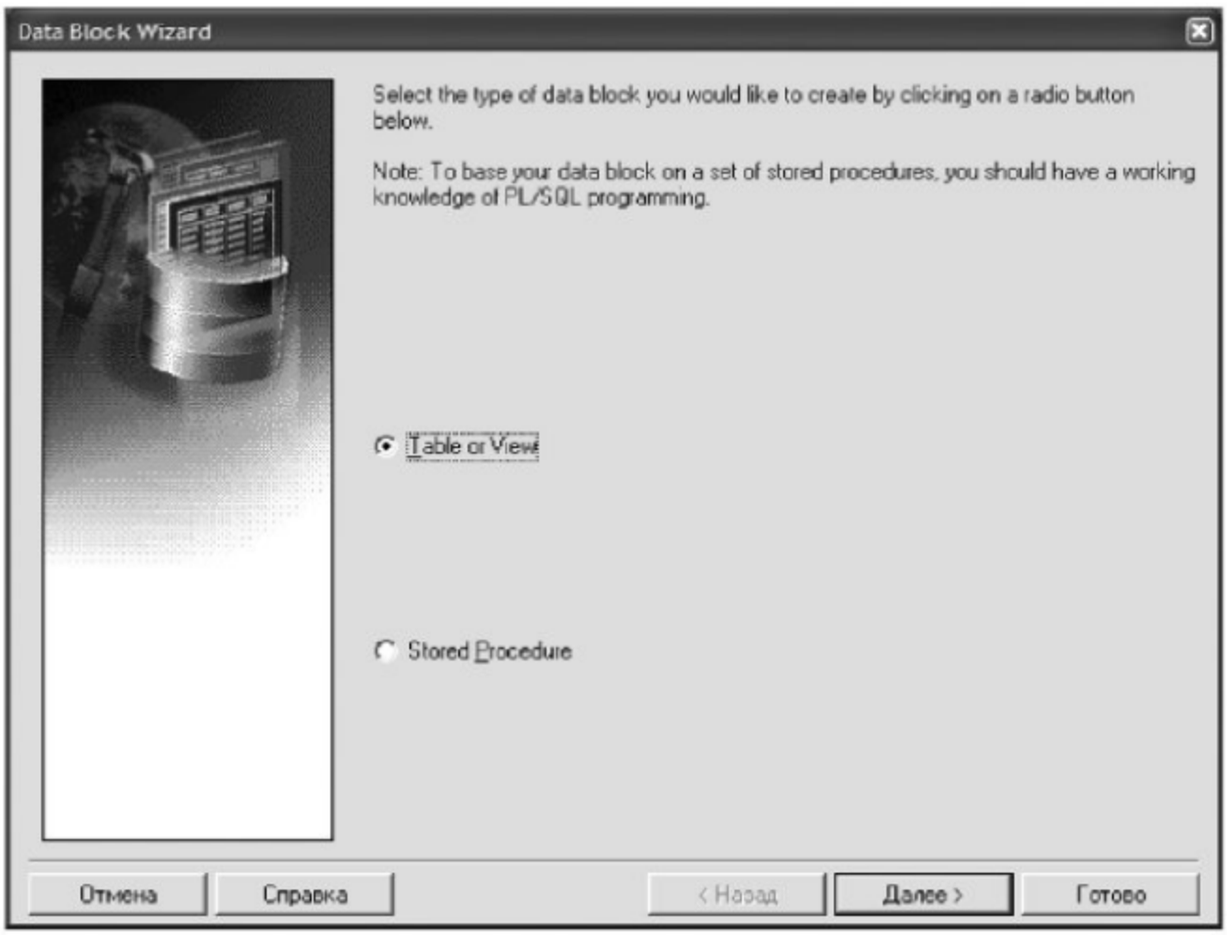

Рис. 7.2. Диалог выбора модели данных

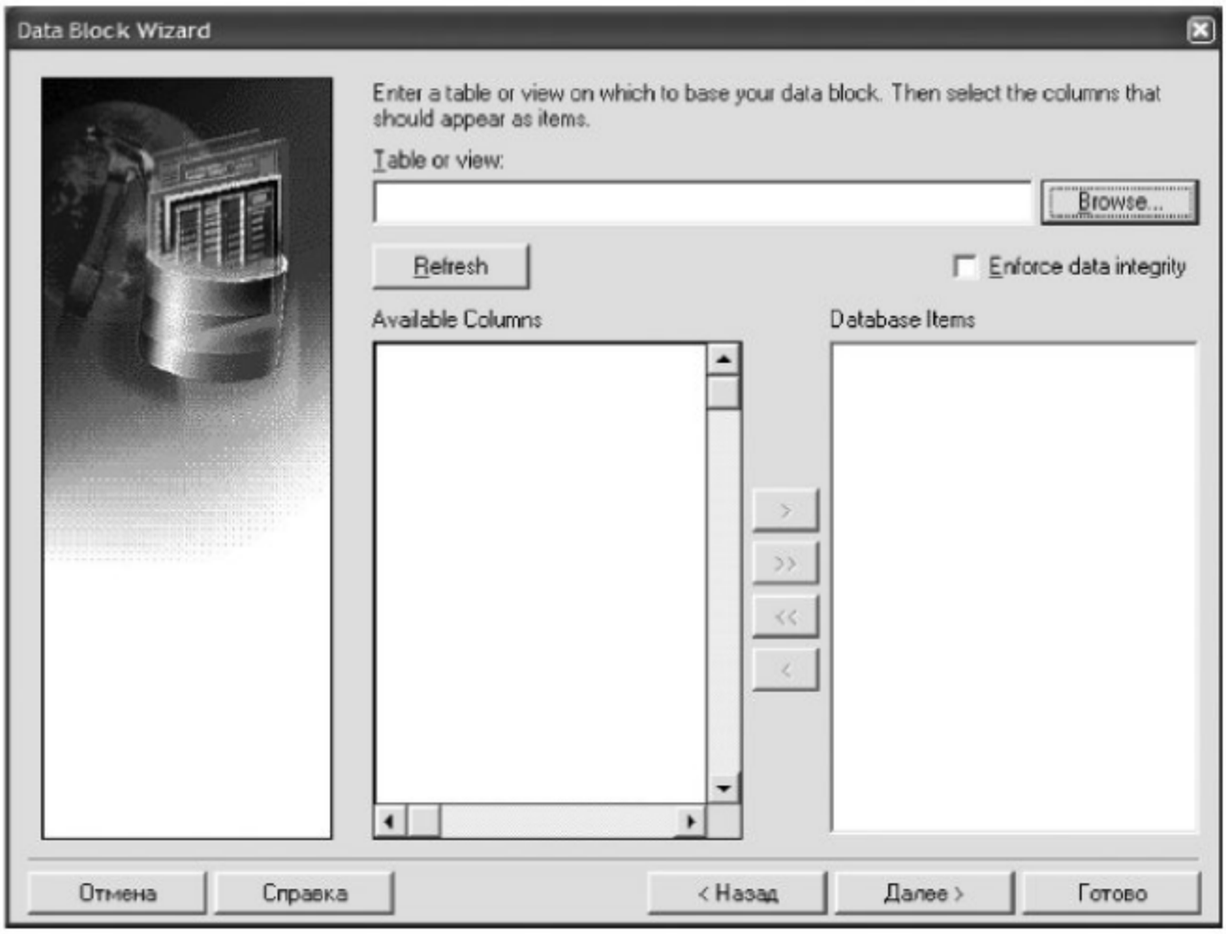

Рис. 7.3. Диалог создания структуры блока данных

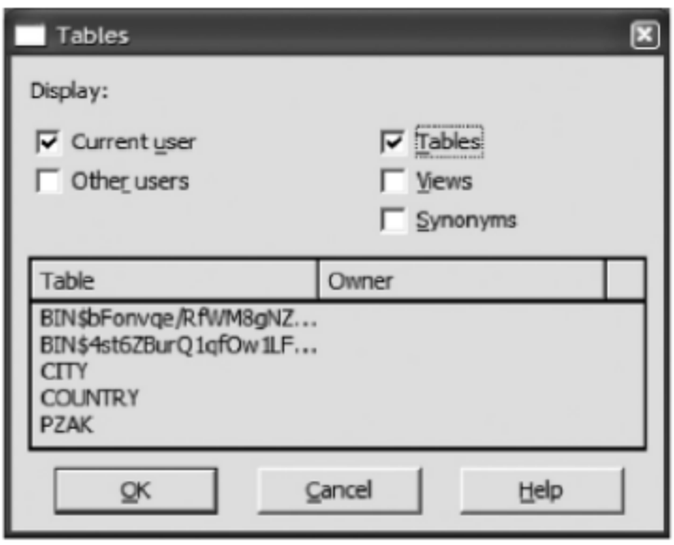

Рис. 7.4. Окно «Tables»

Вы можете управлять отображением таблиц, изменяя состояние флажков.

Current user – если опция включена, то в списке «Table» появляются все таблицы, принадлежащие текущему пользователю (схеме).

Other users – если опция включена, то в списке «Table» появляются все таблицы других пользователей. Если опция отключена, то отображаются таблицы текущего пользователя.

Tables – если опция включена, то в списке «Table» отображаются все объекты БД, имеющие тип «Таблица».
Views – если опция включена, то в списке «Table» отображаются все объекты БД, имеющие тип «Обзор».
Synonyms – если опция включена, то в списке «Table» отображаются все объекты БД, имеющие тип «Синоним».

5. Выберите созданную нами ранее таблицу PZAK и нажмите кнопку «OK» для подтверждения выбора. После того как источник данных выбран, в окне «Available Columns» появится весь перечень столбцов таблицы (рис. 7.5).

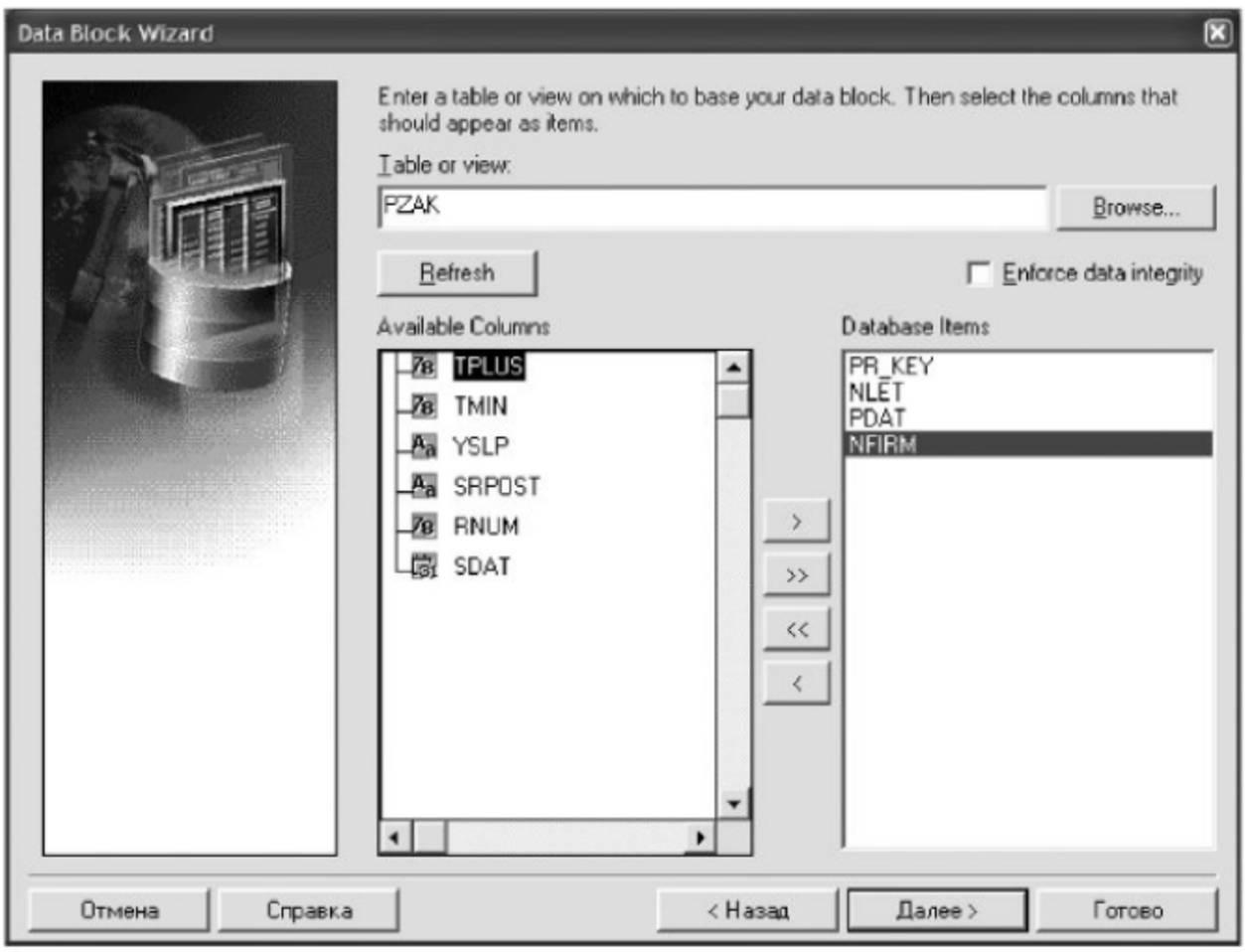

Рис. 7.5. Выбор элементов блока данных

6. Перейдем к завершающему этапу – выбору столбцов блока данных. Выбор элементов блока данных осуществляется с помощью управляющих кнопок с изображением стрелок, где одинарные стрелки означают перемещение одного элемента, а двойные стрелки – перемещение всего набора столбцов. Для создания блока данных нужно, чтобы в списке «Database Items» присутствовал хотя бы один элемент.
7. Блок данных готов, и вы можете нажать кнопку «Готово» для завершения работы мастера, после чего в навигаторе объектов появится блок данных с именем PZAK. Если вы не завершите работу мастера и

нажмете кнопку «Далее», то вам будет предложено ввести имя блока. По умолчанию имя блока идентично имени таблицы, с которой он ассоциирован.

8. Нажмите кнопку «Далее» для продолжения (рис. 7.6). В следующем окне разработчику будет предложено разметить блок данных, создать для него холст и упорядочить элементы с помощью мастера разметки блока данных. Выберите опцию «Just create the Data Block» для завершения работы мастера блоков данных (рис. 7.7). Вы можете продолжить работу над блоком, вызвав мастер разметки блока данных.

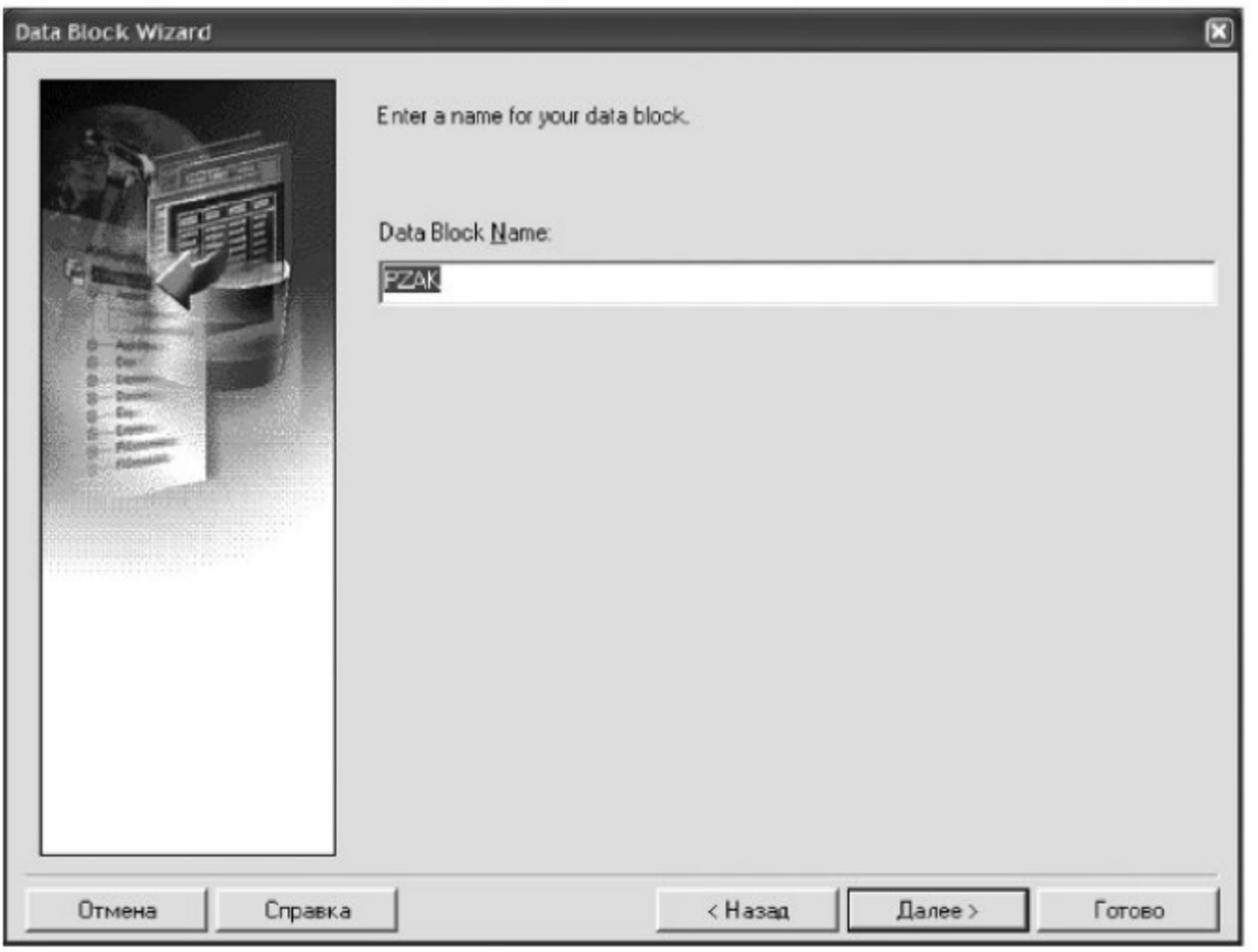

Рис. 7.6. Ввод имени блока данных

9. Нажмите кнопку «Готово» для завершения работы с мастером. После этого диалог мастера исчезнет с экрана и в навигаторе объектов появится созданный вами блок данных со всеми элементами (рис. 7.8).

Создание небазовых блоков

Небазовый, или, как его еще называют, управляющий (Control Block), блок создается намного проще, так как не требует источника данных и может быть создан без привязки к базе данных. Для создания небазового блока данных достаточно выполнить несколько действий:

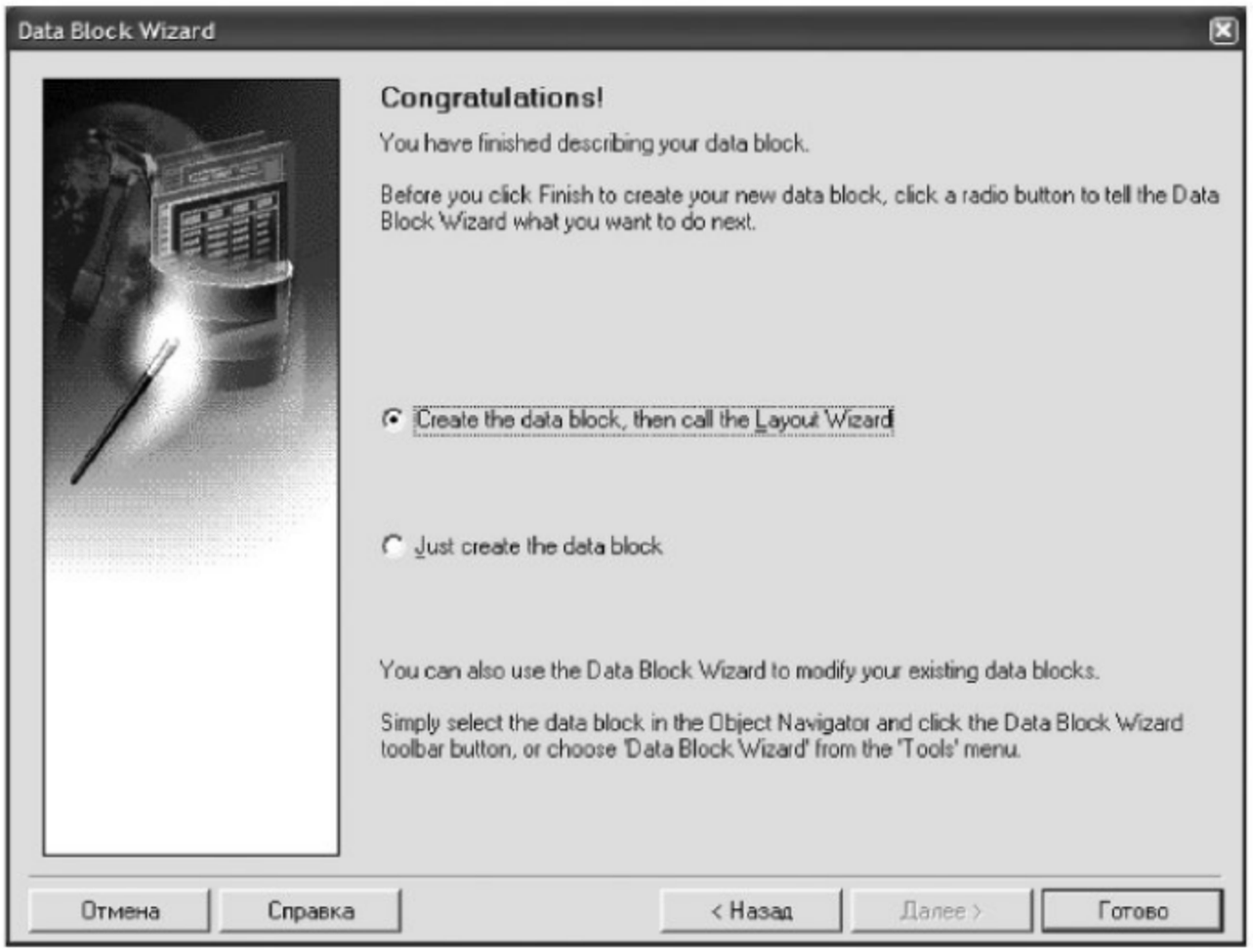

Рис. 7.7. Завершение работы мастера блоков данных

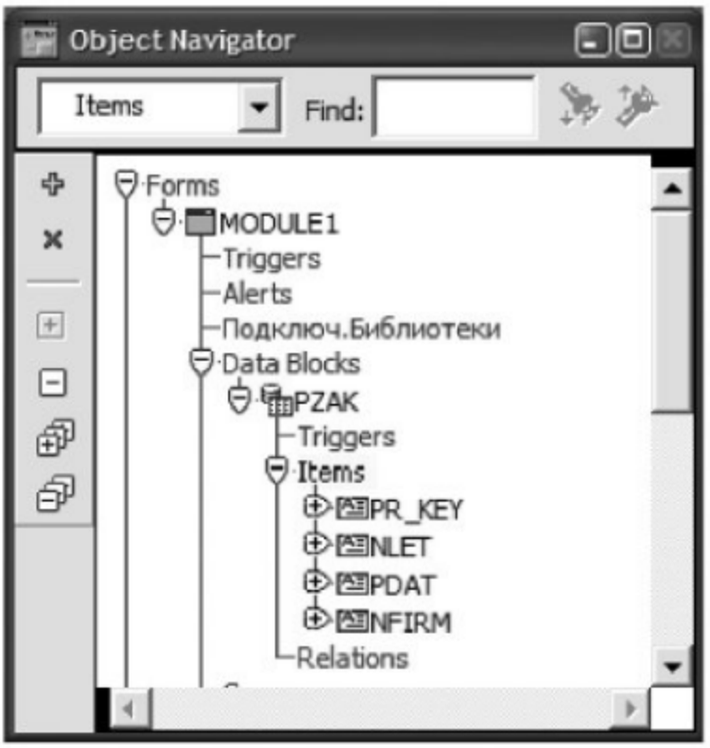

Рис. 7.8. блок данных в навигаторе объектов

1. Находясь в навигаторе объектов, выберите узел Data Blocks и нажмите кнопку «Create» для появления окна «New Data Block» (рис. 7.9).
2. В этом окне есть два переключателя, один из которых предлагает нам создать блок данных с помощью мастера блоков данных («Use the Data Block Wizard»), а другой — создать блок данных вручную («Block a new data block manually»). Выберите опцию «Block a new data block manually» и нажмите «OK».

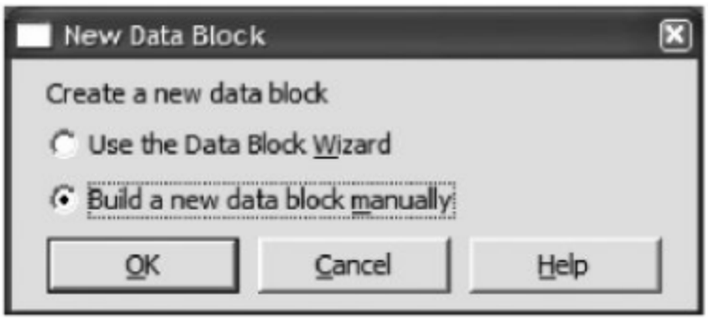

Рис. 7.9. Окно «Создать новый блок»

3. После того как вы подтвердите свой выбор, в навигаторе объектов создастся блок данных с именем по умолчанию BLOCKXX, где XX – это порядковый номер, который Forms присваивает каждому объекту при создании.

Свойства блока

Теперь, когда мы научились создавать блок данных, можно перейти к рассмотрению его атрибутов (свойств). Атрибуты блока – это его неотъемлемая часть, поэтому очень важно знать их назначение. Неправильная настройка атрибутов может привести к различным критическим ситуациям, таким как падение производительности, неправильная работа или возникновение исключительных ситуаций. Если вы будете знать большинство атрибутов блока, вы сможете не только повысить производительность вашего приложения, но и избавить себя от написания лишнего кода.

Сейчас мы ознакомимся со всеми свойствами блока данных, а самые важные рассмотрим более подробно (рис. 7.10).

1. Вкладка **General** (Общие)
   - **Name** – это свойство определяет внутренне имя вашего блока, которое вы будете использовать для обращения к нему в своих программах.
   - **Subclass Information** – при выборе этого свойства запускается окно Subclass (рис. 7.11).

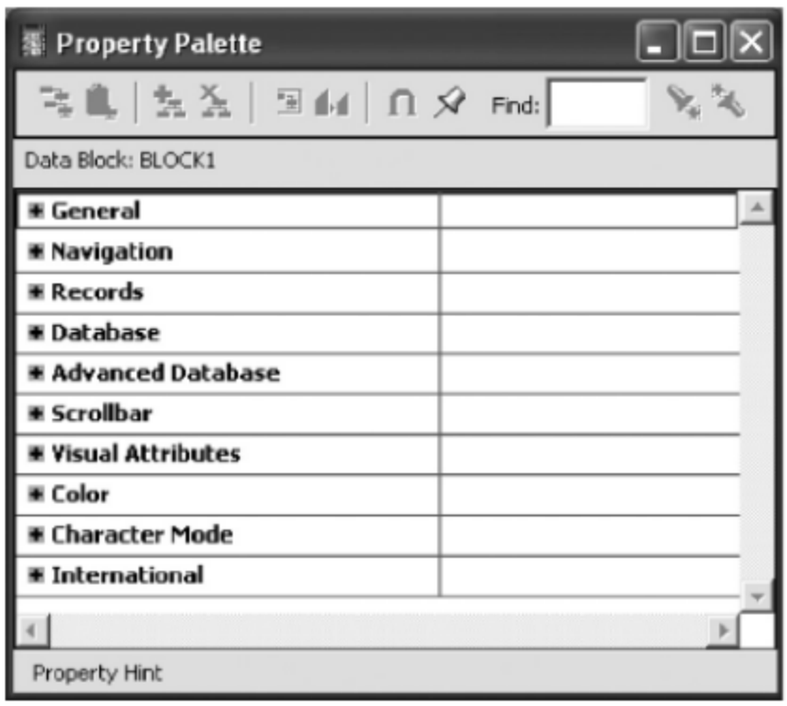

Рис. 7.10. Таблица свойств блока данных

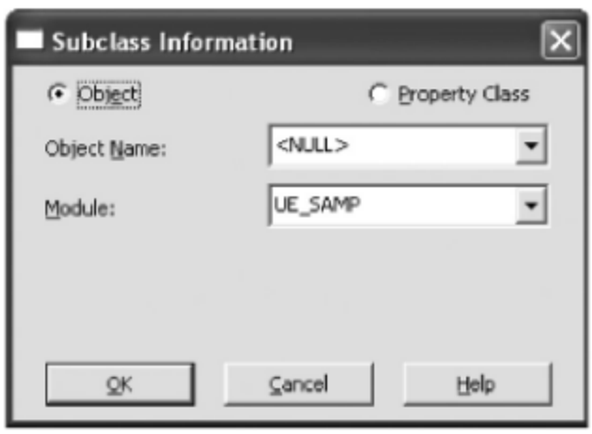

Рис. 7.11. Subclass Information (Информация о подклассе)

В этом окне вы можете выбрать класс свойств или подобъект для блока. Под подобъектом здесь подразумевается блок данных. Чтобы назначить блоку подобъект, нужно:

- установить переключатель в **Object** – этим вы указываете, что в качестве подкласса будет выбран объект;
- перейти в поле **Module** и выбрать имя модуля, в котором находится подобъект. Чтобы в поле Module появился список доступных модулей, нужно, чтобы они были открыты;
- после того как модуль выбран, в поле **Object Name** становятся доступными его объекты. Выберите нужный объект и нажмите кнопку «OK».

После этого блок будет иметь вид и структуру выбранного подобъекта. Чтобы назначить блоку класс свойств, нужно проделать те же действия, что и выше, предварительно установив переключатель на Property Class.

Примечание: блок, в котором было задействовано свойство «Subclass Information», отличается от других тем, что на иконке блока и всех его элементах появляются красные стрелки.

- **Comments** – в этом свойстве вы можете указать комментарии к созданному вами блоку, что позволяет конкретизировать этот объект.

2. Вкладка **Navigation** (Перемещение)
   - **Navigation Style** – это свойство определяет стиль перемещения:
     – Та же запись – этот параметр означает, что при перемещении на следующее или предыдущее поле запись не изменяется и при повторном возврате в это поле статус поля и блока не изменяется. Это значение Forms ставит по умолчанию при создании блока.
     – Изменить запись – этот параметр означает, что при перемещении на другое поле его статус переходит в режим обновления или Update.
     – Изменить блок данных – этот параметр определяется точно так же, как и предыдущий, только с той разницей, что статус Update принимает блок.
   - **Previous Navigation Data Block** – в этом поле вы можете указать, какой блок будет обозначен как предыдущий блок при перемещении. К примеру: у вас в Блоке1 этот параметр имеет значение Блок3; тогда в ситуации, если вы перешли с Блока2 на Блок3 и захотите вернуться на предыдущий блок, то вы вернетесь не Блок3, а на Блок2.
   - **Next Navigation Data Block** – это значение имеет такое же определение, как и предыдущее, с той лишь разницей, что навигация осуществляется на следующий блок, а не предыдущий.

3. Вкладка **Records** (Записи)
   - **Current Record Visual Attribute Group** (группа атрибутов визуализации текущей записи) – это свойство по умолчанию может принимать два типа значения: <Null> и Default. Если же вы создали свою группу атрибутов, то, соответственно, и она будет добавлена в список возможных значений. При создании блока это значение равно <Null>.
   - **Query Array Size** – определяет максимальное количество записей, одновременно выбираемое Forms'ом. Маленькое значение параметра увеличивает время отклика формы, а большое значение уменьшает полное время обработки, делая меньшее количество запросов к БД.
   - **Number of Records Buffered** – определяет минимальное количество строк, буферизуемых в памяти во время выполнения запроса в блоке. Все остальные записи буферизуются во временном файле на диске клиента. Большое значение увеличивает производительность работы формы со значительными объемами данных.

- **Number of Records Displayed** – в этом поле указывается количество отображаемых одновременно записей в блоке. Если значение равно «0» или «1», то в блоке отображается только одна запись; если, к примеру, этому полю присвоить значение «2», то в блоке будет выведено одновременно две записи, причем их элементы – item_text продублируются и создадутся автоматически самим Forms'ом.
- **Query all Records** – определяет, должны ли выбираться все записи, удовлетворяющие критерию запроса, или будет выбрано количество записей, равное параметру Query Array Size блока. В подавляющем большинстве случаев при работе с большими объемами данных в форме не следует устанавливать его в «Yes». Значение «Yes» обычно используется в формах, использующих вычисление итогов.
- **Record Orientation** – определяет ориентацию записи. Принимает два значения – «по вертикали» и «по горизонтали», причем значение «по вертикали» является значением по умолчанию.
- **Single Record** (единичная запись) – принимает значения «Yes» и «No». По умолчанию Forms ставит это значение в «No», что дает возможность отображать более одной записи.

4. Вкладка **Database** (база данных)
   - **Database Data Block** – принимает значения «Yes» и «No». При создании блока данных, как с помощью мастера, так и вручную, Forms по умолчанию ставит этот параметр на «Yes», то есть по умолчанию создаваемый блок становится блоком данных вашей БД. Если параметр имеет значение «No», то блок не является объектом БД.
   - **Enforce Primary Key** – обеспечивает первичный ключ. Если параметр принимает значение «Yes», то, соответственно, блок обеспечивается первичным ключом средствами Forms.
   - **Query Allowed** (запрос разрешен) – если значение этого параметра «Yes», тогда запросы в этом блоке разрешены, и наоборот, если «No».
   - **Query Database Type** (запросить тип исходных данных) – в этом свойстве значением параметра является указание типа данных, на основе которых вы создаете ваш блок данных. Типы данных: «Ничего», «Таблица», «Процедура», «Триггеры транзакций», «Фраза From».
   - **Query Database Name** (запросить имя источника данных) – при входе в это поле Forms запускает редактор, в котором вы определяете имя источника данных в зависимости от выбранного вами параметра в свойстве Query Database Type. Это означает, что если в типе исходных данных вы указали имя таблицы, то и при указании запрашиваемого имени источника данных вы должны указать имя таблицы; если вы указали процедуры триггеров транзакций, то также необходимо указать имена типа этих данных; если же типом данных является фраза

FROM, то именем источника будет являться завершение фразы From с указанием имени таблицы, причем в конце фразы символ «;» опускается.

- **Query Database Columns** – при выборе этого свойства запускается одноименное окно – «Запросить столбцы исходных данных», в котором вы выбираете запрашиваемые столбцы (Column Names), их тип (Type), а также параметры масштабирования (Scale, Length).
- **Query Database Arguments** – этот параметр работает по такому же принципу, что и предыдущий, с той лишь разницей, что в запускаемом окне «Запросить аргументы исходных данных» параметры Argument Names (имена аргументов), тип аргумента (Type), режим (Mode – IN и OUT) и значение аргумента (Value) предназначены для процедуры, а не для таблицы.
- **Alias** (псевдоним) – синоним, связанный с источником данных.
- **Include Ref Item** – если параметру этого свойства присвоить значение «Yes», то создастся дополнительный элемент Ref, содержащий дополнительный ID (идентификатор) объекта.
- **Where Clause** (фраза where) – запускает редактор Forms'a, в котором вы набираете условие фразы where, причем само слово вы уже не набираете; также не забудьте, что в конце вашего SQL-предложения символ «;» опускается.
- **Order by Clause** (фраза order by) – этот параметр определяется так же, как и предыдущий, только в данном случае вы достраиваете фразу «order by».
- **Optimizer Hint** (подсказка оптимизатору) – здесь вы указываете строку подсказки, передаваемую оптимизатору для выбора наилучшего пути построения запроса. Это свойство важно учесть, т. к. Oracle Forms 6i поддерживает материализованные представления (materialized view).
- **Insert Allowed** (вставка разрешена) – если значению этого свойства присвоить «No», то Forms запретит вставку в набор данных, определяемых этим блоком, – и разрешит, если присвоить «Yes». По умолчанию значение этого свойства и родственных ему Delete и Update ставится на «Yes», то есть разрешено.
- **Update Allowed** (обновление разрешено) – если значению этого свойства присвоить «No», то Forms запретит обновление в наборе данных, определяемых этим блоком, – и разрешит, если присвоить «Yes».
- **Locking Mode** – этим свойством задается режим блокирования записей. По умолчанию этому свойству Forms присваивает значение «Автоматическое», то есть Forms сам выбирает режим блокировки из двух: «Немедленный», в котором блокировка происходит сразу же при запросе строк, соответствующих запрашиваемым записям, или

«С задержкой» – в этом случае соответственно блокировка происходит после завершения вашей операции, то есть с задержкой.

- **Delete Allowed** (удаление разрешено) – если значению этого свойства присвоить «No», то Forms запретит удаление в наборе данных, определяемых этим блоком, – и разрешит, если присвоить «Yes».
- **Key Mode** (режим клавиш) – это свойство определяет способ уникальной идентификации строк в базе данных. Свойство может принимать четыре значения:
  - автоматический;
  - уникальный;
  - обновляемый;
  - необновляемый.
- **Update Changed Columns Only** (обновить только изменяемые столбцы) – здесь вроде бы название говорит само за себя, то есть задается при обновлении задается команда обновить все строки (если значение этого свойства «Yes») или обновить только измененные строки (если значение этого свойства «No»); но использование этого параметра очень влияет на производительность вашего приложения. Описание и рекомендации вы найдете в разделе «Как происходит обновление в Oracle Forms 6i».
- **Enforce Column Security** (обеспечить защиту столбца) – этот параметр отвечает за обеспечение привилегий на обновление и вставку для столбца. По умолчанию это свойство имеет значение «No».
- **Maximum Query Time** (максимальное время запроса) – здесь вы указываете максимальное время на ожидание выполнения запроса до прерывания. По умолчанию оно равно «0», то есть время выполнения запроса до прерывания неограниченно.
- **Maximum Record Fetched** (максимум выбранных записей) – этот параметр отвечает за максимальное количество выбранных строк, после которого можно прервать запрос. По умолчанию это значение равно «0».

5. Вкладка **Advanced Database** (Продвинутая база данных)
   - **DML Data Target Type** (тип адресата DML данных) – определяет тип источника данных для выполнения операций DML, имеет четыре возможных параметра: Null (ничего), таблица, процедура и триггеры транзакций. Автоматически это свойство принимает такое же значение, как и тип источника данных.
   - **DML Data Target Name** (имя адресата DML данных) – это свойство принимает значения имени источника данных.
   - **Insert Procedure Name** (вставить имя процедуры) – вставляет в поле значения свойства имени процедуры, заданное вами.

- **Insert Procedure Result Set Columns** (вставить столбцы набора результатов процедуры) – при выборе этого свойства запускается окно редактора процедуры с одноименным названием, имеющее такой же вид и назначение, как и окно свойства Query Database Arguments вкладки Database.
- **Insert Procedure Arguments** (вставить аргументы процедуры) – вставить новые аргументы определенной вами процедуры, аналогично свойству Query Database Arguments вкладки Database.
- **Update Procedure Name** (обновить имя процедуры) – это поле предназначено для ввода нового имени процедуры.
- **Update Procedure Result Set Columns** (обновить столбцы набора результатов процедуры) – запускает окно обновления аргументов процедуры, содержащее такие же опции, как и в случае с Insert Procedure Result Set Columns.
- **Update Procedure Arguments** (обновить аргументы процедуры) – запускает окно обновления аргументов процедуры, аналогичное Insert **Procedure Arguments.**
- **Delete Procedure Name** (удалить имя процедуры) – удаляет имя выбранной процедуры.
- **Delete Procedure Result Set Columns** (удалить столбцы набора результатов процедуры) – запускает окно типа Insert Procedure Result Set Columns, в котором вы можете удалить перечисленные вами столбцы, в поле Column Names.
- **Delete Procedure Arguments** (удалить аргументы процедуры) – запускает окно типа Insert Procedure Arguments, только предназначенное для удаления аргументов процедур.
- **Lock Procedure Name** (блокировать имя процедуры) – в этом свойстве требуется указать имя процедуры, которую вы хотите заблокировать.
- **Lock Procedure Result Set Columns** (блокировать столбцы набора результатов процедуры) – при выборе этого параметра запускается окно типа Insert Procedure Result Set Columns, только в этом случае в поле Column Names вы указываете имена столбцов, которые хотите заблокировать.
- **Lock Procedure Arguments** – при выборе этого свойства запускается окно «Блокировать аргументы процедуры», в котором в поле Argument Names вы указываете имена аргументов вашей процедуры, которые вы хотите заблокировать.
- **DML Array Size** – максимальный размер массива для вставки, обновления и удаления записей за один раз.
- **Precompute Summaries** (предварительный подсчет итогов) – этот параметр говорит Forms, надо ли вычислять значения всех суммируемых элементов в блоке до запуска обычного запроса по блоку.

- **DML Returning Value** (возвращаемое значение командой DML) – определяет, должен ли этот блок использовать команды DML с возвращаемыми значениями.

6. Вкладка **Scrollbars** (Полоса прокрутки)
   - Show Scroll Bar – принимает два значения – «Yes» или «No». Для отображения Scroll Bar выбирайте значения «Yes».
   - Scroll Bar Canvas – в этом поле вам нужно указать вид-картинку (Canvas), на которой вы хотите разместить Scroll Bar.
   - Scroll Bar Tab Page (страница вкладки с полосой прокрутки) – в этом свойстве вы можете указать страницу вкладки, на которой будет находиться ваша полоса прокрутки. Если вкладок нет, то значение этого поля равно «Null».
   - Scroll Bar Orientation (ориентация полосы прокрутки) – в этом свойстве задается значение направления положения полосы прокрутки – «по вертикали» и «по горизонтали».
   - Scroll Bar X Position – позиция Scroll Bar по оси Ox.
   - Scroll Bar Y Position – позиция Scroll Bar по оси Oy.
   - Scroll Bar Width – устанавливает ширину Scroll Bar.
   - Scroll Bar Length – устанавливает высоту Scroll Bar.
   - Reverse Direction – определяет, допускает ли линейка прокрутки просмотр в обратном направлении.

7. Вкладка **Visual Attributes** (Визуальные атрибуты)
   - **Visual Attribute Group** (группа атрибутов визуализации) – это группа свойств, которые вы можете назначить своему объекту, в данном случае блоку. по умолчанию это значение Default.
   - **Character Mode Logical Attribute** (логический атрибут символьного режима) – здесь вы задаете базисные атрибуты устройства символьного режима.
   - **White on Black** – в этом свойстве вы указываете, как объекты будут отображаться на монохромном растровом дисплее. По умолчанию это значение <Не указано>, остальные два параметра – «Да» и «Нет», где «Да» означает, что объект отображается белым цветом на черном фоне.

8. Вкладка **Color** (Цвет)
   - **Foreground Color** (цвет отображаемых на экране символов) – задает цвет для переднего плана объекта, в нашем случае это элементы текста. При выборе этого свойства запускается окно «Набор шаблонов закраски». Для этого запустите окно с палитрой закраски, затем выберите нужный вам цвет и щелкните по нему, после чего окно

автоматически закроется и в поле этого свойства отобразится код выбранного вами цвета.

- **Background Color** (цвет фона) – здесь вы указываете цвет фона (фоновой области). Закраска фона выполнятся точно так же, как и в Foreground Color.
- **Fill Pattern** (шаблон заполнения) – здесь вы выбираете рисунок (орнамент) для заполнения объекта, причем цвет рисунка будет соответствовать цвету отображаемых символов, а цвет фона – фоновой части.

9. Вкладка **Character Mode** (Символьный режим)
   - **Listed in Data Block Menu** (перечислено в меню блока данных) – определяет, должен ли блок быть указан в меню блоков данных символьного режима.
   - **Data Block Description** – в этом поле вы можете дать описание блоку данных, которое будет использовано как в меню блоков данных.

10. Вкладка **International** (Международный)
    - **Direction** (направление) – направление разметки для двунаправленных объектов размеров, используется в «NLS-приложениях (National Language Standart)» . Задается одним из трех возможных параметров: «По умолчанию», «Слева направо», «Справа налево». Первое значение является значением по умолчанию, которое задает Forms этому свойству при создании блока данных.

Настройка некоторых свойств блока

В этом разделе мы рассмотрим свойства блока, настройки которых влияют на производительность приложения. Когда ваша система достигнет огромных размеров и в ваших таблицах будет не одна тысяча записей, вы начнете ощущать падение производительности формы, а именно:

– увеличение времени выполнения запросов;

– задержку вычисления итогов;

– увеличение размеров временных файлов, хранящих буферизованные данные.

На уровне блока вы можете управлять следующими свойствами, которые влияют на производительность работы формы:

- **Query All Records.** Определяет, должны ли выбираться все записи, удовлетворяющие критерию запроса, или будет выбрано количество записей, равное параметру Query Array Size блока. В подавляющем большинстве случаев при работе с большими объемами данных в форме не следует устанавливать это значение на «Yes». «Yes» обычно применяется в формах, где происходит вычисление итогов.

- **Query Array Size.** Определяет максимальное количество одновременно выбираемых записей. Маленькое значение параметра увеличивает время отклика формы, а большое значение уменьшает полное время обработки, делая меньшим количество запросов к БД.
- **Number of Records Buffered.** Определяет минимальное количество строк, буферизуемых в памяти во время выполнения запроса в блоке. Все остальные записи буферизуются во временном файле на диске клиента. Большое значение увеличивает производительность работы формы со значительными объемами данных.
- **DML Array Size.** Определяет максимальный размер массива для вставки, обновления и удаления строк в БД в один момент времени. Большой размер уменьшает время обработки транзакций, снижая сетевой трафик с БД, но требует большего количества памяти. Оптимальный размер – количество строк, изменяемое пользователем в одной транзакции. Существуют некоторые ограничения на использование больших значений этого параметра.

Вы также можете увеличить время отклика формы за счет использования свойства «WHERE Clause», в котором вы можете указать критерий выборки. Если же все-таки вам необходимо вернуть все записи, а не отобранные по какому-либо критерию, вы можете использовать предикаты, которые повлияют на план выполнения запроса. Если ваша таблица содержит первичный ключ или любой другой столбец, имеющий индекс, вы можете сделать так, чтобы оптимизатор пошел не по пути полного просмотра таблицы (FULL SCAN), а использовал при просмотре индекс. Чтобы понять, как это сделать, выполните пример, который приведен ниже.

1. Зайдите в SQL*Plus или любое другое доступное GUI-средство, например, Sql Developer.
2. Создайте таблицу Test:

```
CREATE TABLE test as SELECT ROWNUM VAL FROM
(SELECT * FROM DUAL CONNECT BY LEVEL<=10000)
```

3. Если вы используете SQL*PLUS, то выполните команду утилиты для включения просмотра плана выполнения запроса:

```
SET autotr ON
```

4. Выполните запрос на выборку всех записей и посмотрите на план выполнения запроса – в нем для просмотра таблицы был выбран метод FUUL SCAN.
5. Создайте индекс на столбец VAL таблицы Test, для этого выполните команду SQL:

```
CREATE INDEX I$VAL ON Test (val);
```

6. Повторите запрос на выборку всех записей и посмотрите на план выполнения запроса – он не изменился, оптимизатор все так же выполняет полное сканирования таблицы. Чтобы изменить план запроса, добавьте предикат в условие WHERE:

```
SELECT * FROM Test WHERE val<=100000
```

7. Посмотрите план выполнения запроса – он изменился. Мы всего лишь добавили предикат с условием, которое будет истинно в любом случае, так как в нашем столбце максимальное значение 10000. Вы можете делать то же самое и с оператором BETWEEN, указывая нереальные граничные значения.

Из этого примера можно сделать вывод, что вы можете также добавлять нереальные граничные значения в «WHERE Clause» блока и влиять тем самым на время отклика формы, так как выбор по индексу в таблице с большим объемом данных – это всегда быстрее, чем полный просмотр таблицы.

Лекция 8. Блоки. Создание реляционных отношений. Использование свойств ORDER BY и WHERE Clause

В лекции рассматривается способ организации реляционной зависимости между блоками – relationship и свойства, имитирующие стандартные фразы SQL – Where и Order by.

Ключевые слова: Data Block (блок данных), Data Block Wizard, Layout Editor, ORDER BY и WHERE Clause, relationship.

Цель лекции: научить слушателя создавать блоки данных и реляционные отношения между ними. Ознакомить слушателя с одними из самых главных свойств блока – ORDER BY и WHERE Clause.

Создание Мастер-Деталь блоков

Часто в приложениях необходимо отобразить одну или более записей из одной таблицы, ассоциированных с записями из другой таблицы по какому-либо признаку. В Oracle Forms существует несколько способов реализации отношения Мастер-Деталь между блоками формы. Один из самых быстрых и эффективных способов создания отношения между новыми или существующими блоками – использование Data Block Wizard, то есть мастера создания блоков, который позволяет быстро определить отношения между блоками. Data Block Wizard позволяет не только задавать новые отношения между блоками, но и автоматически создавать связь между блоками, если таковая определена на уровне базы данных. В этом разделе мы научимся создавать реляционные отношения между блоками и корректировать их.

Прежде чем приступить к выполнению конкретного примера, давайте ознакомимся с вариантами представлений отношений и степенью их вложенности.

«Мастер_1-Деталь_1» – это стандартная и наиболее часто используемая схема (рис. 8.1), когда одной таблице Мастер соответствует много записей Деталь, то есть связь один ко многим.

«Мастер_1-Деталь_1 | Мастер_2-Деталь_1 » – эта схема (рис. 8.2) показывает, что у одной таблицы может быть две мастер-таблицы. Такой способ построения отношений между блоками встречается крайне редко, так как неудобен.

«Мастер_1-Деталь_1(Мастер)-Деталь_2» – эта схема (рис. 8.3) аналогична первой, с той лишь разницей, что блок Деталь_1 выступает как Мастер для блока Деталь_2, то есть является одновременно Мастером и Деталью.

Рис. 8.1.

Рис. 8.2.

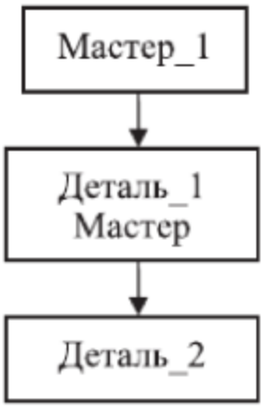

Рис. 8.3.

Рис. 8.4.

«Мастер_1-Деталь_1 | Мастер_1-Деталь_2» – эта схема (рис. 8.4) также аналогична первой, с той лишь разницей, что выступает Мастером для двух не связных между собой деталь-блоков.

Создание отношения Мастер-Деталь

Предварительная подготовка

Ниже приведены три таблицы с описанием структуры и скриптами для их создания.

Таблица 8.1. Описание письма и заказчика (таблица PZAK):

Атрибут	Тип	Значение	Описание
Pr_key	NUMBER		Первичный ключ
Nlet	VARCHAR2	20	Номер письма
Pdat	DATE	YY.MM.DD	Дата письма
Nfirm	VARCHAR2	100	Наименование фирмы
Tplus	NUMBER		Толеранс плюс
Tmin	NUMBER		Толеранс минус
Yslp	VARCHAR2	300	Условия поставки
Srpost	VARCHAR2	300	Срок поставки
sdat	DATE	YY.MM.DD	Дата загрузки письма

Таблица 8.2. Описание товара (таблица PPZAK):

Атрибут	Тип	Значение	Описание
Pr_key	NUMBER		Первичный ключ
Fk_pzak	NUMBER		Внешний ключ на таблицу PZAK
Smarka	VARCHAR2	100	Марка стали
Dopls	VARCHAR2	300	Дополнительные условия
Doptr	VARCHAR2	300	Дополнительные требования
Tmin	NUMBER		Толеранс минус
Tplus	NUMBER		Толеранс плюс
Part	NUMBER		Партия
Stand	VARCHAR2	100	Стандарт на марку стали

Таблица 8.3. Характеристика товара (таблица PZAK_ATR):

Атрибут	Тип	Значение	Описание
Pr_key	NUMBER		Первичный ключ
Fk_ppzak	NUMBER		Внешний ключ на таблицу PPZAK
Tls	NUMBER		Толщина листа
Shr	NUMBER		Ширина листа
Dln	NUMBER		Длина листа
Ton	NUMBER		Тоннаж
Gost	VARCHAR2	100	Стандарт на толщину
Num	NUMBER		Номер позиции заказа

Скрипты для создания таблицы

1. Описание письма и заказчика:

```
Create table PZAK
(
pr_keynumber PRIMARY KEY,
nlet  varchar2(20),
pdat  date,
nfirm varchar2(100),
tplus number,
tmin  number,
```

```
yslp  varchar2(300),
srpostvarchar2(100),
rnum  number,
sdat  date
)
```

2. Описание товара:

```
Create table PPZAK
(
Pr_keynumber PRIMARY KEY,
Fk_pzak  number,
Smarkavarchar2 (100),
Priem    varchar2 (100),
Dopls varchar2 (300),
Doptr varchar2 (300),
Part  number,
Stand varchar2 (100),
)
```

3. Характеристика товара:

```
Create table PZAK_ATR
(
Pr_keynumber PRIMARY KEY,
Fk_ppzak number,
tlsnumber,
shrnumber,
dlnnumber,
tonnumber,
gost  varchar2 (100),
)
```

После того как таблицы созданы, можно перейти к выполнению примера.

Пример, над которым мы сейчас начнем работать, будет использоваться на протяжении всего материала. Наша задача – получить приложение, задействующее основные возможности Oracle Forms. Мы будем создавать программу приема и обработки заказов.

1. Создайте новую форму и сохраните ее под именем Zak.fmb.
2. Вызовите мастер блока данных и выберите тип создаваемого блока данных – таблица или представление. Нажмите кнопку «Далее».

3. Нажмите кнопку «Обзор» и в появившемся окне (рис. 8.5) выберите таблицу PZAK и подтвердите выбор.

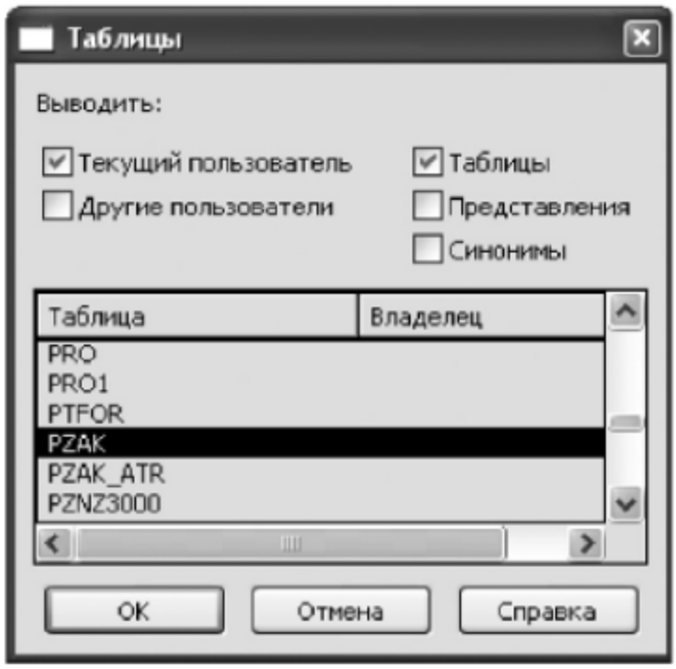

Рис. 8.5. Окно выбора объекта

4. Перенесите все «доступные столбцы» из левого списка в список «базовых элементов» (рис. 8.6) и нажмите кнопку «Далее» для вызова мастера макетов.

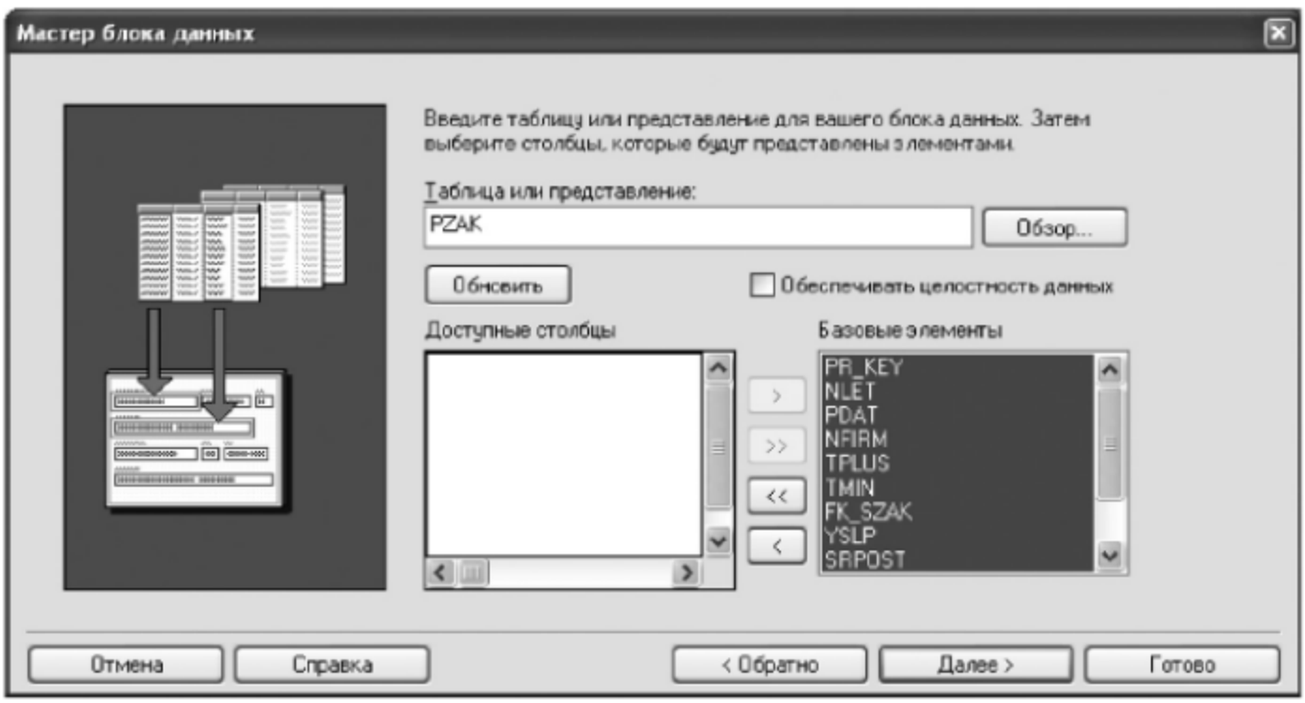

Рис. 8.6. Мастер блока данных. Выбор таблицы и атрибутов

5. В появившемся окне (рис. 8.7) «Мастер макетов» выберите канву, на которой будут расположены элементы блока данных. Тип канвы выберите «Основная». Нажмите кнопку «Далее».

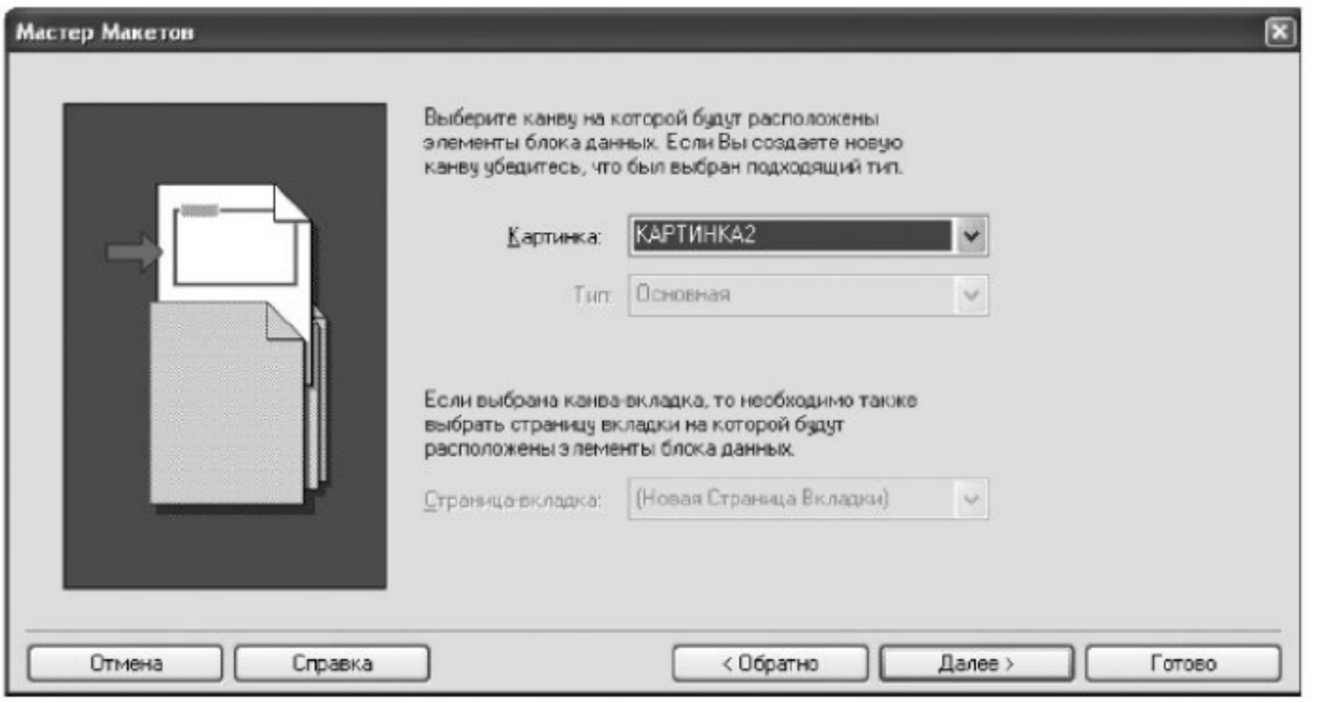

Рис. 8.7. Мастер макетов. Выбор элементов

6. Из списка имеющихся элементов выберите все кроме PR_KEY. После того как вы перенесете все элементы в список «Выводимые элементы», выберите тип элемента (рис. 8.8), соответствующий типу, который указан в таблице PZAK. Нажмите кнопку «Далее».

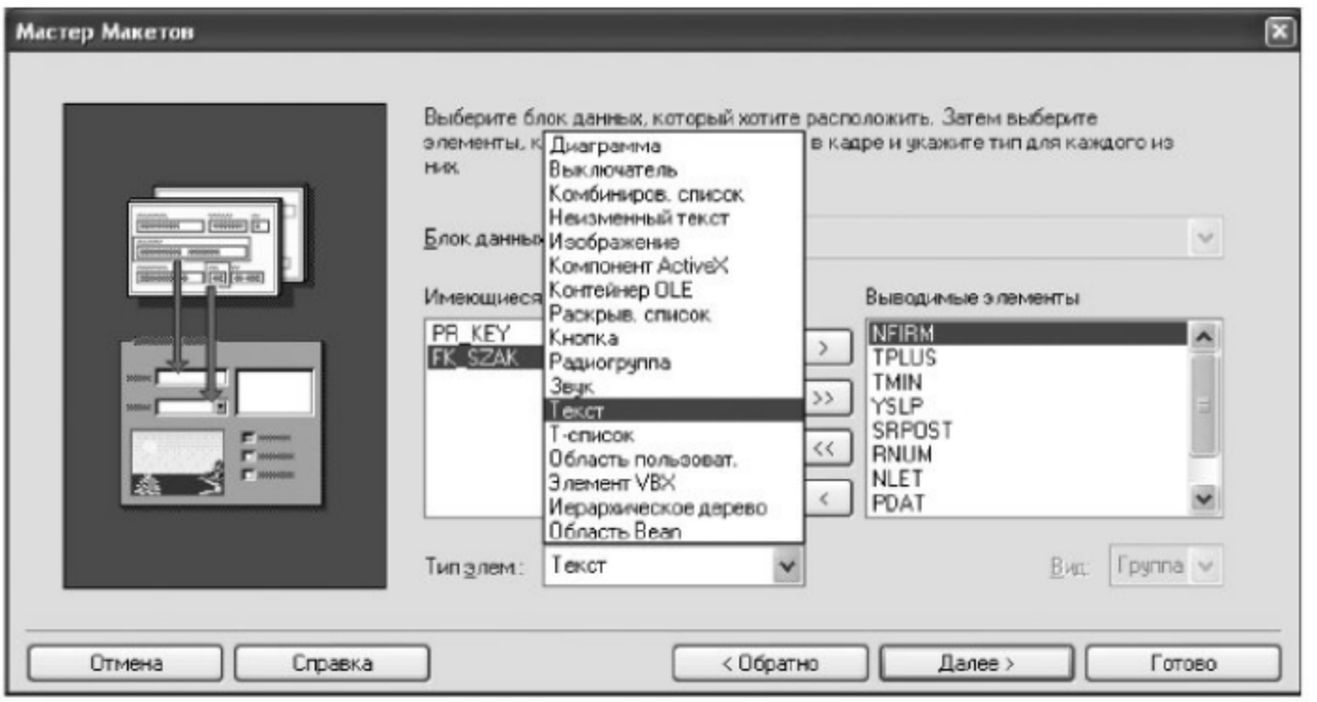

Рис. 8.8. Мастер макета. Выбор типа элемента

7. В этом окне (рис. 8.9) вам предлагается подписать и установить значение ширины и высоты элемента. Вы можете пропустить этот этап и установить эти характеристики позже. Нажмите кнопку «Далее».

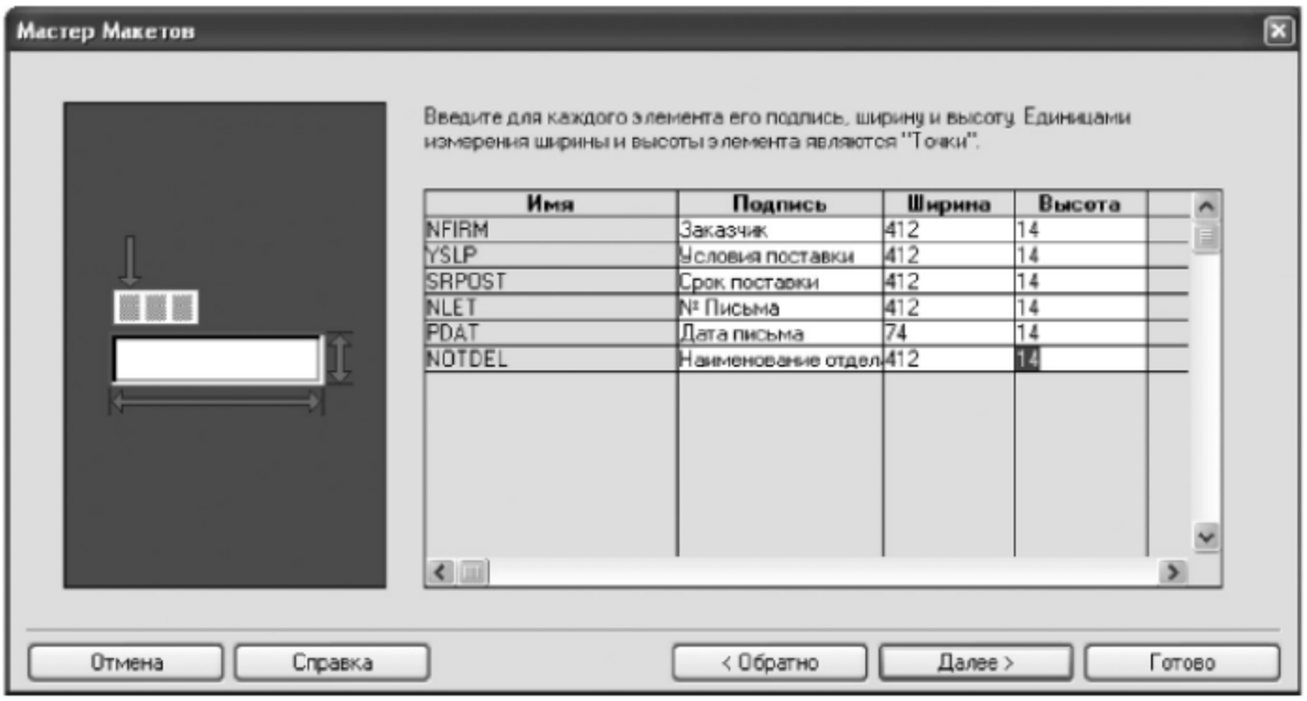

Рис. 8.9. Мастер макетов. Характеристики элементов

8. В этом окне (рис. 8.10) вам предлагается выбрать стиль размещения и отображения элементов. По умолчанию предлагается стиль «Форма», который нам как раз и необходим. Для завершения работы с мастером макетов нажмите кнопку «Далее».

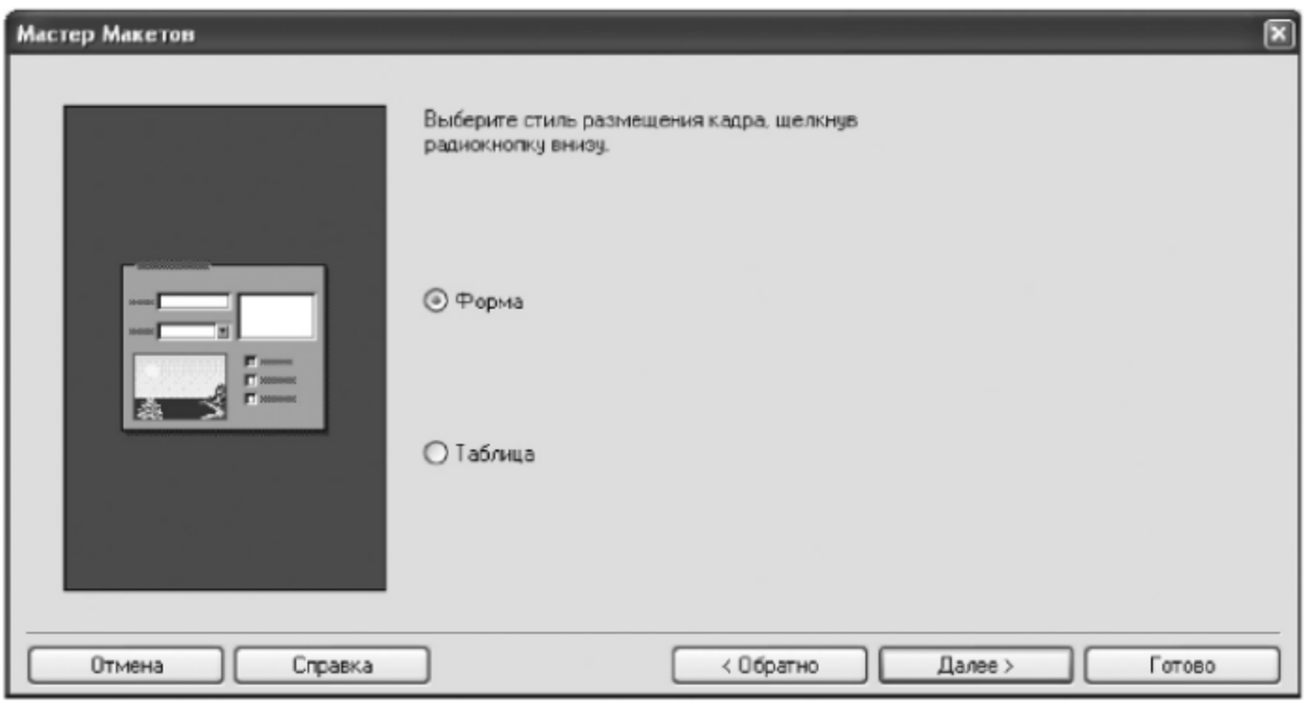

Рис. 8.10. Мастер макетов. Стиль размещения элементов

В нашем примере мы реализуем схему «Мастер_1-Деталь_1(Мастер)-Деталь_2», как показано на (рис. 8.10). Для создания такой схемы требуется как минимум три блока, поэтому нам нужно создать еще два блока. У нас эта схема будет иметь вид:

PZAK-PPZAK-PZAK_ATR

Продолжим выполнение примера и создадим следующий блок – деталь блок PPZAK. Для этого, как и в предыдущем примере, вызовите мастер блока данных и проделайте то же, что и при создании первого блока, только таблицу для вашего блока выберите с именем PPZAK. После выбора таблицы перенесите все доступные столбцы из левого списка в список базовых элементов (см. рис. 8.6) и перейдите на следующий этап. В отличие от первого случая, когда создавался первый блок данных, на экране появится новое окно (рис. 8.11).

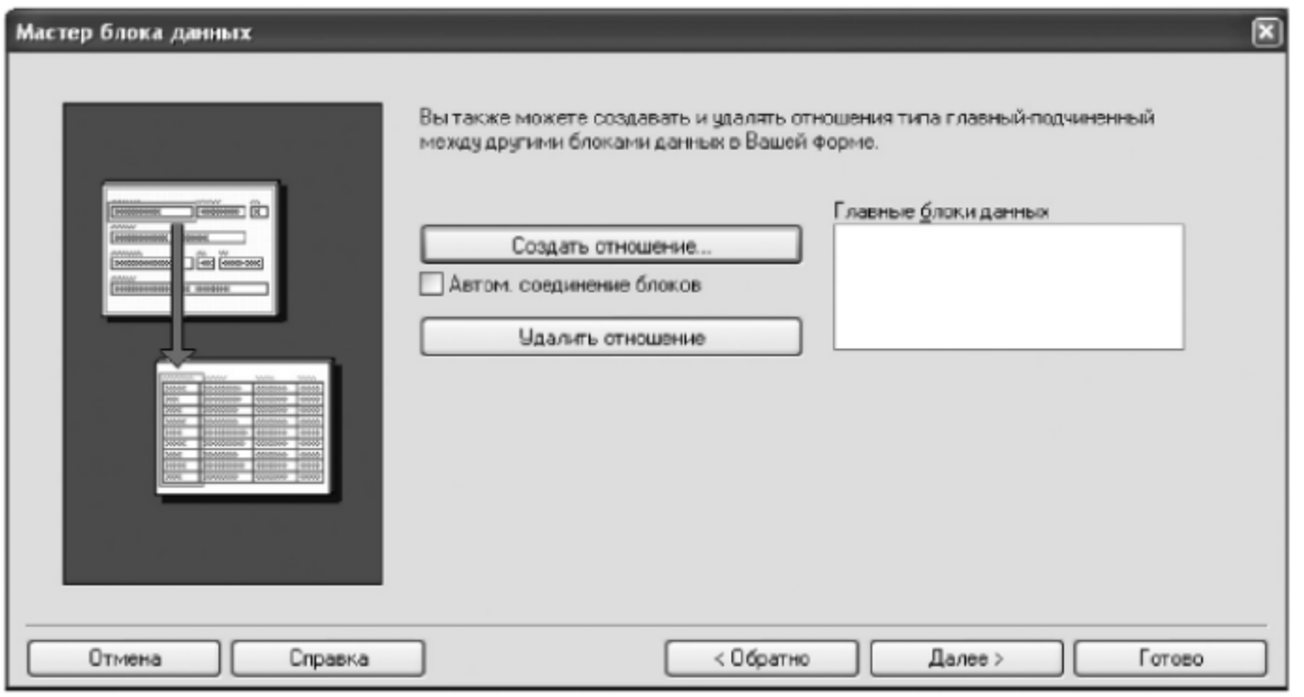

Рис. 8.11. Мастер блока данных. Создание отношений

В появившемся окне вы можете создавать или удалять отношения между блоками данных. Это окно будет появляться всякий раз при создании нового блока, если в вашем модуле имеется хотя бы один блок. Окно имеет три управляющих элемента:

- кнопка «Создать отношение»;
- флажок «Автоматическое сединение блоков»;
- кнопка «Удалить отношение».

Для создания нового отношения нажмите кнопку «Создать отношение». На экране появится диалоговое окно с вариантами создания соединения (рис. 8.12).

В нашем примере будет рассмотрен вариант создания отношения из условия соединения. Для этого установите значение переключателя на первый вариант – «создать новое отношение из условия соединения» – и подтвердите выбор. В появившемся окне (рис. 8.13) выберите блок, который будет мастер для блока PPZAK, то есть блок PZAK, и нажмите кнопку «OK».

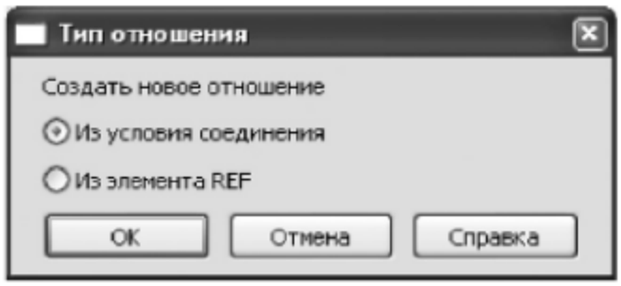

Рис. 8.12. Тип отношения

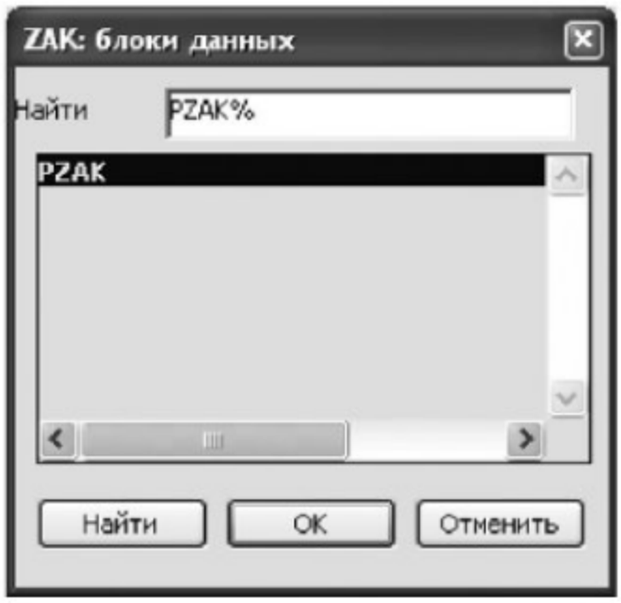

Рис. 8.13. Перечень блоков модуля

После выбора главного блока в первоначальном окне появятся дополнительные элементы управления для создания отношения (рис. 8.14):

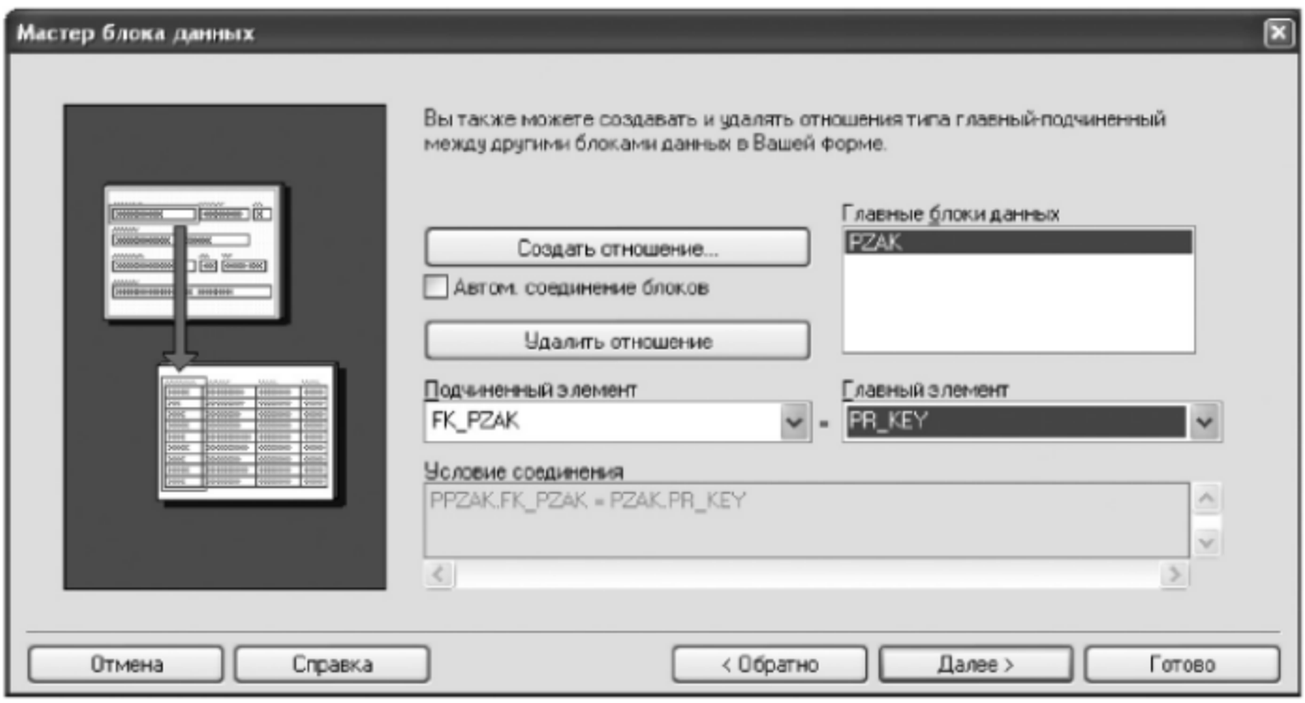

Рис. 8.14. Мастер блока данных. Создание условия соединения

- подчиненный элемент;
- главный элемент.

В качестве подчиненного элемента таблицы PPZAK выберем элемент FK_PZAK, в качестве главного элемента выберем элемент PR_KEY «мастер» таблицы PZAK. Таким образом, одной записи таблицы PZAK будет соответствовать одна или более записей таблицы PPZAK. После того как будут выбраны главный и подчиненный элементы, в списке «Условие соединения» появится условие соединения:

```
PPZAK.FK_PZAK = PZAK.PR_KEY
```

Отношение создано, нажимаем кнопку «Далее» для вызова «мастера макетов». Используя «мастер макетов», расположите элементы блока PPZAK на канве КАРТИНКА 2, на которой уже расположены элементы первого блока:

1. Выберите канву, на которой расположены элементы первого блока – «КАРТИНКА 2», – и нажмите кнопку «Далее».
2. Из списка имеющихся элементов выберите все кроме PR_KEY и FK_PZAK. После того как вы перенесете все элементы в список «Выводимые элементы», выберите тип элемента (рис. 14), соответствующий типу, который указан в таблице PPZAK, и нажмите кнопку «Далее».
3. Стиль размещения выберите «Форма».
4. Подпишите элементы, установите для них значение ширины и высоты и нажмите кнопку «Готово».

Использование свойств блока ORDER BY и WHERE Clause

Сортировка и извлечение записей, удовлетворяющих определенному критерию поиска, являются наиболее часто выполняемыми нами операциями. В этом разделе вы узнаете, как устанавливать значения свойств ORDER BY и WHERE Clause программно и на этапе проектирования.

Свойство ORDER BY

ORDER BY определяет значение одноименной конструкции SQL, добавляемой к команде SELECT по умолчанию и относящейся к блоку. То есть значение этого свойства определяет порядок сортировки записей в блоке. Рассмотрим случай, когда вам необходимо, чтобы выбранные вами записи были отсортированы по какому-либо признаку. Если вы пишете запрос в SQL*Plus или в любой другой среде, это будет выглядеть следующим образом:

```
Select * from table_name ORDER BY col_name asc (desc)
```

Одним словом, чтобы отсортировать записи в таблице, вы просто добавите конструкцию ORDER BY с условием сортировки. В Oracle Forms так нельзя, поскольку все базовые блоки строятся на основе запроса вида select * from table_name, исключением являются лишь представления, в которых вы можете заранее отсортировать записи. Поэтому чтобы добавить условие сортировки к вашему блоку, вам нужно установить значение свойства «Фраза ORDER BY». Значение этого свойства можно установить двумя способами:

- в режиме проектирования формы, определив значение этого свойства;
- в режиме исполнения, то есть меняя порядок сортировки по какому-либо событию или условию.

Рассмотрим первый способ, для этого выполним пример.

1. Откройте форму ZAK.fmb.
2. В объектном навигаторе выберите блок PZAK. Откройте палитру свойств и перейдите к свойству «Фраза ORDER BY» в узле «База данных».
3. Для того чтобы отсортировать записи в блоке по дате заявки в порядке убывания, установите значение свойства равным PDAT desc.
4. Запустите форму и выполните запрос.

Как видите, все записи отсортировались так, как мы задали в свойстве. В качестве критерия сортировки в блоке вы также можете задавать не только имена столбцов, но и параметры, глобальные переменные и подзапросы. Например, не будет ошибкой, если вы установите свойство ORDER BY в:

1. :global.p_dat asc;
2. :parameter.p_dat desc;
3. (select pdat from pzak) desc;
4. pdat desc, nfirm asc.

Первые два варианта позволяют передать в качестве критерия сортировки значение глобальной переменной или параметра. Если значение параметра или глобальной переменной изменится, то изменится и порядок сортировки. Этот способ не самый лучший, хотя и позволяет влиять на сортировку в режиме выполнения программы, т. к. значение параметра или переменной можно изменять динамически. Использование подзапроса в качестве критерия сортировки позволяет более гибко подойти к выбору критерия сортировки, так как иногда критерием может служить столбец из другой таблицы. Последний вариант отображает стандартную составную сортировку.

Несмотря на различные возможности установки критерия сортировки в режиме проектирования формы, этот подход является жестким, так как для изменения или добавления новых требований к отображению данных придется редактировать форму и генерировать ее заново. Такой подход также неприемлем в масштабируемых приложениях, в которых требования к отображению данных меняются очень часто.

Теперь давайте рассмотрим второй способ, в котором условие сортировки можно изменять в ходе выполнения программы в зависимости от требований пользователя. Этот подход является более гибким в отличие от предыдущего, так как условие сортировки можно отключить или поменять в любой момент. В Oracle Forms можно программно влиять практически на все установки блока с помощью процедуры SET_BLOCK_PROPERTY. У этой процедуры достаточно много вариантов описаний, но нам пригодятся только два.

Синтаксис

```
SET_BLOCK_PROPERTY
   (block_id   Block,
   property VARCHAR,
       value   VARCHAR);

SET_BLOCK_PROPERTY
  (block_name  VARCHAR2,
   property VARCHAR,
    value   VARCHAR);
```

Параметры

block_id – уникальный идентификатор, который Forms Builder связывает с элементом при создании. Функция поиска идентификатора – FIND_Block.

block_name – имя блока, свойство которого вы хотите установить. Тип данных – Varchar2.

property – определяет свойство блока, которое вы устанавливаете; в данном случае это свойство «Фраза ORDER BY».

Внимательно изучив синтаксис процедуры, можно приступать к выполнению примеров. Начнем с самого простого.

1. Откройте форму Zak.fmb.
2. В объектном навигаторе выберите блок PZAK. Откройте палитру свойств и перейдите к свойству «Фраза ORDER BY» в узле «База данных». Если свойство заполнено, то отчистите его.

3. Перейдите в объектный навигатор и создайте триггер PRE-QUERY на уровне блока PZAK. В триггере напишите и скомпилируйте следующий код:

```
SET_BLOCK_PROPERTY ('PZAK', ORDER_BY, 'pdat desc');
```

4. Запустите форму и выполните запрос.

Мы выполнили очень простой пример, который наглядно показывает, как программно устанавливать свойство блока. Этот пример не раскрывает всех преимуществ и улучшений в установке свойств программно. Рассмотрим более сложный пример, когда перед нами стоит задача осуществить сортировку в любом поле по нажатии кнопки.

В форме PZAK.fmb удалите триггер PRE_QUERY, а вместо него создайте триггер на уровне блока WHEN_NEW_ITEM_INSTANCE. В теле триггера напишите следующий код:

```
  declare
cr_item varchar2 (20):=:system.current_item;
begin
  : global.sort_param:= cr_item;
  set_block_property ('block_name', ORDER_BY, cr_item ||' asc');
  end;
```

В этом триггере мы объявили глобальную константу global.sort_param – она будет запоминать имя текущего элемента при перемещении, по которому мы будем производить сортировку. Разместите на форме новую кнопку, по нажатию которой будет происходить сортировка. Создайте событие WHEN_BUTTON_PRESSED кнопки и в теле триггера напишите следующий код:

```
   go_block ('block_name');
clear_block;
execute_query;
   go_item (:global.sort_param);
```

Запустите форму и выполните запрос. Для того чтобы отсортировать записи в блоке по какому-нибудь столбцу, достаточно будет перенести фокус курсора в нужное поле и нажать кнопку. Можете переходить на любой элемент и сортировать по нему остальные, причем заметьте, что после извлечения записи курсор остается в том же поле, а не перемещается на исходный элемент благодаря тому, что мы после извлечения записей перемещаем курсор обратно в поле, имя которого запоминаем в глобаль-

ной переменной. Если же мы закомментируем последнюю строчку триггера, то после выполнения сортировки курсор всегда будет возвращаться на начальный элемент.

После того как мы рассмотрели различные способы сортировки в блоке, можно сделать вывод, что управление выбором критерия сортировки в ходе выполнения программы более гибко. В последнем примере мы обращались к полям через системную переменную, что позволило нам не привязываться жестко к элементам, и такой код можно легко перенести в любое приложение. Если мы захотим расширить нашу форму новым элементом, то нам не придется править программу, будет достаточно сгенерировать форму заново.

Свойство WHERE Clause

Выборка записей, удовлетворяющих определенному критерию, является наиболее часто выполняемой нами операцией. В этом разделе вы узнаете, как устанавливать значения свойства WHERE Clause программно и на этапе проектирования.

Конструкция WHERE Clause позволяет ограничить выборку записей или задать определенный критерий выборки.

Свойство WHERE Clause определяет значение одноименной конструкции SQL, добавляемой к команде SELECT по умолчанию и относящейся к блоку. Это свойство аналогично одноименной конструкции в SQL, поэтому все ограничения и правила, применимые к этой конструкции, имеет точно такое же отношение и к фразе WHERE Clause в Oracle Forms.

Рассмотрим случай, когда вам необходимо, к примеру, выбрать все заказы за 2007 год. Если вы пишете запрос в SQL*Plus или в любой другой среде, то вы просто допишете условие WHERE к вашему запросу:

```
Select * from pzak WHERE to_char (pdat, 'YY') ='2007'
```

В результате этого запроса вы получите все заявки за 2007 год. В Oracle Forms вы можете получить такой же результат. Для того чтобы добавить условие выборки к вашему блоку, вам нужно установить значение свойства «Фраза WHERE Clause». Значение этого свойства, как и в случае с ORDER BY, можно установить двумя способами:

- в режиме проектирования формы, определив значение этого свойства;
- в режиме исполнения, то есть меняя порядок сортировки по какому-либо событию или условию.

Рассмотрим первый способ, для этого выполним пример.

1. Откройте форму ZAK.fmb.

2. В объектном навигаторе выберите блок PZAK. Откройте палитру свойств и перейдите к свойству «Фраза WHERE Clause» в узле «База данных».
3. В значении свойства напишите условие отбора: `to_char(pdat, 'YY')='2007'`.
4. Запустите форму и выполните запрос.

В результате выполнения запроса мы получаем требуемый набор данных, то есть все заявки за 2007 год. В свойстве «Фраза WHERE Clause» вы можете задавать различные условия выборки данных с помощью предикатов, булевых операторов и коррелируемых подзапросов. В качестве операндов можно указывать параметры и глобальные переменные. Например, не будет ошибкой, если вы установите свойство «Фраза WHERE Clause»:

1. pdat=:global.p_dat;
2. pdat=:parameter.p_dat;
3. pdat BETWEEN '2006' AND to_char(sysdate, 'YY');
4. to_char(pdat, 'YY')='2007' OR to_char(pdat, 'YY')='2006'.

Примечание: когда вы пишете условие, в свойстве «Фраза WHERE Clause» ключевое слово WHERE опускается. Также не забудьте, что в конце вашего условия символ «;» опускается.

В Oracle Forms пользователи могут в режиме ввода запроса (Enter Query mode) указать в любом из полей значение, которое будет являться критерием отбора записей при выполнении запроса. После того как пользователь получил желаемую выборку данных, он продолжает выполнять какие-либо действия, например, операцию модификации данных. Пользователь зафиксировал данные и теперь ему необходимо запросить их еще раз, – чтобы вновь получить этот набор, придется проделать ту же операцию, что и в начале. Чтобы этого не делать, пользователь может в интерактивном режиме выполнить последовательно команды:

```
ENTER_QUERY (<F7>) ENTER_QUERY (<F7>) EXECUTE_QUERY (<F8>)
```

Если попытаться выполнить эти действия программно, к примеру, по нажатии кнопки, то желаемый результат не будет достигнут. Создайте кнопку и событие кнопки WHEN-BUTTON-PRESSED, в теле которого напишите последовательность команд:

```
ENTER_QUERY;
ENTER_QUERY;
EXECUTE_QUERY;
```

После нажатия кнопки форма всего лишь перейдет в режим ввода запроса. Ниже приведен пример, который позволяет выполнить последний запрос в блоке программно.

1. Откройте форму Zak.fmb. Проверьте, чтобы свойство «Фраза WHERE Clause» блока PZAK было пустым.
2 Создайте триггер KEY-ENTQRY на уровне блока. В теле триггера напишите следующий код:

```
SET_BLOCK_PROPERTY ('items', DEFAULT_WHERE, '');
Enter_Query;
```

3 Создайте триггер PRE-QUERY на уровне блока. В теле триггера напишите следующий код:

```
Declare
  where_cl Varchar2 (1000);
Begin
  where_cl:= UPPER (get_block_property ('pzak', last_query));
  where_cl:= SUBSTR (where_cl, INSTR(where_cl, 'WHERE') + 6)
  SET_BLOCK_PROPERTY ('pzak', DEFAULT_WHERE, where_cl);
End;
```

4. Запустите форму на выполнение. Введите форму в режим запроса и введите в поле NFIRM любое существующее значение, после чего выполните запрос. Измените какую-либо запись и запросите данные еще раз.

Каждый может использовать этот пример для различных целей, то есть не только для выполнения последнего запроса, но и, к примеру, для хранения истории выполненных пользователем запросов.

Лекция 9. Элементы Forms Builder. Обзор элементов и их общих свойств. Элементы ввода

В этой лекции слушатель научится создавать текстовые элементы, радиокнопки, редакторы (Editors), кнопки-флажки (Check Boxes), группы кнопок-переключателей (Radio Groups).

Ключевые слова: создание редакторов, создание Check Box, создание Radio Groups.

Цель лекции: ознакомить слушателя с палитрой элементов Forms Builder и научить создавать кнопки-флажки и радиогруппы.

Элементы блоков (Items)

В этой главе речь пойдет об элементах интерфейса Oracle Forms. В процессе изучения материала вы ознакомитесь с атрибутами элементов, научитесь их устанавливать и модифицировать как на этапе проектирования, так и во время выполнения приложения, используя различные методы. Мы также рассмотрим на конкретных примерах, как, используя определенные методы элемента, можно управлять ими в зависимости от требования пользователей.

Элементы – это объекты интерфейса Oracle Forms, предназначенные для взаимодействия пользователя с программой и управления входной и выходной информацией и ее отображением. В табл. 9.1 приведены и описаны элементы Oracle Forms.

Таблица 9.1. Элементы Oracle Forms 6i

Элемент	Обозначение
Button *(кнопка)*	*Прямоугольник с текстовой меткой или графической иконкой внутри.*
Chart item *(элемент диаграммы)*	*Прямоугольник с окантовкой любого размера, выводящий диаграмму или иное изображение, генерируемое Oracle Graphics. Операторы не могут перемещаться к элементам диаграмм или манипулировать ими.*
Check box *(переключатель)*	*Текстовая метка с графическим индикатором состояния, которая показывает текущее значение или как отмеченное, или как не отмеченное. Выбор переключателя переключает его в противоположное состояние.* Дает пользователю возможность задавать логические значения.

Display item *(неизменяемый текст)*	*«Только читаемая» текстовая рамка, значение которой должно быть извлечено или присвоено программно. Операторы не могут перемещаться к элементу неизменяемого текста или редактировать содержащийся в нем текст.*
Image item *(элемент изображение)*	*Показывает растровое или векторное изображение из файловой системы или Базы Данных.*
List item *(элемент список)*	*Список вариантов, выводимый или как всплывающий список (иногда называемый ниспадающим списком), или как t-список (иногда называемый списочной рамкой), или как combo-рамка.*
Radio group *(радиогруппа)*	*Группа радиокнопок, одна из которых всегда выбрана.*
Text item *(элемент текст)*	*Одно- или многострочная текстовая рамка, поддерживающая множество типов данных, масок формата, а также возможности редактирования.*
OLE container *(OLE-контейнер)*	*Область, которая хранит и выводит объект OLE, создающийся прикладной программой OLE-сервера.*
ActiveX Control *(элемент управления ActiveX)*	*Пользовательское средство управления, которое упрощает построение и усовершенствование пользовательских интерфейсов.*
Bean Area	
Hierarchical Tree *(Иерархическое дерево)*	*Элемент — иерархическое дерево. Элементы дерева можно создавать динамически. Они могут быть любой вложенности родитель-потомок.*
Tab Canvas *(Страница вкладки)*	*Основные функции окна отображаются в группе перекрывающихся объектов-основ с помеченными вкладками.*
Stacked Canvas *(Стековая вид-картинка)*	Канва отображается поверх других объектов-основ, показывая содержимое в зависимости от заданного условия или разделителя.

Все вышеперечисленные элементы представляют собой объекты для построения формы. Каждый элемент имеет набор свойств, как идентичных другим элементам, так и специфичных, применимых только к данному элементу. Элементы содержат все свойства управления отображением

и визуализации. В Forms элементы, как и блоки, могут быть базовыми и небазовыми (управляющими).

- Базовый элемент – это элемент, который связан со столбцом Базы Данных. Базовые элементы могут размещаться как в управляющих, так и в базовых блоках.
- Небазовый элемент (управляющий) – это элемент, не связанный со столбцом Базы Данных. Управляющий элемент может быть использован как переменная или управляющий элемент. Ниже приведена область применения управляющих элементов.
  - Вычисляемые элементы – отображение формул и групповых обработок.
  - Хранение временной информации – использование элемента вместо переменной в PL/SQL-программах.
  - Заполнение данными из других таблиц.

Редактируемый текст (Text_Item)

Редактируемый текст (Text_Item) – это элемент интерфейса, предназначенный для ввода, корректировки и отображения информации в однострочном или многострочном режиме. Текстовый элемент может быть базовым (ассоциирован со столбцом Базы Данных) и управляющим.

Создание текстового элемента

Для того чтобы создать элемент «Редактируемый текст», находясь в Редакторе Разметки, найдите на панели инструментов элемент «Редактируемый текст» и начертите его на канве.

Текстовый элемент поддерживает различные типы данных, приведенные в табл. 9.2.

Для того чтобы выбрать тип данных для текстового элемента, выполните следующие действия:

1. Находясь в Палитре Свойств выбранного элемента, найдите свойство «Тип Данных».
2. Находясь в значении свойства «Тип данных», вызовите выпадающий список с перечнем доступных типов данных. Выберите тот тип данных, который соответствует вашим требованиям.

Примечание: *текстовый элемент при создании имеет по умолчанию тип данных CHAR с размером (Maximum Length) 30 символов.*

Рассмотрим основные свойства, которые могут понадобиться при первом знакомстве с элементом «Редактируемый текст».

Таблица 9.2. Типы данных

Тип данных	Значение
CHAR	Символ
NUMBER	Число
DATE	Дата
LONG	Длинное целое
ALPHA	Символьный
DATETIME	Дата, включая время
EDATE	Дата
JDATE	Integer – Целые
LONG	Julian Date – Юлианская Дата
INT	Целый
MONEY	Денежный
RINT	Right Integer – Целые по Правому
RMONEY	Right Money – Денежный по Правому
RNUMBER	Right Number – Числовой по Правому
TIME	Время

Функциональные

- Выравнивание – значение этого свойства определяет положение (выравнивание) текста в элементе. Доступно пять значений:
  - влево;
  - вправо;
  - по центру;
  - в начало;
  - в конец.
- Многострочный – значение этого свойства определяет, как будет отображаться текст в редактируемой области – как однострочный или многострочный:
  - Однострочный элемент – это элемент текста, в котором данные вводятся в одну строчку и отображаются соответственно так же. Когда свойство «Многострочный текст» установлено в «Нет», это значит, что элемент текста будет выводить одну строку текста.

  - Многострочный элемент позволяет вывести на экран несколько строк текста. В таком элементе при возврате каретки курсор перемещается в начало новой строки. Вывод строк на экран зависит от настройки ширины, длины и шрифтов.
- Стиль завершения строки определяет поведение текста в случае превышения строкой текста ширины элемента. Например, если выбран стиль «Слово», то при превышении строкой ширины столбца ее значение будет переноситься на следующую строку на обрыве слов. Существует три стиля переноса:
  - слово;
  - символ;
  - ничего.
- Ограничение на регистр определяет регистр символов отображаемых элементов:
  - Смешанный – отображает символы верхнего и нижнего регистра, то есть текст будет отображен так, как его ввели.
  - Прописной – при выводе текста переводит его в нижний регистр.
  - Строчный – при выводе текста переводит его в верхний регистр.
- Спрятать данные – это свойство служит для скрытия символов, вводимых в элемент.
- Сохранить позицию курсора – сохраняет текущую позицию для повторного входа в элемент.
- Автоматический пропуск – если значение этого элемента истинно, это значит, что при заполнении последнего символа курсор переместится на следующий элемент. Увеличивает скорость работы с приложением, так как нет необходимости в дополнительном нажатии функциональных клавиш.

Данные

- Фиксированная длина – максимальное количество символов, которое может быть введено в элемент. Если свойство «Автоматический пропуск» истинно, то после того как случится попытка набрать символ, превышающий максимальную длину, курсор переместится в следующий элемент.
- Начальное значение – значение по умолчанию, которое будет присваиваться каждый раз при создании новой записи.
- Обязательный – требует обязательного ввода данных; если свойство истинно, то курсор не покинет элемент, пока не будет введено значение.
- Наименьшее Допустимое Значение – минимальное значение, которое может быть введено. Если, предположим, значение равно пяти, то пять также является разрешенным значением.

- Наибольшее Допустимое Значение – максимальное значение, которое может быть введено. Если, предположим, значение равно десяти, то десять также является разрешенным значением.

Список значений (LOV)

- Список Значений – имя списка значений (LOV), привязанного к элементу. Если в этом свойстве задать список значений, то этот список будет выводиться командой LIST_VALUES.
- Проверять по Списку – если значение свойства истинно, то вводимое значение будет сравниваться со значениями в ассоциированном списке.

Использование элемента

Вы можете использовать элемент «Редактируемый текст» как базовый или управляющий.

Ниже перечислены основные свойства для работы с элементом «Редактируемый текст» как с элементом Базы Данных.

- Имя столбца – имя столбца, с которым ассоциирован элемент.
- Первичный ключ – определяет, соответствует ли элемент столбцу первичного ключа, определенного в Базе Данных.
- Только запрос – если значение свойства истинно, то элемент нельзя задействовать в командах DML.
- Запрос Разрешен – значение этого элемента разрешает или запрещает выполнения запроса.
- Длина Запроса – определяет максимальное количество символов, разрешенное для ввода критерия запроса в элементе.
- Запрос без Учета Регистра – значение этого элемента разрешает или запрещает выполнение запроса без учета регистра.
- Вставка Разрешена – значение этого элемента разрешает или запрещает изменять данные в элементе, то есть выполнять команду UPDATE.
- Обновление Разрешено – значение этого элемента разрешает или запрещает вставку в элементе, то есть выполнять команду INSERT.
- Обновить только если отсутствует – если значение элемента истинно, то изменять данные в элементе можно только в том случае, если оно не определено, то есть NULL.
- Блокировать запись – если значение этого свойства истинно, Forms заблокирует запись БД, соответствующую текущей записи в блоке.

Управляющий (небазовый) элемент – это элемент, который не ассоциирован со столбцом Базы Данных. Вы можете использовать небазовые

элементы как разновидность параметров или переменных, действующих в области всей формы. Чаще всего такие элементы применяют как вычисляемые поля.

Форматирование элемента

Вы можете управлять отображением содержимого элемента, не изменяя при этом самого значения, с помощью форматных масок. Маски данных также служат для упрощения ввода оператором данных, так как автоматически проставляют различные специальные символы, такие как тире и денежные символы. Форматные маски разделяются по типам данных:

- форматы масок для значений NUMBER;
- форматы масок для значений DATE;
- форматы масок для значений DATETIME;
- форматы масок для значений TIME.

Ниже будут приведены три таблицы с примерами форматных масок для конкретного типа данных.

Форматы масок для значений NUMBER

К примеру, нам необходимо вести справочник телефонных номеров, в котором номера телефонов имеют тип NUMBER, скажем, номер 80629528361. Перед нами стоит задача создать форму просмотра телефонного справочника, причем номер для лучшего восприятия должен отображаться как 8(0629) 52-83-61, то есть символы в нем должны отделяться тире и скобками. Рассмотрим пример.

1. Создайте в форме элемент «Редактируемый текст».
2. Откройте Палитру Свойств и установите следующие свойства:
   - Тип данных: NUMBER;
   - Маска: 9”(“9999”) “99”-”99”-”99.
3. Запустите форму и введите в элемент набор символов 80629528361.

После подтверждения ввода или перехода в другой элемент вы увидите, что элемент отображает введенное вами значение в соответствии с заданной маской. Маска довольно проста для понимания, достаточно знать таблицу специальных символов (табл. 9.3). Например, в нашем случае для вставки символов «тире» и «скобка» достаточно заключить эти символы в двойные кавычки. Этот пример очень хорошо показывает, как маска может упростить работу оператора, позволяя не тратить время на набор специальных символов, и разработчика, которому не придется менять тип данных столбца.

Таблица 9.3. Специальные символы маски формата для типа NUMBER

Символ	Описание
9	Представляет один цифровой символ. Количество девяток определяет то, сколько цифр сможет выводить элемент текста.
0	Выводит ведущие нули, если они имеются.
$	Предваряет число символом доллара.
B	Выводит предшествующие нули как пробелы.
MI	Выводит «–» после отрицательного значения.
PR	Выводит отрицательное значение в <угловых скобках>.
, (запятая)	Выводит в этой позиции запятую, если требуется.
. (точка, десятичный разделитель)	Выводит в этой позиции десятичную точку.

Таблица 9.4. Примеры использования специальных символов

Маска формата	Значение	Результат
99	125	12
9”(“9999”) “99”-”99”-”99	80629528361	8(0629) 52-83-61
$9,999.99	1256.78	$1,256.78
99.9	0041.8	41.8
0.9	0.45	0.5
B99.9	0023.13	23.1
99.9MI	–12	12.0–
99.99”т. ”	12.12	12.12т.

Форматы масок для значений DATE, DATETIME и DATE

В этом разделе, как и в предыдущем, будут приведены основные символы, использующиеся при определении маски формата.

Таблица 9.5. Специальные символы маски формата типа DATE

Маска формата	Значение
9	Представляет один цифровой символ. Число девяток определяет количество цифр, которое сможет выводить элемент текста.
MM	Месяц (01–12).
MON	Имя месяца в 3-буквенном сокращении.
MONTH	Имя месяца с дополненными пробелами до длины 9 символов.
DD	Число месяца (1–31).
DY	Имя дня в 3-буквенном сокращении.
DAY	Имя дня в 3-буквенном сокращении.
Y, YY, YYY, YYYY	4-, 3-, 2- или 1-цифровой год.
HH (HH12)	Час дня (1–12).
HH24	HH24 Час дня (0–23).
MI	Минуты (0–59).
SS	Секунды (0–59).
AM (A.M) или PM(P.M)	Индикатор полудня.
TH	Порядковое число (например, «DDHH» для «15TH»).
SP	Названное число (например, «DDSP» для «FIFTEEN»).
SPTH (THSP)	Названное порядковое число (например, «DDSPTH» для «FIFTEENTH»).
FM	Префикс, используемый с символами, такими как MONTH и DAY, для подавления заполнения, добавляемого этими символами. Подавляет также ведущие нули для числовых символов, таких как HH и MM. FM – это переключатель. Если FM в модели формата даты присутствует дважды, то после первого FM заполнение пробелами и ведущие нули для элементов подавляются, а после второго – включаются снова.

Таблица 9.6. Примеры использования специальных символов

Маска формата	Значение	Результат
MM:DD:YYYY	DATE	01/01/2007
Day, «the» DDTH «of» fmMonth	DATE	Monday, the 10TH of April
J	DATE	10012007
HH:MI PM	TIME	11:15 PM
HH24:MI	TIME	23:27
MM-DD-YY HH:MI AM	DATETIME	01-01-99 05:15 AM
FmMM-DD-YY fmHH:MI AM	DATETIME	1-01-99 05:15 AM
fmMonth YY	DATETIME	April 99

Мы рассмотрели основные маски, применимые в Oracle Forms. Не забывайте об их существовании и используйте в приложениях, это упростит работу не только вам, но и оператору.

Вы можете управлять элементами текста во время выполнения, используя все тот же набор подпрограмм:

- GET_RADIO_BUTTON_PROPERTY;
- SET_RADIO_BUTTON_PROPERTY;
- CONVERT_OTHER_VALUE;
- DISPLAY_ITEM.

Рисунок (Image)

Рисунок (Image) – это элемент, который отображает растровое или векторное изображение из файловой системы или Базы Данных. Рисунок имеет вид прямоугольной рамки. Элемент «Рисунок» недоступен для отображения только в терминальном режиме.

Вообще в Forms существует два вида изображения – это изображение как элемент шаблонной графики и как элемент интерфейса, который может быть ассоциирован со столбцом Базы Данных.

Создание Рисунка

Как уже было сказано выше, элемент «Рисунок» может быть базовым или управляющим. Для создания изображения выполните следующие действия:

1. Находясь в Редакторе Разметки, найдите на панели инструментов элемент «Рисунок» и начертите его на канве. Присвойте изображению имя IMG.

2. Вы можете заполнить изображение двумя способами:
   - извлечь из ассоциированного с элементом столбца Базы Данных;
   - с помощью встроенной процедуры READ_IMAGE_FILE.
3. Рассмотрим пример, в котором заполним изображение процедурой READ_IMAGE_FILE. Для этого разместите на канве слева от изображения новую кнопку и создайте для нее триггер WHEN-BUTTON-PRESSED. В свойстве «Метка» кнопки укажите значение «Загрузить картинку». В теле триггера напишите следующий код:

```
WHEN-BUTTON-PRESSED
declare
   filename varchar2(100);
begin
   filename:=get_file_name(filename);
   read_image_file(filename, 'BMP', 'img');
   end;
```

4. Запустите форму и нажмите кнопку, после чего на экране появится диалоговое окно для выбора файла изображения.
   Вы можете записать изображение в файл операционной системы WRITE_IMAGE_FILE, для этого:
5. Создайте дополнительно вторую кнопку на канве и определите для нее триггер WHEN-BUTTON-PRESSED. В свойстве «Метка» кнопки укажите значение «Загрузить картинку». В теле триггера напишите следующий код:

```
WHEN-BUTTON-PRESSED
write_image_file('C:\izo_copy.bmp', 'BMP', 'img');
```

6. Запустите форму еще раз и нажмите сначала первую кнопку «Загрузить картинку», а затем вторую кнопку «Записать изображение». После того как вы нажмете вторую кнопку, изображение из элемента «Рисунок» запишется в файл izo_copy.bmp.
   Вы также можете управлять изображением, используя свойства:
   - Формат изображения – в этом свойстве вы выбираете формат хранения изображения из выпадающего списка. Для работы с изображением вам доступны следующие форматы:
     – BMP;
     – CALS;
     – GIFF;
     – JFIF;

  - PICT;
  - RAS;
  - TIFF;
  - TPIC.
- Глубина изображения – это свойство задает глубину изображения при чтении или записи. Существует четыре вида установки глубины изображения:
  - первоначальный;
  - монохромный;
  - серый;
  - LUT;
  - RGB.
- Качество Сжатия – определяет степень сжатия файла изображения при чтении или записи вне зависимости от того, был ли он сжат ранее. Существует шесть видов сжатия:
  - ничего (нулевой);
  - минимум;
  - низкий;
  - средний;
  - высокий;
  - максимум.
- Качество Отображения – это свойство определяет качество отображения рисунка. Доступно три вида качества:
  - высокое;
  - среднее;
  - низкое.
- Способ масштабирования – определяет, как будет выводиться изображение в случае, если оно превышает размеры элемента. Существует два способа:
  - усеченный – изображение уместится лишь частично;
  - полный – изображение будет масштабироваться так, чтобы уместиться в элементе.

Программное управление изображением

Вы можете загружать изображение в элемент во время выполнения программы, например, двойным щелчком мыши по изображению или при попадании фокуса на картинку. Для управления изображением программно используйте триггеры:

- WHEN-IMAGE-ACTIVATED;
- WHEN-IMAGE-PRESSED.

Переключатель

Переключатель – это элемент интерфейса, позволяющий пользователю задавать логические значения в виде отметок (флажков). В GUI и в браузере этот элемент имеет вид прямоугольника с надписью справа и индикатором отметки внутри. Если переключатель включен, то в прямоугольнике отображается индикатор отметки, иначе – пустая белая область. В текстовом режиме (Forms6i и ниже) переключатель отображается в виде квадратных скобок с надписью справа и индикатором отметки внутри в виде крестика. Вы можете использовать переключатели для:

- установки состояния объекта. Вы можете создать переключатель, который будет управлять отображением элемента;
- хранения значений. Переключатели хранят значения, которые определены состоянием элемента – «Включен» или «Выключен». Переключатели могут хранить три основных типа данных: DATE, NAMBER или CHAR;
- установки значения столбца Базы Данных.

Создание переключателя

Для того чтобы создать переключатель, находясь в Редакторе Разметки, выберите на панели инструментов элемент «Переключатель» (Check box) и разместите его на канве. Рассмотрим основные свойства переключателя.

- Метка (Label) – надпись, выводимая справа от прямоугольника переключателя.
- Значение при выборе (Value when Checked) – это значение, которое будет определять состояние переключателя как «Включен». Если хранимое значение переключателя равно «значению при выборе», то переключатель включается.
- Значение при отмене выбора (Value when Unchecked) – это значение, которое будет определять состояние переключателя как «Выключен». Если хранимое значение переключателя равно «значению при отмене выбора», то переключатель не включен.
- Присвоение Выключателю значения Другие (Check Box Mapping of Other Values) – определяет поведение переключателя, если в нем хранится или ему присваивается значение, отличное от заданного при выборе или отмене выбора.

Если переключатель определен как базовый, то есть свойство «Элемент Базы Данных» истинно, при выборе состояния переключателя в ассоциированный с ним столбец будет копироваться значение этого состояния.

Программное управление переключателями

Для того чтобы инициировать какое-либо действие при выборе состояния переключателя, нужно ассоциировать с ним триггер WHEN-CHECKBOX-CHANGED и выполнить в нем какое-либо действие.

1. Создайте элемент переключатель и присвойте ему имя C_BX.
2. Создайте элемент «Текст» (Text Item), назовите его T_item и разместите его слева от переключателя.
3. Создайте триггер для переключателя WHEN-CHECKBOX-CHANGED и напишите в нем приведенный ниже код:

```
WHEN-CHECKBOX-CHANGED
IF :c_bx=checked THEN
SET_ITEM_PROPERTY('t_item', font_size, 20);
ELSE
   …
END IF;
```

4. Запустите форму, введите любое символьное значение в элемент «Текст» и включите переключатель. При включении переключателя размер шрифта в текстовом элементе увеличится в соответствии с заданным значением.

Вы также можете управлять свойствами переключателя в режиме выполнения программы с помощью процедуры SET-ITEM-PROPERTY и получать значения свойств с помощью функции GET-ITEM-PROPERTY.

Радиогруппа

Радиогруппа – это совокупность переключателей (радиокнопок), объединенных в одну структуру (радиогруппу), в которой каждый элемент взаимно исключает остальные. Радиогруппу можно представить как набор опций, в котором при выборе одной опции все остальные становятся недоступными. Радиокнопка изображается в виде окружности с жирной точкой внутри, если она выбрана, или в виде пустой окружности, если не выбрана. Радиогруппа может содержать любое число радиокнопок. Каждая радиокнопка, несмотря на то что она является частью группы, может иметь свои личные атрибуты, такие как значение, размер и подпись.

Радиогруппа может быть определена как базовая или как управляющая. Если радиогруппа базовая, то она представляет конкретный столбец Базы Данных. Любая выбранная опция определяет значение всей радиогруппы, поэтому, переключая радиокнопки, вы тем самым изменяете

значение записи в ассоциированном с ней столбце. Рассмотрим свойства, специфичные для этого элемента.

- Метка (Label) – надпись, выводимая справа от окружности радиокнопки.
- Значение радиокнопки (Radio Button Value) – определяет значение радиокнопки при выборе. Значение выбранной радиокнопки представляет конкретное значение всей радиогруппы, поэтому каждый раз при выборе радиокнопки вы устанавливаете новое значение для всей радиогруппы.

Создание радиогруппы

Само слово «группа» дает представление о том, что необходимо как минимум две радиокнопки для создания радиогруппы. Создадим радиогруппу.

1. В запущенном Редакторе Разметки на панели инструментов выберите элемент «Радиокнопка» и начертите его на канве.
2. Одна радиокнопка как элемент ничего собой не представляет, так как является единственной опцией и выбирать, собственно, нечего. При создании радиокнопки автоматически создается радиогруппа с номером ниже, чем у только что созданного элемента. Присвойте радиокнопке значение 1, а метку определите как «Разрешить операции DML».
3. Находясь в Навигаторе Объектов, запустите Палитру Свойств радиогруппы. Назовите радиогруппу «RGroup» и присвойте свойству «Начальное значение» числовое значение 2.
4. Создайте вторую радиокнопку. Как только вы попытаетесь начертить вторую радиокнопку, на экране появится окно «Radio Groups» (рис. 9.1).

   Это окно представляет собой список доступных радиогрупп. Каждый раз, создавая новую радиокнопку, Forms будет автоматически запускать это окно, чтобы вы смогли выбрать радиогруппу, в которой планируете разместить новый элемент.
5. Выберите радиогруппу для размещения нового элемента и подтвердите выбор. Присвойте новой радиокнопке значение 2, а метку определите как «Разрешить операции DML». Начальное значение определяет радиокнопку, которая будет выбрана при запуске формы. Радиогруппа должна быть обязательно инициирована заданием свойств «Значением по умолчанию» или «Присваивание Значений Другие»; исключением является случай, когда в радиогруппе существует кнопка со значением NULL.

Рис. 9.1. Окно «Радиогруппа»

6. Несмотря на то что радиогруппа включает в себя набор радиокнопок, триггеры создаются для всей группы, а не для каждого элемента в отдельности. Создайте триггер WHEN-RADIO-CHANGED радиогруппы и напишите в нем следующий код:

```
WHEN-RADIO-CHANGED
IF :RGroup =1 THEN
SET_BLOCK_PROPERTY('block_name', update_allowed, property_false);
SET_BLOCK_PROPERTY('block_name', delete_allowed, property_false);
SET_BLOCK_PROPERTY('block_name', insert_allowed, property_false);
ELSIF :RGroup =2 THEN
SET_BLOCK_PROPERTY('block_name', update_allowed, property_true);
SET_BLOCK_PROPERTY('block_name', delete_allowed, property_true);
SET_BLOCK_PROPERTY('block_name', insert_allowed, property_true);
END IF;
```

В этом примере показано, как радиокнопка определяет значение радиогруппы. Поэтому для программного управления группой вам достаточно использовать имя самой группы. Как видно из примера, обращение к радиокнопке осуществляется через ее значение.

Вы также можете программно управлять свойствами радиогруппы и получать их значения с помощью встроенных процедур и функций:

- GET_RADIO_BUTTON_PROPERTY;
- SET_RADIO_BUTTON_PROPERTY;
- CONVERT_OTHER_VALUE;
- DISPLAY_ITEM.

Рассмотрим процедуру CONVERT_OTHER_VALUE.

1. Выберите только что созданную нами радиогруппу и запустите Палитру Свойств.
2. Найдите свойство «Присваивание Значений Другие» и установите его значение равным 2.
3. Создайте кнопку и триггер WHEN-BUTTON-PRESSED, в котором напишите следующий код:

```
WHEN-BUTTON-PRESSED
convert_other_value('RGroup');
```

В этом примере показано, как по нажатию кнопки выбирается радиокнопка со значением, указанным в свойстве «Присваивание Значений Другие».

Элемент отображения (Неизменный текст)

Неизменный текст – это элемент интерфейса, предназначенный для отображения информации. Особенность этого элемента в том, что он предоставляет данные только для чтения и при наведении курсора на этот элемент вы не получаете фокуса входа в этот элемент. Несмотря на то что элемент отображения предоставляет информацию только для чтения, вы можете манипулировать содержимым этого элемента. Элемент «Неизменный текст» может быть базовым, то есть отображать данные ассоциированного с ним столбца Базы Данных, и управляющим.

Создание элемента отображения

Чтобы создать элемент отображения, выполните следующие действия.

Находясь в Редакторе Разметки, найдите на панели инструментов элемент «Элемент отображения» и начертите его на канве.

Обращаясь к элементу отображения в своих PL/SQL-программах, вы можете манипулировать его отображаемым содержимым. Выполним небольшой пример.

1. Запустите Редактор Разметки и разместите на канве элемент отображения и кнопку. Присвойте элементу отображения имя D_item.
2. В триггере кнопки WHEN-BUTTON-PRESSED напишите следующий код:

```
WHEN-BUTTON-PRESSED
:d_item:=10;
```

Если вы выполните этот пример, то увидите, как при нажатии на кнопку изменится отображаемое содержимое элемента.

Текст

Текст – это статическая текстовая метка, значение которой определяется только на этапе проектирования. Текст относится к графическим элементам, перечень которых вы можете увидеть, раскрыв узел Картинка (Canvas). К этому элементу нельзя обращаться в программах, так как вы сразу получите ошибку «Неверная переменная привязки». Для того чтобы создать элемент «Текст», находясь в Редакторе Разметки, найдите на панели инструментов одноименный элемент и начертите его на канве.

Лекция 10. Элементы Forms Builder. Элементы списка и невводные элементы

В этой лекции слушатель научится создавать элементы отображения, картинки, кнопки, вычисляемые элементы. Также в этой лекции слушатель научится создавать элемент-список, наполнять элемент статически и динамически, использовать его как комбинированный список, делать снимок элемента.

Ключевые слова: создание неизменного текста, создание кнопок, картинки, создание элемента списка, зеркальные списки, удаление списков, снимок списка, группа записей, метка, выделение элемента в списке элементов.

Цель лекции: ознакомить слушателя с элементом списка и невводными элементами.

Создание элементов списков (List Items)

Элемент списка (List Item) – это элемент Oracle Forms, предназначенный для отображения данных, причем отображение и возможность форматирования данных списка напрямую зависит от выбранного стиля списка. Различают три типа элементов списка:

- **Всплывающий список** – элемент списка стиля «всплывающий список», первоначально выглядит как одно поле (подобно полю элемента текста). Когда оператор выбирает иконку списка, появляется список доступных вариантов.
- **Текстовый список** – элемент списка стиля «текстовый список», выглядит как прямоугольная рамка, которая выводит фиксированное количество значений. Если текстовый список содержит значения, которые нельзя вывести (из-за области возможного вывода этого элемента), то появляется вертикальная линия прокрутки, позволяющая оператору просмотреть и выбрать невыведенные значения.
- **Комбинированный список** – элемент списка стиля «комбинированная рамка», объединяет характеристики элементов текста и списка. В отличие от элементов списка стиля всплывающего или текстового списка, элемент списка стиля комбинированная рамка будет как выводить фиксированные значения, так и принимать одно вводимое оператором значение.

Основные функциональные свойства

- *Включен* – этот параметр отвечает за состояние элемента списка, по умолчанию имеет значение «Да», то есть включен.
- *Элементов в списке* – при выборе этого свойства запускается окно List Elements, где в поле List Elements вводятся данные, которые будут отображены в списке, а в поле List Item Value – значения этих данных.
- *Присваивание значений другие* – определяет, как понимать выбранные или присвоенные значения в отличие от заданных. Это свойство не пригодится вам только в случае использования комбинированных списков, поскольку в этом случае пользователь может вводить и корректировать данные. Вот два варианта задания этого свойства:
  - Укажите, что элемент списка будет отвергать другие значения как неразрешенные.
  - Укажите, какой составляющий элемент должен выбираться, если значение элемента списка будет другим значением.
- *Класс выполнения* – в этом параметре указывается класс для выполнения Java Bean.
- *Ограничение на регистр* – определяет регистр для текста, введенного в элемент данных или в параметр подстановки для меню.
- *Значения по умолчанию* – вводимое вами здесь значение по умолчанию указывает для элемента списка первоначальное значение и таким образом определяет, какой элемент в списке будет первоначально выбран. Значение по умолчанию не указывается, когда:
  - элемент списка принимает другие значения;
  - значение, связанное с одним из составляющих элементов списка, определено как NULL.

Есть и другой способ сделать элемент текущим: извлечь его в элемент списка или присвоить ему значение NULL. Вид элемента списка NULL во время выполнения зависит от стиля вывода:

1. для всплывающего списка – в элементе текста не выводится никакого текста;
2. для текстового списка – в списке не подсвечивается ни один элемент;
3. для комбинированной рамки – в элементе текста не выводится никакого текста.

В режиме ввода запроса операторы, если они не хотят включать значение элемента списка в критерий запроса, могут устанавливать это значение на NULL:

1. для всплывающего списка – операторы исключают элемент, выбирая составляющий элемент NULL;
2. для текстового списка – операторы исключают элемент, отменяя выбор текущего выбранного составляющего элемента;

3. для комбинированной рамки – операторы исключают элемент, выбирая составляющий элемент NULL.

Примечание: *учтите, что значение элемента списка будет равным NULL до тех пор, пока блок не будет содержать записей, «да» – если вы установите свойство Default Value (значение по умолчанию), а поскольку Forms инициирует создание записи при перемещении к блоку, то вам достаточно программно произвести перемещение с помощью GO_BLOCK либо GO_ITEM.*

Создание элемента списка

При создании элемента списка в окне свойства List Elements, определяя каждый новый элемент списка, вы должны связать его со значением List Value (значение элемента списка). При выборе пользователем составляющего элемента значение элемента списка изменяется на значение, связанное с этим составляющим, то есть на значение, указанное в List Value. То же происходит и при извлечении значения из базы данных или присваивании такового программно; при этом составляющий элемент, значение которого соответствует извлеченному или присвоенному значению, становится выбранным составляющим в этом списке.

Элемент списка может также хранить извлеченное или присвоенное значение, которое не является одним из значений, связанным с определенным составляющим элементом в списке.

Перед тем как перейти к созданию элемента списка, будет логично сразу перечислить процедуры и функции для работы с элементами списков во время выполнения: вставку, удаление, изменение, снимки, оценивание значений:

- ADD_LIST_ELEMENT;
- CONVERT_OTHER_VALUE;
- CLEAR_LIST;
- DELETE_LIST_ELEMENT;
- GET_LIST_ELEMENT_COUNT;
- GET_LIST_ELEMENT_LABEL;
- GET_LIST_ELEMENT_VALUE;
- POPULATE_LIST;
- RETRIEVE_LIST.

Для ответа на события интерфейса, являющиеся следствием манипулирования операторами элемента списка, в Oracle Forms существуют следующие триггеры:

- When-List-Activated – этот триггер позволяет вам инициировать какое-либо действие всякий раз, когда элемент списка становится текущим.
- When-List-Changed – этот триггер позволяет вам инициировать какое-либо действие всякий раз, как вы выбираете какой-либо элемент из списка.

Теперь перейдем к рассмотрению примера пошагового создания различных типов элемента списка. Чтобы создать элемент списка:

1. Откройте созданный нами ранее модуль ABOUT.fmb.
2. В редакторе разметки выберите на панели инструментов кнопку с изображением элемента List item, после чего ваш курсор примет вид крестика – теперь он предназначен для выделения региона, на котором будет создан, то есть для вычерчивания элемента на области разметки List Item:
   - List Item1: присвойте свойству «Метка» этого элемента – «Popup_list», «Стиль списка» – Всплывающий список (popup list).
   - List Item2: присвойте свойству «Метка» этого элемента – «Tlist Item», «Стиль списка» – Текстовый список (TList).
   - List Item3: присвойте свойству «Метка» этого элемента – «Combo_list», «Стиль списка» – Комбинированный список (combo box).

Теперь этап создания разделим на три части, рассматривая каждый стиль списка отдельно, а также с различным формированием данных и значений.

1. **Для элемента «Popup_list»**
   - Перейдите в палитру свойств этого элемента и выберите свойство «Элементов в списке».
   - В появившемся окне List Elements введите в первую строку слово «Куртка» и присвойте этому полю значение в List Value – «Красная», теперь перейдите ко второму полю и создайте второй элемент списка – «Штаны» со значением «Черные».

2. **Для элемента «Tlist Item»**
   - Создайте кнопку Button, измените значение свойства «Метка» на «Tlist Item».
   - Перейдите в Навигатор и создайте триггер для этой кнопки – WHEN_BUTTON_PRESSED:

```
/*WHEN_BUTTON_PRESSED*/
declare
  cursor C1 is
  select * from ABOUT;
```

```
  v_param C1%rowtype;
  n number;
begin
 select count (1) into n from about;
 open C1;
for i in 1..n loop
   fetch C1 into v_param;
    ADD_LIST_ELEMENT ('LI1', i, v_param.Name, v_param.FName);
end loop;
end;
```

- Скомпилируйте триггер.

3. **Для элемента «Combo_list»**
   - Создайте кнопку Button, измените значение свойства «Метка» на «Combo_list».
   - Перейдите в навигатор и создайте триггер для этой кнопки – WHEN_BUTTON_PRESSED:

```
/*WHEN_BUTTON_PRESSED*/
POPULATE_LIST ('COMBO_LIST', 'ABOUT_GROUP');
```

   - Скомпилируйте триггер.

Теперь проверим, как работают наши примеры.

1. Запустите форму.
2. Нажмите на первый созданный нами элемент списка – «Popup_list». В результате появится всплывающий список, где будут указаны данные, которые мы определили во время проектирования. Остальные два списка у нас пока остаются пустыми.
3. Теперь перейдем ко второму элементу списка – «Tlist Item». Для формирования этого списка была использована процедура ADD_LIST_ELEMENT, которая добавляет одну строку в элемент списка; ниже приведен ее синтаксис:

```
ADD_LIST_ELEMENT (ITEM_NAME IN VARCHAR2, ITEM_INDEX IN NUMBER,
LABEL_STR IN VARCHAR2, VALUE_STR IN VARCHAR2);
```

В нашей программе: ADD_LIST_ELEMENT ('Tlist_Item', i, v_param.Name, v_param.FName); теперь, для большего понимания синтаксиса и подставляемых нами значений на входе выполнения программы, рассмотрим параметры этой процедуры обобщенно.

- **ITEM_NAME** ('TLIST_ITEM') – этот параметр принимает имя вашего элемента списка, либо его ID.
- **ITEM_INDEX** (i) – индекс элемента списка, указывает на порядковый номер элемента в списке. В примере этот параметр принимает значения счетчика цикла, то есть после каждого прохождения цикла индекс элемента в списке равен: i=i+1. Поэтому, если у нас цикл будет работать в диапазоне от 1 до 4, то индексы добавленных элементов будут 1, 2, 3, 4.
- **LABEL_STR** (v_param.Name) – этот параметр принимает данные типа varchar2, которые будут видны после формирования списка, то есть отображает те данные, которые мы непосредственно будем видеть. V_PARAM.NAME – это переменная типа %ROWTYPE, данные из которой мы выбираем для заполнения элемента списка из столбца NAME.
- **VALUE_STR** (v_param.FName) – этот параметр принимает значение типа varchar2, которое мы связываем с предыдущим параметром. V_PARAM.FNAME – это переменная типа %ROWTYPE, данные из которой мы выбираем для заполнения элемента списка значениями из столбца FNAME. Вы также можете присвоить этому параметру значение NULL, то есть к элементу списка с названием, к примеру, Sergey добавится значение FNAME – отчество.

Теперь нажмите кнопку Tlist Item для заполнения элемента списка.

И последний рассматриваемый элемент списка – «COMBO_LIST». Для формирования этого списка была использована процедура POPULATE_LIST, которая формирует элемент списка с помощью группы записей; ниже приведен ее синтаксис:

```
POPULATE_LIST (ITEM_NAME IN VARCHAR2, RG_NAME IN VARCHAR2);
```

Параметры

- **ITEM_NAME** – имя элемента списка, для которого мы исполняем запрос группы записей.
- **RG_NAME** – имя Группы записей, которая формирует элемент списка. Эта группа должна быть создана программно.

Примечание: *группа записей, на основе которой формируется элемент списка, не только должна быть создана программно, но и должна содержать две колонки символьного типа – одна для указания данных, отображаемых в списке, а другая для значений, связанных с этими данными.*

Для заполнения COMBO_LIST

- Нажмите кнопку «Создать запросную группу» для создания группы «about_group».
- После получения сообщения «Group create succesfull» нажмите кнопку «COMBO_LIST». Теперь элемент списка «COMBO_LIST» заполнен данными из группы, в чем вы можете убедиться, щелкнув по нему.

Как видно из примеров, в Oracle Forms встроено достаточно функций для обеспечения гибкости работы с данным элементом, что позволяет управлять заполнением элементов списка не только статически, но и программно. Далее мы рассмотрим еще несколько очень важных и специфических процедур этого элемента.

Создание «снимка»

Под «снимком» здесь имеется в виду извлечение и немедленное сохранение содержимого списка. Эта процедура поможет вам не только извлекать и сохранять содержимое списков, но и манипулировать восстановлением списков со старым содержимым, динамически делать подмену.

RETREIVE_LIST – процедура создания так называемого «снимка».

Синтаксис:

RETRIEVE_LIST (ITEM_NAME IN VARCHAR2, RECGRP_ID IN FORMS4C.RECORDGROUP);

Параметры:

- **ITEM_NAME** – этот параметр принимает имя списка, из которого мы собираемся извлекать содержимое;
- **RECGRP_ID** – этот параметр принимает имя группы записей, в которую мы будем добавлять содержимое.

Примечание: *в этом случае условие извлечения содержимого элемента списка в группу записей такое же, как и при заполнении списка из группы записей, то есть необходимо существование двух символьных столбцов для заполнения данных списка и их значений.*

Давайте рассмотрим пример, в котором содержимое элемента списка, созданного на основе запросной группы, извлекается в другую запросную группу, состоящую, в свою очередь, из двух колонок. Для этого в форме ABOUT создайте новую кнопку меткой «снимок». Для кнопки «снимок» создайте триггер WHEN_BUTTON_PRESSED:

```
/*WHEN_BUTTON_PRESSED*/
  DECLARE
       group_id       RecordGroup;
       query_ok       number;

  BEGIN
      group_id := Create_Group_From_Query('snapshot_group','SELECT
NAME, FNAME FROM ABOUT');
      query_ok:=Populate_Group('snapshot_group');
    Retrieve_List ('COMBO_LIST', 'snapshot_group');
    Populate_List('TLIST_ITEM','snapshot_group');
  END;
```

Теперь давайте подробно рассмотрим, как это все будет происходить:

1. После запуска формы нажмите указанные кнопки в заданной последовательности: «Создать запросную группу»->«COMBO LIST»->«Снимок».
2. По нажатии первой кнопки у нас создается запросная группа, на основе которой мы будем формировать элемент списка COMBO LIST.
3. Кнопка COMBO LIST формирует одноименный список на основе запросной группы, заполняя его данными из нее.
4. Кнопка «снимок»: при выполнении триггера этой кнопки вначале создается «snapshot_group» – это запросная группа, в которую мы будем «сливать» содержимое списка COMBO LIST. Далее на основе этой группы мы формируем список List Item, причем данные List Item должны полностью соответствовать данным COMBO LIST.

Удаление составляющих списка

Для удаления определенных элементов списка либо для полной его отчистки в Oracle Forms предложены следующие функции.

CLEAR_LIST – эта процедура служит для полной очистки элемента списка от его содержимого. Процедура принимает один-единственный параметр – это имя очищаемого списка:

```
CLEAR_LIST  (ITEM_NAME IN VARCHAR2);
```

Пример:

```
CLEAR_LIST ('COMBO_LIST')
```

DELETE_LIST_ELEMENT – эта процедура служит для удаления определенного элемента списка, поскольку мы указываем конкретный номер элемента, который собираемся удалить:

```
DELETE_LIST_ELEMENT (ITEM_NAME IN VARCHAR2, ITEM_INDEX IN NUMBER);
```

Пример:

```
DELETE_LIST_ELEMENT ('COMBO_LIST', 2);
```
– удалить второй элемент из списка.

Ниже приведен пример с использованием процедуры DELETE_LIST_ELEMENT, удаляющей из списка все элементы, в которых встречается значение «Sergey»:

```
declare
     n number;
begin
   n:=GET_LIST_ELEMENT_COUNT('COMBO_LIST');
for i in 1..n loop
   if GET_LIST_ELEMENT_VALUE('COMBO_LIST',i)='Sergey'
   then
     DELETE_LIST_ELEMENT ('COMBO_LIST', i);
 end loop;
end;
```

По этому примеру можно построить множество аналогичных процедур, которые позволят гибко обрабатывать данные – в данном случае удалять. В вышеприведенном примере были задействованы также и две незнакомые нам функции:

1. GET_LIST_ELEMENT_COUNT – эта функция возвращает количество строк в элементе списка:

```
GET_LIST_ELEMENT_COUNT (ITEM_NAME IN VARCHAR2) RETURN VARCHAR2;
```

Пример:

```
declare
LIST_COUNT NUMBER;
begin
LIST_COUNT:=GET_LIST_ELEMENT_COUNT('COMBO_LIST');
```

```
... // какое-то действие
end;
```

2. GET_LIST_ELEMENT_VALUE – эта функция возвращает значение элемента в списке по заданному индексу:

```
GET_LIST_ELEMENT_VALUE (ITEM_NAME IN VARCHAR2, ITEM_INDEX IN NUMBER)
RETURN VARCHAR2;
```

Пример:

```
declare
str_value varchar2(20);
begin
str_value:=GET_LIST_ELEMENT_VALUE('COMBO_LIST',i);
... // какое-то действие
end;
```

Примечание: *не удаляйте составляющий список и не очищайте весь список, если свойство Other Values этого элемента списка установлено на False и запрос открыт. Это может лишить Oracle Forms возможности выводить уже извлеченные записи, т. к. Oracle Forms будет пытаться вывести ранее извлеченные значения, но не сможет, потому что составляющий список удален. Поэтому перед удалением составляющего списка проверьте, нет ли открытых запросов, и для закрытия любых запросов используйте встроенную подпрограмму ABORT_QUERY.*

Кнопка

Кнопка – это элемент интерфейса, который позволяет пользователю самостоятельно инициировать события интерфейса нажатием на кнопку. Кнопки обычно используются для:

- выполнения различных расчетов;
- запуска окон и канвы;
- вызова списков значений (LOV);
- вызова других подпрограмм модулей;
- выполнения команд операционной системы;
- выполнения стандартных операций, назначаемых кнопкам, например, вызов различных диалогов, запуск отчетов и других приложений.

Создание кнопок

Для создания кнопки достаточно выполнить одно действие – в редакторе разметки на панели инструментов выбрать элемент «Кнопка» и нарисовать его на канве. Кнопка, как и любой другой элемент, имеет большой набор свойств. Рассмотрим основные свойства кнопки.

- Метка (Label) – надпись, выводимая на кнопке.
- Пиктограмма (Iconic) – это свойство определяет, является ли кнопка пиктографической.
- Имя файла пиктограммы (Icon Filename) – в этом свойстве вы определяете имя иконки для отображения на кнопке. Чтобы назначить иконку, необходимо указать путь к ней, причем название иконки можно указывать без расширения. Например, c:\pict или c:\pict.ico. Чтобы иконка отобразилась на вашей кнопке, необходимо, чтобы она (кнопка) была пиктографической.
- Кнопка по умолчанию (Default Button) – если значение этого свойства «True», то пользователь может выбрать эту кнопку нажатием командной клавиши, которая позволяет не переходить к ней и не активировать ее мышью. На одной канве может быть только одна кнопка по умолчанию.

Программное управление кнопками

Кнопка, как и все остальные элементы, имеет ассоциированные с ней события интерфейса, то есть триггеры. Для того чтобы выполнить какое-либо действие по нажатии кнопки, нужно ассоциировать с ней триггер WHEN-BUTTON-PRESSED и инициировать в нем какое-либо действие:

```
WHEN-BUTTON-PRESSED
DECLARE
   filename varchar2(1000);
BEGIN
filename:=get_file_name(filename);
END;
```

Вы также можете управлять свойствами кнопки в режиме выполнения с помощью процедуры SET_ITEM_PROPERTY:

```
if user='sqaime' then
SET_ITEM_PROPERTY('block_name.btn_name', visible, property_false);
end if;
end;
```

Элемент отображения (неизменный текст)

Неизменный текст – это элемент интерфейса, предназначенный для отображения информации. Особенность этого элемента в том, что он предоставляет данные только для чтения и при наведении курсора на этот элемент вы не получаете возможности входа в этот элемент. Несмотря на то что элемент отображения предоставляет информацию только для чтения, вы можете манипулировать содержимым этого элемента. Элемент «Неизменный текст» может быть базовым, то есть отображать данные ассоциированного с ним столбца базы данных, и управляющим.

Создание элемента отображения

Чтобы создать элемент отображения, выполните следующие действия: находясь в Редакторе Разметки, найдите на панели инструментов элемент «Элемент отображения» и начертите его на канве.

Обращаясь к элементу отображения в своих PL/SQL-программах, вы можете манипулировать его отображаемым содержимым. Выполним небольшой пример.

1. Запустите Редактор Разметки и разместите на канве элемент отображения и кнопку. Присвойте элементу отображения имя D_item.
2. В триггере кнопки WHEN-BUTTON-PRESSED напишите следующий код:

```
WHEN-BUTTON-PRESSED
:d_item:=10;
```

Если вы выполните этот пример, то увидите, как при нажатии на кнопку изменится отображаемое содержимое элемента.

Элементы рисования

Часто в программах требуется как-то обозначить или выделить группу объектов, разбить рабочую область на отдельные фрагменты. В Oracle Forms для этого существуют элементы рисования, их всего восемь:

- прямоугольник;
- линия;
- эллипс;
- дуга;
- многоугольник;
- ломаная;

- округленный прямоугольник;
- перо.

Элемент рисования – это элемент шаблонной графики, который не имеет идентификатора и не может использоваться в ваших программных единицах. Все перечисленные элементы приемлемы для отображения в GUI и браузере, что же касается терминального режима, то там отображаются только прямые линии – горизонтальные и вертикальные, а все остальные, даже если они и нарисованы на канве, игнорируются.

Все элементы рисования имеют небольшой набор свойств, управляющих отображением на канве.

Лекция 11. Элементы Forms Builder. Создание холстов, виды и свойства. Настройка визуальных эффектов

В этой лекции слушатель научится создавать фиксированные и вложенные холсты, панели инструментов и управлять их свойствами. Также в этой лекции слушатель научится создавать визуальные атрибуты и назначать их элементам, как на этапе проектирования, так и в режиме выполнения.

Ключевые слова: создание холстов, виды холстов, кадр, Frame, визуальный атрибут, визуализация, шрифт, фон, именованный атрибут, класс свойств.

Цель лекции: ознакомить слушателя с понятиями «холст», «кадр» и «визуальные эффекты».

Элемент холст (Canvas)

Когда вы делаете разметку блока с помощью мастера макетов или просто, находясь в навигаторе объектов, выделяете блок и вызываете редактор разметки, Forms автоматически генерирует для него и его элементов объект холст (далее канва), на котором они будут размещены. Вы можете иметь любое количество холстов в вашей форме.

Холст (Canvas) – это объект, который представляет собой фоновую основу, где вы размещаете элементы интерфейса, такие как радиокнопки, переключатели и текстовые элементы. Существует пять видов холстов:

- основная (Content), или заполнение;
- с вкладками (Tabbed);
- стековая (Stacked), или наложение;
- вертикальная панель инструментов (Vertical Toolbar);
- горизонтальная панель инструментов (Horizontal Toolbar).

Взаимосвязь элементов формы, окон и канвы можно представить в виде схемы (рис. 11.1).

Как показано на схеме, каждый элемент или группа элементов может располагаться на разных канвах. Канва или группа канв, в свою очередь, могут размещаться в одном окне. Окно должно содержать минимум одну канву. Канва может не только отображаться в окне, но еще и прокручиваться.

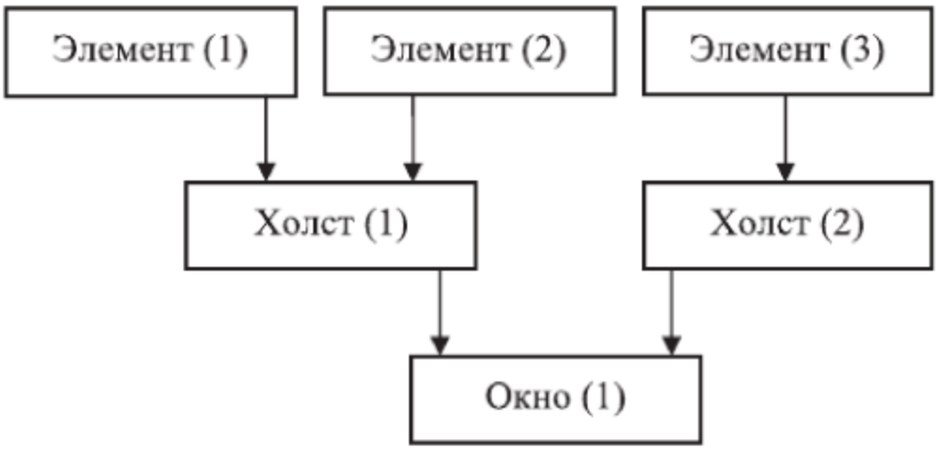

Рис. 11.1. Схема «Элемент-Холст-Окно»

Основная (Content)

Основная (Content) – это канва, которая создается по умолчанию и заполняет всю область окна. Для того чтобы создать основную канву, нужно выполнить следующие действия:

1. Находясь в навигаторе объектов, выделите узел Канвы (Canvas) и нажмите иконку «Создать» или выберите пункт меню Навигатор | Создать. Вы можете изменить вид существующей канвы, установив свойство «Тип канвы» – «Основная».

Несмотря на то что окно в один момент может отображать только одну вид-картинку, вы можете назначить одному окну более одной канвы. При создании канвы вида «Основная» в навигаторе объектов слева от объекта отображается иконка с изображением прямоугольника.

С вкладками (Tabbed)

С вкладками (Tabbed) – это канва, которая состоит из одной или более канв заполнения, помеченных вкладками. Каждая канва заполнения имеет свой набор свойств и имя. Другим словами, это канва с закладками, содержимое которых можно активировать, переключая закладки. Для создания канвы с вкладками:

1. Находясь в навигаторе объектов, выделите узел «Канвы» (Canvas) и нажмите иконку «Создать» или выберите пункт меню Навигатор | Создать. Вы можете изменить вид существующей канвы, установив свойство «Тип канвы» – «Вкладка». Если у вас на канве уже были размещены элементы, то при изменении типа канвы на «Вкладка» все элементы исчезнут и разместятся на NULL-канве. После того как вы создаете канву-вкладку, в узле «Канвы» появляется дочерний узел «Страницы вкладки», в котором создаются и размещаются закладки.

2. Для того чтобы создать закладку, перейдите в узел «Страницы вкладки» и нажмите иконку «Создать» или выберите пункт меню Навигатор | Создать. Вы не можете изменять вид закладки, то есть делать ее стековой или какой-либо другой.

Канва-вкладка очень удобна для размещения большого числа элементов, которые можно логически разбить по страницам. При создании канвы вида «Вкладка» в навигаторе объектов слева от объекта отображается иконка с изображением прямоугольника с вкладками.

Стековая (Stacked)

Стековая (Stacked) – эта канва, которая отображается поверх других канв, назначенных тому же окну, и показывает содержимое в зависимости от заданного условия. Этот тип канвы еще называют «наложением», так как она накладывается поверх других. Вы можете управлять поведением стековой канвы, указывая, отображать ли стековую канву немедленно или оставлять ее невидимой до вызова пользователем. Чтобы создать стековую канву, находясь в навигаторе объектов, выделите узел «Канвы» (Canvas) и нажмите иконку «Создать» или выберите пункт меню Навигатор | Создать. Вы можете изменить вид существующей канвы, установив свойство «Тип канвы» – «Дополнительная».

Несмотря на то что окно в один момент может отображать только одну вид-картинку, вы можете назначить одному окну более одной стековой канвы. При создании канвы вида «Дополнительная» в навигаторе объектов слева от объекта отображается иконка с изображением двух прямоугольников, один из которых налагается на другой.

Вертикальная и горизонтальная панели инструментов (Vertical/Horizontal Toolbars)

Вертикальная и горизонтальная панели инструментов (Vertical/Horizontal Toolbars) – это канва, которая представляет собой вертикальную или горизонтальную панель инструментов. Вертикальная панель инструментов располагается в левом крае окна, а горизонтальная – между окном и панелью меню. Для того чтобы создать вертикальную или горизонтальную панель инструментов:

1. Находясь в навигаторе объектов, выделите узел «Канвы» (Canvas) и нажмите иконку «Создать» или выберите пункт меню Навигатор | Создать.
2. Запустите палитру свойств созданной канвы и установите свойство «Тип канвы» — «Горизонтальная панель инструментов» или «Вертикальная панель инструментов». Выберите горизонтальную панель инструментов и назовите ее «HToolbar».

3. Чтобы назначить панель инструментов окну, находясь в объектном навигаторе, выделите окно и запустите для него палитру свойств. Найдите свойство «Горизонтальная панель инструментов» и выберите из выпадающего списка панель «HToolbar».
4. Установите следующие свойства канвы HToolbar:
   - Высота: 20;
   - Окно: ОКНО1.
5. Вернитесь в редактор разметки канвы HToolbar и начертите на канве одну кнопку. Установите для кнопки следующие свойства:
   - Ширина: 16;
   - Высота: 16;
   - В виде Пиктограммы: Да;
   - Имя файла Пиктограммы: C:\pic.ico.
6. Создайте триггер WHEN-BUTTON-PRESSED и напишите в нем приведенный ниже код:

```
WHEN-BUTTON-PRESSED
Message('Выход из формы');
Exit_Form;
```

Для того чтобы увидеть, как выглядит панель инструментов в режиме выполнения, запустите форму и нажмите кнопку, расположенную на панели инструментов. Создайте дополнительные кнопки на панели инструментов и создайте для них различные события.

Удаление холста

Если вы удаляете канву, то вместе с ней удаляются все принадлежащие ей элементы шаблонной графики, а все элементы, которые были на ней расположены, меняют свое свойство «Канва» на NULL.

Контур (вид)

Контур (вид) – это прямоугольная рамка, которая является видом для канвы, то есть контур фактически задает область видимости канвы в окне. Если контур меньше канвы, то есть охватывает ее частично, то в режиме выполнения вы увидите только ту область канвы, которая очерчена контуром. Если вы не видите контура на канве, выполните следующее действие: находясь в редакторе разметки, выберите пункт главного меню Представление | Показать представление.

Кадр (Frame)

Кадр – это элемент шаблонной графики, который располагается на канве. Кадр предназначен не для улучшения внешнего вида формы, а для управления визуальным отображением и размещением элементов в блоке. Кадр работает с блоком и его элементами как с единой структурой, распространяя свои действия на все объекты. Используя кадр, вы получаете возможность:

- управлять количеством выводимых объектов;
- упорядочивать элементы на канве;
- связывать кадр с блоком данных;
- включать кадр в состав объектной библиотеки.

Создание кадра

Вы можете создавать любое количество кадров в вашей форме, используя редактор разметки или навигатор объектов. В принципе, зачастую нет необходимости в создании элемента кадра вручную, так как он создается автоматически при создании разметки для «блока данных в мастере макетов». Для того чтобы создать кадр вручную и связать с блоком данных, выполните следующие действия:

1. Находясь в редакторе разметки, найдите на панели инструментов элемент «Кадр» и начертите его на канве.
2. Находясь в навигаторе объектов, найдите и разверните узел Картинка(CANVAS) | Графика(Graphics). Выделите Кадр и запустите Палитру Свойств.
3. Найдите свойство «Блок Данных Разметки (Layout Data Block)» и из выпадающего списка доступных блоков выберите тот блок, с которым хотите ассоциировать кадр.

После этого кадр создан и связан с блоком данных, можно настроить другие свойства. Рассмотрим основные свойства.

- Блок данных разметки (Layout Data Block) – определяет блок данных, ассоциируемый с кадром.
- Обновить разметку (Update Layout) – определяет стиль обновления рамки. Если вы выбираете стиль обновления рамки «Автоматически», нужно помнить, что любое ручное вмешательство в расположение элементов в кадре будет аннулировано при первом же взаимодействии с кадром. Так что даже если вы вынесете какой-либо элемент за пределы кадра, при малейшем изменении положения на канве или размерной части элемент будет возвращен обратно. Существует три типа обновления:

  - автоматически (Automatically);
  - вручную (Manually);
  - блокирован (Locked).
- Стиль Разметки (Layout Style) – определяет стиль размещения элементов блока данных на канве. Различают два типа разметки:
  - форма (Form);
  - таблица (Tabular).
- Выравнивание рамки (Frame Alignment) – определяет стиль выравнивания объектов в кадре. Различают пять видов автоматического выравнивания элементов:
  - начало (Start);
  - конец (End);
  - по центру (Center);
  - заполнить (Fill);
  - столбец (Column).
- Выравнивание единичного объекта (Single Object Alignment) – определяет поведение кадра при попытке выровнять однострочные объекты. Различают три типа:
  - в начало;
  - по центру;
  - в конец.
- Горизонтальная граница (Horizontal Margin) – определяет расстояние между левой (правой) границей кадра и близлежащими объектами.
- Вертикальная граница (Vertical Margin) – определяет расстояние между верхней границей кадра и первым близлежащим объектом.
- Сдвиг горизонтального объекта (Horizontal Object Offset) – определяет горизонтальный зазор между объектами.
- Вертикальный сдвиг объекта (Vertical Object Offset) – определяет вертикальный зазор между объектами.
- Разрешить расширение (Allow Expansion) – если значение этого свойства истинно, то в случае если элементы не попадают в область рамки, кадр будет автоматически расширяться до тех пор, пока область рамки не покроет этот элемент.
- Масштабировать (Shrinkwrap) – если значение этого свойства истинно, то пустое место в кадре будет автоматически удалено.
- Вертикальное заполнение (Vertical Fill) – если значение этого свойства истинно, значит, при переходе в стиль разметки «Форма» пустые места в кадре вокруг объектов будут использоваться.
- Максимальное число объектов на строку (Maximum Objects per line) – определяет количество выводимых объектов в одной строке кадра.

Вы также можете создать кадр, не связанный с блоком данных, для использования в графических целях.

Визуальные эффекты в Oracle Forms

В этой главе речь пойдет о том, как настроить внешний вид вашего приложения, используя атрибуты визуализации. Разрабатывая приложение, вы должны акцентировать свое внимание не только на функциональности и быстродействии, но и на приятном, дружественном для пользователя интерфейсе. Когда программа имеет приятный вид, с ней хочется работать, в ней хочется все детально рассмотреть. В Forms вы можете устанавливать атрибуты визуализации для элементов, записей, блоков, подсказок. Применяя именованные атрибуты, вы с легкостью можете изменять их значение не только на этапе проектирования приложения, но и во время выполнения.

Атрибуты визуализации – это атрибуты элементов интерфейса, представляющие свойства шрифта, цветов и шаблона заполнения. Исходя из определения, визуальные атрибуты можно классифицировать по свойствам:

- атрибуты, включающие свойства шрифта;
- атрибуты, включающие свойства цветов и шаблонов.

Виды атрибутов визуализации

Создавая внешний вид своего приложения, вы можете использовать три типа атрибутов визуализации:

- пользовательский (Custom);
- по умолчанию (Default);
- именованный объект атрибута визуализации (Named).

Пользовательский Атрибут визуализации (Custom)

Пользовательский атрибут визуализации – это свойства шрифта и цвета, установленные в палитре свойств элемента на этапе проектирования формы. Определяя настройки атрибутов в палитре свойств, вы указываете специфические значения, которые будут установлены для элемента при выводе на экран или заполнении данными. Пользовательский атрибут не создается как таковой, он лишь является набором специфических свойств, установленных для конкретного элемента. Для того чтобы определить пользовательский атрибут, достаточно вызвать палитру свойств какого-либо элемента интерфейса:

1. Создайте новый элемент интерфейса, например, элемент «Редактируемый текст», и вызовите для него палитру свойств.
2. Найдите свойства, перечисленные ниже, и установите для них следующие значения:
   - Название шрифта (Font Name): Tahoma;
   - Стиль шрифта (Font Style): курсив (Italic);

- Цвет отображаемых на экране символов (Foreground color): blue.

Значение свойств атрибутов визуализации изменены, но они действительны только для конкретного элемента и выполняются вручную отдельно для каждого элемента, поддерживающего свойства визуализации, поэтому и называются пользовательскими.

Атрибут визуализации по умолчанию (Default)

Атрибут визуализации по умолчанию (Default) – это атрибут, который устанавливает значение всех свойств атрибутов на значения по умолчанию, или, как их еще называют, системные настройки. Чтобы определить атрибут визуализации по умолчанию, достаточно изменить свойство «Группа атрибутов визуализации» на DEFAULT. Если атрибут относится к записи, то, соответственно, свойство «Группа визуализации текущей записи» должно быть также установлено на DEFAULT.

Именованный атрибут визуализации (Named)

Именованный атрибут визуализации (Named) – это именованный объект, который содержит набор атрибутов, включающих свойства шрифта и цвета. Преимущество именованного атрибута в том, что, однажды создав и определив для него свойства, вы можете применять его к любым элементам, которые будут наследовать его свойства. К тому же, если понадобится перенести ваше приложение на другую платформу или просто изменить какое-либо свойство, вам будет достаточно переопределить только набор свойств атрибута визуализации, а не настраивать каждый элемент по отдельности.

Чтобы создать атрибут визуализации и применить его к объекту, выполните следующие действия:

1. Находясь в навигаторе объектов, выделите узел «Атрибуты визуализации» и нажмите иконку «Создать» или выберите пункт меню Навигатор | Создать.
2. Запустите палитру свойств созданного атрибута и установите для него следующие свойства:
   - Имя: Item_atr;
   - Цвет отображаемых на экране символов (Foreground color): blue;
   - Цвет фона (Background Color): dark yellow;
   - Шаблон заполнения (Fill Pattern): gray3.3;
   - Название шрифта (Font Name): Tahoma;
   - Размер шрифта (Font Size): 12;
   - Вес шрифта (Font Weight): Полужирный (DemiBold);
   - Стиль шрифта (Font Style): курсив (Italic);
   - Разрядка для шрифта (Font Spacing): Нормальный (Normal).

3. Находясь в редакторе разметки, начертите на канве элемент «Редактируемый текст» и вызовите для него палитру свойств. Найдите свойство «Группа атрибутов визуализации» и выберите из выпадающего списка именованный атрибут «Item_atr».
4. Запустите форму и попробуйте ввести текст. Вы также можете назначить именованный атрибут визуализации свойству элемента «Группа атрибутов визуализации текущей записи» – тогда элемент, отображающий текущую запись, будет наследовать свойства ассоциированного с ним атрибута визуализации.

Вы можете управлять именованным атрибутом визуализации программно, используя процедуру SET_ .

1. Продолжая предыдущий пример, начертите кнопку на канве и создайте для нее триггер WHEN-BUTTON-PRESSED. В теле триггера напишите следующий код:

```
WHEN-BUTTON-PRESSED
IF Name_IN('Bl_atr.t_item')=5
  THEN
set_item_property('t_item', visual_attribute, 'item_atr');
Message('5 -Слишком большое значение для этого элемента!',
ACKNOWLEDGE);
END IF;
```

2. Запустите форму на выполнение и попробуйте ввести значение 5. После того как вы нажмете кнопку, вид элемента поменяется в соответствии с назначенными ему свойствами. Вы также можете управлять атрибутом визуализации программно, используя процедуру DISPLAY_ITEM:

```
WHEN-BUTTON-PRESSED
IF Name_IN('Bl_atr.t_item')=5
  THEN
Display_Item('Bl_atr.t_item','item_atr');
Message('5 - Слишком большое значение для этого элемента!',
ACKNOWLEDGE);
END IF;
```

DISPLAY_ITEM (ITEM_NAME IN varchar2, DISPLAY_ATTRIBUTE_NAME IN varchar2) модифицирует способ, которым элемент выводится на экран или терминал, и назначает указанный атрибут вывода этому элементу. Названия возможных атрибутов вывода хранятся в карте Oracle*Terminal

(с которой форма запускается) для терминального режима и в узле «Атрибут визуализации» навигатора объектов для GUI. Если вы определили неверный атрибут вывода, Forms назначает атрибут вывода по умолчанию для данного элемента. Любое изменение, сделанное встроенной процедурой DISPLAY_ITEM, действительно до тех пор, пока аналогичная процедура не обратится к этому элементу или текущая форма не завершится.

Классы свойств

Oracle Forms предоставляет мощное средство «Класс свойств», которое позволяет подходить к разработке более глобально и гибко. Если провести аналогию, то класс свойств можно сравнить с именованным атрибутом визуализации, включающим в себя конкретизированный набор свойств, которые впоследствии могут быть наследованы различными элементам. Класс свойств в отличие от атрибута визуализации может включать в себя все свойства Forms и может разрешить наследовать их другим объектам в пределах более чем одного модуля. Используя классы свойств в своем приложении, вы получаете возможность одноразовым переопределением свойства переопределить идентичные свойства всех объектов, которые базируются на этом классе. Классы свойств можно базировать друг на друге.

Класс свойств – это именованный объект, который содержит список свойств и их значений, заданных разработчиком. Чтобы создать класс свойств, выполните следующие действия:

1. Находясь в Навигаторе Объектов, выделите узел «Классы свойств» и нажмите иконку «Создать» или выберите пункт меню Навигатор|Создать. Назовите созданный объект «TFONT».
   Запустив палитру свойств этого объекта, вы увидите всего три свойства. Чтобы добавить атрибут в список свойств класса, нужно нажать кнопку «Добавить свойство («Add property»)» (рис. 11.2) на панели инструментов палитры свойств. После нажатия кнопки на экране появится окно «Свойства».

Рис. 11.2. Панель инструментов палитры свойств

2. Выберите свойства Font Name, Font Size, Font Style и Font Weight и установите для них значения. Чтобы выбрать атрибут в список класса свойств, выделите его и нажмите кнопку «OK» (рис. 11.3).

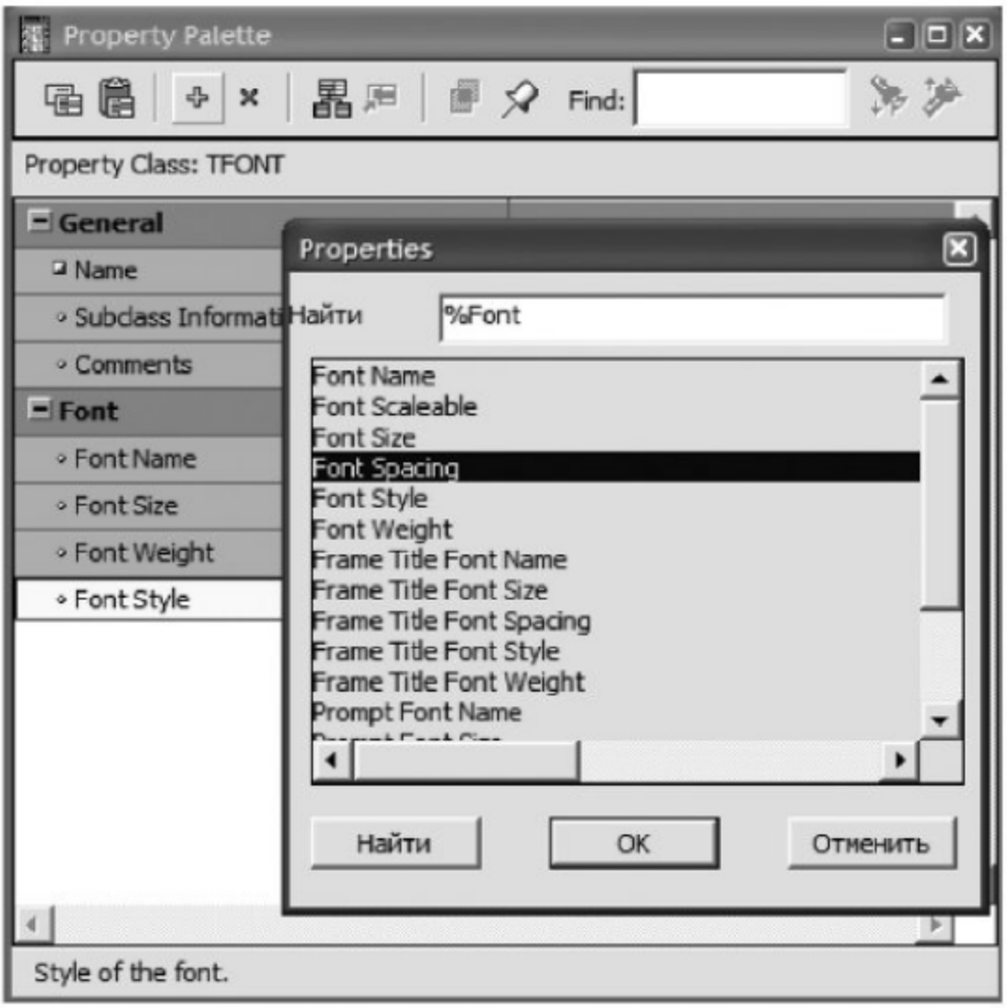

Рис. 11.3. Добавление атрибута в объект класса свойств

Теперь, когда класс свойств создан, можно попробовать базировать на нем какой-либо объект.

1. Находясь в редакторе разметки, начертите на канве элемент «Редактируемый текст» и вызовите для него палитру свойств. Найдите свойство «Информация о подклассе» и нажмите кнопку «Найти» для вызова окна «Информация о подклассе» (рис. 11.4).

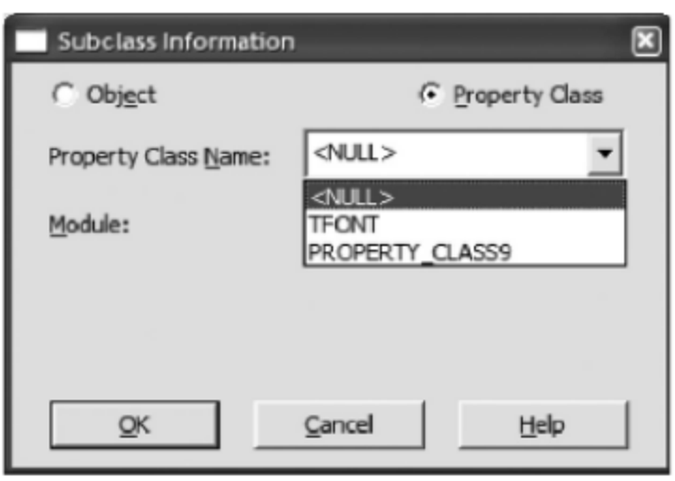

Рис 11.4. Окно «Информация о подклассе»

2. В появившемся окне выберите радиокнопку «Класс свойств», затем перейдите в поле «Имя класса свойств» и выберите из выпадающего списка созданный нами ранее класс TFONT. Подтвердите выбор и запустите форму на выполнение для просмотра результатов.

Триггеры классов свойств

Вы можете определять триггеры для классов свойств, причем триггер, который определен для элемента, базирующегося на классе свойств, будет исполняться подобно тому, как если бы он был создан конкретно для этого класса. Все формы, блоки и элементы, базирующиеся на классе свойств, для которого определен триггер, также наследуют этот триггер.

Как показали примеры, классы свойств – это действительно мощное средство в руках разработчика, так как позволяет быстро и эффективно управлять атрибутами большого числа объектов. Несмотря на всю мощь этого объекта, все же имеются ограничения в его использовании:

- если в элементе определен именованный атрибут визуализации и класс свойств, то предпочтение будет отдано именованному атрибуту;
- классами свойств нельзя управлять программно.

Лекция 12. Триггеры Oracle Forms. Классификация триггеров. Пользовательские триггеры

В этой лекции слушатель ознакомится с понятием триггера, его назначением и классификацией.

Ключевые слова: область триггера, тип триггера, код триггера, транзакционный триггер, триггер события интерфейса, триггер обработки событий блока, триггеры обработки сообщений и ошибок, триггеры главный-подчиненный, триггеры проверки допустимости, триггеры периода запроса.

Цель лекции: ознакомить слушателя с понятием триггер, его структурой и областью действия.

Компоненты триггера

Компоненты триггера – это три главные составляющие триггера, которые определяют его основные свойства и назначение (рис. 12.1).

- Область триггера (Trigger scope) – это уровень модуля, на котором определен триггер.
- Тип триггера (Trigger Type) – определяет принадлежность к тому или иному событию, во время которого он будет срабатывать.
- Код триггера (Trigger code) – это блок PL/SQL, определяющий действия триггера во время срабатывания.

Рис. 12.1. Компоненты триггера

Самое главное при написании, определении и кодировании триггера – точно понимать, когда и в какой момент происходит событие по отношению к другим событиям.

Сфера триггера (Trigger scope)

Область триггера определяет его позицию в иерархии объектов Oracle Forms, а именно – под каким объектом он будет создан. Ассоциированный с триггером объект определяет его уровень. Вы можете подсоединить триггер к форме, блоку, элементу или записи.

- Форма – триггер срабатывает для всей формы. Если вы, к примеру, создадите триггер KEY-COMMIT, то событие, описанное в нем, будет применено ко всему модулю.
- Блок – триггер срабатывает, если событие происходит в пределах блока. Если же у вас форме определено два блока и более, то триггер будет отвечать только на события того блока, в котором он определен. Если то же самое событие возникает в пределах другого блока, то оно касается только его.
- Элемент – триггер срабатывает, если событие происходит в пределах элемента, в котором он определен.

Если, к примеру, триггер WHEN-BUTTON-PRESSED создан для кнопки «BUTTON1», то он сработает только для этой кнопки, но не при нажатии «BUTTON2» или «BUTTON3». Если, к примеру, вы определите триггер WHEN-NEW-ITEM-INSTANCE на уровне элемента и для конкретного элемента, то действие, которое вы определили в теле триггера, будет срабатывать в момент, когда фокус ввода попадет именно в этот элемент, а не в какой другой. Если, к примеру, вы создали триггер WHEN-NEW-ITEM-INSTANCE на уровне блока, то в этом случае триггер будет срабатывать при попадании фокуса ввода в любой из элементов, принадлежащих блоку.

Кроме понятия уровень триггера существует еще четыре понятия:

- приоритет – если один и тот же триггер определен на всех трех уровнях иерархии, то первым сработает триггер с наименьшим уровнем – Record Level, уровень элемента (записи);
- ограничение по уровню определения – это значит, что не каждый из представленных триггеров в Oracle Forms может быть определен на том или ином уровне. По умолчанию при вызове списка триггеров отображаются только те, которые доступны для этого уровня, не давая вам возможность совершить ошибку. Например, триггер ON-LOGON можно определить только на уровне формы. Если вы хотите выполнить проверку записи, то сделать это вы можете, лишь подсоединив триггер WHEN-VALIDATE-RECORD к форме или блоку, но не к элементу;
- ограничение режима выполнения – это ограничение, которое запрещает выполнение того или иного триггера в режиме запроса, то есть свойство «Fire in Enter Query Mode» такого триггера по умолчанию установлено на No;

- ограничение на выполнения команд – в зависимости от типа триггера существуют ограничения на выполнение различных команд: навигация, сохранение, откат и так далее. Например, если вы попытаетесь выполнить в триггере с префиксом POST команду go_item, то получите ошибку: FRM-40737: Недопустима ограниченная процедура GO_ITEM в POST триггере.

Вы можете управлять приоритетами срабатывания триггеров, используя свойство «Иерархия выполнения» триггера, которое поможет вам решить, как поступить в случае, если подобный триггер уже определен на одном из уровней иерархии.

Код триггера

Код триггера – это анонимный блок PL/SQL, который выполняется при срабатывании триггера. Блок может состоять из трех разделов:

- раздел декларирования – этот раздел необходим для объявления переменных, констант, курсоров и исключений. Если у вас нет потребности в вышеперечисленных операциях, то этот раздел можно опустить;
- раздел исполняемых операторов – это секция, которая содержит основу вашего PL/SQL-блока, а именно – исполняемые операторы. Этот раздел обязателен;
- раздел обработчиков исключений – эта секция необязательна и используется для обработки исключений.

1. Общий синтаксис блока PL/SQL выглядит так:

```
DECLARE – декларативные операторы (необязательные)
BEGIN – исполняемые операторы (обязательно)
EXCEPTION – обработчики исключений (необязательно)
END;
```

2. Если вы не объявляете переменные, то секция DECLARE не обязательна:

```
BEGIN
Message('Первый блок');
BEGIN
Message('Второй блок');
END;
END;
```

3. Операторы BEGIN и END необязательны, если вы не объявляете переменные и не используете вложенные блоки.

```
IF :SYSTEM.CURSOR_RECORD=1 THEN
Message('Вы на первой записи');
END IF;
```

При написании тела триггера вы можете использовать:

- SQL-операторы, которые можно использовать в PL/SQL-блоке;
- стандартные PL/SQL-конструкции (константы, операторы управления и так далее);
- хранимые процедуры, функции и пакеты;
- встроенные подпрограммы и пакеты.

1. Использование операторов SQL в триггере:

```
DECLARE
Val NUMBER;
BEGIN
SELECT 1 INTO val FROM DUAL;
Insert into table table_name (col) values (val);
END;
```

2. Стандартные PL/SQL-конструкции:

```
FOR I IN 1..10 LOOP
(действие);
END LOOP;
```

3. Встроенные подпрограммы и пакеты:

```
SET_BLOCK_PROPERTY ('block_name', WHERE_CLAUSE, 'ROWNUM<=10');
```

4. Хранимые процедуры, функции и пакеты:

```
DECLARE
Val number;
BEGIN
My_procedure('value');
Val:=my_function('value');
END;
```

Классификация триггеров

Триггеры можно классифицировать по имени и по функциональности (табл. 12.1). Имя триггера состоит из двух основных частей – префикса и имени. Префикс в имени триггера означает момент или время его срабатывания (табл. 12.2), а имя – ассоциированное с ним событие. В Oracle Forms существует более 100 триггеров.

Таблица 12.1. Классификация триггеров

Функция	**Имя/Тип**
Триггеры обработки блоков	WHEN-Event триггеры
Триггеры событий интерфейса	ON-Event триггеры
Триггеры главный-подчиненный	PRE-Event триггеры
Триггеры обработки сообщений	POST-Event триггеры
Навигационные триггеры	KEY триггеры
Триггеры периода запроса	USER-named – пользовательские триггеры
Транзакционные триггеры	
Триггеры проверки допустимости	

Таблица 12.2. Описание префиксов триггера

Префикс триггера	**Описание**
WHEN-	Сигнализирует о точке, в которой можно нарастить стандартную (по умолчанию) обработку Oracle Forms дополнительными условиями, задачами и исключениями.
ON-	Срабатывает во время обработки события, сообщая о точке, в которой можно заменить стандартную обработку этого события, выполняемую Oracle Forms по умолчанию.
PRE-	Срабатывает до выполнения действия, описанного в имени триггера. То есть если вы опишете какое-либо действие в триггере Pre-Query, то оно выполнится до того, как запрос будет обработан.

POST-	Срабатывает после выполнения действия, описанного в имени триггера. То есть если вы опишете какое-либо действие в триггере Post-Query, то оно выполнится после того, как запрос будет обработан.
KEY-	Срабатывает во время нажатия функциональной клавиши, имя которой ассоциировано с действием.

Типы событий

В общем все события можно разбить на два типа.

События интерфейса – это внешние события интерфейса, результат действия которых выявляется сразу. К событиям интерфейса можно отнести следующие действия:

- нажатие на кнопку;
- изменение состояния выключателей и радиокнопок;
- нажатие командных клавиш.

События внутренней обработки – это события, происходящие вследствие выполнения какого-либо действия, в результате которого возникает ряд логически последовательных событий. Возникновение событий внутреннего интерфейса инициировано обработчиком Oracle Forms в соответствии с общей моделью обработки данных.

Пример № 1

У нас имеется форма из одного блока и двух элементов. Для того чтобы перейти из одного элемента в другой, нам достаточно нажать клавишу ENTER или кликнуть мышкой на втором элементе, то есть физически совершить одно действие – нажатие или клик. На самом же деле при перемещении из одного элемента в другой совершается ряд событий:

1. Проверка элемента – WHEN-VALIDATE-ITEM.
2. Покидание элемента – POST-ITEM.
3. Проверить запись – WHEN-VALIDATE-RECORD.
4. Покинуть запись – POST-RECORD.
5. Вхождение в запись – PRE-RECORD.
6. Вхождение в элемент – PRE-TEXT-ITEM.
7. Запись готова для ввода – WHEN-NEW-RECORD-INSTANCE.
8. Элемент готов для ввода – WHEN-NEW-ITEM-INSTANCE.

Итак, при переходе из одного элемента в другой имеем восемь событий – события внутренней обработки. Этот пример наглядно показывает, как при выполнении одного действия выполняется ряд других событий.

Пример № 2

На форме имеется два блока, в каждом из которых есть один текстовый элемент. Как и в предыдущем примере, чтобы перейти из одного блока в другой, нам достаточно выполнить одно действие, которое влечет за собой ряд событий внутренней обработки и в отличие от предыдущего примера будет содержать события блока.

1. Проверка элемента – WHEN-VALIDATE-ITEM.
2. Покидание элемента – POST-ITEM.
3. Проверить запись – WHEN-VALIDATE-RECORD.
4. Покинуть запись – POST-RECORD.
5. Покинуть блок – POST-BLOCK.
6. Войти в блок – PRE-BLOCK.
7. Вхождение в запись – PRE-RECORD.
8. Вхождение в элемент – PRE-TEXT-ITEM.
9. Блок готов для ввода – WHEN-NEW-BLOCK-INSTANCE.
10. Запись готова для ввода – WHEN-NEW-RECORD-INSTANCE.
11. Элемент готов для ввода – WHEN-NEW-ITEM-INSTANCE.

Приведенные примеры помогут вам понять общую модель обработки Oracle Forms.

Свойства триггера

Для управления поведением триггера используйте его свойства, перечисленные в палитре свойств (рис. 12.2).

- **Имя триггера** – это зарезервированное слово, определенное в Oracle Forms для именования события. Если вы создадите стандартный триггер, а затем измените его имя на другое, не входящее в перечень зарезервированных имен триггеров Oracle Forms, то такой триггер автоматически становится пользовательским. Также стандартный триггер становится пользовательским, если его имя не соответствует области его определения. Так, например, триггер WHEN-VALIDATE-RECORD недоступен на уровне элемента, поэтому если вы создадите пользовательский триггер с таким именем или переименуете существующий стандартный триггер в WHEN-VALIDATE-RECORD, то он все равно будет пользовательским.

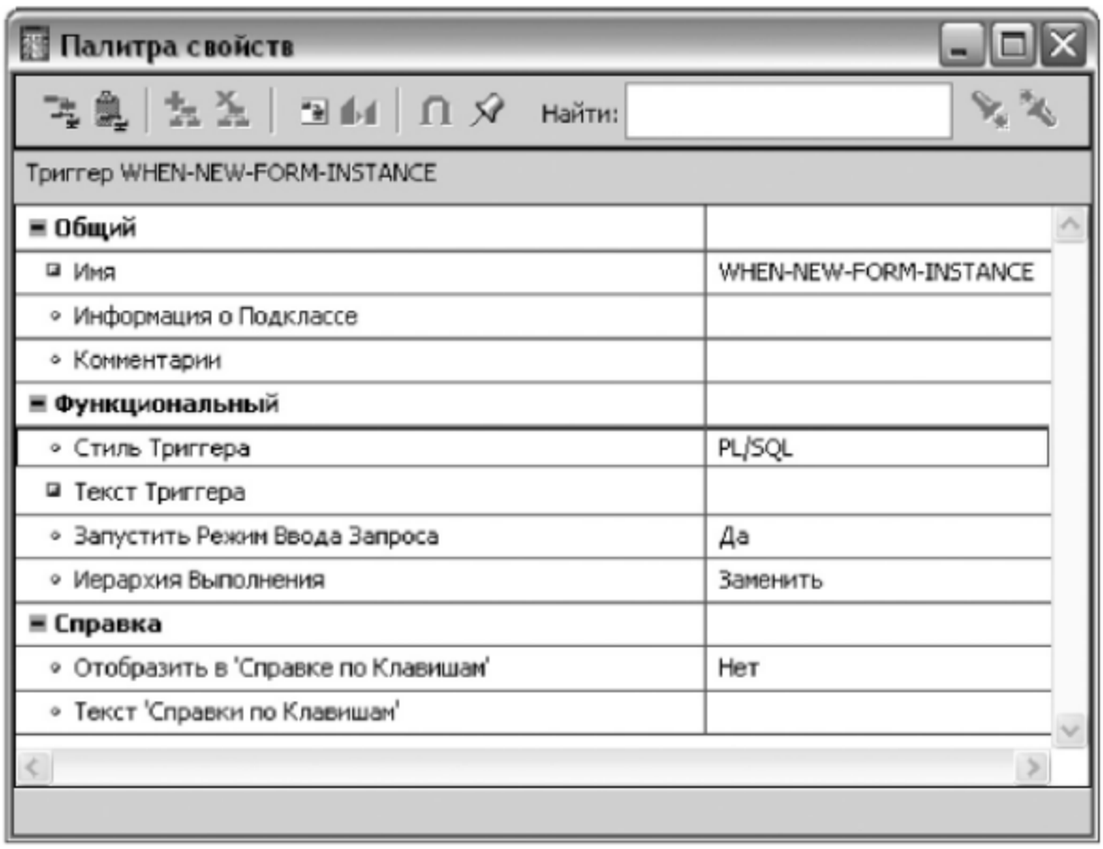

Рис. 12.2. Свойства триггера

- **Стиль триггера** – это тип выполняемой команды. Значение этого свойства неизменно, так как тело триггера может содержать только PL/SQL-код.
- **Текст триггера** – это код триггера (тело триггера), выполняемый при его срабатывании. При выборе этого свойства появляется PL/SQL-редактор.
- **Режим запроса** – определяет поведение триггера во время режима запроса. Если значение свойства установлено в «Yes», триггер срабатывает в режиме запроса.
- **Иерархия выполнения** – значение этого свойства определяет поведение триггера, в случае если на более высоком уровне объявлен триггер с тем же самым именем. Для этого свойства доступны три значения:
  - заменить – выполняется только код текущего триггера;
  - перед – текущий триггер выполняется до выполнения триггера с более высоким уровнем определения;
  - после – текущий триггер выполняется после выполнения триггера с более высоким уровнем определения.
- **Отобразить в «Справке по клавишам»** – определяет, выводится ли триггер в «Справке по клавишам».

- **Текст «Справки по клавишам»** – описание триггера, выводимое в «Справке по клавишам».

Например, триггеры типа «KEY-» срабатывают независимо от того, в каком режиме находится форма. Чтобы запретить срабатывание этого триггера, в режиме запроса установите значение свойства «Режим запроса» на «No».

Триггеры обработки блоков

Триггеры обработки блоков – это триггеры, которые срабатывают в ответ на события, связанные с модификацией и управлением записями в блоке, например:

- WHEN-CREATE-RECORD – это событие возникает, когда Forms пытается создать запись в блоке;
- WHEN-CLEAR-BLOCK – это событие возникает, когда Forms пытается отчистить блок, то есть удалить все записи;
- WHEN-DATABASE-RECORD – это событие возникает, когда Forms пытается изменить статус записи;
- WHEN-REMOVE-RECORD – это событие возникает, когда Forms пытается удалить или отчистить запись.

Триггеры события интерфейса

Триггеры события интерфейса – это триггеры, которые срабатывают в ответ на события интерфейса формы, обусловленные действием оператора или программным управлением.

- WHEN-BUTTON-PRESSED – это событие возникает, когда оператор нажимает на кнопку, причем независимо от того, как было осуществлено нажатие – с помощью мыши или клавиатуры.
- WHEN-CHECKBOX-CHANGED – это событие возникает, когда оператор изменяет состояние переключателя, включая и выключая его, причем независимо от того, как было осуществлено нажатие – с помощью мыши или клавиатуры.
- WHEN-IMAGE-ACTIVATED – это событие возникает, когда оператор дважды щелкает по элементу изображения.
- WHEN-IMAGE-PRESSED – это событие возникает, когда оператор щелкает по элементу изображения.
- WHEN-RADIO-CHANGED – это событие возникает при выборе оператором радиокнопки в радиогруппе.
- WHEN-TIMER-EXPIRED – это событие возникает, когда истекает период времени, определенный в таймере.

- WHEN-WINDOW-ACTIVATED – это событие возникает, когда активизируется окно.
- WHEN-WINDOW-CLOSED – это событие возникает в ответ на закрытие окна.
- WHEN-WINDOW-DEACTIVATED – это событие возникает, когда окно становится неактивным, например, когда оператор активизирует другое окно.
- WHEN-WINDOW-RESIZED – это событие возникает в ответ на изменение размеров окна.
- KEY-[ALL] – это событие, возникающие в ответ на нажатие функциональной клавиши. К этой категории относятся все триггеры с префиксом KEY.

Триггеры главный-подчиненный

Триггеры главный-подчиненный или Мастер-Деталь – это триггеры, которые управляют координацией между записями главной и подчиненной таблиц и срабатывают в ответ на модификацию записей в Мастер-блоке или Деталь-блоке. Триггеры главный-подчиненный генерируются Oracle Forms автоматически при создании отношения, например:

- ON-CHEK-DELETE-MASTER – это событие возникает при попытке Oracle Forms удалить запись в главном блоке;
- ON-CLEAR-DETAILS – это событие возникает при очистке Oracle Forms записей подчиненного блока, в момент, когда они уже не соответствуют записи главного блока;
- ON-POPULATE-DETAILS – это событие возникает в момент, когда Oracle Forms необходимо извлечь записи в подчиненный блок.

Триггеры обработки сообщений и ошибок

Триггеры обработки сообщений – это триггеры, которые срабатывают в ответ сообщения, которые возникают во время выполнения формы, например:

- ON-ERROR – это событие возникает в ответ на возникновения ошибки. Эти триггеры предназначены для того, чтобы вы могли подавить сообщения об ошибке, заменить его свои сообщением или выполнить иное действие;
- ON-MESSAGE – это событие, возникающее в ответ на сообщение Oracle Forms и предназначенное для замены стандартного сообщения или его подавление.

Эти триггеры следует использовать для обработки наиболее обобщенных ситуаций, так как основная обработка ошибок должна быть выполнена непосредственно в том блоке, который ее возбуждает.

Навигационные триггеры

Триггеры навигации – это триггеры, которые срабатывают в ответ на какие-либо навигационные действия в форме, например, перемещение фокуса на другой объект или переход от одного элемента к другому. Навигационные триггеры действуют на всех уровнях иерархии объектов Forms. Навигационные триггеры можно разделить на две категории: PRE-, POST- и WHEN-.

PRE-, POST – это триггеры внутренней навигации, которые срабатывают в ответ на внутреннее перемещение между иерархиями объектов. Так, например, при перемещении фокус ввода одним щелчком мыши из одного элемента блока в элемент, находящийся в другом блоке, выполняется целый ряд внутренних навигационных событий.

- PRE-FORM – это событие возникает перед переходом на уровень формы, например, перед запуском формы.
- PRE-BLOCK – это событие возникает перед тем, как Oracle Forms перейдет с уровня формы на уровень блока.
- PRE-RECORD – это событие возникает перед тем, как Oracle Forms перейдет с уровня блока на уровень записи.
- PRE-TEXT-ITEM – это событие возникает перед переходом с уровня записи на уровень элемента.
- POST-TEXT-ITEM – это событие возникает во время того, как Oracle Forms покидает элемент и переходит на уровень записи.
- POST-RECORD – это событие возникает во время того, как Oracle Forms покидает запись и переходит на уровень блока.
- POST-BLOCK – это событие возникает во время того, как Oracle Forms покидает блок и переходит на уровень формы.
- POST-FORM – это событие происходит во время того, как Oracle Forms перейдет к «вне» формы, то есть покинет уровень формы, например, выход из формы или переход к другой форме.

WHEN – это триггеры, срабатывающие в завершении последовательности навигации, то есть уже после перемещения фокуса ввода в элемент. Триггеры этого типа не реагируют на события внутреннего интерфейса.

- WHEN-NEW-FORM-INSTANCE – это событие возникает при старте формы.

- WHEN-NEW-BLOCK-INSTANCE – это событие возникает при перемещении фокуса ввода в блок.
- WHEN-NEW-RECORD-INSTANCE – это событие возникает при переходе фокуса ввода в новую запись.
- WHEN-NEW-ITEM-INSTANCE – это событие возникает после того, как фокус ввода переместится в другой элемент.

Триггеры перечислены в порядке их срабатывания – так, например, если вы перемещаетесь с записи одного блока на новую запись другого блока, то триггеры срабатывают в следующем порядке:

```
WHEN-NEW-BLOCK-INSTANCE - WHEN-NEW-RECORD-INSTANCE
```

Триггеры периода запроса

Триггеры периода запроса – это триггеры, которые срабатывают в ответ на события, связанные с обработкой запроса, до или после его выполнения в блоке, причем независимо от того, инициировано оно пользователем или программой, например:

- PRE-QUERY – это событие возникает до того, как в базу данных будет отправлен оператор SELECT;
- POST-QUERY – это событие возникает после того, как записи извлечены, причем данный триггер срабатывает для каждой извлекаемой записи отдельно.

Например, для того чтобы изменить критерий запроса перед его выполнением используйте триггер PRE-QUERY. Если же вам необходимо откорректировать извлекаемое значение, перед тем как отобразить его в блоке, задействуйте триггер POST-QUERY. Ниже приведен пример, в котором корректируется POST-QUERY.

```
DECLARE
Select cena*0.10 INTO :cena from price;
END;
```

Транзакционные триггеры

Транзакционные триггеры – это триггеры, которые срабатывают в ответ на события, происходящие во время взаимодействия формы с источником данных. Примерами таких событий могут служить: подсое-

динение к Oracle, фиксация или откат транзакции, обработка операций DML во время и после посылки транзакций, например:

- ON-DELETE – это событие сигнализирует о точке, в которой можно заменить обработку процесса удаления записей Oracle Forms по умолчанию;
- ON-INSERT – это событие сигнализирует о точке, в которой можно заменить обработку процесса вставки записей Oracle Forms по умолчанию;
- ON-LOCK – это событие сигнализирует о точке, в которой можно заменить обработку блокировок Oracle Forms по умолчанию;
- ON-LOGON – это событие сигнализирует о точке, в которой можно заменить обработку подсоединения Forms к базе данных Oracle или к любому другому источнику;
- ON-LOGOUT – это событие сигнализирует о точке, в которой можно заменить обработку Forms по умолчанию для отсоединения от Oracle;
- ON-UPDATE – это событие сигнализирует о точке, в которой можно заменить обработку процесса обновления записей Oracle Forms по умолчанию;
- POST-DATABASE-COMMIT – это событие возникает после фиксации изменений, позволяя вам наращивать обработку по умолчанию;
- POST-DELETE – это событие возникает после удаления строки из базы данных;
- POST-INSERT – это событие возникает после вставки строки в базу данных;
- POST-UPDATE – это событие возникает после обновления строки в базе данных;
- PRE-COMMIT – это событие возникает, когда Oracle Forms определяет, что в форме есть изменения для посылки или фиксации, иначе говоря – перед процессом Post и Commit Transactions;
- PRE-DELETE – это событие возникает до удаления записи из базы данных, позволяя вам выполнять различного виды проверки во время процессов Post и Commit Transactions;
- PRE-INSERT – это событие возникает до вставки записи в базу данных, позволяя вам выполнять различного вида проверки во время процессов Post и Commit Transactions;
- PRE-UPDATE – это событие возникает до обновления записи в базе данных, позволяя вам выполнять различного виды проверки во время процессов Post и Commit Transactions.

В качестве примера рассмотрим триггер PRE-INSERT, с помощью которого создадим аналог автоинкремента, то есть автоматического генератора первичных ключей для таблицы.

PRE-INSERT

```
Select seq.nextval INTO :block_name.prkey_item from dual;
```

Триггеры проверки допустимости

Триггеры проверки допустимости – это триггеры, которые срабатывают в момент, когда Oracle Forms проверяет допустимость данных в элементе или зап55иси. Проверка допустимости данных осуществляется во время навигации, которая может быть вызвана действиями оператора или программным управлением. Срабатывание триггеров проверки допустимости может также инициировать обработка по умолчанию, которая осуществляется Oracle Forms при выполнении какого-либо действия – например, выполнение команды сохранения commit (commit_form) инициирует обработку по умолчанию.

WHEN-VALIDATE-ITEM – это событие возникает во время обработки проверки допустимости, которую Oracle Forms выполняет по умолчанию. Вы можете использовать этот триггер, чтобы нарастить стандартную проверку допустимости для элемента дополнительными условиями и исключениями.

WHEN-VALIDATE-RECORD – это событие возникает во время обработки проверки допустимости, которую Oracle Forms выполняет по умолчанию. Вы можете использовать этот триггер, чтобы нарастить стандартную проверку допустимости для записи дополнительными условиями и исключениями.

В качестве примера рассмотрим триггер WHEN-VALIDATE-ITEM, с помощью которого выполним проверку элемента на введенные значения. Если значение элемента будет превышать 1000, то программа вернет фокус ввода обратно в элемент и отчистит его.

WHEN-VALIDATE-ITEM

```
IF :item_name>1000 THEN
Message('Значение не должно превышать 1000');
Go_item('item_name');
:item_name:=null;
END IF;
```

Компиляция триггеров

Когда вы генерируете (компилируете) модуль, автоматически компилируются и все триггеры, принадлежащие этому модулю. Триггер автоматически компилируется при запуске формы, если включен параметр

«Generate before run». Также вы можете скомпилировать все триггеры командой главного меню Program | Compile All.

Создание пользовательских триггеров

В Oracle Forms помимо триггеров событий существуют пользовательские триггеры, которые не связаны с какими-либо событиями.

Пользовательский триггер – это триггер, который не связан ни с одним событием Oracle Forms и имеет уникальное имя, данное ему разработчиком. Вы можете создавать пользовательские триггеры на уроне формы, блока и элемента.

Чтобы создать пользовательский триггер, выполните следующие действия:

1. Находясь в навигаторе объектов, выберите узел «Триггеры» на уровне формы и нажмите иконку «Создать» или выберите пункт меню Edit | Create для запуска окна триггеров (рис. 12.3).

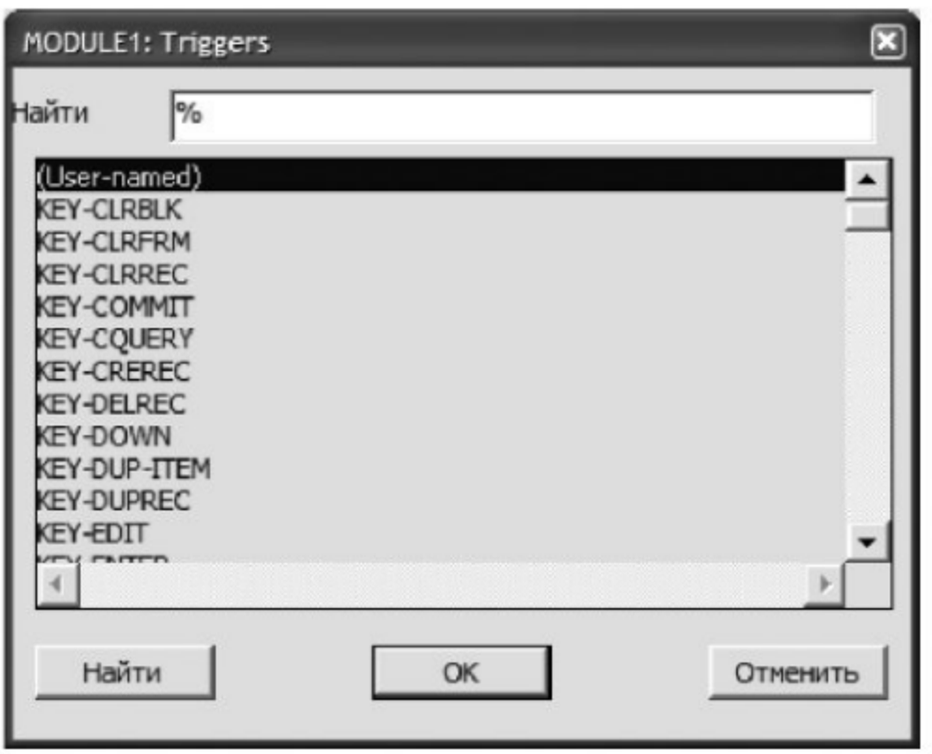

Рис. 12.3. Окно «Триггеры»

2. Выберите из списка надпись «(User_Named)», она указана самой первой в списке, и подтвердите выбор.
3. После создания триггер появляется в ветке «Триггеры» с именем, сгенерированным Oracle Forms. Вызовите для триггера палитру свойств и переименуйте его в нужное вам имя.

Поскольку триггер не отвечает ни на одно событие Forms, его срабатывание необходимо инициировать. Для вызова пользовательского триггера существует встроенная процедура EXECUTE_TRIGGER:

```
EXECUTE_TRIGGER('my_trigger');
```

Пользовательские триггеры имеют точно такие же свойства, как и триггеры событий. По большому счету особой необходимости в них нет, так как, создав хранимую программу (функцию, процедуру), вы можете так же вызвать ее из любого элемента меню, триггера или программы и получить ту же самую функциональность без дополнительного вызова.

Лекция 13. Группы записей (Record Groups)

В этой лекции слушатель ознакомится с понятием «Группа записей», научится создавать различные виды групп записей. Также будут рассмотрены функции и процедуры для программного управления этой структурой.

Ключевые слова: группа записей, Record Group, создание группы записей, заполнение группы записей, удаление группы записей.

Цель лекции: научить слушателя создавать группы записей и ознакомить со стандартными встроенными подпрограммами, предназначенными для работы с Record Group.

Разобьем изучение этой главы на несколько частей, т. к. этот материал довольно объемный и требует внимательного изучения, поскольку используется более чем с одним объектом.

- Описание и свойства групп записей.
- Создание групп записей.
- Примеры использования.

Описание, назначения и свойства

Группы записей – это структурированные наборы данных, которые могут использоваться для передачи данных между модулями прикладных программ либо для заполнения списков значений или других элементов списка. Группа записей подобна как таблицам, так и представлениям базы данных. Как и таблица, группа записей содержит столбцы и строки, которые принадлежат той рабочей области, в которой определены. Так же, как и представление, группа записей создается на основе запроса, но является более гибкой, и не только по той причине, что группа записей для Oracle Forms локальна, но и потому что запрос, на основе которого создается группа записей, можно конструировать динамически, во время выполнения программы.

Группы записей можно разделить на два основных типа, которые вы можете определить в таблице свойств этого объекта:

- **Запросные** – это группа записей, имеющая связанный с ней оператор выборки – SELECT. При создании группы записей Forms дает столбцам этой группы имена по умолчанию; все остальные данные, такие как размерность столбца и тип данных, Forms наименует исходя из тех данных, которые вы указали в SELECT. Записи для такой группы извлекаются тем же запросом, что и при построении группы записей.

Колонки запросной группы записей берут свои имена по умолчанию, типы данных и размерность – из колонок базы данных, на которые делаются ссылки в операторе SELECT.

- **Статические** – это группа записей, не связанная с запросом, поэтому структуру такой группы вы определяете во время проектирования, и, следовательно, во время выполнения формы ее значения остаются фиксированными, т. к. вы их задаете заранее при построении группы.

Основные функциональные свойства группы записей:

Тип запроса – здесь вы можете выбрать тип запроса «Статический» или «Запрос». Их различия описаны выше. В зависимости от выбранного типа группа записей принимает разные функциональные свойства.

Если выбран тип «Запрос»:

- **Запрос группы записей** – в этом свойстве вы пишете запрос, которым будете выбирать записи в группу и создавать группу;
- **Размер выборки группы записей** – это число строк, выбираемое Запросом группы записей;
- **Спецификация столбца** – при выборе этого свойства запускается окно «STUDY: Спецификация столбца», в котором вы можете указать имя столбца, длину и тип.

Если выбран тип «Статический»:

- Спецификация столбца – при выборе этого свойства запускается точно такое же окно «STUDY: Спецификация столбца», как и в предыдущем случае, с той лишь разницей, что помимо полей, в которых вы можете указать имя столбца, длину и тип, вы также можете указать и значение столбцов (Column Values).

Применение групп записей

Группы записей являются неотъемлемой частью списков значений LOV, так как LOV связан с определенной группой записей. Когда в мастере создания LOV – LOV Wizard – вы выбираете столбцы из одной или более таблицы для их отображения в LOV, вы на самом деле создаете группу записей, которая после завершения работы мастера появится в соответствующем узле навигатора. Если же вы создаете LOV вручную, то вам придется самим создавать группу записей, а затем присоединять ее к LOV. Подсоединение группы запросов к LOV может осуществляться как статически, так и динамически, во время выполнения программы.

Помимо LOV группы записей использует объект TREE VIEW – в этом объекте группа записей является одним из двух вариантов заполне-

ния этого элемента. Создав группу записей, вы можете ее присоединить к TREE VIEW через его палитру свойств. Однако запросная группа для этого элемента строится не через обычный запрос, а через иерархический запрос. Создание групп записей для этого объекта мы рассмотрим в разделе, посвященном объекту TREE VIEW.

С помощью специального набора процедур и функций для работы с группами записей можно не только менять структуру группы в ходе выполнения программы, но и переопределять группу записей как для LOV, так и для TREE VIEW. Для динамического подключения групп записей к LOV и TREE VIEW используются две процедуры:

- *SET_LOV_PROPERTY* – содержит аргумент для установки группы записей в LOV;
- *SET_TREE_PROPERTY* – содержит аргумент для установки группы записей в TREE VIEW.

Строки групп записей внутренне пронумерованы последовательно начиная с 1, то есть 1, 2, 3 и т. д., поэтому если вам будет необходимо обратиться к каким-либо определенным строкам, то достаточно указать номер этой строки.

Создание группы записей

Поскольку существует два типа запросных групп —*статическая* и з*апросная*, этапы создания статической и запросной групп рассмотрим отдельно.

Для создания запросной группы выполните следующие шаги:

1. Перейдите в навигатор и выберите узел «группы записей». В меню навигатора выберите Navigator->Create.
2. В появившемся окне New Record Group выберите тип запроса Based on the Query below.
3. В поле Query Text введите ваш запрос. В запросе вы можете без проблем использовать как параметры формы, так и глобальные переменные. После окончания ввода запроса нажмите кнопку «OK», причем ввод символа «;» в конце запроса необязателен. После этого Forms проверит ваш запрос на наличие ошибок и закроет окно, если таковые отсутствуют.

Созданная вами группа пока еще не имеет строк, поэтому для так называемого заполнения вам необходимо программно исполнить запрос с помощью процедуры POPULATE_GROUP*. Исполняемый запрос, естественно, должен быть связан с запросной группой.

* Если на запросной группе записей основывается LOV, то Oracle Forms исполняет запрос, связанный с этой группой записей, при каждом вызове LOV либо при проверке LOV Validation.

После создания Запросной группы записей перейдите в палитру свойств группы записей и выберите свойство «Спецификация столбца» (Column Specification). При выборе этого свойства запускается окно Query Record Group Column Specification, в котором вы увидите выбранные вами столбцы (Column Names), их тип (Data Type) и размер (Length).

При выборе определенной колонки отображаются соответствующие ей структуры (тип, размер). Если вы удалите какой-нибудь столбец из Column Names, ваш запрос абсолютно не пострадает, а вот изменение запроса, будь то удаление, добавление или изменение запрашиваемого столбца в запросе, приведет и к его изменению в Query Record Group Column Specification.

> ***Примечание.*** *При изменении группы запросов не забудьте сделать соответствующие преобразование в LOV, потому что в этом случае вам придется обновить определение этого LOV.*

Для создания статической группы выполните следующие шаги:

1. Перейдите в навигатор объектов и выберите узел «Группы записей». В меню навигатора выберите Navigator->Create.
2. В появившемся окне New Record Group выберите тип запроса Static Values и нажмите «OK».
3. В появившемся окне «Спецификация столбца» (Column Specification) для каждого столбца выполните следующее:
   - в списке Column Name введите имена колонок для группы записей в нужном порядке;
   - установите тип данных и длину каждой колонки, причем длины необходимо указывать только для данных типа VARCHAR и LONG, для всех остальных типов Oracle автоматически расставляет значения.

Введите значение для каждой ячейки в группе записей, выбирая нужную колонку в списке Column Name и затем вводя нужные значения в списке Column Value. Очень важно правильно указать размер, т. к. в противном случае такие данные будут недоступны.

Что касается случая, когда запрос в запросной группе не связан с таковой, то такая группа называется «**Незапросной группой**». Поскольку у незапросной группы нет связанного с ней запроса, в отличие от запросной группы, для заполнения которой мы программно исполняем запрос, в случае с незапросной группой вы должны программно устанавливать значения столбцов.

Функции и процедуры для работы с группами записей

В Oracle Forms существует достаточно много процедур и функций для работы с группами записей, которые дают возможность делать следующее:

- изменять запрос, связанный с запросной группой, при этом добавляя новые столбцы и строки в структуру группы;
- заполнять и очищать группу;
- программно создавать группу;
- управлять размером массива извлекаемых данных;
- добавлять и удалять строки;
- устанавливать и получать значения колонок;
- помечать и отменять пометки строк как «выбранные»;
- получать и устанавливать свойства группы.

Разобьем все процедуры и функции на несколько категорий, определяющих их назначение.

Создание и удаление групп:

- **CREATE_GROUP** (RECORDGROUP_NAME IN VARCHAR2, SCOPE IN NUMBER, ARRAY_SIZE IN NUMBER) RETURN FORMS4C.RECORDGROUP;
- **CREATE_GROUP_FROM_QUERY** (RECORDGROUP_NAME IN VARCHAR2, QUERY IN VARCHAR2, SCOPE IN NUMBER, ARRAY_SIZE IN NUMBER) RETURN FORMS4C.RECORDGROUP;
- **DELETE_GROUP** (RECORDGROUP_NAME IN VARCHAR2).

Изменение структуры группы – эти подпрограммы неприменимы к статическим запросным группам:

- **ADD_GROUP_COLUMN** (RECORDGROUP_NAME IN VARCHAR2, GROUPCOLUMN_NAME IN VARCHAR2, COLUMN_TYPE IN NUMBER) RETURN FORMS4C.GROUPCOLUMN;
- **ADD_GROUP_ROW** (RECORDGROUP_NAME IN VARCHAR2, ROW_NUMBER IN NUMBER);
- **DELETE_GROUP_ROW** (RECORDGROUP_NAME IN VARCHAR2, ROW_NUMBER IN NUMBER).

Заполнение групп:

- **GET_GROUP_SELECTION_COUNT** (RECORDGROUP_ID IN FORMS4C.RECORDGROUP) RETURN NUMBER;
- **POPULATE_GROUP_WITH_QUERY** (RECORDGROUP_ID IN FORMS4C.RECORDGROUP, QUERY IN VARCHAR2) RETURN NUMBER;

- **SET_GROUP_CHAR_CELL** (GROUPCOLUMN_ID IN FORMS4C. GROUPCOLUMN, ROW_NUMBER IN NUMBER, CELL_VALUE IN VARCHAR2);
- **SET_GROUP_DATE_CELL** (GROUPCOLUMN_NAME IN VARCHAR2, ROW_NUMBER IN NUMBER, CELL_VALUE IN DATE);
- **SET_GROUP_NUMBER_CELL** (GROUPCOLUMN_NAME IN VARCHAR2, ROW_NUMBER IN NUMBER, CELL_VALUE IN NUMBER).

Получение значений ячеек:

- **GET_GROUP_CHAR_CELL** (GROUPCOLUMN_NAME IN VARCHAR2, ROW_NUMBER IN NUMBER) RETURN VARCHAR2;
- **GET_GROUP_DATE_CELL** (GROUPCOLUMN_ID IN FORMS4C. GROUPCOLUMN, ROW_NUMBER IN NUMBER) RETURN DATE;
- **GET_GROUP_NUMBER_CELL** (GROUPCOLUMN_NAME IN VARCHAR2, ROW_NUMBER IN NUMBER) RETURN NUMBER.

Обработка строк:

- **GET_GROUP_ROW_COUNT** (RECORDGROUP_ID IN FORMS4C. RECORDGROUP) RETURN NUMBER;
- **GET_GROUP_SELECTION** (RECORDGROUP_NAME IN VARCHAR2, SELECTION_NUMBER IN NUMBER) RETURN NUMBER;
- **GET_GROUP_SELECTION_COUNT** (RECORDGROUP_ID IN FORMS4C.RECORDGROUP) RETURN NUMBER;
- **RESET_GROUP_SELECTION** (RECORDGROUP_NAME IN VARCHAR2);
- **SET_GROUP_SELECTION** (RECORDGROUP_ID IN FORMS4C. RECORDGROUP, ROW_NUMBER IN NUMBER);
- UNSET_GROUP_SELECTION (RECORDGROUP_ID IN FORMS4C. RECORDGROUP, ROW_NUMBER IN NUMBER).

Функции идентификаторов объектов:

- **FIND_GROUP** (RECORDGROUP_NAME IN VARCHAR2) RETURN FORMS4C.RECORDGROUP;
- **FIND_COLUMN** (GROUPCOLUMN_NAME IN VARCHAR2) RETURN FORMS4C.GROUPCOLUMN.

Примечание. *Учтите, что программно можно создавать и модифицировать только запросные и незапросные группы, в отличие от статических групп. Процедуры и функции, используемые для обработки как запросной, так и незапросной группой, не всегда одинаковы.*

Создание групп записей программно

Перед тем как перейти к рассмотрению этой темы и проработке представленных примеров, выполните следующие два шага:

1. Создайте таблицу ABOUT в SQL*Plus или любой другой утилите:

```
SQL> create table ABOUT (
  2  Name varchar2(10),
  3  FName varchar2(12),
  4  SName varchar2(12));
```

2. После создания таблицы перейдите в Forms и с помощью Data Block Wizard создайте блок данных, выбрав в качестве структуры таблицу ABOUT.

Мы с вами уже детально ознакомились с созданием групп записей в режиме проектирования. Теперь перейдем к следующему этапу – созданию групп записей программно. Рассмотрим два примера: создание запросной и незапросной группы.

Создание запросной группы:

- Откройте созданный вами модуль ABOUT.fmb.
- Откройте редактор разметки и создайте в нем следующие элементы:
  - Button1 – в палитре свойств этого элемента установите свойство «Метка» в «Создать запросную группу»;
  - Button2 – в палитре свойств этого элемента установите свойство «Метка» в «Создать незапросную группу»;
  - Button3 – в палитре свойств этого элемента установите свойство «Метка» в «Добавить колонку»;
  - Button4 – в палитре свойств этого элемента установите свойство «Метка» в «Добавить строку»;
  - Button5 – в палитре свойств этого элемента установите свойство «Метка» в «Удалить строку»;
  - Button6 и Button7 – «Метка» этих двух кнопок установите в «>>»;
  - Button8 – в палитре свойств этого элемента установите свойство «Метка» в «Удалить группу»;
  - Text-item1 (элемент текста) – в палитре свойств этого объекта измените 2 свойства: «Имя» – на ZAP_GR, а «Элемент базы данных» – на «НЕТ»;
  - Text-item2 – в палитре свойств этого объекта измените 2 свойства: «Имя» – на NO_ZAP_GR, а «Элемент базы данных» – на «НЕТ».

Если хотите разместить элементы на канве самостоятельно, то в свойстве «Обновить разметку» CANVAS (вид-картинки) выберите пункт «Вручную» и разместите элементы на канве в удобном для вас порядке.

- Для кнопки «Создать запросную группу» создайте триггер WHEN-BUTTON_PRESSED:

```
/*WHEN-BUTTON_PRESSED*/
DECLARE
group_id RecordGroup;
query_ok NUMBER;
row_count number;
BEGIN
/* создаем группу prod_group и присваиваем ее идентификатор
** переменной group_id  */
group_id := Create_Group_From_Query('about_group',
'SELECT Name,FName, SName
FROM ABOUT
WHERE rownum<10');
/* теперь исполним запрос новой группы, используя переменную
** group_id для идентификации группы  */
query_ok := Populate_Group(group_id);

/* если запрос неудачный, то прерываем этот триггер вызовом
** предварительно определенного исключения  */
IF query_ok <> 0 THEN
RAISE Form_Trigger_Failure;
END IF;
IF query_ok = 0 THEN
message('SQL maked succesfull');
END IF;
END;
```

- Для кнопки «Создать незапросную группу» создайте триггер WHEN-BUTTON_PRESSED:

```
/*WHEN-BUTTON_PRESSED*/
DECLARE
group_id    RecordGroup;
col1_id     GroupColumn;
col2_id     GroupColumn;
col3_id     GroupColumn;
```

```
query_ok number;
total_rows  NUMBER;
new_row     NUMBER;
BEGIN
/* Создаем незапросную группу с именем my_group и присваиваем ее
** идентификатор переменной group_id.
*/
group_id := Create_Group('about_no_query_group');
/* Добавляем в новую группу три колонки, используя для идентификации
** группы переменную group_id. Первые две колонки типа CHAR_COLUMN и
** их длина должна определяться.
** Третья типа NUMBER_COLUMN и параметр длины не принимает
*/
col1_id := Add_Group_Column(group_id,'col1_id',CHAR_COLUMN,50);
col2_id := Add_Group_Column(group_id,'col2_id',CHAR_COLUMN,50);
col3_id := Add_Group_Column(group_id,'col3_id',NUMBER_COLUMN);
query_ok := Populate_Group(group_id);
/* если запрос неудачный, то прерываем этот триггер вызовом
** предварительно определенного исключения  */
END;
```

- Для кнопки «Добавить колонку» создайте триггер WHEN-BUTTON_PRESSED:

```
/*WHEN-BUTTON_PRESSED*/
Add_Group_Column('about_no_query_group','N_AME',CHAR_COLUMN);
Set_Group_Char_Cell('about_no_query_group .N_AME',2,'Nik');
```

В первом параметре процедуры Add_Group_Column вы указываете группу, в которую собираетесь добавить колонку, во втором параметре вы указываете имя новой колонки, а в третьем указываете тип, создаваемой колонки: CHAR_COLUMN, DATE_COLUMN, NUMBER_COLUMN.

Примечание. *Новую колонку можно добавить только в группу, не содержащую строк.*

- Для кнопки «Добавить строку» создайте триггер WHEN-BUTTON_PRESSED:

```
/*WHEN-BUTTON_PRESSED*/
Add_Group_Row('about_group',2);
```

Первый параметр about_group указывает на имя группы, в которую мы хотим добавить запись, а второй параметр – это номер строки, в которую мы хотим вставить запись. Для добавления новой строки в группу с уже имеющимся набором строк используйте константу **END_OF_GROUP**, например:

```
Add_Group_Row('about_group', END_OF_GROUP);
```

Примечание. *В новой строке значения колонок по умолчанию определяются как NULL. Поэтому после добавления новой строки используйте процедуру SET_GROUP_CHAR (NUMBER, DATE)_CELL для установки значений новой строки.*

- Для кнопки «Удалить строку» создайте триггер WHEN-BUTTON_PRESSED:

```
/*WHEN-BUTTON_PRESSED*/
Delete_Group_Row('about_group',1);
```

Первый параметр about_group указывает на имя группы, в которой мы хотим произвести удаление. Второй параметр указывает на номер удаляемой строки; вы также можете использовать в качестве параметра константу **ALL_ROWS**, которая выполняет удаление всех строк группы.

- Для кнопки «(Q)->>» (подсчет строк запросной группы) создайте триггер WHEN-BUTTON_PRESSED:

```
/*WHEN-BUTTON_PRESSED*/
:ABOUT.zap_gr:=Get_Group_Row_Count('about_group');
```

В функции Get_Group_Row_Count вам достаточно указать имя группы или ее ID для нахождения количества строк в группе.

- Для кнопки «(NoQ)->>» (подсчет строк незапросной группы) создайте триггер WHEN-BUTTON_PRESSED:

```
/*WHEN-BUTTON_PRESSED*/
:ABOUT.no_zap_gr:=Get_Group_Row_Count('about_no_query_group');
```

- Для кнопки «Удалить группу» создайте триггер WHEN-BUTTON_PRESSED:

```
/*WHEN-BUTTON_PRESSED*/
DELETE_GROUP ('about_no_query_group');
DELETE_GROUP ('about_group');
```

Примечание. *Удалять группы записей программно, то есть с помощью вышеупомянутой функции, можно, но лишь те группы записей, которые создавались программно, а не во время проектирования.*

Теперь запустите форму и попробуйте создать запросную и незапросную группу, посчитать количество строк в группе до и после удаления или вставки. Для ответа на вопрос «Создана ли группа или удалена?» можно выполнить такую проверку:

- создать, к примеру, запросную группу одноименной кнопкой;
- затем еще раз нажать на эту кнопку, пытаясь создать такую же группу. После повторной попытки Oracle Forms выведет вам следующее сообщение: FRM – 41072 «Невозможно создать группу about_group» – это означает, что группа с таким именем уже существует. После этой проверки мы знаем, что группа создана;
- теперь нажмем кнопку DELETE_GROUP, то есть удалим созданную нами группу. Теперь, нажав кнопку «Создать запросную группу», мы получим сообщение о создании группы, следовательно, группа действительно была удалена.

Пометка строк в группе записей

Программно вы можете не только считать количество строк, устанавливать новые или изменять предыдущие значения, но и выделять (помечать) из существующей группы интересующий вас набор данных, удовлетворяющий, к примеру, какому-то одному значению. Функции и процедуры, связанные с пометкой, имеют окончание «_ SELECTION». Опишем назначение каждой из них более подробно.

- SET_GROUP_SELECTION (процедура) – для пометки строки как выбранной;
- GET_GROUP_SELECTION_COUNT (функция) – для определения количества выбранных строк в группе;
- GET_GROUP_SELECTION (функция) – для получения номера строки, помеченной как выбранная;
- UNSET_GROUP_SELECTION (функция) – для отмены пометки выбранной строки;
- RESET_GROUP_SELECTION (процедура) – для отмены выбора всех выбранных в текущий момент строк в группе.

Теперь рассмотрим простой пример: «Пометить как выбранные все строки, в которых встречается имя Вова». Для этого выполните следующие шаги:

1. Откройте форму ABOUT и создайте кнопку с меткой «Пометить».
2. Создайте триггер WHEN_BUTTON_PRESSED:

```
/*WHEN-BUTTON_PRESSED*/
DECLARE
C_OUNT  NUMBER;
count_names NUMBER;
group_id  RecordGroup;
BEGIN

group_id:= FIND_GROUP('about_group');

C_OUNT := Get_Group_Row_Count(group_id);

FOR i IN 1..C_OUNT LOOP
IF Get_Group_Char_Cell('ABOUT.Name',i) =
'Вова' THEN
Set_Group_Selection(group_id,i);
END IF;
END LOOP;
count_names := Get_Group_Selection_Count(group_id);
:ABOUT.ZAP_GR:=TO_CHAR (count_names);
END;
```

3. Запустите форму на выполнение. Нажмите кнопку «Пометить», после чего у вас в поле «Количество строк в запросной группе» появится значение, равное количеству помеченных строк в запросной группе с именем Вова. Но теперь учтите, что функция GET_GROUP_SELECTION будет вам возвращать значение из набора, сформированного процедурой SET_GROUP_SELECTION, причем номера строк в новом наборе будут расставлены заново начиная с единицы. Поэтому если до этого выбранные вами строки, к примеру, имели номера 2, 7, 10, 15, ..., то после формирования нового набора 2 будет иметь порядковый номер 1, 7 – порядковый номер 2, 10 – порядковый номер 3 и так далее:

```
Get_Group_Selection('about_group',2);
```

Здесь about_group – имя группы, а 2 – номер помеченной строки.

Лекция 14. Списки значений (LOV)

В этой лекции слушатель ознакомится с понятием «Список значений», научится создавать списки значений, управлять их свойствами и поведением в форме.

Ключевые слова: списки значений, List of Values, LOV, создание LOV, удаление LOV, заполнение LOV, динамические LOV, статические LOV.

Цель лекции: научить слушателя создавать списки значений, управлять их свойствами и ознакомить со встроенными подпрограммами, предназначенными для обработки LOV.

LOV (List Of Values – список значений) – список значений представляет данные, содержащиеся в именованном объекте, и предназначенные для выбора возможных значений из некоторого поля. LOV выводит на дисплей диалоговое окно с возможными данными на основе запроса или групп записей. Список значений LOV можно задать двумя способами:

- вручную;
- с помощью LOV Wizard.

В LOV список возможных значений может определяться не только одним столбцом данных, но и несколькими столбцами, что очень удобно и часто используется в моделях типа «Главный-подчиненный» (master-detail), а также позволяет работать с уже сгруппированными и предварительно отсортированными данными. Также списки позволяют оперативно и быстро вводить данные, т. к. свойства автоуменьшения и поиска по LOV позволяют пользователям быстро находить конкретные значения – для этого вам достаточно ввести хотя бы один символ искомого значения; по умолчанию поле поиска содержит символ «%», что означает «вывести все элементы, соответствующие запросу LOV».

Основные свойства LOV в таблице свойств

Перечислим здесь только основные свойства LOV, так как остальные свойства типа General или Font имеют такие же значения, как и у других объектов.

1. Функциональные (Functional):
   - **Record Group** – здесь вы указываете имя Record Group (группы записей), с которой будет работать ваш LOV. При выборе этого значения,

если у вас больше одной Record Group, поле значения будет иметь вид выпадающего списка.

- **Column Mapping Properties** – при выборе этого свойства запускается окно LOV Column Mapping, в котором вы можете задать такие свойства, как Column Title (синоним столбца), Column Names (имена отображаемых столбцов), Return item (возвращаемый элемент), Display Width (указать ширину столбца). Для выбора значения поля Return item нажмите кнопку «Browse», после чего появится окно с возможными вариантами.
- **Фильтровать перед отображением** – это свойство отвечает за вывод диалога критериев запроса перед выводом LOV, где «Yes» означает, что диалог с критерием запроса будет выведен до списка возможных значений. После того как вы выбрали критерий запроса, нажмите кнопку «Найти».
- **Automatic Display** (автоматическое отображение) – имеет всего два значения, «Yes» и «No», отвечающих за автоматическое отображение значений LOV на дисплее, то есть если вы установите значение этого свойства на «Yes», то при навигации к элементу, к которому оно подсоединено, будет автоматически отображаться окно LOV.
- **Automatic Refresh** (автоматическое обновление) – имеет два значения, «Yes» и «No», отвечающих за автоматическое обновления дисплея LOV. «Yes» означает «применить автоматическое обновление» – выполнить запрос при заполнении LOV.
- **Automatic Select** (автоматический выбор) – имеет два значения, «Yes» и «No», где «Yes» говорит о том, что при сокращении элементов в LOV до одного это значение автоматически выберется в поле.
- **Automatic Skip** (автоматический пропуск) – в этом поле вы указываете, следует ли перемещать курсор на следующий элемент при заполнении последнего символа в текущем элементе.
- **Automatic Position** (автоматическая позиция) – имеет два значения, «Yes» и «No», где «Yes» означает, что LOV будет автоматически располагаться возле того поля, из которого был вызван.
- **Automatic Column Width** (автоматическая ширина столбца) – автоматическое определение ширины колонок. Это свойство в режиме «Yes» разрешает Forms самому определить ширину колонки, что очень удобно, т. к. дает возможность не заполнять эти параметры вручную.

2. Физические (Physical):
   - **X Position** – указание координаты размещения LOV по оси Ox;
   - **Y Position** – указание координаты размещения LOV по оси Oy;
   - **Height** – в этом поле указывается высота диалогового окна LOV;
   - **Width** – в этом поле указывается ширина диалогового окна LOV.

Вот, в принципе, и все основные свойства для работы LOV, которые вам необходимо знать.

Приступим к созданию LOV. Создавать LOV будем с помощью LOV Wizard в 5 шагов:

1. **Специфицируем отображаемые LOV значения**. Когда вы создаете новый LOV, вы создаете SQL-запрос, который возвращает данные для отображения списка значений LOV. Этот запрос создает новую рабочую группу автоматически, если вы пишете его с помощью LOV Wizard. Если же вы при создании LOV выбираете опцию «Build a new LOV manually», то вы должны помнить, что значения LOV происходят от внутренней структуры данных, называемой Record Group. Поэтому когда вы создаете LOV, вы связываете его с именной группой записей.
2. **Формат отображения LOV**. В LOV элементы отображаются в виде таблицы, и в этом шаге вы выбираете, модифицируете названия поля (столбца), ширину, длину, количество отображаемых столбцов, определяете поля, отображаемые в LOV, и указываете элемент формы, в который будет возвращено значение.
3. **Привязка LOV к текстовому полю**. Вы должны ассоциировать ваш LOV с соответствующим текстовым элементом формы. Это нужно для того, чтобы при входе курсора в элемент или при нажатии какой-либо кнопки автоматически открывалось диалоговое окно LOV.
4. **Переименование**. При создании LOV мы пользовались мастером создания списков значений LOV Wizard, который при создании LOV и Record Group автоматически присваивает имя, состоящее из имени объекта и порядкового номера, поэтому лучше вам изменить имена по умолчанию на более понятные, что придаст вашему приложению большую читабельность.
5. Создать вызов LOV в приложении с помощью кнопки либо в ответ на любое другое событие.

Теперь выполним следующее упражнение:

1. Откройте ранее созданную нами форму MOBILS.fmb, которую мы делали в предыдущем упражнении. Нам необходимо создать LOV для поля Models блока Mobils. Это делается для того, чтобы пользователь при вводе новой модели мог точно определить фирму для создаваемой модели, проверить, существует ли эта фирма вообще, дабы не нарушить ссылочную целостность, организовать корректный ввод значения, определить для пользователя формат вводимых данных. Также списком значений очень удобно пользоваться в режиме ввода запроса, т. к. вы можете легко просмотреть все возможные критерии выборки.
2. Для запуска LOV выберите *Tools->LOV Wizard* либо в Навигаторе – узел LOV, затем выполните команду Create в меню Навигатора или LOV Wizard – во всплывающем меню.

3. После запуска LOV Wizard вы увидите окно (рис. 14.1).

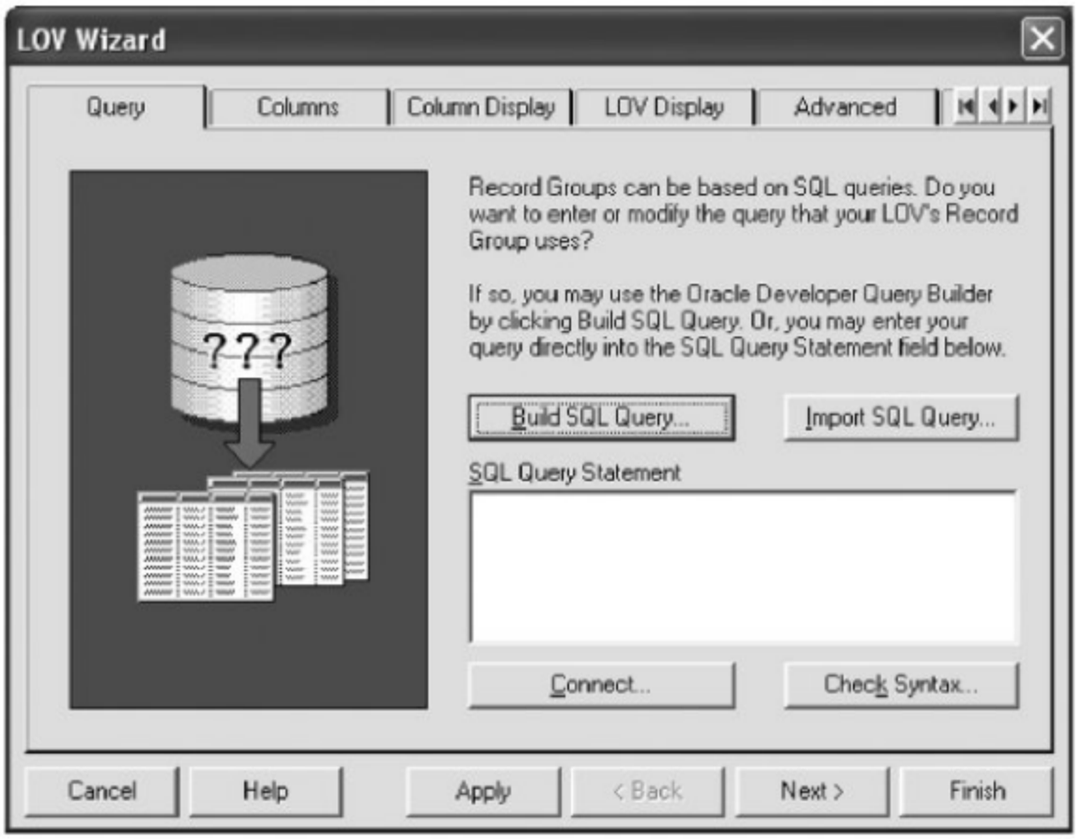

Рис. 14.1. LOV Wizard

Здесь вам предлагается два способа построения запроса:

- **Build SQL Query** – построить запрос с помощью мастера запросов. После нажатия этой кнопки запустится окно, где вам будут предложены следующие возможности: выбрать одну или более таблиц, связать таблицы (если вы выбрали более одной таблицы) и выбрать нужные вам столбцы, пометив нужный вам столбец галочкой.
- **Import SQL Query** – предлагает вам выбрать из вашей файловой системы ранее созданный вами запрос.

SQL Query Statement – это поле «прямого ввода», в этом поле вы можете сами написать нужный вам запрос. При этом в конце запроса не нужно ставить знак «;» (точка с запятой), т. к. Forms сразу же выдаст вам сообщение об ошибке.

Кнопки, расположенные под редактором SQL Query Statement, выполняют следующие действия:

- **Connect** – соединяет пользователя с БД, запуская окно соединения;
- **Check Syntax** – проверяет введенный вами запрос на ошибки.

В нашем случае в поле SQL Query Statement введем запрос:

```
Select MOBILS.MODEL, Firm.Firm_Name From MOBILS m, Firm f where
f.Firm_ID=m.Firm_Fk
```

Затем для подтверждения нажмите кнопку *Apply*, **после чего Forms сделает неявную проверку Check Syntax введенного вами SQL-предложения**, и если проверка завершится успешно, Forms, пока невидимо для вас, создаст Группу записей (**Record Group**) на основе вашего запроса (**SQL Query Statement**).

4. Затем появится следующее окно, в котором будут отображены выбранные вами столбцы. Нажмите кнопку с иконкой «>>», чтобы переместить сразу все столбцы, и нажмите кнопку *Next*.
5. В следующем окне – окно форматирования LOV Display – вам будет предложено задать размеры (*Width*) колонок в дисплее LOV, а также возвращаемое значение Return Value Column, определяющее текстовое поле формы, с которым будет связан наш LOV. Для определения этого значения будет выведено окошко

К примеру, когда пользователь переведет фокус в поле, связанное с LOV, будет выведено диалоговое окно LOV, в котором будут перечислены возможные значения этого поля. После выбора одного из возможных значений поле, с которым вы работаете, примет выбранное вами значение из списка. Чтобы создать связку, выполните следующие действия:

- Выберите в окне поле, которое хотите связать, затем нажмите кнопку Look up return item и в появившемся окне выберите:
  - **Models** – выберем элемент MOBILS.MODEL;
  - **Firm** – элемент FIRM.Firm_Name.
- После этого при вводе курсора в поле Model будет открываться список значений с данными из элементов MOBILS.MODEL и FIRM. Firm_Name.

Теперь создадим кнопку, по нажатию которой будет вызываться наш LOV, и напишем обработчик для вызова списка значений.

1. Создайте в блоке MOBILS кнопку, с помощью навигатора или нажав соответствующую кнопку на панели инструментов Редактора разметки (Layout Editor).
2. Создав кнопку, перейдите в навигатор, выделите созданную вами кнопку и нажмите правую кнопку мыши. Во всплывающем меню выберите пункт **Smart Triggers** (специальные триггеры и наиболее часто используемые)-> WHEN_BUTTON_PRESSED.
3. В запустившемся PL/SQL-редакторе наберите и скомпилируйте следующий код:

```
GO_ITEM ('MOBILS.Model');          //- перейти к элементу
LIST_VALUES;                       //- вызвать список значений
```

4. После этого нажмите кнопку **Compile PL/SQL Code** для компилирования вашего триггера и выполните команду Run Form.

Результат будет следующим: после нажатия кнопки курсор перейдет в элемент MOBILS.Model и вызовет LOV.

Если же вы хотите вызывать LOV автоматически при входе в поле и при этом не писать обработчик для каждого поля, выполните следующие действия.

В объектном навигаторе выделите блок Mobils, затем вызовите всплывающий список правым щелчком мыши и выберите пункт Smart Triggers -> WHEN_NEW_ITEM_INSTANCE. Этот триггер срабатывает всякий раз, когда курсор будет входить в элемент.

Теперь в редакторе PL/SQL напишите следующий обработчик:

```
Declare
      a varchar2(20):=:system.current_item;
      orient varchar2(20):=:system.block_status;
begin
  IF ( orient = 'NEW' or orient = 'INSERT' )
  THEN
select a into :client.C_name from dual;
  GO_ITEM ( 'Client.'||a) ;
  LIST_VALUES;
  END IF;
END;
```

Мы рассмотрели два основных и наиболее часто применяемых вызова LOV. Можно посоветовать вам использовать второй вариант, так как он более гибкий за счет того, что вы можете управлять выводом LOV в зависимости от статуса блока, и за счет того, что вам не придется узнавать каждый раз имя элемента, для которого нужно вывести соответствующий список.

Лекция 15. PL/SQL в Oracle Forms. Блоки и переменные PL/SQL

В этой лекции слушатель ознакомится с декларативной средой PL/SQL и ее возможностями, научится создавать блоки и переменные PL/SQL.

Ключевые слова: операторские скобки, именованный блок, анонимный блок, вложенный блок, исключение, Exception, комментарии, косвенные ссылки, копирование.

Цель лекции: ознакомить слушателя с базовыми конструкциями и основными концепциями языка PL/SQL.

PL/SQL – это язык программирования, объединяющий в себе возможности процедурных языков и SQL. Иначе говоря, он представляет собой процедурное расширение языка SQL, отсюда и название – Procedural Language SQL. PL/SQL является родным языком Oracle, так как он интегрирован с сервером базы данных и его код выполняется непосредственно сервером, поэтому программы, написанные на PL/SQL, работают быстро и эффективно. Возможность использовать SQL в блоках PL/SQL – одна из важнейших его характеристик. PL/SQL применяется для доступа к базам данных Oracle из различных сред разработки, одной из которой является Oracle Forms. В этой главе мы ознакомимся с декларативной средой PL/SQL и ее возможностями, научимся создавать блоки и переменные PL/SQL.

Блоки PL/SQL

PL/SQL, как и любой другой процедурный язык программирования, состоит из логически связанных элементов, объединенных в программные единицы, которые называются блоками. Каждый модуль PL/SQL состоит как минимум из одного блока. Блоки PL/SQL могут содержать любое количество подблоков, то есть иметь различный уровень вложенности. Блок как структурная единица логически связанных элементов определяет область их действия, делает код читабельным и простым для понимания. В PL/SQL различают два типа блока:

- анонимные блоки;
- именованные блоки.

Анонимные блоки

Анонимные блоки – это блоки, которые не имеют имени. Анонимные блоки не могут быть вызваны другими блоками, так как у них нет имени, на которое можно ссылаться.

Триггеры Oracle Forms и Reports, которые также называются клиентскими триггерами, являются анонимными блоками PL/SQL. Триггеры базы данных и сценарии в SQL*Plus, заключенные в операторские скобки BEGIN и END, также являются анонимными блоками. Ниже приведена структура анонимного блока:

```
DECLARE
<имя переменной > <тип данных><(значение)>;

BEGIN
  < исполняемый оператор>;

EXCEPTION
  < оператор обработки исключения >;

END;
```

- DECLARE – раздел объявлений. В этом разделе идентифицируются переменные, курсоры и вложенные блоки, на которые имеются ссылки в исполняемом блоке и блоке исключений. Этот раздел необязательный.
- BEGIN-END – исполняемый раздел. В этом разделе находятся операторы, которые выполняются ядром PL/SQL во время работы вашего приложения. Этот раздел является обязательным.
- EXCEPTION – раздел исключений. В этом разделе обрабатываются различные исключения, предупреждения и ошибки. Этот раздел необязателен.

Из всех ключевых слов в представленной структуре для анонимного блока ключевые слова BEGIN и END являются обязательными, и между ними должен быть как минимум один исполняемый оператор:

```
BEGIN
Null;
END;
```

Несмотря на то что ключевые слова BEGIN и END обязательны, в Oracle Forms их можно опустить. Так, к примеру, простейший анонимный блок имеет вид:

```
BEGIN
Message ('Hello!');
END;
```

В Oracle Forms этот анонимный блок можно записать и без ключевых слов BEGIN и END:

```
Message ('Hello!');
```

Рассмотрим различные виды анонимных блоков.

1. Блок с разделом объявлений и исключений:

```
DECLARE
<имя переменной > <тип данных><(значение)>;

BEGIN
  < исполняемый оператор>;

EXCEPTION
  < оператор обработки исключения >;

END;
/
```

Пример:

```
BEGIN
  NULL;
EXCEPTION
  WHEN OTHERS THEN
    NULL;
END;
```

2. Вложенный* блок – это такой вид блока, когда один блок объявляется в исполняемом разделе другого. В большинстве случаев вложенный блок создается с целью размещения в нем раздела EXCEPTION для обработки исключений. Это очень удобно, так как в случае возникновения исключения программа не завершается с ошибкой, а продолжает функционировать, обработав ошибку. Ниже приведен пример вложенного блока:

* Вложенный блок в PL/SQL называют также дочерним блоком или подблоком. Внешний блок называют родительским.

Синтаксис:

```
BEGIN
    < исполняемый оператор >;
  BEGIN
    < исполняемый оператор >;
  EXCEPTION
    < оператор обработки исключения >;
  END;
EXCEPTION
  < оператор обработки исключения >;
END;
/
```

Практический пример:

```
 BEGIN
  NULL;
  BEGIN
    NULL;
  EXCEPTION
    WHEN OTHERS THEN
     NULL;
  END;
  NULL;
EXCEPTION
  WHEN OTHERS THEN
    NULL;
END;
/
```

3. Вложенный анонимный блок с разделом объявлений и исключений:
Синтаксис:

```
DECLARE

<имя переменной > <тип данных><(значение)>;

BEGIN
  < исполняемый оператор >;
  BEGIN
    < исполняемый оператор >;
```

```
  EXCEPTION
    < оператор обработки исключения >;
  END;
EXCEPTION
  < оператор обработки исключения >;
END;
```

Практический пример:

```
DECLARE
 x NUMBER(4);
BEGIN
  x := 1000;
  BEGIN
    x := x + 100;
  EXCEPTION
    WHEN OTHERS THEN
      x := x + 2;
  END;
  x := x + 10;
  DBMS_OUTPUT.PUT_LINE(x);
EXCEPTION
  WHEN OTHERS THEN
    x := x + 3;
END;
```

Когда вы связываете код PL/SQL с триггером или полем, пользуясь таким инструментальным средством, как Forms Builder, этот код составляет анонимный блок PL/SQL. При этом можно создать полный блок с объявлениями, исполняемыми операторами и разделом обработки исключений или же ограничиться только исполняемыми операторами.

Именованные блоки

Именованные блоки – это блоки, которые имеют имя, например, функция или процедура. Несмотря на то что анонимные блоки используются часто, каждый разработчик пытается оформить свою PL/SQL-программу как именованный блок. Преимущество именованного блока в том, что у него есть имя и на него можно ссылаться из других блоков. Если считать главным отличием между анонимным блоком и именованным отсутствие имени у первого, то тогда в Forms понятие анонимного блока очень размыто. В Oracle Forms, несмотря на то что пользовательский

триггер будет считаться анонимным блоком, его все же можно вызвать и на него можно ссылаться. Ниже приведен пример именованного блока функции и процедуры:

```
PROCEDURE [схема.] имя [(параметр [, параметр ...])]
   [AUID {DEFINER | CURRENT_USER}]
FUNCTION [схема.] имя [(параметр [, параметр ...])]
   RETURN тип_возвращаемых_данных
   [AUID {DEFINER | CURRENT_USER}]
   [DETERMINISTIC]
   [PARALLEL ENABLE ...]
   [PIPELINED]
```

Заголовок функции отличается от заголовка процедуры лишь ключевым словом RETURN.

Именованные блоки, так же как и анонимные, могут быть вложенными, причем анонимный блок может быть вложен в именованный блок. Ниже приведен пример такого блока:

```
PROCEDURE calc_totals
IS
   year_total NUMBER;
BEGIN
   year_total := 0;

   /* Начало вложенного блока */
   DECLARE
      month_total NUMBER;
   BEGIN
      month_total := year_total / 12;
   END set_month_total;
   /* Конец вложенного блока */
END;
```

В этом разделе мы ознакомились с понятием блока в PL/SQL, научились определять типы блоков и их структуру. Остается лишь сделать вывод из всего вышеперечисленного: PL/SQL предоставляет разработчику возможность писать удобочитаемый и гибкий код, подходить к написанию программы как к творческому процессу, требующему нестандартного мышления.

Переменные PL/SQL

Традиционно в программировании принято считать, что *переменная* — это именованная ячейка памяти, имя которой используется для доступа к данным. Можно сказать просто, что *переменная* – это обозначение, заменяющее какое-либо значение, будь то число, массив, строка или еще какой-нибудь тип. Переменные используются для следующих целей:

- Временное хранение данных.
- Манипуляция хранимыми значениями.
- Возможность многократного использования.
- Простота обслуживания.
- Повышение читаемости программы.

С понятием переменная чаще всего связываются такие понятия, как область видимости переменной, идентификаторы, ее тип и имя.

Типы переменных PL/SQL

PL/SQL, как и любой другой язык программирования, оперирует различными типами данных (табл. 15.1). Переменные PL/SQL:

- Скалярные (Scalar) – скалярные типы данных содержат одно значение. К скалярным типам данных относятся все основные типы данных сервера Oracle, за исключением Boolean, которые не могут быть назначены для столбцов.

Таблица 15.1. Типы данных PL/SQL

Типы данных	Описание
CHAR (size)	Строка фиксированной длины Максимум 255 символов Умолчание – 1 байт
VARCHAR2(size)	Переменная длинна записи Максимально – 2000 байт. Не допускается для применения в параметрах функций
FLOAT (p)	Числа с плавающей точкой (позиция точки не зафиксирована) Максимально FLOAT (126)
NUMBER (p, s)	Фиксированные с фиксированной позицией точки. Максимально 38 значащих цифр

DATE	Формат типа дата. Хранение в Юлианском календаре (количество дней после 1 января 4712 до нашей эры; максимальная дата 31 декабря 4712 нашей эры) Формат преобразования по умолчанию определяется переменной NLS_DATE_FORMAT или ALTER SESSION параметром

- Составные (Composite) – это составной тип данных, такой как записи, который позволяет объявить группу полей и манипулировать ей блоке PL/SQL.
- Ссылочные (Reference) – ссылочные типы данных содержат значения, называемые указателями.
- LOB (Large Objects) – позволяют хранить неструктурированные данные (картинки, текстовые файлы, видео, звук) объемом до 4 гигабайт. LOB поддерживает произвольный доступ к данным. Различают четыре типа LOB:
  - CLOB (character large object) – позволяет хранить большие блоки символьных данных;
  - BLOB (binary large objects) – служит для хранения больших бинарных объектов;
  - BFILE (binary file) – служит для хранения больших бинарных объектов за пределами Базы Данных;
  - NCLOB (national-language character large objects) – служит для хранения больших single-byte или fixed-width NCHAR Unicode данных.
  - EXPR – любое значение, которое подходит по смыслу и типу данных.

Примечание: *в PL/SQL также используются связываемые переменные (bind variable) и переменные операционной системы, но они не относятся к PL/SQL.*

Объявление переменных

Все переменные PL/SQL необходимо объявлять в декларативном разделе – DECLARE. Рассмотрим синтаксис объявления переменной:

```
Identifier [CONSTANT] datatype [NOT NULL] [:=| DEFAULT expr]
```

Identifier – имя переменной. После объявления имени переменой на нее можно ссылаться по имени из любой части программы.

CONSTANT – это переменная, значение которой нельзя изменить. Эта переменная должна быть инициализирована. Инициализация в данном контексте означает, что значение переменной при создании должно быть определенно, то есть значение этой переменной необходимо присвоить в разделе DECLARE. Инициализация переменной производится с помощью оператора присвоения (:=) или ключевого слова DEFAULT.

Datatype – определяет тип данных: скалярный, составной, ссылочный и LOB.

NOT NULL – ограничение, которое говорит о том, что значение переменой не может быть пустым. Эта переменная должна быть инициализирована.

Рассмотрим пример объявления некоторых типов переменных.

1. Создайте новую форму и сохраните под именем Var.fmb.
2. Перейдите в узел «Блоки Данных». Правым щелчком мыши вызовите всплывающее меню и выберите пункт «Редактор Разметки».
3. Разместите новую кнопку на канве. После того как вы разместите кнопку на канве, Oracle Forms автоматически создаст блок.
4. Выделите кнопку и вызовите всплывающее меню. Во всплывающем меню выберите пункт Универсальные триггеры | WHEN-BUTTON-PRESSED. В появившемся PL/SQL-редакторе наберите и скомпилируйте нижеприведенный код:

```
DECLARE
Height NUMBER (5, 2) := 172.43;
Age NUMBER:= 21;
Weight CONSTANT NUMBER := 74.5;
Name VARCHAR2 (10) := 'Сергей';
S_name CHAR (20) := NULL;
F_name VARCHAR2 (20) NOT NULL:= 'Васильевич';
Man BOOLEAN NOT NULL:= TRUE;
City VARCHAR2 (50) DEFAULT 'Мариуполь';
Born DATE default sysdate;
BEGIN
Message('height: '||height);
Message('Age: '||age);
Message('Weight: '||weight);
Message('Name: '||name);
Message('S_name: '||s_name);
Message('F_name: '||f_name);
Message('City: '||city);
synchronize;
END;
```

- Height – переменная численного типа с пятью знаками до запятой и двумя после, хранит значение высоты;
- Age – переменная численного типа без определения разряда, хранит значение возраста;
- Weight – константа численного типа. Значение инициализации 74.5;
- Name – переменная символьного типа, хранит значение имени. Максимальный размер – десять символов;
- S_name CHAR (20) – переменная символьного типа, хранит значение фамилии. Максимальный размер – двадцать символов;
- F_name – значение символьного типа. Не может быть пустым, инициализировано значением «Васильевич»;
- Man – переменная логического типа, инициализирована в TRUE; значение TRUE или FALSE указывать без кавычек. Переменная типа Boolean может принимать только три типа значений:
  – TRUE;
  – FALSE;
  – NULL.
- City – переменная символьного типа, инициализированная значением по умолчанию – «Мариуполь». Максимальный размер – пятьдесят символов;
- Born – переменная типа даты, по умолчанию хранит значение текущей даты.

5. Запустите форму и нажмите на кнопку. После нажатия кнопки на экране последовательно будут появляться сообщения со значениями переменных.

В PL/SQL существует множество зарезервированных слов (табл. 2), которые не могут быть использованы для объявления переменных. Также не следует называть переменные именами столбцов. Ниже приведен пример неправильного объявления переменных:

```
    DECLARE
Select varchar(20);
Pr_key number;
I number;
BEGIN
Select age from about where pr_key=pr_key;
END;
```

Объявление переменных с атрибутом %TYPE

Префикс %TYPE используется для объявления переменных, ассоциированных с ранее объявленными переменными или столбцами Базы Данных. Основное преимущество префикса в том, что при объявлении переменная, которая впоследствии будет ассоциирована со столбцом Базы Данных, должна быть точно объявлена (имя, разряд) во избежание ошибок. Вы не можете помнить все значения типов данных столбцов таблицы, разрядность численных типов и размерность символьных полей, поэтому понимание использования этого префикса очень важно. Ниже приведен синтаксис и пример объявления префикса:

```
Identifier table_name.column_name%TYPE;
Identifier previous_var_name%TYPE;

DECLARE
City VARCHAR2 (50);
t_city city%TYPE:='Мариуполь';
pr_key about.pr_key%TYPE
…
```

t_city – переменная типа city. Переменная символьного типа, хранит значение – «Мариуполь»;
pr_key – хранит значение столбца таблицы about.pr_key.

Примечание: *NOT NULL столбцы не могут быть объявлены с префиксом %TYPE.*

Область видимости переменной

Область видимости переменной – это область, в которой она определена. В большинстве случаев все переменные имеют единую область видимости. Переменная может быть локальной, глобальной или одновременно и той и другой. Локальная переменная – это переменная, локальная для блока, в котором она объявлена. Глобальная переменная – это переменная, действующая для всех вложенных блоков, которые находятся внутри блока, где она объявлена. Если на одном из уровней вложенности эту переменную переопределить, то она станет глобальной для всех нижестоящих уровней. Блоку, стоящему на одном уровне вложенности с другим, нельзя ссылаться на переменную, объявленную в этом блоке.

В этом разделе мы рассмотрели основные типы данных, применяемых в PL/SQL, а также правила и способы их объявления.

Обращение к элементам Oracle Forms в операторах PL/SQL

Для того чтобы обращаться к элементам, блокам, окнам и параметрам, нужно перед названием объекта поставить двоеточие. Ниже приведены основные примеры обращения к объектам Oracle Forms:

- :parameter.param_name – обращение к параметру. Слово parameter обязательно.
- :block_name.item_name – обращение к элементу блока. Block_name — это идентификатор блока данных, которому принадлежит элемент, к коему мы обращаемся. Если в форме элемент один, то название блока можно будет опустить, в противном случае вы получите ошибку – «Неверная переменная привязки».
- :block – обращение к блоку. В этом случае никаких вспомогательных префиксов не требуется.
- :column_name – обращение к элементу. Если элемент в форме один, к нему можно обращаться без указания имени блока.
- :system.variable_name – обращение к системной переменной. Слово SYSTEM является обязательным.
- :global.global_name – обращение к глобальной переменной. Слово global является обязательным.

Косвенное обращение к элементам

В Oracle Forms существует два вида непрямого обращения к элементам:

- Процедура COPY.
- Функция NAME_IN.

Процедура COPY

COPY – копирует указанное значение в указанный элемент. Ниже рассмотрено два вида обращения к элементу.

: about.age:=20 – это прямое обращение, элементу age присваивается значение 20.

COPY(20, ‘about.age’) – непрямое обращение, в элемент age помещается значение 20.

Функция NAME_IN

NAME_IN – возвращает содержимое элемента или переменной. Эта функция может быть вложена в другие функции. Ниже приведены примеры обращения к элементам с помощью функции NAME_IN:

```
DECLARE
Val varchar2(20);
BEGIN
Val:=NAME_IN('about.age');
END;

Val:=TO_NUMBER(NAME_IN('about.age'));
Val:=TO_DATE (NAME_IN('about.date_born'));
```

Функция NAME_IN может использоваться совместно с процедурой COPY, например, копирование в элемент REF указанного значения:

```
COPY('Сергей', NAME_IN('about.ref_item'));
COPY(NAME_IN('about.name'), 'about.f_i_o');
```

Косвенное обращение часто используется в программных модулях и библиотеках, где прямое обращение к элементам нежелательно или недопустимо.

Комментарии

Каждая программа, претендующая на простоту, изящность и понимание, должна содержать пояснительный текст – комментарии. Для того чтобы программа была понятной и доступной, не всегда хватает исчерпывающих имен переменных или именованных блоков, поэтому в PL/SQL, как и в других языках программирования, можно использовать комментарии. В PL/SQL существует два вида комментариев:

- однострочные;
- многострочные.

Однострочные комментарии

Однострочный комментарий удобен для комментирования небольших фрагментов кода, одиночных строк или коротких фраз. Однострочные комментарии начинаются двумя дефисами (--), текст после которых до конца строки считается закомментированным. Ниже приведен пример

однострочного комментария, который объясняет назначение процедуры sum_price:

```
  IF price>100 THEN
Sum_price (5); -- процедура sum_price понижает прайсовую стоимость
               --всех товаров на 5%
  END IF;
```

Многострочные комментарии

В случае когда необходимо закомментировать большой фрагмент программы, расстановка однострочных комментариев займет приличное время, так как они удобны скорее для небольших фрагментов кода. Многострочный комментарий полезен для написания больших пояснений к тексту программы. В многострочном комментарии текст размещается между парой символов (/*) и (*/) – весь текст, размещенный между этой парой символов считается закомментированным. Ниже приведен пример многострочного комментария:

```
/* Эта процедура предназначена для выгрузки и данных
в EXCEL и обратно
Файл источник - Out.xls
Файл приемник - In.xls
Выполнение этой процедуры требует наличие Microsoft Excel
*/

DECLARE
…
BEGIN
…
END;
```

Используйте комментарии, это поможет не только вам ориентироваться в вашем же коде, но и тем разработчикам, которым, возможно, придется дорабатывать или изучать вашу программу.

Лекция 16. PL/SQL в Oracle Forms. Управляющие структуры. Глобальные переменные и параметры

В этой лекции слушатель научится работать с циклами, условным и последовательным управлением, объявлять глобальные переменные и параметры.

Ключевые слова: Global Variable, Erase, System variable, Parameter, GOTO, метка, while, for, цикл, условие.

Цель лекции: ознакомить слушателя с различными видами переменных PL/SQL, управляющими и циклическими конструкциями.

Глобальные переменные и параметры

Глобальные переменные

Global Variable (глобальная переменная) – эта переменная среды Oracle Forms, значение которой доступно всему приложению. Глобальная переменная имеет символьный тип и может хранить до 255 символов. Переменная этого типа не объявляется, а инициализируется. Вы можете инициализировать глобальную переменную в любом месте программы, просто присвоив ей значение. Вы не можете инициализировать глобальную переменную в секции объявления переменных – DECLARE. Ниже приведен пример инициализации такой переменной.

Синтаксис:

:global.variable_name,

где: global – обязательное ключевое слово для инициализации и обращения к переменной;
variable_name – название переменной global.

```
DECLARE
My_age number:=20;
My_name varchar2(10);
   BEGIN
:global.s_name:='Сергеенко'
DEFAUL_VALUE('Сергей', 'global.my_name');
…
END;
```

В этом примере помимо обычного присвоения мы использовали процедуру DEFAULT_VALUE, которая инициализирует глобальную переменную global.my_name значением – «Сергей».

> ***Примечание:*** *инициализация глобальной переменной происходит только после присвоения ей какого-либо значения. Если вы используете глобальную переменную, предварительно не инициализировав ее, то, несмотря на успешную компиляцию, вы гарантировано получите ошибку выполнения.*

Глобальная переменная удаляется процедурой ERASE:

```
ERASE('global.my_name');
```

После удаления переменной также освобождается память.

В последних версиях Forms глобальные переменные поддерживаются, но не рекомендуются, так как они прежде всего расходуют существенное количество ресурсов. В качестве альтернативы глобальным переменным можно использовать небазовые элементы, размещенные на NULL-канве. Такие переменные, в отличие от глобальных, могут хранить различные типы данных (DATE, NUMBER) и позволяют контролировать размер с помощью свойства Maximum Length. Недостатком их использования является тот факт, что такая переменная доступна только в пределах одного модуля.

Параметры

Параметры – это особый вид переменных Oracle Forms, использующийся для передачи значений между модулями и другими средствами разработки, например, Reports и Graphics. Параметры – это одна из самых лучших альтернатив глобальным переменным, так как они имеют относительно большую гибкость:

- Хранение и передача специфичных данных. Параметры могут хранить различные типы данных.
- Параметр не нужно объявлять или инициализировать, так как он создается как объект навигатора.
- Значение параметра может быть доступно другим модулям и компонентам пакета Developer (Reports, Graphics).

Параметры могут быть использованы для передачи данных в другие модули с помощью процедур:

- CALL_FORM;
- OPEN_FORM;

- NEW_FORM;
- RUN_PRODUCT.

Параметры могут применяться для передачи данных в Reports с помощью процедур:

- RUN_PRODUCT – используется в версиях Oracle Forms 6i и ниже. Неприменимо к Oracle Forms 9i и выше.
- RUN_REPORT_OBJECT – используется в версиях Oracle Forms 9i и выше. Неприменимо к Oracle Forms 6i и ниже.

Далее мы рассмотрим примеры создания, обращения и передачи параметров.

Создание параметра

1. В навигаторе объектов выберите узел «Параметры». Создайте параметр двойным щелчком мыши по узлу «Параметры» или выберите пункт меню Навигатор | Создать.
2. Созданный параметр появится в одноименном узле. При создании параметру присваивается имя ПАРАМЕТР+ПОСЛЕДОВАТЕЛЬНЫЙ НОМЕР, например, ПАРАМЕТР1. Запустите палитру свойств параметра (рис. 16.1).

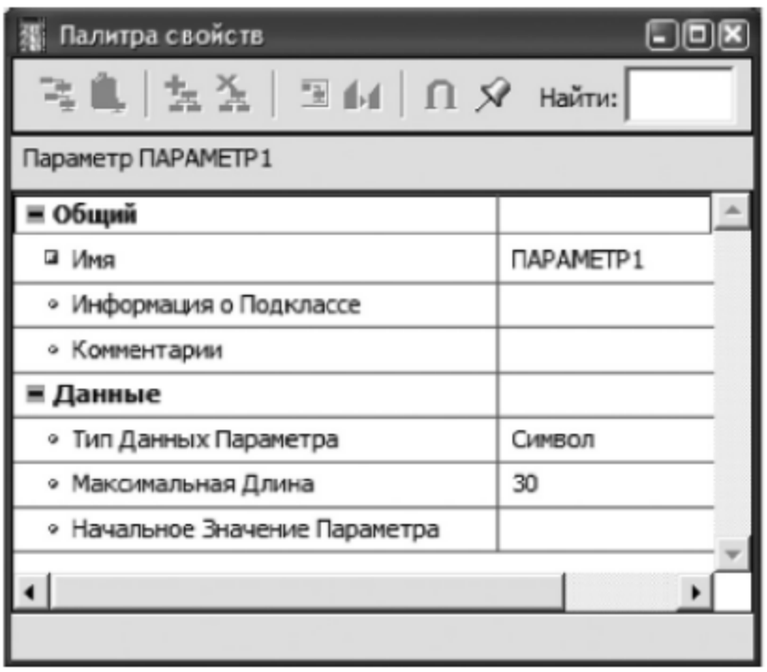

Рис. 16.1. Палитра свойств параметра

3. Определите внутренне имя параметра в свойстве «Имя». Определяя имя параметра, не забывайте, что оно будет использовано для обращения к нему в программе.

4 Выберите тип данных для параметра – CHAR, DATE или NUMBER – в свойстве «Тип данных параметра». Если вы указываете тип CHAR, то необходимо также указать максимальную длину значения данных.

5. В свойстве «Начальное значение параметра» определите начальное значение параметра.

Ранее мы уже рассматривали примеры обращения к параметрам в операторах PL/SQL, но повторим еще раз.

Обращение к параметрам в программах PL/SQL

Для обращения к параметру используется обязательное ключевое слово :PARAMETER. Ниже приведен синтаксис определения параметра:

:PARAMETER.parameter_name

где PARAMETER – зарезервированное слово, использующее для обращения;
Parameter_name – уникальное имя параметра в объектном навигаторе.

```
DECLARE
Ma_name varchar2(20):=:parameter.my_name;
BEGIN
:parameter.s_name:='Сергеенко';
:about.age:=:parameter.my_age;
END;
```

К параметрам, так же как и к элементам, допустимо косвенное обращение с помощью функции NAME_IN и COPY, как показано ниже:

```
My_name:=Name_in('PARAMETER.my_name')
Copy('Сергей', 'PARAMETER.my_name')
```

При обращении к параметрам, как и в случае с глобальными переменными, опускается двоеточие перед ключевым словом, а сам параметр заключается в двойные кавычки.

Вы также можете обращаться к параметрам:

1. В свойствах блока ORDER BY и WHERE CLAUSE. Например:

ORDER BY – :parameter.age asc
WHERE CLAUSE – my_name=:parameter.my_name

2. При создании LOV, указывая условие WHERE при формировании группы записей и в свойстве RETURN ITEM LOV (свойство Column Mapping). Например:

```
select * from names where name=:parameter.my_name
```

Передача параметров. Создание списков параметров

Как уже было написано выше, параметры могут передаваться между модулями и приложениями с помощью списков параметров – Parameter List. Список параметров – это структура, которая подобно элементу списка хранит список имен параметров. В списке параметров вы можете определить два типа данных:

- Текстовый параметр – хранит и передает символьный набор данных.
- Параметр данных – хранит и передает в качестве ключа имя группы записей (Record Group).

Процедуры и функции для работы со списком параметров и параметрами приведены в таблице 16.1.

Таблица 16.1. Процедуры и функции для работы со списком параметров и параметрами.

Метод	**Значение**
ADD_PARAMETER	Добавление параметра в список
CREATE_PARAMETER_LIST	Создание списка параметров
DELETE_PARAMETER	Удаление параметра
DESTROY_PARAMETER_LIST	Удаления списка параметров
GET_PARAMETER_ATTR	Получение значения параметра
GET_PARAMETER_LIST	Получение значения списка параметров
SET_PARAMETER_ATTR	Установка значения атрибута параметра

Рассмотрим пример, в котором затронуты практически все вышеперечисленные функции и процедуры.

```
DECLARE
        pl_id ParamList;
        pl_name VARCHAR2(255) := 'PL_REP';
```

```
BEGIN
   pl_id := Get_Parameter_List(pl_name);
     IF NOT ID_Null (pl_id) THEN
      Destroy_Parameter_List(pl_id);
     END IF;
     pl_id := Create_Parameter_List (pl_name);
Run_Product(REPORTS, 'c:\ zak\zak' ,ASYNCHRONOUS, RUNTIME,
FILESYSTEM, pl_id, null);
END;
```

Системные переменные и константы

Системные переменные – это переменные сессии Oracle Forms. Как только вы запускаете приложение, вам становятся доступны все его переменные окружения, выполнения и внутреннего состояния. Системные переменные, их имена и типы не привязаны к какому-либо объекту конкретно, они лишь передают текущее состояние или значение объекта, поэтому системная переменная – это, скорее, ссылка на текущее состояние приложения, элемента или события. Практически все системные переменные имеют тип данных VARCHAR и поэтому могут возвращать различные типы данных, такие как число, дата, строка. Системные переменные – это не только путь к написанию правильного и «гибкого» кода, это еще и огромный шаг к написанию переносимых и независимых программных единиц. Для наглядного примера напишем триггер, в котором нам нужно будет определить статус блока и, если блок находится в режиме CHANGED, сохранить изменения.

1. Пример с использованием системной переменной:

```
If :SYSTEM.BLOCK_STATUS = 'CHANGED' Then
      Commit_Form ;
End if ;
```

2. Пример без использования системной переменной:

```
   if get_block_property('block_name', status)='CHANGED' then
Commit_Form;
end if;
```

Если сравнивать эти два примера, то основной и самой существенной разницей между ними будет отсутствие в первом имени блока, статус

которого мы проверяем. Первый пример вы можете перенести в любое другое приложение, и оно точно будет работать, а вот со вторым так не получится, так как в нем указано имя блока.

Все системные переменные, за исключением четырех, являются Read_Only, то есть «только для чтения». Ниже приведена таблица 16.2, в которой перечислены все основные системные переменные.

Таблица 16.2. Основные системные переменные

Системная переменная
SYSTEM.BLOCK_STATUS
SYSTEM.COORDINATION_OPERATION
SYSTEM.CURRENT_BLOCK
SYSTEM.CURRENT_DATETIME
SYSTEM.CURRENT_ITEM
SYSTEM.CURRENT_FORM
SYSTEM.CURRENT_VALUE
SYSTEM.CURSOR_BLOCK
SYSTEM.CURSOR_ITEM
SYSTEM.CURSOR_RECORD
SYSTEM.CURSOR_VALUE
SYSTEM.CUSTOM_ITEM_EVENT
SYSTEM.CUSTOM_ITEM_EVENT_PARAMETERS
SYSTEM.DATE_THRESHOLD
SYSTEM.EFFECTIVE_DATE
SYSTEM.EVENT_WINDOW
SYSTEM.FORM_STATUS
SYSTEM.LAST_QUERY
SYSTEM.MESSAGE_LEVEL
SYSTEM.MODE
SYSTEM.MOUSE_BUTTON_PRESSED
SYSTEM.MOUSE_BUTTON_SHIFT_STATE
SYSTEM.MOUSE_CANVAS
SYSTEM.MOUSE_FORM
SYSTEM.MOUSE_RECORD

SYSTEM.MOUSE_WINDOW
SYSTEM.MOUSE_X_POS
SYSTEM.MOUSE_Y_POS
SYSTEM.RECORD_STATUS
SYSTEM.SUPPRESS_WORKING
SYSTEM.TRIGGER_BLOCK
SYSTEM.TRIGGER_ITEM
SYSTEM.TRIGGER_RECORD
SYSTEM.MASTER_BLOCK
SYSTEM.TAB_NEW_PAGE
SYSTEM.TAB_PREVIOUS_PAGE

Полный перечень системных переменных вы можете получить, просмотрев одноименный узел в отладчике Forms. Далее приведено описание наиболее используемых функций.

SYSTEM.BLOCK_STATUS – показывает статус блока, точнее, в каком режиме он находится: Enter_Query (ввода запроса), Insert (вставки), New (новой записи).

SYSTEM.CURRENT_BLOCK – возвращает имя текущего блока.

SYSTEM.CURRENT_DATETIME – возвращает системную дату.

SYSTEM.CURRENT_ITEM – возвращает имя (если это кнопка, то также возвращает имя, а не метку) текущего элемента блока.

SYSTEM.CURRENT_FORM – возвращает имя текущей формы.

SYSTEM.CURRENT_VALUE – возвращает текущее значение поля.

SYSTEM.CURSOR_BLOCK – возвращает имя курсора блока.

SYSTEM.CURSOR_ITEM – возвращает курсор на текущий элемент блока в формате block_name.item_name.

SYSTEM.CURSOR_RECORD – возвращает последовательный номер курсора записи.

SYSTEM.FORM_STATUS – ‘CHANGED’, ‘NEW’, ‘QUERY’.

SYSTEM.LAST_QUERY – возвращает последний запрос.

SYSTEM.LAST_RECORD – возвращает TRUE, если достигнута последняя запись, и ‘FALSE’, если наоборот.

SYSTEM.MESSAGE_LEVEL – эта одна из немногих системных переменных, которая не только возвращает значение, но и принимает его, позволяя разработчику контролировать уровень строгости сообщения. Чем больше уровень строгости, тем меньше системных сообщений

на экране. Если в процессе работы формы требуется вывести сообщение, то оно будет выведено только в том случае, когда уровень этого сообщения больше, чем значение SYSTEM.MESSAGE_LEVEL. В противном случае сообщение будет проигнорировано.

По умолчанию значение SYSTEM.MESSAGE_LEVEL равно 0, и это значит, что Oracle Forms будет выводить любые сообщения. Исходя из списка возможных уровней, мы знаем 6 значений, которые имеет смысл присваивать переменной SYSTEM.MESSAGE_LEVEL: 0, 5, 10, 15, 20, 25. Каждый вышестоящий уровень подавляет нижестоящий. Присвоение переменной SYSTEM.MESSAGE_LEVEL значения 10 будет означать, что все сообщения, имеющие уровень 5 или 10, больше выводиться не будут.

SYSTEM.MODE – возвращает текущий режим форма запроса.

SYSTEM.MOUSE_BUTTON_PRESSED – возвращает значение состояния кнопки. Состояние TRUE означает, что кнопка мыши нажата. SYSTEM.MOUSE_BUTTON_SHIFT_STATE – возвращает состояние курсора на кнопке.

SYSTEM.MOUSE_X_POS – возвращает значение позиции курсора мыши на экране по X.

SYSTEM.MOUSE_Y_POS – возвращает значение позиции курсора мыши на экране по Y.

SYSTEM.RECORD_STATUS – эта переменная показывает, в каком режиме находится запись: CHANGED, INSERT, NEW, QUERY.

SYSTEM.SUPPRESS_WORKING – подавляет стандартные рабочие сообщения, если SYSTEM.SUPPRESS_WORKING:= TRUE.

SYSTEM.TRIGGER_BLOCK – возвращает курсор блока, когда текущий триггер инициализирован.

SYSTEM.TRIGGER_ITEM – возвращает курсор элемента блока, когда текущий триггер инициализирован.

SYSTEM.TRIGGER_RECORD – возвращает последовательный номер записи.

После того как мы рассмотрели основные принципы теории использования системных переменных, перейдем непосредственно к выполнению примеров, которые более наглядно помогут вам разобраться в их нужности и важности.

1. :SYSTEM.CURSOR_BLOCK

```
Declare
Curr_bl varchar2(90):= :system.current_block
If :system.current_block='Zak' then
Set_block_property(curr_bl, update_allowed, property_false);
```

```
Set_block_property(curr_bl, delete_allowed, property_false);
Set_block_property(curr_bl, insert_allowed, property_false);
…
End if;
End;
```

В этом примере системная переменная используется для проверки имени блока, и в случае совпадения имени со значением условия выполняется некоторое действие, а именно запрет на операции DML в блоке.

2. :SYSTEM.CURSOR_RECORD

Вы можете использовать эту переменную для генерирования последовательности записей. Например, у вас в блоке одновременно отображается десять записей и вам нужно пронумеровать каждую новую запись. Для этого можно создать триггер WHEN-NEW-ITEM-INSTANCE на уровне блока и в теле триггера написать:

```
:zak.num:=:SYSTEM.CURSOR_RECORD
```

Если вы выполните этот пример, то увидите, как при переходе на следующую запись элемент :num принимает значение последовательного номера записи. Можно записать пример намного проще:

```
Message('Номер записи: ':SYSTEM.CURSOR_RECORD);
```

3. :SYSTEM.DATE_THRESHOLD

```
:System.Date_Threshold := '01:00' ;
```

4. :SYSTEM.EFFECTIVE_DATE

```
:System.Effective_Date := '01-Декабрь-2007 11:00:00' ;
```

5. :SYSTEM.MOUSE_BUTTON_PRESSED

```
Declare
  Btn_press  Varchar2(21) := :System.Mouse_Button_Pressed ;
Begin
  If Btn_press  = '1' Then
     Message('Нажата левая кнопка' );
  ElsIf Btn_press  = '2' Then
```

```
      Message('Нажата правая кнопка' );
   Else
      Message('Другая клавиша' );
   End if ;
End ;
```

В этом примере показано, как с помощью системной переменной проверяется нажатие клавиши.

6. :SYSTEM.LAST_RECORD

```
Begin
    First_Record ;
    Loop
       Exit when :System.last_Record = 'TRUE' ;
:num:=: system.cursor_record;
Message('Это была запись №_ '||:system.cursor_record)
       Next_Record ;
    End loop ;
End ;
```

В этом примере системная переменная :system.last_record используется для проверки условия выхода из цикла, так как возвращает значение 'TRUE', если текущая запись последняя.

7. :SYSTEM.MESSAGE_LEVEL

Уровни строгости сообщения:

- 0;
- 5;
- 10;
- 15;
- 20;
- 25.

```
Declare
  Mess_lev Pls_Integer := :System.Message_Level ;
Begin
Insert into Zak  values(...)
  :System.Message_Level := 5 ;
  Commit_Form ;
  :System.Message_Level := Mess_lev;
End ;
```

В этом примере показан стандартный прием для обхода сообщения «Нет изменений для сохранения». Для начала переменной Mess_lev присваивается временное значение начального уровня строгости сообщения, а затем переопределяется значение самой системной переменной для фиксации изменений. После того как фиксация прошла, системной переменной присваивается первоначальное значение.

8. :SYSTEM.FORM_STATUS
   - CHANGED;
   - NEW;
   - QUERY.

```
If :SYSTEM.FORM_STATUS = 'CHANGED' Then
   Message('Форма находится в режиме:'||:SYSTEM.FORM_STATUS);
…;
End if;
```

В этом примере проверяется статус формы и выводится на экран в виде сообщения.

9. :SYSTEM.BLOCK_STATUS
   - CHANGED;
   - NEW;
   - QUERY.

```
If :SYSTEM.BLOCK_STATUS = 'Query' Then
   Go_item(:num);
LIST_VALUES;
End if ;
```

В этом примере проверяется статус блока, и если он находится в режиме запроса, то выполняется навигация к элементу и вызывается список значений.

10. :SYSTEM.RECORD_STATUS
    - CHANGED;
    - INSERT;
    - NEW;
    - QUERY.

```
If :SYSTEM.RECORD_STATUS in ('CHANGED') then
Commit_form;
```

```
Go_block('pzak');
Execute_query;
End if;
```

В этом примере проверяется статус блока, и если он находится в режиме изменения, то выполняется сохранение и запрос данных.

11. :SYSTEM.CURRENT_VALUE

```
If :system.current_value>100 then
…
End if;
```

В этом примере идет проверка значения элемента, и если оно больше заданного, то выполняется какое-либо действие.

12. :SYSTEM.MASTER_BLOCK

```
Copy(name_in(:System.Master_Block||'.primary_item'),
'detail_block.detail_item');
```

В этом примере системная переменная используется для определения имени Мастер блока. С помощью команды COPY значение первичного ключа Мастер блока (primary_item) копируется в элемент подчиненного блока (detail_block.detail_item).

Использование циклов

В ваших приложениях вам не раз придется исполнять какую-то группу операторов неоднократно. Повторять операторы можно двумя способами: указав конкретное число проходов в цикле или выполняя цикл до встречи с некоторым условием сравнения, пока условие не будет истинно. Поскольку Oracle Forms поддерживает PL/SQL, то, следовательно, и циклы в Forms будут такого же типа, как и в PL/SQL:

- LOOP-EXIT;
- WHILE-LOOP;
- FOR-LOOP.

LOOP-EXIT – это один из самых простых циклов PL/SQL. Ключевое слово LOOP указывает на начало повторяемого программного блока, а END LOOP – на его конец. Ключевое слово EXIT, указываемое отдельно, означает, что цикл нужно прервать, а ключевое слово EXIT WHEN – что

предлагаемый оператор сравнения нужно проверить на необходимость завершения выполнения операторов.

Рассмотрим пример с применением такого цикла:

```
   Declare
         i number:=0;
   Begin
          LOOP
   i:=C_ount+1
    insert into EXEM1  (SUM)
    values (i);
  IF i=10 THEN
     EXIT;
     end if;
end loop;
end;
```

После выполнения этого цикла в таблицу EXEM1 будет вставлено 10 записей, причем здесь наглядно видно, что как только наше условие будет истинно, то есть значение счетчика i будет равным 10, – цикл прервется.

Теперь рассмотрим этот же пример, только с использованием оператора EXIT-WHEN:

```
Declare
         i number:=0;
   Begin
          LOOP
   i:=C_ount+1;
    insert into EXEM1  (SUM)
    values (i);
  EXIT WHEN i=10;
end loop;
end;
```

Из примера видно, что такая конструкция, в отличие от предыдущей, дает возможность разработчику писать менее громоздкий код.

WHILE-LOOP – цикл этого типа похож на цикл с условием LOOP-EXIT WHEN, разница заключается в том, что в WHILE-LOOP проверка условия выхода осуществляется в начале цикла. Если сравнивать два этих цикла, то по гибкости, конечно, выигрывает LOOP-EXIT, т. к. позволяет указывать условие EXIT в любом месте цикла, а если сравнивать по элегантности и краткости оформления кода, то WHILE-LOOP выигрывает.

Рассмотрим тот же самый пример, что и в предыдущем случае, только с использованием конструкции WHILE-LOOP:

```
Declare
          i number:=0;
   Begin
           while i<=10 LOOP
   i:=C_ount+1;
    insert into EXEM1  (SUM)
    values (i);
   end loop;
end;
```

В этом примере мы еще больше упростили наш цикл, т. к. не использовали оператор EXIT.

FOR_LOOP – этот цикл отличается от двух других, т. к. позволяет указать конкретное количество выполнений программы до ее прерывания. Данная циклическая конструкция состоит из счетчика и диапазона циркуляции цикла. Счетчик в операторах FOR-LOOP является встроенным и автоматически увеличивает значение на единицу. Диапазон циркуляции определяют нижняя и верхняя граница, причем границы должны иметь целочисленный тип. Нижняя граница диапазона циркуляции цикла может начинаться не только с единицы, но и с любого другого целого числа.

Для примера создадим таблицу EXEM1 с одним столбцом типа number – SUM, который содержит различные числовые значения. Теперь выполним цикл:

```
DECLARE
C_OUNT NUMBER;
BEGIN
for i in 1..10 loop
insert into EXEM1  (SUM)
values (i);
C_OUNT:=C_OUNT+i;
end loop;
SELECT C_OUNT INTO :block_name.item_name from DUAL;
end;
```

После выполнения этого цикла в таблицу EXEM1 будет вставлено 10 записей с значением от 1 до 10, а затем результат суммирования значений вставленных записей будет выведен в элемент формы :block_name.

item_name. В рассмотренном нами цикле встроенным счетчиком будет переменная i, а диапазоном циркуляции будет интервал от 1 до 10, где 1 – это нижняя граница диапазона, а 10 – верхняя. Если рассмотреть пример, который мы приводили в двух предыдущих случаях, но с применением операторов FOR-LOOP, то вы увидите, что использование этой конструкции делает код элегантнее:

```
Declare
          i number:=0;
   Begin
          for i in 1..10 LOOP
     insert into EXEM1  (SUM)
     values (i);
           end loop;
end;
```

FOR_LOOP также позволяет исполнять цикл в другом направлении с помощью оператора REVERSE, причем при изменении направления цикла вам не надо как-то иначе задавать границы диапазона, т. к. Oracle сделает это за вас:

```
Declare
i number:=0;
Begin
for i in REVERSE 1..10 LOOP
insert into EXEM1  (SUM)
values (i);
end loop;
end;
```

Существуют также операторы FOR-LOOP для работы с курсорами, которые будут рассмотрены в разделе описания курсоров.

Oracle Forms позволяет создавать циклические конструкции любого уровня вложенности. Поэтому рассмотренные нами конструкции также могут быть вложенными.

Условное управление

Условным управлением называется оператор IF - THEN (если-то), который говорит о том, что при истинности условия (или наоборот, если оно ложно) нужно выполнить следующий программный блок.

Использование этого оператора позволяет разработчику структурировать код программы таким образом, что определенные операторы будут или не будут выполняться – это определяется достоверностью операции сравнения. Причем логика этой конструкции следующая: если (IF) сравнение истинно, то (THEN) выполнить следующее. В PL/SQL существуют еще две дополнительные конструкции ELSE (иначе) и ELSIF (иначе если). Логика оператора ELSE: «в противном случае сделать то, что указано в конструкции ELSE», а логика оператора ELSIF: «в противном случае, если...». Оператор ELSIF позволяет вам исключить дополнительное вложение оператора IF внутрь конструкции ELSE, поэтому позволяет реализовать операцию взаимоисключения более четко. Приведем простой пример:

WHEN_NEW_FORM_INSTACE:

```
BEGIN
IF TO_CHAR (TO_DATE (SYSDATE, 'DAY')='SATURDAY' or TO_CHAR (TO_
DATE(SYSDATE, 'DAY')='SUNDAY'
THEN
set_block_property('Block1',update_allowed, property_false);
set_block_property('Block1',insert_allowed, property_false);
set_block_property('Block1',delete_allowed, property_false);
ELSE
message ('Сегодня  '||TO_CHAR (SYSDATE,'DAY')||' удачного рабочего
дня');
end if;
end;
```

В вышеприведенном примере перед запуском формы будет проверено, выходной день сегодня или рабочий: если условие истинно, то есть сегодня выходной, то запретить какие-либо операции, связанные с манипуляцией данных, если же условие ложно, то в строке состояния появится сообщение: «Сегодня понедельник, удачного рабочего дня». Теперь рассмотрим пример с использованием оператора ELSE:

```
Declare
  but_no  Varchar2(1) := :System.Mouse_Button_Pressed ;
Begin
  If but_no = '1' Then
     Message('Нажата левая кнопка');
  Else  If LN$Btn = '2' Then
     Message('Нажата правая кнопка' );
  End if ;
```

```
End if;
End ;
Рассмотрим тот же случай, только с использованием оператора ELSIF:
Declare
  but_no  Varchar2(1) := :System.Mouse_Button_Pressed ;
Begin
  If but_no = '1' Then
     Message('Нажата левая кнопка');
  ELSIF LN$Btn = '2' Then
     Message('Нажата правая кнопка' );
  End if;
End ;
```

Как видим, построенная конструкция IF-THEN с помощью оператора ELSIF менее громоздка и более понятна.

И в заключение рассмотрим пример, в котором используются как условные, так и итеративные операторы:

```
DECLARE
OriPos number;
BEGIN
OriPos := TO_NUMBER(:System.Trigger_Record);
         First_Record;
         LOOP
           IF (:System.Last_Record = 'TRUE') THEN
              Go_Record ('OriPos');
              EXIT;
           ELSE
              Next_Record;
           END IF;
         END LOOP;
   END;
```

Последовательное управление

В PL/SQL существует оператор, который называется «метка» (label). Оператор GOTO — это последовательность символов, отмечающая некоторый исполняемый блок, который используется для выхода из условной структуры обработчика исключительных ситуаций или программного блока. Оператор GOTO дает возможность перейти с места его указания в программе непосредственно к метке.

```
 LOOP
IF (:System.Last_Record = 'TRUE') THEN
 goto first_rec;
        ELSE
            Next_Record;
         END IF;
       END LOOP;
<<first_rec>>
 First_record;
 END;
```

Ограничения на использование GOTO

Нельзя разместить метку перед неисполняемым оператором, например, перед IF, END LOOP и т. д. Если же вам все-таки необходимо разместить метку там, где нет исполняемого оператора, следует воспользоваться ключевым словом NULL, которое, как известно, не производит никаких действий, но при этом удовлетворяет требования, предъявляемые к меткам. Для примера возьмем часть из предыдущей программы:

```
 LOOP
IF (:System.Last_Record = 'TRUE') THEN
 goto first_rec;
 .......
           <<first_rec>>
    END;
```

Обратите внимание, что между меткой и оператором завершения (end) нет исполняемого оператора, поэтому такое объявление метки вызовет ошибку. Для того чтобы избежать этой ошибки, достаточно написать ключевое слово NULL:

```
 LOOP
IF (:System.Last_Record = 'TRUE') THEN
 goto first_rec;
 .......
           <<first_rec>>
    NULL;
 END;
```

Оператор GOTO используется только для выхода из структуры, но не для входа в нее. Приведем пример такого объявления, все из того же программного блока:

```
    goto first_rec;
 LOOP
IF (:System.Last_Record = 'TRUE') THEN
  <<first_rec>>
       ELSE
           Next_Record;
        END IF;
      END LOOP;
First_record;
 END;
```

Совет: *старайтесь как можно более элегантней оформлять свой код с помощью вышеуказанных операторов для повышения читабельности вашей программы, чтобы ваш программный блок имел вид осмысленной структуры.*

Лекция 17. Элемент дерево (Tree view)

В этой лекции слушатель научится создавать и заполнять элемент дерево. Также будут рассмотрены примеры поиска по дереву и обработки триггеров.

Ключевые слова: дерево, лист, ветка, создание деревьев, заполнение деревьев, виды деревьев.

Цель лекции: ознакомить слушателя с элементом Tree View и пакетом FTREE, предназначенным для обработки деревьев.

Элемент Tree View является специфичным не только потому, что требует особых условий и настройки свойств, а еще и потому, что для управления этим элементом в Oracle Forms создан специальный встроенный пакет FTREE. Элемент дерево является очень функциональным, так как позволяет скоординировать и компактно отобразить большой набор данных, разбивая его на логические единицы – узлы. Вы можете заполнять деревья статически и динамически данными из таблиц. В дереве вы можете реализовать отношение Мастер-Деталь, выбирая в качестве родительского узла запись таблицы Мастер, а в качестве дочернего – запись из таблицы Деталь. Вы также можете использовать дерево как навигатор вашего приложения, в котором перечислены все блоки и элементы. Пакет FTREE содержит довольно небольшой объем процедур и функций, которые перечислены ниже:

- ADD_TREE_DATA;
- ADD_TREE_NODE;
- DELETE_TREE_NODE;
- FIND_TREE_NODE;
- POPULATE_GROUP_FROM_TREE;
- POPULATE_TREE;
- SET_TREE NODE_PROPERTY.

Создание Tree View

Tree View представляет собой прямоугольную область, в которой отображается иерархическое дерево. Для того чтобы создать дерево, выполните следующие действия:

1. Создайте новую форму и сохраните ее как TREE.
2. Находясь в навигаторе объектов, создайте два небазовых блока. Первый блок назовите «TREE», а второй «CONTR».

3. Находясь в редакторе разметки, создайте одну канву. Разместите на канве элемент Tree View и две кнопки.
4. Установите для элемента TREE VIEW блок TREE, а все остальные элементы ассоциируйте с блоком «CONTR».

В этом примере мы создали дополнительный управляющий блок данных «CONTR» для последующего использования в примерах.

Прежде чем построить дерево, рассмотрим основные методы пакета FTREE, которые нам для этого понадобятся.

Функция ADD_TREE_NODE

ADD_TREE_NODE (*item_id [item_id]* ITEM, *node* NODE, *offset_type* NUMBER, *offset* NUMBER, *state* NUMBER, *label* VARCHAR2, *icon* VARCHAR2, *value* VARCHAR2) RETURN FTREE.NODE – добавляет новый элемент в дерево.

Метод	Параметр	Тип	Значение
FUNCTION ADD_TREE_NODE (параметры) RETURN NODE	item_name [id_name]	VARCHAR2 [FORMS4C.ITEM]	Определяет имя элемента или его идентификатор
	node	FTREE.NODE	Определяет вершину дерева
	offset_type	NUMBER	Определяет тип сдвига для вершины, возможные значения: PARENT_OFFSET, SIBLING_OFFSET
	offset	NUMBER	Определяет позицию новой вершины: если это PARENT_OFFSET, то позиция 1-n,, или LAST_CHILD. Если тип сдвига SIBLING_OFFSET, то позицией сдвига может быть NEXT_NODE или PREVIOUS_NODE
	state	NUMBER	Определяет состояние вершины: COLLAPSED_NODE, EXPANDED_NODE, LEAF_NODE
	label	VARCHAR2	Отображаемый текст для вершины
	icon	VARCHAR2	Имя файла иконки для вершины
	value	VARCHAR2	Значение, которое содержит вершина

Примечание: элемент дерево всегда должен размещаться в отдельном блоке и быть в нем единственным элементом. В предыдущем примере для того чтобы разместить кнопки, нам пришлось создать второй блок, так как блок, содержащий элемент Tree View, не может содержать других элементов. Если вы все же разместите еще один элемент в блоке «TREE», то получите ошибку «FRM-32089 — Деревья должны размещаться в однозаписных блоках с одним элементом».

Следующий пример описывает процесс добавления нового элемента в существующее дерево.

Листинг 17.1. Добавление элемента к дереву

```
DECLARE
Itree ITEM;
top_node Ftree.Node;
new_node Ftree.Node;
i_value VARCHAR2(30);
BEGIN
Itree := Find_Item('tree_block.tree_item ');
new_node := Ftree.Add_Tree_Node(Itree, Ftree.ROOT_NODE,
Ftree.PARENT_OFFSET, Ftree.LAST_CHILD,
Ftree.EXPANDED_NODE, i_value, NULL, i_value);
END;
```

Процедура DELETE_TREE_NODE

DELETE_TREE_NODE (*item_name [item_id]* VARCHAR2, *node* FTREE.NODE) – эта процедура выполняет действие, обратное ADD_TREE_NODE, то есть удаляет элемент дерева.

Метод	Параметр	Тип	Значение
PROCEDURE DELETE_TREE_NODE (параметры)	item_name [id_name]	VARCHAR2 [FORMS4C.ITEM]	Определяет имя элемента или его идентификатор
	node	FTREE.NODE	Определяет вершину дерева, которую необходимо удалить

Следующий пример описывает процесс удаления вершины и всех ее детей.

Листинг 17.2. Удаление элемента дерева

```
DECLARE
Itree item := find_item('tree_block.tree_item ');
del_node Ftree.Node;
i_value VARCHAR2(30);
BEGIN
del_node := Ftree.Find_Tree_Node(Itree, i_value,
Ftree.FIND_NEXT, Ftree.NODE_LABEL,
Ftree.ROOT_NODE, Ftree.ROOT_NODE);
```

```
IF NOT Ftree.ID_NULL(del_node) then
Ftree.Delete_Tree_Node(Itree, del_node);
END IF;
END;
```

Функция FTREE.GET_TREE_NODE_PARENT

FTREE.GET_TREE_NODE_PARENT (*item_name [item_id]* VARCHAR2, *node* FTREE.NODE) – возвращает родителя указанной вершины дерева.

Метод	Параметр	Тип	Значение
FUNCTION DELETE_TREE_NODE (параметры) RETURN NODE	item_name [id_name]	VARCHAR2 [FORMS4C.ITEM]	Определяет имя элемента или его идентификатор
	node	FTREE.NODE	Вершина дерева, для которой отыскивается родитель

Следующий пример описывает процесс получения родителя указанного элемента.

Листинг 17.3. Получение родителя элемента дерева. Триггер WHEN-TREE-NODE-SELECTED

```
DECLARE
Itree item := find_item('tree_block.tree_item ');
par_node FTREE.NODE;
BEGIN
par_node := Ftree.Get_Tree_Node_Parent(Itree,
:SYSTEM.TRIGGER_NODE);
...
END;
```

Функция FTREE.GET_TREE_NODE_PROPERTY

FTREE.GET_TREE_NODE_PROPERTY (*item_name [item_id]* VARCHAR2, *node* NODE, *property* NUMBER) RETURN VARCHAR2 – возвращает значение свойства указанной вершины дерева.

Метод	Параметр	Тип	Значение
FUNCTION GET_TREE_NODE_PROPERTY (параметры) RETURN VARCHAR2	item_name [id_name]	VARCHAR2 [FORMS4C.ITEM]	Определяет имя элемента или его идентификатор
	node	FTREE.NODE	Вершина дерева, для которой отыскивается родитель
	property	NUMBER	Возвращает значение одного из перечисленных свойств: NODE_STATE, NODE_DEPTH, NODE_LABEL, NODE_ICON, NODE_VALUE

Следующий пример проверяет состояние указанной вершины, и если она не раскрыта, то есть NODE_STATE установлено в COLLAPSED_NODE, то вершина автоматически раскрывается, то есть NODE_STATE устанавливается в EXPANDED_NODE.

Листинг 17.4. Получение значения свойства элемента дерева. Триггер WHEN-TREE-NODE-SELECTED

```
DECLARE
Itree item := find_item('tree_block.tree_item ');
BEGIN
IF Ftree.Get_Tree_Node_Property(Itree,
:SYSTEM.TRIGGER_NODE, Ftree.NODE_STATE)=Ftree.COLLAPSED_NODE
  THEN
ftree.set_tree_node_property(Itree, :SYSTEM.TRIGGER_NODE, ftree.
node_state, ftree.expanded_node);
END IF;
END;
```

Используйте функцию FTREE.GET_TREE_NODE_PROPERTY для получения значения вершины.

Листинг 17.5. Получение значения элемента дерева. Триггер WHEN-TREE-NODE-SELECTED

```
DECLARE
Itree item := find_item('tree_block.tree_item ');
BEGIN
Ftree.Get_Tree_Node_Property(Itree, :SYSTEM.TRIGGER_NODE, Ftree.
```

```
NODE_VALUE);
...
END;
```

Функция FTREE.GET_TREE_PROPERTY

FTREE.GET_TREE_PROPERTY (*item_name [item_id]* VARCHAR2, *property* NUMBER) RETURN VARCHAR2 – возвращает значение свойства указанного элемента дерева.

Метод	Параметр	Тип	Значение
FUNCTION GET_TREE_PROPERTY (параметры) RETURN VARCHAR2	item_name [id_name]	VARCHAR2 [FORMS4C.ITEM]	Определяет имя элемента или его идентификатор
	property	NUMBER	Возвращает значение одного из перечисленных свойств: DATASOURCE, RECORD_GROUP, SET_TREE_PROPERTY, QUERY_TEXT, NODE_COUNT, SELECTION_COUNT, ALLOW_EMPTY_BRANCHES, ALLOW_MULTI-SELECT

Следующий пример описывает процесс получения количества вершин указанного дерева.

Листинг 17.6. Получение значения свойства дерева. Триггер WHEN-TREE-NODE-SELECTED

```
DECLARE
Itree item := find_item('tree_block.tree_item ');
node_cnt NUMBER;
BEGIN
node_cnt := Ftree.Get_Tree_Property(Itree,
Ftree.NODE_COUNT);
Message('Количество вершин дерева: '||node_cnt);
END;
```

Функция FTREE.GET_TREE_SELECTION

GET_TREE_SELECTION (item_name *[item_id]* VARCHAR2, selection NUMBER) RETURN Ftree.NODE – возвращает выделенную вершину дерева.

Метод	Параметр	Тип	Значение
FUNCTION GET_TREE_SELECTION (параметры) RETURN VARCHAR2	item_name [id_name]	VARCHAR2 [FORMS4C.ITEM]	Определяет имя элемента или его идентификатор
	selection	NUMBER	Индекс выделенной вершины дерева

Следующий пример описывает процесс получения выделенных элементов указанного дерева. Все элементы, помеченные как выделенные, будут удалены, причем удалять элементы нужно в обратном порядке, как показано в примере.

Листинг 17.7. Получение выделенной вершины

```
DECLARE
Itree item := find_item('tree_block.tree_item ');
Select_cnt NUMBER;
del_node FTREE.NODE;
BEGIN
Select_cnt:= Ftree.Get_Tree_Property(Itree,
Ftree.SELECTION_COUNT);
FOR i IN REVERSE 1..select_cnt LOOP
del_node := Ftree.Get_Tree_Selection(Itree, i);
Ftree.Delete_Tree_Node(Itree, del_node);
END LOOP;
END;
```

Процедура FTREE.SET_TREE_NODE_PROPERTY

SET_TREE_NODE_PROPERTY *(item_name [item_id]* VARCHAR2 [FORMS4C.ITEM], *node* FTREE.NODE, *property* NUMBER, *value* NUMBER [VARCHAR2]) – устанавливает свойства элемента дерева.

Метод	Параметр	Тип	Значение
PROCEDURE SET_TREE_NODE_PROPERTY (параметры) RETURN VARCHAR2	item_name [id_name]	VARCHAR2 [FORMS4C.ITEM]	Определяет имя элемента или его идентификатор
	node	FTREE.NODE	Вершина дерева, для которой устанавливается свойство
	property	NUMBER	Устанавливает одно из перечисленных свойств равным value: NODE_STATE, NODE_LABEL, NODE_ICON, NODE_VALUE
	value	NUMBER [VARCHAR2]	Значение, устанавливаемое для заданного свойства

Следующий пример описывает процесс поиска корневого узла и его развертывания.

Листинг 17.8. Установка значения свойства элемента дерева

```
DECLARE
Itree item := find_item('tree_block.tree_item ');
branch_node Ftree.NODE;
i_value varchar2(30);
begin
select_node := Ftree.Find_Tree_Node(Itree, i_value,

   Ftree.FIND_NEXT, Ftree.NODE_VALUE,

   Ftree.ROOT_NODE, Ftree.ROOT_NODE);
ftree.set_tree_node_property(Itree, select_node, ftree.node_state,
ftree.expanded_node);
END;
```

С помощью процедуры SET_TREE_NODE_PROPERTY вы можете менять иконку узла в зависимости от какого-либо условия, как показано в листинге 17.8.

Листинг 17.9. Установка новой иконки для элемента дерева

```
DECLARE
Itree item := find_item('tree_block.tree_item ');
```

```
BEGIN
IF Ftree.Get_Tree_Node_Property(Itree,
:SYSTEM.TRIGGER_NODE, Ftree.NODE_STATE)=Ftree.COLLAPSED_NODE
  THEN
Ftree.Set_Tree_Node_Property(Itree, :SYSTEM.TRIGGER_NODE,
Ftree.NODE_ICON, 'Close');
ELSIF
Ftree.Get_Tree_Node_Property(Itree,
:SYSTEM.TRIGGER_NODE, Ftree.NODE_STATE)= Ftree.EXPANDED_NODE
  THEN
Ftree.Set_Tree_Node_Property(Itree, :SYSTEM.TRIGGER_NODE,
Ftree.NODE_ICON, 'Open');
END IF;
END;
```

Процедура SET_TREE_PROPERTY

SET_TREE_PROPERTY (*item_name [item_id]* VARCHAR2 [FORMS4C.ITEM], *property* NUMBER, *value* NUMBER [VARCHAR2, RECORDGROUP]) – устанавливает значение указанного свойства дерева.

Метод	Параметр	Тип	Значение
PROCEDURE SET_TREE_PROPERTY (параметры)	item_name [id_name]	VARCHAR2 [FORMS4C.ITEM]	Определяет имя элемента или его идентификатор
	property	NUMBER	Устанавливает одно из перечисленных свойств равным value: RECORD_GROUP, QUERY_TEXT, ALLOW_EMPTY_BRANCHES
	value	NUMBER [VARCHAR2, RECORDGROUP]	Значение, устанавливаемое для заданного свойства. Для ALLOW_EMPTY_BRANCHES это всегда PROPERTY_TRUE или PROPERTY_FALSE

В следующем примере разрешается наличие в дереве пустых узлов, то есть элементов, не содержащих детей.

Листинг 17.10. Получение выделенной вершины

```
DECLARE
Itree item := find_item('tree_block.tree_item ');
```

```
BEGIN
Ftree.Set_Tree_Property(Itree, Ftree.ALLOW_EMPTY_BRANCHES,
property_true);
END;
```

Процедура SET_TREE_SELECTION

SET_TREE_SELECTION *item_name [item_id]* VARCHAR2 [FORMS4C.ITEM], node NODE, *selection_type* NUMBER) – устанавливает или сбрасывает выделение указанной вершины.

Метод	Параметр	Тип	Значение
PROCEDURE SET_TREE_SELECTION (параметры)	item_name [id_name]	VARCHAR2 [FORMS4C.ITEM]	Определяет имя элемента или его идентификатор
	node	NUMBER	Вершина дерева, для которой устанавливается свойство
	selection_type	NUMBER [VARCHAR2, RECORDGROUP]	Определяет тип выделения: SELECT_ON, SELECT_OFF, SELECT_TOGGLE

Следующий пример описывает процесс поиска корневого узла и его выделение.

Листинг 17.11. Выделение заданной вершины

```
DECLARE
Itree item := find_item('tree_block.tree_item ');
select_node Ftree.NODE;
i_value VARCHAR2(30);
BEGIN
   select_node:= Ftree.Find_Tree_Node(Itree, i_value,

   Ftree.FIND_NEXT, Ftree.NODE_VALUE,

   Ftree.ROOT_NODE, Ftree.ROOT_NODE);
Ftree.Get_Tree_Property(Itree, select_node, ftree.select_on);
END;
```

Теперь, когда мы рассмотрели все основные процедуры и функции для работы с деревом, можно приступить к его построению. Для начала

выполним простой пример, который описывает создание корневого и дочернего узла, причем для построения дерева мы будем использовать шаблон, созданный в предыдущем примере.

1. Установите для кнопки метку «Добавить родительский узел» и создайте для нее событие WHEN-BUTTON-PRESSED. В теле триггера напишите и скомпилируйте строки, которые приведены ниже.

```
WHEN-BUTTON-PRESSED
DECLARE
r_Node Ftree.Node;
c_Node Ftree.Node;
BEGIN
r_Node:=Ftree.Add_Tree_Node('TREE.TREE',
Ftree.ROOT_NODE,
Ftree.PARENT_OFFSET,
Ftree.LAST_CHILD,
Ftree.EXPANDED_NODE, 'Главный' , NULL, '1');
END;
```

2. Установите для второй кнопки метку «Добавить дочернюю ветку» и создайте для нее событие WHEN-BUTTON-PRESSED. В теле триггера напишите и скомпилируйте строки, которые приведены ниже.

```
WHEN-BUTTON-PRESSED
DECLARE
h_Node Ftree.Node;
l_Node Ftree.Node;
begin
h_Node:=Ftree.Add_Tree_Node(' TREE.TREE ',
Ftree.ROOT_NODE,
Ftree.PARENT_OFFSET,
Ftree.LAST_CHILD,
Ftree.EXPANDED_NODE, 'Дочерний' , NULL, '1');

l_node:=Ftree.Add_Tree_Node('TREE.TREE', h_node,
 Ftree.Parent_offset,
  ftree.NEXT_NODE,
  Ftree.LEAF_NODE,
  'Дочерний', NULL, '2');
end;
```

Далее приведен пример с заполнением дерева моделью Мастер-Деталь.

1. Для создания блоков COUNTRY и CITY выполните скрипт, приведенный в листинге 17.12.

Листинг 17.12. Создание таблиц COUNTRY и CITY

```
SQL: CREATE TABLE COUNTRY
     (PRkey number PRIMARY KEY,
      cntr varchar2(100))
      /

SQL: CREATE TABLE CITY
     (FKkey number,
      town varchar2(100))
      /

-- вставка значений в таблицу COUNTRY

SQL: INSERT INTO COUNTRY VALUES (1,'Украина');
SQL: INSERT INTO COUNTRY VALUES (2,'Россия');
SQL: INSERT INTO COUNTRY VALUES (3,'Германия');

-- вставка значений в таблицу CITY

SQL: INSERT INTO CITY VALUES (1,'Киев');
SQL: INSERT INTO CITY VALUES (1,'Львов');
SQL: INSERT INTO CITY VALUES (1,'Мариуполь');
SQL: INSERT INTO CITY VALUES (2,'Москва');
SQL: INSERT INTO CITY VALUES (2,'Санкт-Петербург');
SQL: INSERT INTO CITY VALUES (2,'Калининград');
SQL: INSERT INTO CITY VALUES (3,'Берлин');
SQL: INSERT INTO CITY VALUES (3,'Франкфурт');
SQL: INSERT INTO CITY VALUES (3,'Магдебург');
```

2. Сохраните модуль как relation_tree.
3. Создайте два блока COUNTRY и CITY и свяжите их между собой, установив JOIN CONDITION равным COUNTRY.PRKEY=CITY.FKKEY.
4. Находясь в навигаторе объектов, создайте небазовый блок и назовите его «TREE».
5. Находясь в редакторе разметки, создайте одну канву. Разместите на канве элемент Tree View и назовите его TREE.

6. Создайте триггер WHEN-NEW-FORM-INSTANCE и напишите в нем нижеприведенный код.

```
WHEN-NEW-FORM-INSTANCE

DECLARE
z varchar2(100);
x number;
h_Node Ftree.Node;
l_Node Ftree.Node;
cursor c1 is select * from country;
cursor a1 is select * from city;
c2 c1%rowtype;
a2 a1%rowtype;

begin
open c1;
for i in (select * from country) loop begin
   :parameter.param:=:parameter.param+1;
h_Node:=Ftree.Add_Tree_Node('TREE.TREE',
ftree.ROOT_NODE,
Ftree.PARENT_OFFSET,
Ftree.LAST_CHILD,
Ftree.EXPANDED_NODE, i.contr , NULL, :parameter.param);
   for j in (select * from city) loop
if j.fkkey=i.prkey then
   z:=j.c;
l_node:=Ftree.Add_Tree_Node('TREE.TREE', h_node,
 Ftree.Parent_offset,
  ftree.NEXT_NODE,
  Ftree.LEAF_NODE,
  j.c, NULL, :parameter.param); ELSE null;
   end if;
end loop;
end;
end loop;
end;
```

При запуске формы дерево заполняется значениями из таблиц. Далее мы рассмотрим несколько примеров, которые не только продемонстрируют нам методы пакета FTREE, но и расширят функциональность формы.

Вы можете синхронизировать дерево с главным блоком, отражая перемещение на следующую запись в дереве, выделяя и развертывая соответствующий узел. Для того чтобы выполнить этот пример, создайте триггер WHEN-NEW-RECORD-INSTANCE для блока COUNTRY и напишите в нем нижеприведенный код.

```
WHEN-NEW-RECORD-INSTANCE
DECLARE
Itree item := find_item('tree.tree');
select_node ftree.node;
begin
   select_node := ftree.find_tree_node(Itree, :prkey,
   ftree.find_next, ftree.node_value,
   ftree.root_node, ftree.root_node);
ftree.set_tree_selection(Itree, select_node, ftree.select_on);
ftree.set_tree_node_property(Itree, select_node, ftree.node_state,
ftree.expanded_node);
end;
```

Запустите форму и выберите записи. Попробуйте перейти на следующую или предыдущую запись, и вы увидите, как синхронно разворачивается и выделяется узел в дереве при перемещении на новую запись. Теперь создадим кнопку, по нажатию которой будет удаляться выбранный элемент дерева. Для этого создайте кнопку и установите для нее метку «Удалить узел». Создайте триггер WHEN-BUTTON-PRESSED для кнопки и напишите в нем нижеприведенный код.

```
WHEN-BUTTON-PRESSED. Удаление выделенного узла
DECLARE
Itree item := find_item('tree.tree');
select_node Ftree.NODE := Ftree.Get_Tree_Selection(Itree,
:country.prkey);
begin
Ftree.Delete_Tree_Node(Itree, select_node);
end;
```

Лекция 18. Эффективное программирование в PL/SQL. Встроенные подпрограммы, функции, процедуры и пакеты

В этой лекции слушатель научится создавать гибкий код, используя системные переменные и встроенные подпрограммы. В лекции также рассматривается создание пользовательских функций, процедур и пакетов.

Ключевые слова: Procedure, Function, Set, Get, Find, поиск объектов, установка значений, получение значений.

Цель лекции: слушатель ознакомится с основными встроенными подпрограммами и научится создавать пользовательские функции, процедуры и пакеты PL/SQL.

В Oracle Forms для работы с элементами в режиме выполнения существуют три основные разновидности встроенных подпрограмм пакета STANDARTS:

1. процедуры SET_ – служат для установки свойств объектов;
2. функции GET_ – служат для получения свойств объектов;
3. функции FIND_ – служат для поиска идентификаторов объектов.

Процедуры установки свойств объектов

Процедуры установки значений предназначены для установки свойств объекта во время выполнения программы. В Oracle Forms существуют следующие разновидности процедур установки:

- SET_BLOCK_PROPERTY;
- SET_CANVAS_PROPERTY;
- SET_FORM_PROPERTY;
- SET_ITEM_PROPERTY;
- SET_LOV_PROPERTY;
- SET_MENU_ITEM_PROPERTY;
- SET_PARAMETER_ATTR;
- SET_RADIO_BUTTON_PROPERTY;
- SET_RECORD_PROPERTY;
- SET_RELATION_PROPERTY;
- SET_VIEW_PROPERTY;
- SET_WINDOW_PROPERTY.

Синтаксис процедуры SET_ имеет достаточно много вариаций, поэтому перечислим только основные.

```
Синтаксис процедуры SET_
SET_ITEM_PROPERTY  (item_id   ITEM, property  NUMBER, value
VARCHAR2);
SET_ITEM_PROPERTY  (item_name VARCHAR2, property NUMBER, value
VARCHAR2);
```

Каждая процедура в зависимости от объекта, с которым она ассоциирована, может принимать в качестве значений константы и различные переменные подстановки. Например, рассмотрим функцию SET_RECORD_PROPERTY.

SET_RECORD_PROPERTY (rec_num NUMBER, block_name VARCHAR2, property NUMBER, value NUMBER) – устанавливает свойства записи.

Параметры:

record_number – номер записи;

block_name – имя блока, содержащего целевую запись. Тип данных – VARCHAR2;

property – свойство, значение которого собираемся изменить, – это константа STATUS;

value – используйте возможное значение:

- CHANGED_STATUS;
- INSERT_STATUS;
- NEW_STATUS;
- QUERY_STATUS.

```
«Установка свойства WHERE Clause блока»
BEGIN
  Set_Record_Property (:system.cursor_record, 'EMP', STATUS,
INSERT_STATUS);
END;
```

Область применения этой процедуры очень большая, поэтому вы будете часто использовать ее в своих программах. Ниже приведено несколько примеров, в которых рассмотрены возможные способы применения этой процедуры.

1. «Замена группы записей для LOV» – в этом примере проверяется текущая группа (group_name1), на которой основан LOV, и если данная группа записей пустая, то мы подменяем ее другой группой (group_name2).

```
Пример 1. «Замена группы записей для LOV»
DECLARE
```

```
  lov_id LOV;
      rg_id RecordGroup;
BEGIN
rg_id:= Find_Group ('groupe_name1');
IF Get_Group_Row_Count( rg_id )=0 THEN
  lov_id:= Find_LOV('LOV1');
  IF Get_LOV_Property(lov_id,GROUP_NAME) = 'group_name1' THEN
     Set_LOV_Property(lov_id,GROUP_NAME,'group_name2');
  END IF;
END IF;
END;
```

2. «Установка свойства WHERE Clause блока» – в этом примере показано, как с помощью процедуры SET_ можно ограничить выборку. После того как свойство будет установлено, количество записей, которое может выбрать пользователь командой EXECUTE QUERY, ограничивается до десяти.

Пример 2. «Установка свойства WHERE Clause блока»

```
Set_Block_Property(block_name,DEFUALT_WHERE ,'rownum<=10');
```

3. «Стиль перемещения» – в этом примере показано, как, изменяя атрибут Navigation_Style, можно управлять обработкой следующего или предыдущего элемента, когда фокус находится в первом или доступном последнем элементе.

Пример 3. «Стиль перемещения»

```
Set_Block_Property('pzak',navigation_style,change_block);
```

4. «Сделать элемент недоступным» – используя константу ENABLED процедуры SET_, вы можете сделать элемент недоступным, то есть запретить перемещать фокус ввода и выполнять навигацию к нему.

Пример 4. «Сделать элемент недоступным»

```
IF :system.form_status='ENTER QUERY'
   THEN
   Set_item_Property('item_name',enabled, property_false);
END IF;
...
```

Функции получения свойств объектов

Функции GET_ предназначены для получения значений свойств объекта во время выполнения программы. В Oracle Forms существуют следующие разновидности рассматриваемых функций:

1. GET_BLOCK_PROPERTY.
2. GET_CANVAS_PROPERTY.
3. GET_FORM_PROPERTY.
4. GET_ITEM_PROPERTY.
5. GET_LOV_PROPERTY.
6. GET_MENU_ITEM_PROPERTY.
7. GET_MESSAGE.
8. GET_PARAMETER_ATTR.
9. GET_RADIO_BUTTON_PROPERTY.
10. GET_RECORD_PROPERTY.
11. GET_RELATION_PROPERTY.
12. GET_VIEW_PROPERTY.
13. GET_WINDOW_PROPERTY.

Синтаксис функции, как и в предыдущем случае, имеет достаточно много вариаций, поэтому перечислим основные.

Синтаксис процедуры GET_

```
GET_ITEM_PROPERTY  (item_id   ITEM, property  NUMBER) RETURN
VARCHAR2;
GET_ITEM_PROPERTY  (item_name VARCHAR2, property NUMBER) RETURN
VARCHAR2;
```

Использование функций типа GET_ дает возможность узнать текущие значения элементов в ходе выполнения программы, позволяя вам контролировать состояние объекта. Ниже приведены различные примеры применения этой функции.

```
BEGIN
Go_block(Get_Form_Property(curform,FIRST_BLOCK));
END;
```

Вы можете использовать функцию Get_Record_Property вместо системной переменной – :system.record_status.

```
BEGIN
IF Get_Record_Property(1,'item_name',STATUS) = 'NEW' THEN
Message('Введите запись');
RAISE Form_trigger_Failure;
END IF;
END;
```

Вы можете получить значение сообщения, отображающегося в статусной строке формы, используя функцию GET_MESSAGE.

```
DECLARE
mes VARCHAR2(200);
BEGIN
mes := Get_Message;
END;
```

Поиск идентификаторов объектов

FIND_ – эта функция предназначена для поиска идентификатора объекта, который характеризует внутреннее значение объекта, использующееся в Forms как внутреннее средство управления объектами.

Идентификатор – это значение, связываемое с объектом при его создании. Также стоит отметить, что использование идентификаторов объектов повышает производительность приложения.

Возвращаемое значение функции FIND_ зависит от класса искомого элемента (объекта). Ниже приведена таблица 18.1, где показаны все возможные вариации этой функции в зависимости от класса объекта, с которым она используется.

Таблица 18.1. Варианты функции FIND

Объект	Функция	Возвращаемый тип
Alert	FIND_ALERT	ALERT
Block	FIND_BLOCK	BLOCK
Canvas	FIND_CANVAS	CANVAS
Record Group Column	FIND_COLUMN GROUP	COLUMN
Editor	FIND_EDITOR	EDITOR

Form	FIND_FORM	FORM MODULE
Record Group	FIND_GROUP	RECORDGROUP
Item	FIND_ITEM	ITEM
List of Values	FIND_LOV	LOV
Menu Item	FIND_MENU_ITEM	MENU ITEM
Parameter List	GET_PARAMETER_LIST	PARAMLIST
Relation	FIND_RELATION	RELATION
Timer	FIND_TIMER	TIMER
View	FIND_VIEW	VIEWPORT

Используя функцию поиска FIND_, вы можете присвоить полученный идентификатор переменной, который можете применять для управления объектом, не обращаясь к нему по имени. Ниже приведен пример, который демонстрирует использование этой функции.

```
«Поиск идентификатора окна»
DECLARE
   id_var WINDOW;
BEGIN
   id_var:=Find_Window('my_window');
END;
```

В следующем примере показано, как с помощью функции FIND_ мы проверяем существование редактора; если таковой существует, то он будет отображен процедурой Show_Editor на экран.

```
«Поиск идентификатора редактора»
DECLARE
ed_id Editor;
status BOOLEAN;
BEGIN
ed_id := Find_Editor('Happy_Edit_Window');
IF NOT Id_Null(ed_id) THEN
Show_Editor(ed_id, NULL, :emp.comments, status);
ELSE
Message('Editor not found');
RAISE Form_trigger_Failure;
END IF;
END;
```

В этом примере перед отображением списка значений (LOV) на экран проверяется его существование.

```
«Поиск идентификатора LOV»
DECLARE
lv_id LOV;
status BOOLEAN;
BEGIN
lv_id := Find_LOV('My_Shared_LOV');
IF Id_Null(lv_id) THEN
lv_id := Find_LOV('My_Private_LOV');
END IF;
status := Show_LOV(lv_id,10,20);
END;
```

Программные единицы Oracle Forms

Эта тема довольно обширная, и ее рассмотрение достойно отдельной книги, поэтому наша задача – не столько погружение во все тонкости техники и стиля программирования, сколько базовое ознакомление с основными методологиями, которое даст вам отправную точку в изучении построения методов и в Oracle Forms, и в PL/SQL.

Разрабатывая приложения, вы неоднократно сталкиваетесь с повторным использованием кода или его фрагментов. Подпрограммы изначально появились как средство оптимизации – они позволили не повторять в программе идентичные блоки кода. Подпрограммы улучшают читаемость программы и ее сопровождение за счет структуризации кода и логического выделения целостной задачи. Используя подпрограммы, вы не только избавляетесь от лишней работы, но и экономите время на отладке и модификации кода, так как изменяете не весь код, а конкретную подпрограмму.

Подпрограмма (*subprogram*) – это именованная часть программы, содержащая описание определенного набора действий. Подпрограмма может быть многократно *вызвана* из разных частей программы. Различают два вида подпрограмм: функции и процедуры.

Мы с вами рассмотрим различные виды программных структур, представленных в Oracle Forms, а именно:

- Пакет
- Функция
- Процедура

Любая программная структура, созданная в Oracle Forms, не доступна в Базе Данных, она доступна только в модуле, в котором создана.

Функция (Function)

Функция – это один из видов подпрограммы, на вход которой поступают значения в виде аргументов различных типов данных, а на выходе получается результат ее выполнения. Результат, возвращаемый функцией, может быть отличным от входных параметров. Отличие функции от другого вида подпрограмм – процедуры – состоит в том, что функция возвращает значение, а ее вызов может использоваться в программе как выражение.

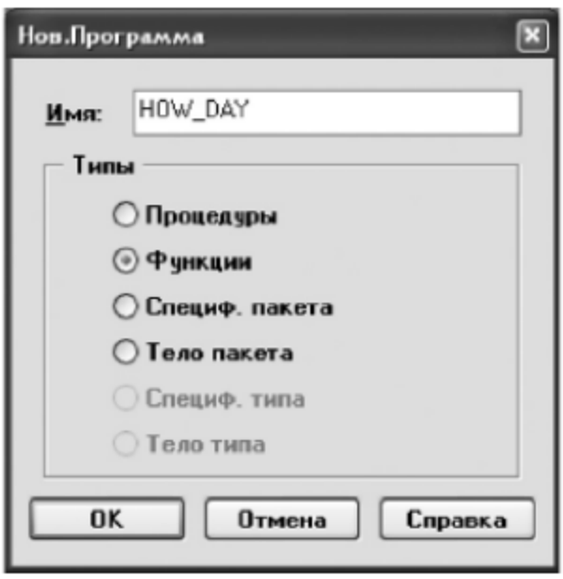

Рис. 18.1. Окно «Новая программа»

Для того чтобы ознакомится с этим понятием поближе, выполним упражнение, в котором создадим три функции, выполняющие операции с датой:

- HOW_DAY – возвращает количество недель определенного дня недели в текущем месяце.
- DAYS_IN_MONTH – возвращает количество дней в месяце.
- FIRST_DAY – возвращает первый день месяца.

1. Находясь в навигаторе объектов, выберите узел «Программы», затем выберите пункт меню Навигатор | Создать.
2. В окне «Программы» (рис. 1) введите имя первой функции – HOW_DAY и выберите радиокнопку «Функции». Подтвердите выбор для вызова PL/SQL-редактора. Вы можете ввести любое имя функции за исключением зарезервированных слов. Имя функции, указанное в окне «Программы», не окончательное, так как вы всегда можете его изменить на этапе написания функции. Поле «Имя» должно быть обязательно заполнено.

3. В появившемся PL/SQL-редакторе уже содержится небольшой фрагмент кода – это конструкция функции (рис. 18.2.).

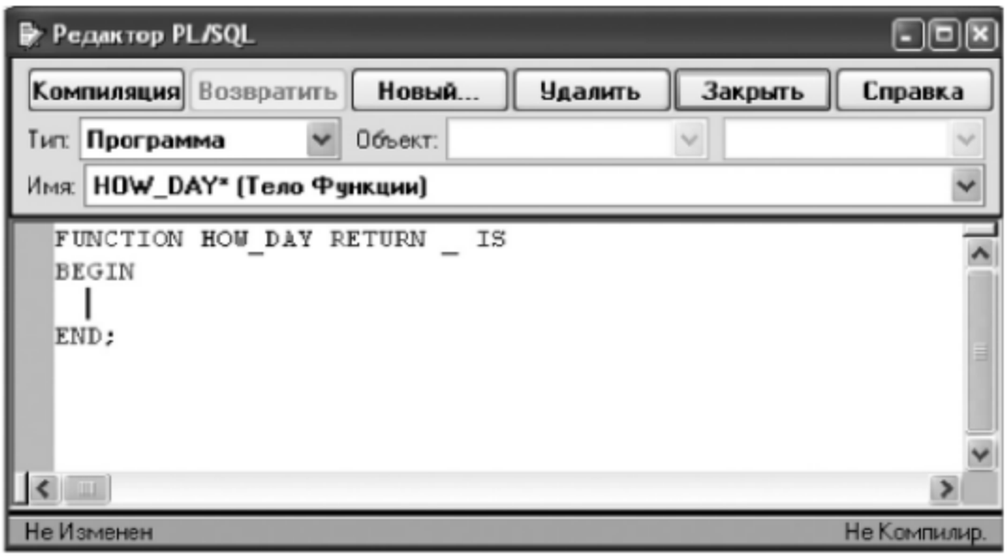

Рис. 18.2. Окно редактора PL/SQL. Конструкция функции

4. Фраза «RETURN _IS» – указывает на тип возвращаемого значения функцией. Вместо подчеркивания вы должны вставить тип возвращаемого значения, в нашем случае это NUMBER. То есть теперь фраза должна выглядеть как «RETURN NUMBER IS».
5. Ниже приведен листинг функции HOW_DAY.

```
«HOW_DAY»
FUNCTION   "HOW_DAY" (
        dai varchar2
                          )
return number
         as cnt number;
begin
Select Count (a.h_day) INTO cnt From
(Select Rtrim (To_char
                  (To_date ( Rownum|| '-'||
                    To_char (Sysdate, 'mon-yy')),
'DAY'
  )
    ) As h_day From user_all_tables
    Where Rownum Between 1 And To_char (Last_day (Sysdate), 'DD')) a
Where a.h_day = upper(dai);
return cnt;
end;
```

6. Наберите этот код и скомпилируйте функцию.

Создайте самостоятельно вторую функцию – DAYS_IN_MONTH. В теле функции напишите и скомпилируйте следующий код:

```
«DAYS_IN_MONTH»
FUNCTION            "DAYS_IN_MONTH" (
    val IN DATE

  )
RETURN NUMBER
IS
Sum_date number := 0 ;
cnt number:=0;
BEGIN
  FOR i IN  1 ..31 LOOP
   IF TO_CHAR(to_date(i||substr(val,3)), 'DAY') = 'SUNDAY' THEN
     Sum_date := Sum_date+ 1 ;
     cnt:=sum_date;
   END IF ;
     END LOOP ;
  RETURN cnt ;
END ;
```

Создайте заключительную, третью функцию – FIRST_DAY. В теле функции напишите и скомпилируйте следующий код:

```
«First_day»
FUNCTION "First_day" (val date) return date
is
begin
return to_date('01'||substr(val, 3));
end First_day;
```

Примечание: *созданные функции и процедуры могут дублировать имена хранимых программ в базе данных, так как доступны только в области видимости Forms, но при этом в пределах одного модуля имена подпрограмм должны быть уникальны, как все остальные объекты.*

Обращение к функциям в программе

Функции, созданные в Oracle Forms, не могут участвовать в SQL-запросах напрямую. Обойти это ограничение можно, передав значение функции какой-либо переменной, которую затем можно подставить в запрос. Например, следующий PL/SQL-блок вызовет ошибку компиляции – «Функция HOW_DAY не может быть использована в SQL»:

```
DECLARE
Val number;
BEGIN
SELECT HOW_DAY('MONDAY') INTO Val FROM dual;
END;
```

Для того чтобы этой ошибки не возникало, следует поступить следующим образом:

```
DECLARE
Val number;
BEGIN
Val:=HOW_DAY('MONDAY');
END;
```

Если вам необходимо использовать значение функции в условии запроса, то вам нужно сначала присвоить это значение какой-либо переменной:

```
DECLARE
Val varchar2(20);
BEGIN
Val:=FIRST_DAY(SYSDATE);
SELECT col_name INTO Val FROM table_name where day=val;
/* Если же вы напишете: SELECT col_name INTO Val FROM table_name
where day= FIRST_DAY(SYSDATE) - это вызовет такую же ошибку, как и
в первом случае */
END;
```

Процедура

Процедура – это любая подпрограмма, которая не является функцией. Вы можете оборачивать в процедуру любую типовую задачу. Как уже было сказано ранее, процедура в отличие от функции не возвращает значение и не может быть использована как выражение, поэтому исполняется как отдельный оператор. Можно привести множество примеров, которые продемонстрируют, как заменить блок кода одной строкой. Например, если перед вами стоит задача запретить удаление строки, и если она единственная, то вместо того чтобы несколько раз писать один и тот же блок кода, можно обернуть его в процедуру.

Листинг 18.1. Процедура «DONT_DELETE_LAST_RECORD»

```
PROCEDURE DONT_DELETE_LAST_RECORD
 IS
BEGIN
IF :System.Last_Record = 'TRUE' AND :System.Cursor_Record ='1'
THEN
Message('Последняя запись не может быть удалена');
ELSE
Delete_Record;
END IF;
END;
```

Теперь, когда мы создали процедуру DONT_DELETE_LAST_RECORD, для выполнения этого ограничения в каждом блоке вам достаточно будет написать всего одну строку.

Листинг 18.2. Пример вызова процедуры

```
BEGIN
DONT_DELETE_LAST_RECORD
END;
```

Вы также можете передавать параметры в процедуру, причем параметры могут быть различного типа. Для примера рассмотрим процедуру, которая создает таблицу из N столбцов.

Листинг 18.3. Пример вызова процедуры

```
PROCEDURE N_Table (tab_name varchar2, n NUMBER) IS
my_ddl VARCHAR2(2000);
BEGIN
my_ddl := 'create table '||tab_name||'(COL1 NUMBER';
```

```
FOR I in 2..N LOOP
my_ddl := my_ddl||',COL'||TO_CHAR(i)||' NUMBER';
END LOOP;
my_ddl := my_ddl||')';
Forms_DDL(my_ddl);
IF NOT Form_Success THEN
Message ('Таблица не создана');
ELSE
Message ('Таблица создана');
END IF;
END;
```

Создание тела и спецификации пакета

Пакет – это набор процедур, функций и переменных, объединенных с целью их совместного использования в приложении. При создании пакета создается интерфейсная часть – спецификация пакета и часть пакета, называемая «Телом». В нашем примере мы объединим все ранее созданные подпрограммы в единую структуру – пакет. Для начала мы создадим спецификацию пакета, в которой будут определены все подпрограммы.

Спецификация пакета

```
PACKAGE "MY_PKG" AS
function HOW_DAY (dai varchar2) return number;
function DAYS_IN_month (val IN date) return number;
function first_day(val date) return date;
PROCEDURE N_Table (tab_name varchar2, n NUMBER);
PROCEDURE DONT_DELETE_LAST_RECORD;
END;
```

Тело пакета

```
PACKAGE BODY "MY_PKG" AS
function First_day (val date) return date
is
begin
return to_date('01'||substr(val, 3));
end First_day;

FUNCTION            DAYS_IN_MONTH (
    val IN DATE
  )
```

```
RETURN NUMBER
IS
  LN number := 0 ;
s number:=0;
BEGIN
  FOR i IN 1..31 LOOP
   IF TO_CHAR(to_date(i||substr(val,3)), 'DAY') = 'SUNDAY' THEN
     LN := LN+ 1 ;
     s:=ln;
   END IF ;
     END LOOP ;
  RETURN s ;
END ;

FUNCTION   "HOW_DAY" (
       dai varchar2
                        )
return number
         as val number;
begin
Select Count (a.L_day) INTO val From
(Select Rtrim (To_char
                 (To_date ( Rownum|| '-'||
                   To_char (Sysdate, 'mon-yy')),
'DAY'
  )
    ) As L_day From user_all_tables
    Where Rownum Between 1 And To_char (Last_day (Sysdate), 'DD')) a
Where a.L_day = upper(dai);
RETURN val;
END;

PROCEDURE "N_Table" (tab_name varchar2, n NUMBER) IS
my_ddl VARCHAR2(2000);
BEGIN
my_ddl := 'create table '||tab_name||'(COL1 NUMBER';
FOR I in 2..N LOOP
my_ddl := my_ddl||',COL'||TO_CHAR(i)||' NUMBER';
END LOOP;
my_ddl := my_ddl||')';
Forms_DDL(my_ddl);
IF NOT Form_Success THEN
```

```
Message ('Таблица не создана');
ELSE
Message ('Таблица создана');
END IF;
END;
PROCEDURE DONT_DELETE_LAST_RECORD
 IS
BEGIN
IF :System.Last_Record = 'TRUE' AND :System.Cursor_Record ='1' THEN
Message('Последняя запись не может быть удалена');
ELSE
Delete_Record;
END IF;
END;
END;
```

Вы можете расширять пакет, добавляя в него новые процедуры и функции.

Обращение к подпрограммам пакета

Для того чтобы вызвать подпрограмму пакета, необходимо перед вызовом подпрограммы указать имя пакета.

```
Спецификация пакета
DECLARE
Vdmonth number;
Vfday varchar2(20);
BEGIN
Message(my_pkg.HOW_DAY ('FRIDAY') return number;
Vdmonth:= my_pkg.DAYS_IN_MONTH (sysdate);
Vfday:= my_pkg.first_day(sysdate);
my_pkg.N_Table ('test',3);
my_pkg.DONT_DELETE_LAST_RECORD;
END;
```

Создание эффекта отмены действий в элементе

В Oracle Forms, если пользователь вошел в запись и изменил ее значение, он не может отменить свое действие, не перезапросив данные.

Перезапросить данные – хоть и самый простой способ, но не самый лучший и не единственный выход из ситуации. Рассмотрим еще два способа:

- Хранение первоначального значения в параметре.
- Использование константы DATBASE_VALUE.

Хранение первоначального значения в параметре

Для того чтобы выполнить это упражнение, создайте или откройте любую существующую форму и сохраните ее под именем «Undo».

1. Находясь в Навигаторе Объектов, перейдите в узел «Параметры» и создайте два параметра.
2. Установите для первого параметра следующие свойства:
   - Имя: old_value.
   - Тип данных: Varchar2.
   - **Размер: 200.**
3. Установите для второго параметра следующие свойства:
   - Имя: block_item.
   - Тип данных: Varchar2.
   - Размер: 200.
4. Определите на уровне блока триггер WHEN-NEW-ITEM-INSTANCE и напишите в теле триггера нижеприведенный код:

```
WHEN-NEW-ITEM-INSTANCE
:parameter.old_value:=:system.current_value;
IF Get_Item_Property(:system.current_item, ITEM_TYPE)
          IN ('DISPLAY ITEM', 'CHECKBOX', 'LIST',
               'RADIO GROUP',  'TEXT ITEM')
    AND Get_Item_Property ((:system.current_item, BASE_TABLE) = 'TRUE'
    THEN
:parameter.block_item:=:system.trigger_item;
ELSE
:parameter.block_item:=null;
END IF;
```

5. Определите какой-нибудь триггер для отмены изменений в элементе и напишите в нем следующий код:

```
Триггер отмены
DECLARE
Cur_val varchar2(200):=:system.current_value;
BEGIN
IF :SYSTEM.RECORD_STATUS = 'CHANGED'
```

```
            THEN
    IF :parameter.old_value<>cur_val AND :parameter.block_item IS
NOT NULL
            THEN
COPY(:parameter.old_value, :parameter.block_item)
     END IF;
END IF;
END;
```

Запустите форму для просмотра результатов, попробуйте изменить значение и вернуть предыдущее. Этот пример действует только в пределах текущего элемента, то есть вы можете отменить изменение только в одном элементе.

Процедура UNDO

Процедура UNDO предназначена для отмены всех сделанных изменений в блоке без перезапроса данных. В этой процедуре определяется статус записи, и если она находится в режиме редактирования, то ее значение будет восстановлено на первоначальное.

Листинг 18.4. Процедура «UNDO» (вы можете использовать эту в процедуру в триггере отмены действий)

```
PROCEDURE Undo IS
  V_Block  VARCHAR2 (30) := :SYSTEM.CURSOR_BLOCK;
  V_Field  VARCHAR2 (61);
  V_Item   VARCHAR2 (61);
BEGIN
  Validate (Item_Scope);
  IF :SYSTEM.RECORD_STATUS = 'CHANGED' THEN
    V_Field := Get_Block_Property (V_Block, FIRST_ITEM);
    V_Item := V_Block || '.' || V_Field;
    WHILE V_Field IS NOT NULL
    LOOP
      IF Get_Item_Property (V_Item, ITEM_TYPE)
         IN ('DISPLAY ITEM', 'CHECKBOX', 'LIST',
             'RADIO GROUP',  'TEXT ITEM')
      AND Get_Item_Property (V_Item, BASE_TABLE) = 'TRUE'
      THEN
        COPY (Get_Item_Property (V_Item, DATABASE_VALUE),
              V_Item);
      END IF;
```

```
        V_Field := Get_Item_Property (V_Item, NextItem);
        V_Item := V_Block || '.' || V_Field;
      END LOOP;
    END IF;
  END;
```

Запустите форму для просмотра результатов, попробуйте изменить значение нескольких элементов, затем попробуйте вернуть им первоначальные значения с помощью процедуры UNDO.

Лекция 19. Блоки на основе FROM CLAUSE

В этой лекции слушатель научится создавать новый источник данных для блока, а также ознакомится с конвейерными (pipelined) функциями, которые будут рассматриваться как один из возможных источников.

Ключевые слова: FROM CLAUSE, Data Source.

Цель лекции: слушатели научатся создавать блоки данных на основе фразы FROM CLAUSE и конвейерных функций.

Ранее мы рассматривали создание блоков на основе таблиц, представлений, синонимов. В принципе это достаточный набор объектов, так как для реализации иерархических запросов JOIN (cross, natural, inner, outer и т.д.) или запросов с группировкой вам достаточно будет создать блок на основе представления. А что делать, если у разработчика нет привилегии на создание представления? Или необходимо сделать блок на основе функции? Ответом на эти вопросы как раз и является свойство блока Query Data Source Type — FROM CLAUSE.

Свойства для работы с FROM CLAUSE

Для начала определимся, что такое FROM CLAUSE.

FROM CLAUSE используется в качестве источника данных (data source) для блока, позволяющего использовать вложенные операторы SELECT в конструкции FROM. Значение, возвращаемое FROM Clause, – это подмножество записей первоначального (original query) запроса. В основном FROM CLAUSE используется для реализации соединения (joins), связывания (lookups), вычисления и агрегации без создания представления на сервере, поэтому FROM CLAUSE может также применяться как прототип представления и повысить производительность вашего приложения.

> ***Примечание:*** *блоки, основанные на FROM CLAUSE, – это блоки типа Query Only, то есть блоки, позволяющие выполнять только запросы.*

Перейдем к практической части, в которой мы выполним два несложных упражнения:

- создание блока на основе запроса с группировкой;
- **с** оздание блока на основе Pipelined Function.

Блок на основе запроса с группировкой

Для начала рассмотрим пример с созданием блока на основе запроса с группировкой.

1. Создайте таблицу NUMB:

```
create table numb
             (A number);
```

2. Заполните таблицу значениями:

```
begin
   for i in 1..10
      loop
      insert into NUMB values(i);
        end loop;
        end;
```

3. Измените таблицу так, чтобы появились дубли:

```
update NUMB set A=4 where A=7;
update NUMB set A=5  where A=6;
```

4. Проверьте содержимое таблицы:

```
select * from NUMB;
          A
----------
          1
          2
          3
          4
          5
          5
          4
          8
          9
         10

10 rows selected.
```

5. Выполните запрос с группировкой, который будет источником для блока:

```
select A from numb group by a;
         A
----------
         1
         2
         3
         4
         5
         8
         9
        10

8 rows selected.
```

Теперь, когда все готово для выполнения примера, создадим новую форму FROM_1. Создайте блок данных вручную. Выберите блок в объектном навигаторе и вызовите палитру свойств. Установите свойства блока следующим образом:

- Query Data Source Type – определяет тип источника данных для блока. Установите этот параметр на FROM Clause.
- Query Data Source Name – текст вложенного запроса (nested query). В этом свойстве введите текст запроса:

```
select A from numb group by a
```

- Свойство Database item установите равным «Yes».
- Name – значение этого свойства установите равным «Num».

Создайте одну кнопку и один текстовый элемент с параметрами:

- Свойство Database item установите равным «Yes» (иначе получите ошибку unable to perform query).
- Column name – это свойство установите равным A (имя должно совпадать с именем столбца в запросе, иначе также получите ошибку unable to perform query).

Создайте триггер WHEN-BUTTON-PRESSED:

```
go_block ('NUMB');
execute_query;
```

Мы рассмотрели очень простой, но показательный пример, который поможет вам понять назначение FROM CLAUSE и способы его использования в реальных и более сложных задачах.

Создание блока на основе Pipelined Function

Данный пример практически ничем не будет отличаться от предыдущего, за тем исключением, что в качестве источника данных мы возьмем функцию, возвращающую массив данных. В примере мы рассмотрим функцию с параметрами, чтобы вы могли ясно представить, какие возможности может вам дать использование данного типа источника. Выполнение примера можно разбить на две части:

- написание функции;
- создание формы.

Приступим к первой части.

Создадим простую Pipelined-функцию, возвращающую набор данных в виде последовательности чисел:

```
create type «virtual_table_type» as
    table of number;

create or replace  function «pipe_table»  (p_num_rows in number,
    l_order varchar2)
             return virtual_table_type
           pipelined
                is
           type t_data is ref cursor;
           c_data t_data;
    cursor c1 is select * from numb;
    c2 c1%rowtype;
begin
 open c_data for 'select * from numb order by'||l_order;
for i in 1 .. p_num_rows
loop
fetch c_data into c2;
 pipe row(c2.a);
 end loop;
 return;
end pipe_table;

select * from table(vt(5,' a desc'));
```

```
COLUMN_VALUE
------------
          10
           9
           8
           5
           5
```

Функция готова, теперь можно перейти к следующему шагу. Но прежде чем начать, давайте рассмотрим еще один небольшой пример, чтобы не ограничиваться отображением данных из Pipelined-функции и показать, как можно другому блоку назначить новый источник данных.

1. К существующей форме добавьте еще один базовый блок на таблицу Numb с одним текстовым полем A. В блоке NUM измените свойство Query Data Source Name на select * from table(vt(5,' a desc')).
2. Добавьте в триггер WHEN-BUTTON-PRESSED следующие строки:

```
go_block('NUM');
execute_query;
go_block('NUMB');
set_block_property('NUMB', query_data_source_type,
get_block_property( 'NUM', query_data_source_type);
set_block_property('NUMB', query_data_source_name,
get_block_property( 'NUM', query_data_source_name);
execute_query;
```

Запустите форму на выполнение. В результате у вас в обоих элементах должны отобразиться одинаковые данные. Этот простой пример должен помочь вам закрепить ранее выполненное упражнение, и вы сможете применить данную функциональность в своих приложениях. Приведем простой пример, когда, имея форму с одним блоком данных, вы хотите просматривать набор не только отсортированных данных или ограниченных условием WHERE Clause, но и сгруппированных. Вам не придется создавать много представлений на сервере – достаточно создать несколько блоков с различными источниками данных и назначать их основному при нажатии кнопки.

Лекция 20. Сообщения, предупреждения и таймеры

В этой лекции слушатель научится создавать предупреждения и сообщения, управлять их свойствами и отображением. Также в этой лекции будет рассказано, как создавать таймеры, обрабатывать их события, и будет рассмотрен пример создания системы оповещения с помощью таймера.

Ключевые слова: Alert, предупреждение, создание предупреждения, динамические предупреждения, Timer, Iterate, Expired, создание таймера, удаление таймера, система оповещения.

Цель лекции: слушатели научатся создавать и обрабатывать предупреждения и таймеры.

Предупреждение (**Alert**) – это модальное окно, которое отображает сообщение, предупреждающее оператора о каком-либо событии в приложении.

Используйте предупреждения для вывода дополнительной информации, различных информативных и функциональных сообщений, требующих от пользователя подтвержения или выполнения какого-либо другого действия.

В Oracle Forms вы можете работать с тремя типами предупреждений: «Стоп», «Предупреждение» и «Замечание» – все эти типы можно установить в таблице свойств этого объекта. У каждого типа предупреждения свой уровень строгости. Для каждого типа в таблице свойств вы можете определить три кнопки с нужными вами метками, по умолчанию это «Да» (Кнопка1), «Стоп» (Кнопка2), «Отмена» (Кнопка3). Сообщение, отображаемое предупреждением, задается в свойстве «Сообщение».

При создании Alert вам также необходимо написать триггер для вывода этого предупреждения, а также указать основную информацию, такую как сообщение, которое вы бы хотели в нем выводить.

Далее мы подробнее рассмотрим создание и отображение Предупреждений.

Создание предупреждения

В этом разделе мы пошагово рассмотрим создание сообщения на этапе проектирования.

1. В объектном навигаторе выберите узел предупреждения (Alerts).
2. Создайте кнопку на форме и назовите ее «Расчет».
3. Вызовите таблицу свойств Alert двойным щелчком по иконке.
4. Установите стиль Alert, отвечающий вашим требованием, выберите «Стоп», «Предупреждение» или «Замечание».

5. Во время выполнения формы рядом с сообщением в окне сигнала будет выводиться иконка, представляющая выбранный вами стиль.
6. В окне свойства «Сообщение» наберите следующий текст: «Вывести ли дополнительно информацию по накладной за каждый месяц?» – именно это сообщение отобразится при выводе предупреждения. В редакторе «Сообщение» этого свойства можно ввести не более 200 символов (этот показатель зависит от используемой платформы).

   Примечание: *старайтесь писать сообщения как можно более сжато и осмысленно, так как большие сообщения будут занимать много места на экране и к тому же время вывода такого окна увеличится.*

7. Определите одну или более кнопок для вашего предупреждения путем ввода текста в значения свойств «Метка Кнопки1», «Метка Кнопки2», «Метка Кнопки3» либо оставьте значения по умолчанию: «Да», «Стоп», «Отмена».

   Примечание: *хотя бы одна кнопка в предупреждении должна иметь именованную метку. Кнопки, не имеющие своих меток, отображаться не будут. При выводе «Предупреждения» кнопки будут располагаться в таком же направлении, как и в таблице свойств.*

8. Далее выберите кнопку, которая будет активной по умолчанию при каждом запуске вашего предупреждения.

В этом разделе мы с вами детально рассмотрели все шаги создания «предупреждения». В следующем разделе рассмотрим один из способов программного отображения сообщения на экран.

Отображение Предупреждения

Для отображения предупреждения ваша PL/SQL-программа должна содержать встроенную функцию Forms – SHOW_ALERT, которая возвращает цифровую константу, указывающую, на какую из трех кнопок вы нажали. В программе эти кнопки представлены в виде констант численного типа: ALERT_BUTTON1, ALERT_BUTTON2, ALERT_BUTTON3. Порядковый номер констант соответствует номеру метки кнопки. В вашем PL/SQL-коде вызов предупреждения должен выглядеть следующим образом.

Синтаксис функции отображения сообщения:

```
Show_Alert (alert_name in varchar2) Return NUMBER;
```

Теперь перейдем к выводу предупреждения.

1. Перейдите в Навигатор, затем выделите созданную вами ранее кнопку «Расчет» и вызовите триггер WHEN_BUTTON_PRESSED.
2. В теле триггера напишите следующий пример:

```
When-Button-Pressed Trigger:

declare
  a_lert  number;
 begin
  a_lert:=Show_alert('Change');
 end;
```

Обратите внимание на то, что функция не вызывается таким образом:

```
 begin
  Show_alert('Change');
 end;
```

Редактор PL/SQL, естественно, сразу выдаст вам ошибку – error 221: «Show_Alert не является процедурой или не определено», поэтому вызывать предупреждение нужно так, как это было показано в предыдущем примере, то есть через присваивание, т. к. это функция.

3. Теперь запустите форму и вызовите предупреждение с помощью созданной кнопки. Поскольку мы не инициировали какие-либо действия при нажатии той или иной кнопки в окне предупреждения, нажатие любой из двух кнопок приведет к закрытию предупреждения.
4. Расширим код нашего триггера, добавим к нему обработку кнопок:

```
When-Button-Pressed Trigger:
 declare
        a_lert  number;
 begin
        a_lert:=Show_alert('Change');
                  IF alert_button = ALERT_BUTTON1
                  THEN  Go_Block('Расчет');
      END IF;
end;
```

Запустите форму и вызовите предупреждение. Теперь при нажатии на кнопку «Да» вы перейдете в блок «Расчеты». Поскольку мы написа-

ли в триггере, что выбранная кнопка – ALERT_BUTTON1, необходимо выполнить навигацию к указанному блоку. Если же вы выбрали второй вариант, то есть «Нет», то предупреждение просто закроется, причем это действие мы не инициировали, т. к. вы уже знаете, что оно выполняется по умолчанию.

Мы рассмотрели наиболее актуальные примеры с использованием вызова предупреждений, которые научили вас не только выводить дополнительную информацию, но и инициировать различные события при выборе одного из предложенных вариантов. Теперь давайте рассмотрим пример, в котором некоторые свойства предупреждения будут меняться в ходе выполнения программы.

Set_Alert_Property – устанавливает значение свойства предупреждения.

Синтаксис:

```
Set_Alert_Property (alert_name, alert_property, property_value);
```

Пример:

```
DECLARE
    a_lert NUMBER;
BEGIN
Set_Alert_Property ('Change', ALERT_MESSAGE_TEXT,
'Сегодня '||TO_CHAR (SYSDATE));
    a_lert:= Show_Alert ('Change');
END;
```

При срабатывании этого триггера выводится предупреждение, отображающее текущую дату.

Свойство Set_Alert_Property очень полезно, т. к. программно, изменяя сообщение сигнала, вы можете неплохо сэкономить, повторно используя один и тот же объект предупреждения вместо создания нескольких. Также это дает вам возможность создавать динамические сообщения, которые вы можете менять в зависимости от ситуации, текущего объекта, выбранного элемента или его значения.

Таймер

Таймер (Timer) – это средство измерения времени в Oracle Forms. Таймер начинает работать с момента своего создания и работает до тех пор, пока не закроется приложение или таймер не будет удален. Если задать временной интервал, то по истечении заданного времени таймер

вызовет событие WHEN-TIMER-EXPIRED и начнет отсчет времени заново. Таймером очень хорошо пользоваться для непрерывного отслеживания каких-либо действий. Вы также можете использовать таймер как планировщик, закрывая формы по истечении указанного времени, или, например, для выполнения автосохранения в форме.

Программное управление таймером

Oracle Forms позволяет создавать таймер только программным путем, то есть вы не можете создать таймер в навигаторе объектов. Вы можете удалять таймер и устанавливать его характеристики программно, используя встроенные подпрограммы. Ниже перечислены процедуры и функции для работы с таймером:

- CREATE_TIMER;
- DELETE_TIMER;
- SET_TIMER;
- FIND_TIMER.

Создание таймера

CREATE_TIMER – создает таймер с заданным числом миллисекунд и возможностью повторного срабатывания. Возвращает числовой идентификатор таймера.

Синтаксис:

```
CREATE_TIMER (TIMER_NAME IN varchar2, MILLISECONDS IN
number, ITERATE IN number) RETURN FORMS4C.TIMER;
```

- TIMER_NAME – определяет имя таймера;
- MILLISECONDS – определяет количество миллисекунд, по истечении которых таймер срабатывает, вызывая триггер WHEN-TIMER-EXPIRED;
- ITERATE – определяет количество срабатываний таймера. Это свойство принимает два параметра. Если в ITERATE задано 1000, значит, триггер сработает через секунду.

1. REPEAT – при указании этой константы триггер WHEN-TIMER-EXPIRED будет срабатывать каждый раз по истечении времени, указанного в ITERATE, до тех пор, пока таймер не будет удален или форма не будет закрыта.
2. NO_REPEAT – при указании этой константы триггер WHEN-TIMER-EXPIRED сработает всего один раз по истечении времени, указанного в ITERATE.

Пример:

```
DECLARE
TimerId Timer;
   BEGIN
            TimerId:= Find_Timer ('T_Timer');
             IF NOT Id_Null (TimerId) THEN
                Delete_Timer (TimerId);
             END IF;
            TimerId:= Create_Timer ('T_TIMER', 1000, NO_REPEAT);
   END;
```

Удаление таймера

DELETE_TIMER – удаляет таймер, принимая в качестве параметра его идентификатор, полученный при создании командой CREATE_TIMER, или его имя.

Синтаксис возможный процедур DELETE_TIMER:

- DELETE_TIMER(TIMER_NAME IN varchar2);
  – TIMER_NAME – определяет имя таймера;
- DELETE_TIMER(TIMER_ID IN FORMS4C.TIMER);
  – TIMER_ID – идентификатор, полученный при создании командой CREATE_TIMER.

Пример:

```
1. DELETE_TIMER('T_Timer');
2. DECLARE
TimerId Timer;
   BEGIN
            TimerId:= Find_Timer ('T_Timer');
             IF NOT Id_Null (TimerId) THEN
                Delete_Timer (TimerId);
             END IF;
```

Установка характеристик таймера

SET_TIMER – устанавливает значения свойств таймера, таких как временной интервал и количество итераций.

Синтаксис:

SET_TIMER (TIMER_NAME IN varchar2 [TIMER_ID IN FORMS4C.TIMER], MILLISECONDS IN number, ITERATE IN number) RETURN FORMS4C.TIMER;

- TIMER_NAME – определяет имя таймера;
- TIMER_ID – идентификатор, полученный при создании командой CREATE_TIMER;
- MILLISECONDS – определяет количество миллисекунд, по истечении которых таймер срабатывает, вызывая триггер WHEN-TIMER-EXPIRED;
- ITERATE – определяет количество срабатываний таймера. Это свойство принимает два параметра:
  - REPEAT – при указании этой константы триггер WHEN-TIMER-EXPIRED будет срабатывать каждый раз по истечении времени, указанного в ITERATE, до тех пор, пока таймер не будет удален или форма не будет закрыта;
  - NO_REPEAT – при указании этой константы триггер WHEN-TIMER-EXPIRED сработает всего один раз по истечении времени, указанного в ITERATE.

Создание системы оповещения

В этом примере будет показано, как можно с помощью таймера быстро создать несложную систему оповещения. Для наглядности примера мы создадим две формы – форму-оповещение (Client_1) и форму-приемщик (Client_2).

Предварительная подготовка

Создайте тестовую таблицу T_taime для демонстрации примера:

```
CREATE TABLE t_taime (
Mes_str varchar2(100),
Prz number);
```

1. Создайте новую форму и сохраните ее как Client_1.fmb.
2. Создайте Блок Данных на основе таблицы T_taime.
3. Для элемента Mes_str установите свойство Prompt – «Сообщение клиента 1», а для PRZ – «Отправить сообщение». Для элемента PRZ установите следующие свойства:
   - Тип элемента: Checkbox (переключатель).
   - Метка: «Отправить сообщение».
   - Значение при выборе: 1.

- Значение при отмене выбора: 0.
- Присвоение Выключателю значение Другие: Выключен (Unchecked).

4. Создайте триггер WHEN-CHECK-BOX-CHANGED для элемента PRZ. В теле триггера напишите следующий код:

```
WHEN-CHECK-BOX-CHANGED
commit_form;
```

Форма оповещения готова, теперь перейдем к созданию второй формы, которая будет принимать сигнал оповещения и выполнять какое-либо действие.

1. Создайте новую форму и сохраните ее как Client_2.fmb.
2. Создайте блок данных на основе таблицы T_taime. В свойстве WHERE CLAUSE укажите условие PRZ=1.
3. Для элемента Mes_str установите свойство Prompt – «Сообщение клиента 1». Элемент PRZ разместите на NULL-канве, то есть оставьте на канве только один элемент.
4. Создайте триггер WHEN-NEW-FORM-INSTANCE на уроне формы и напишите в нем следующий код:

```
WHEN-NEW-FORM-INSTANCE
DECLARE
    timer_id Timer;
    one_second NUMBER:= 1000;
BEGIN
timer_Id:= Find_Timer ('mes_timer');
               IF NOT Id_Null (Timer_Id) THEN
                  Delete_Timer (Timer_Id);
               END IF;
 timer_id:= CREATE_TIMER('mes_timer', one_second, REPEAT);
END;
```

5. Создайте параметр OLD_REC_CNT и установите нулевое начальное значение.
6. Создайте триггер WHEN-TIMER-EXPIRED на уровне формы. В теле триггера напишите следующий код:

```
WHEN-TIMER-EXPIRED
DECLARE
    rec_cnt number;
BEGIN
      SELECT count(*) INTO rec_cnt FROM t_ime;
```

```
IF :parameter.old_rec_cnt<>rec_cnt  THEN
execute_query;
END IF;
:parameter.old_rec_cnt:=rec_cnt;
END;
```

Для того чтобы проверить выполненный нами пример, запустите две формы одновременно и расположите их на экране так, чтобы было видно два окна. Перейдите в форму Client_1, введите новую запись и включите переключатель, после чего в форме Client_2 через секунду появится новая запись. В триггере WHEN-TIMER-EXPIRED мы отслеживаем изменение количества записей в таблице, поэтому если запись была выключена из списка передаваемых значений, то она также будет выключена из списка полученных сообщений. Запись исключается из общего списка передаваемых сообщений отменой выбора значения переключателя.

Лекция 21. Работа с отчетами в Oracle Forms

В этой лекции слушатель немного ознакомится со средой создания отчетов Oracle Reports, научится запускать отчеты и передавать параметры из приложения в отчет. В лекции будут рассмотрены различные способы запуска отчетов.

Ключевые слова: Report, Report Object, создание отчетов, запуск отчетов, скрытие URL.

Цель лекции: слушатели научатся запускать отчеты из Oracle Forms и передавать параметры из Forms в Reports.

Oracle Reports Developer (Построитель отчетов) – это мощное средство для проектирования отчетов в составе среды Oracle Developer. Оно позволяет структурировать и форматировать информацию на основе разных стилей, как из базы данных, так и из файловой системы, а также комбинировать ее с текстом и графикой для представления в отчетах на бумаге и в веб-среде, используя Oracle AS Services. Oracle Reports предполагает создание отчетов, работающих как в архитектуре «клиент-сервер», так и в Web, для построения которых можно использовать Java-апплеты или Java-скрипты. Также вы получаете возможность получения различных типов выходных отчетов в форматах Adobe Acrobat Reader (*.pdf), Microsoft Excel (*.xls), HTML и многих других.

В этой главе мы рассмотрим, как Forms можно связать с построителем отчетов и какие для этого предусмотрены возможности. Между двумя этими продуктами, как, впрочем, и между остальными продуктами Oracle Developer, существует тесная взаимосвязь и гибкий обмен данными. Обмениваясь данными с Reports, вы делаете это аналогично тому, как вы бы это делали, передавая параметры в другую форму. Далее будут рассмотрены темы:

- Запуск Oracle Reports из Oracle Forms.
- Передача параметров из Oracle Forms в Oracle Reports.

Запуск Oracle Reports из Oracle Forms

В Oracle Forms в зависимости от версии Reports можно запустить через две различные процедуры:

- RUN_PRODUCT;
- WEB.SHOW_DOCUMENT.

Сначала мы с вами рассмотрим процедуру RUN_PRODUCT, которая предназначена в первую очередь для запуска отчетов из версий Forms

6i и ниже. Эта процедура поддерживается для совместимости и в более поздних версиях Forms. С помощью этой процедуры вы можете запускать отчет, передавать в него параметры и группы записей. Ниже приведен синтаксис этой процедуры, который может быть оформлен в двух вариантах, отличающихся двумя выделенными параметрами.

Синтаксис процедуры RUN_PRODUCT

```
RUN_PRODUCT (PRODUCT IN NUMBER, DOCUMENT IN VARCHAR2, COMMMODE IN
NUMBER, EXECMODE IN NUMBER, LOCATION IN NUMBER, PARAMLIST_ID IN
PARAMLIST, DISPLAY IN VARCHAR2);
RUN_PRODUCT (PRODUCT IN NUMBER, DOCUMENT IN VARCHAR2, COMMMODE IN
NUMBER, EXECMODE IN NUMBER, LOCATION IN NUMBER, PARAMLIST_NAME IN
VARCHAR2, DISPLAY IN VARCHAR2);
```

Описание принимаемых параметров:

PRODUCT – имя запускаемого продукта, им может быть Graphics, Forms или Reports;

DOCUMENT – имя исполняемого модуля, который должен быть исполнен вызываемым продуктом;

COMMODE – определяет тип (режим) запуска, который будет использован для вызываемого продукта. Вы можете устанавливать один из нижеперечисленных режимов:

- SYNCHRONOUS (синхронный) – указывает на то, что управление будет передано форме только после закрытия вызываемого продукта;
- ASYNCHRONOUS (асинхронный) – указывает на то, что управление форме будет возвращено немедленно после отображения модуля, то есть вы можете работать одновременно и с запущенным модулем, и с формой;

EXECMODE – определяет режим выполнения вызываемого продукта. Если вызывается Reports или Graphics, то режим можно устанавливать BATCH или RUNTIME; если вы запускаете Forms, то режим выполнения всегда RUNTIME;

LOCATION – определяет место размещения вызываемого модуля, им может быть файловая система (FILESYSTEM) или База Данных (DB);

PARAMLIST_NAME OR PARAMLIST_ID – определяет список параметров, передаваемых вызываемому продукту. В качестве принимаемого параметра вы можете указать имя списка параметров или его идентификатор. Если вы не предаете никаких параметров, то можно написать NULL;

DISPLAY – определяет имя элемента диаграммы (block_name.chart_item_name), в который будет выводится график, сгенерированный Graphic Builder. Этот параметр определяется, только если вы вызываете Graphic Builder, во всех остальных случаях этот параметр должен быть NULL.

Теперь, когда мы знаем синтаксис процедуры и все необходимые параметры, выполним несколько примеров различной сложности, которые приведены ниже.

1 Если вы просто хотите запустить отчет без передачи параметров, вам достаточно будет написать в вашей программе одну строчку, как показано в листинге 21.1 «Запуск отчета без передачи параметров».

Листинг 21.1. Запуск отчета без передачи параметров

```
Run_Product(REPORTS,'c:\myrep.rep', SYNCHRONOUS, RUNTIME,
FILESYSTEM, NULL, NULL);
```

2. Для того чтобы передать параметр в отчет, создайте список параметров и передайте его определение в процедуру, как показано в листинге 21.2 «Запуск и передача параметров в отчет».

Листинг 21.2. Запуск и передача параметров в отчет

```
  DECLARE
pl_id ParamList;
BEGIN
   pl_id := Get_Parameter_List('list_data');
  IF NOT Id_Null(pl_id) THEN
    Destroy_Parameter_List( pl_id );
  END IF;
  pl_id := Create_Parameter_List('list_data');
/* Наполняем список параметров различными данными, где
:block_name.item_name - значение элемента передаем в параметр
p1 Reports
:parameter.param_name - значение параметра передаем в параметр
p2 Reports
'Rec_group_name' - передаем группу записей в именнованную группу
записей Reports
'PARAMFORM' - передаем системный параметр в Reports */
  Add_Parameter(pl_id,'p1',TEXT_PARAMETER,:block_name.item_name);
  Add_Parameter(pl_id,'p2',TEXT_PARAMETER,:parameter.parameter _name);
  Add_Parameter(pl_id,'query_group',DATA_PARAMETER,'Rec_group_name');
  Add_Parameter(pl_id, 'PARAMFORM', TEXT_PARAMETER, 'NO');
  Run_Product(REPORTS, 'c:\my_rep', SYNCHRONOUS, RUNTIME, FILESYSTEM,
pl_id, NULL);
END;
```

Как показано в листинге 21.2 «Запуск и передача параметров в отчет», вы можете передавать различные типы параметров:

- значения элементов;
- значения параметров;
- значения глобальных переменных;
- **значения групп записей;**
- значения переменных;
- системные параметры Reports.

> ***Совет:*** *используйте параметры для передачи значений в лексические параметры Reports, это позволит вам не только передавать значения критерия выборки, но и добавлять другие конструкции к фразе SELECT, такие как WHERE, GROUP BY, ORDER BY, HAVING, CONNECT BY и START WITH.*

Остановимся подробнее на последнем пункте, так как все остальные варианты понятны и были уже рассмотрены ранее. Системные параметры Reports – это те параметры, которые управляют выполнением, отображением и формированием отчета. Используя их определения при добавлении параметра в список, вы тем самым получаете возможность влиять на ход выполнения вызываемого продукта. Далее перечислены основные системные параметры.

COPIES – определяет число копий, которое будет задано, когда отчет будет отправлен на печать.

CURRENCY – определяет индикатор денежного символа.

DESFORMAT – определяет выходной формат отчета. Oracle Reports поддерживает следующие форматы:

- PDF (Forms9i и выше);
- HTML;
- RTF;
- HTMLCSS;
- XML (Forms9i и выше);
- DELIMITED;
- DELIMITEDDATA;
- SPREADSHEETS (Forms10g).

DESNAME – определяет имя выходного драйвера (имя файла, имя принтера, mail пользователя). Например, если вы хотите вывести отчет на матричный принтер, вам необходимо указать имя файла *.prt – dec180.

DESTYPE – определяет тип вывода отчета. Вы можете определить следующие типы: просмотр, файл, mail, принтер или вывод на просмотр с использованием PostScript.

MODE – определяет режим запуска отчета: растровый (BITMAP) или символьный (CHARACTERS).

ORIENTATION – определяет ориентацию печати, допустимы следующие значения:

- DEFAULT;
- PORTRAIT;
- LANDSCAPE.

PRINTJOB – управляет отображением диалога печати перед запуском отчета.

THOUSENDS – указывает символ разделитель.

Используя вышеописанные параметры, вы сможете управлять печатью, например, вывести печать в файл: пример показан в листинге 21.3 «Формирование отчета в файл».

Листинг 21.3. Формирование отчета в файл

```
  DECLARE
pl_id ParamList;
BEGIN
   pl_id := Get_Parameter_List('list_data');
  IF NOT Id_Null(pl_id) THEN
    Destroy_Parameter_List( pl_id );
  END IF;
  pl_id := Create_Parameter_List('list_data');
    Add_Parameter(pl_id,'DESNAME',TEXT_PARAMETER,'my_rep.txt');
    Add_Parameter(pl_id, 'DESTYPE', TEXT_PARAMETER, 'FILE');
    Run_Product(REPORTS, 'c:\my_rep', SYNCHRONOUS, RUNTIME,
FILESYSTEM, pl_id, NULL);
END;
```

Вы можете вывести отчет немедленно на печать без предварительного просмотра или формирования файла, как показано в листинге 21.4 «Формирование отчета на принтер без предварительного просмотра».

Листинг 21.4. Формирование отчета на принтер без предварительного просмотра

```
  DECLARE
pl_id ParamList;
BEGIN
   pl_id := Get_Parameter_List('list_data');
  IF NOT Id_Null(pl_id) THEN
    Destroy_Parameter_List( pl_id );
  END IF;
  pl_id := Create_Parameter_List('list_data');
```

```
        Add_Parameter(pl_id, 'BATCH',TEXT_PARAMETER,'YES');
        Add_Parameter(pl_id, 'DESNAME', TEXT_PARAMETER, 'dec180');
        Add_Parameter(pl_id, 'DESTYPE', TEXT_PARAMETER, 'PRINTER');
        Run_Product(REPORTS, 'c:\my_rep', SYNCHRONOUS, RUNTIME,
FILESYSTEM, pl_id, NULL);
END;
```

Теперь, когда мы рассмотрели процедуру RUN_PRODUCT и запуск отчета в GUI, перейдем к рассмотрению встроенной подпрограммы WEB.SHOW_DOCUMENT для запуска отчетов в браузере.

Запуск отчета с помощью WEB.Show_Document

Встроенная процедура WEB.SHOW_DOCUMENT, как уже было сказано выше, предназначена для запуска отчета в браузере. Эта процедура поддерживается в версиях Forms 6i и выше. Из всех перечисленных в этой главе эта процедура является наиболее удобной и компактной в кодировании, и нижеприведенный синтаксис лишний раз это доказывает.

Синтаксис:

```
WEB.SHOW_DOCUMENT(URL, DESTINATION);
```

Параметры:

URL – определяет имя ресурса, с которого будет загружен документ. URL в данном случае — это не только адресная строка, но и набор передаваемых параметров, так как в определении URL используются системные параметры (DESNAME, DESTYPE и другие).

DESTINATION – назначение целевого окна. Различают следующие назначения:

_SELF – заставляет отчет загружаться в ту же самую структуру или окно, что и первоисточник.

_PARENT – заставляет отчет загружаться в родительское окно или frameset, содержащий ссылку гипертекста. Если ссылка находится в окне или верхнем уровне фрейма, то _PARENT отрабатывает аналогично _SELF.

_TOP – заставляет отчет загружаться в окно, содержащее гипертекстовую ссылку, заменяя все текущие фреймы, которые отражены в этом окне.

_BLANK – загружает отчет в новое неименованное окно.

Строка URL в области определения Reports имеет свою особую специфику, так как является не просто адресной строкой с указанием места

расположения ресурса, но и содержит перечень различных параметров. Ниже приведен синтаксис URL с перечислением возможных параметров:

```
http://<hostname>:<port>/reports/rwservlet?server=<reportserver_tns>
&report=<report>.rdf&desformat=[htmlcss|pdf|xml|delimited|]&destype=c
ache&userid=<user/pw@database>&paramform=[no|yes]
```

Теперь, когда мы ознакомились с синтаксисом этой процедуры, попробуем ее в действии. В листинге 21.5 «Запуск отчета с помощью процедуры WEB.SHOW_DOCUMENT» приведен пример запуска отчета в Web.

Листинг 21.5. Запуск отчета с помощью процедуры WEB.SHOW_DOCUMENT

```
DECLARE
rep_url varchar2(300);
BEGIN
rep_url:='http://<hostname><port>/reports/rwservlet?server='
||'Repsrv&report=my_rep.rdf&desformat=htmlcss&destype=cache '
||'&userid=user/pw@database&P_NAME ='||:parameter.param_
name||'&paramform
=no';
WEB.SHOW_DOCUMENT(rep_url,'_blank');
END;
```

Как вы уже, наверно, успели заметить, в URL помимо системных параметров можно передавать также и пользовательские параметры. К недостаткам такого метода запуска отчетов можно отнести адресную строку URL, так как при запуске отчета отображается весь гипертекст, в котором указано имя, логин и пароль пользователя. Далее мы еще не раз вернемся к этой процедуре, а также рассмотрим пример, в котором устраним вышеописанный недостаток.

Запуск отчета с помощью функции RUN_REPORT_OBJECT

В Oracle Forms для работы с отчетами существует объект Report, который представляет собой модуль отчета, построенный в Oracle Reports Builder. Для того чтобы создать объект «отчет», выполните следующие действия:

1. Находясь в навигаторе объектов, нажмите кнопку «Создать» для создания нового объекта отчета. Сразу после нажатия кнопки на экране появится окно «Новый отчет» (рис. 21.1).
2. В появившемся окне выберите радиокнопку «Использовать существующий отчет» и нажмите кнопку «Обзор» для поиска существующего модуля отчета из файловой системы.

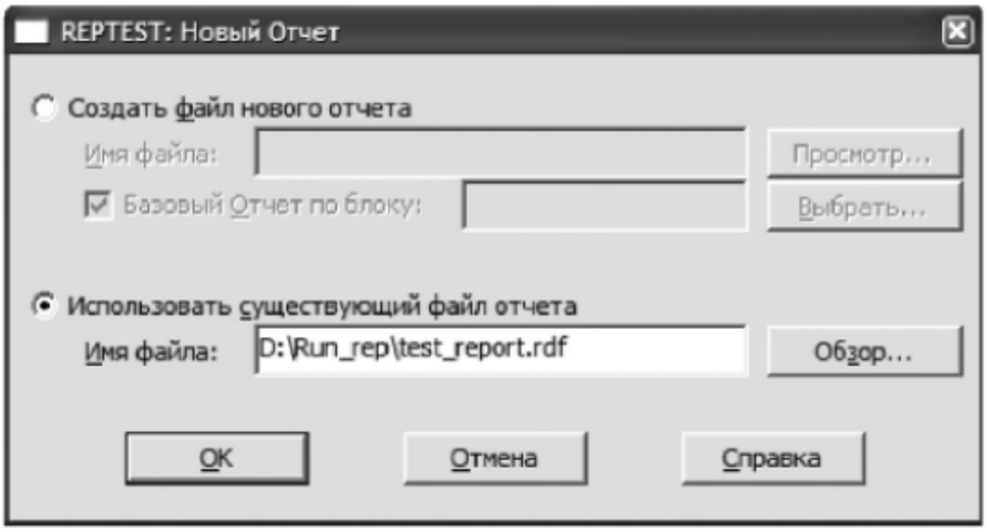

Рис. 21.1. Окно «Новый отчет»

3. Подтвердите выбор для закрытия окна. Созданный отчет появится в узле «Reports» со сгенерированным Forms именем, а не с именем выбранного отчета.

4. Запустите Палитру Свойств объекта отчета и посмотрите на параметры, которыми можно управлять на этапе проектирования (рис. 21.2).

Рис. 21.2. Палитра свойств отчета

5. В разделах Oracle Developer Integration и Reports вы можете управлять параметрами отчета. Для примера измените режим отображения отчета, установив параметр «Report Destination Type» на Preview. Назовите отчет «My_rep».
6. Находясь в редакторе разметки, создайте кнопку и определите для нее триггер WHEN-BUTTON-PRESSED. В теле триггера наберите и скомпилируйте следующие строки:

Листинг 21.6. WHEN-BUTTON-PRESSED (Запуск отчета с помощью Run_Report_Object)

```
DECLARE
rep_id Report_Object;
Rep VARCHAR2(100);
BEGIN
rep_id:= find_report_object('My_Rep');
SET_REPORT_OBJECT_PROPERTY(rep_id,REPORT_COMM_MODE,ASYNCHRONOUS);
SET_REPORT_OBJECT_PROPERTY(rep_id,REPORT_DESTYPE,PREVIEW);
SET_REPORT_OBJECT_PROPERTY(rep_id,REPORT_SERVER,'Repsrv');
Rep:=run_report_object(rep_id);
END;
```

7. Запустите форму и нажмите на кнопку для проверки выполненного примера.

В последнем листинге мы использовали пока неизвестные нам процедуру SET_REPORT_OBJECT_PROPERTY и функцию RUN_REPORT_OBJECT. Далее приведен синтаксис этих двух встроенных подпрограмм.

Функция RUN_REPORT_OBJECT предназначена для запуска отчета локально и на удаленном сервере Базы Данных. Ниже приведен синтаксис этой функции и листинг с примером.

RUN_REPORT_OBJECT (report_id REPORT_OBJECT) RETURN VARCHAR2;

Параметры: report_id – определяет идентификатор отчета.

Листинг 21.7. Пример использования функции Run_Report_Object

```
DECLARE

Rep_id REPORT_OBJECT;
   rep  VARCHAR2(100);
BEGIN
```

```
    Rep_id := find_report_object('my_rep');
    rep := RUN_REPORT_OBJECT(rep_id);
    ......
END;
```

Процедура SET_REPORT_OBJECT_PROPERTY предназначена для программной установки свойств объекта. Ниже приведен один из возможных синтаксисов это процедуры:

SET_REPORT_OBJECT_PROPERTY(report_id REPORT_OBJECT, property NUMBER, value [VARCHAR2 , NUMBER]);

Параметры:

report_id [report_name] – этот параметр в качестве значения принимает идентификатор отчета или его имя;

property – определяет имя свойства отчета;

value – определяет передаваемое значение для устанавливаемого свойства.

Параметр property процедуры SET_REPORT_OBJECT_PROPERTY может принимать следующие значения:

- REPORT_EXECUTION_MODE – режим выполнения отчета BATCH или RUNTIME;
- REPORT_COMM_MODE – определяет тип (режим) запуска, который будет использован для вызываемого продукта. Вы можете устанавливать один из нижеперечисленных режимов:
  - SYNCHRONOUS (синхронный) – указывает на то, что управление будет передано форме только после закрытия вызываемого продукта;
  - ASYNCHRONOUS (асинхронный) – указывает на то, что управление форме будет возвращено немедленно после отображения модуля, то есть вы можете работать одновременно и с запущенным модулем, и с формой;
- REPORT_DESTYPE – определяет тип вывода отчета. Вы можете определить следующие типы: просмотр, файл, mail, принтер, кеш или экран;
- REPORT_FILENAME – определяет имя отчета;
- REPORT_SOURCE_BLOCK – имя блока источника данных для отчета;
- REPORT_QUERY_NAME – определяет имя запроса;
- REPORT_DESNAME – определяет целевое имя отчета;
- REPORT_DESFORMAT: The report destination format;

- REPORT_SERVER – определяет имя сервера отчетов;
- REPORT_OTHER – определяет пользовательские параметры.

В следующем листинге приведен еще один пример с использованием процедуры SET_REPORT_OBJECT_PROPERTY.

Листинг 8. Запуск отчета с помощью Run_Report_Object

```
DECLARE
rep_id Report_Object;
Rep VARCHAR2(100);
BEGIN
rep_id:= find_report_object('My_Rep');
SET_REPORT_OBJECT_PROPERTY(rep_id, REPORT_EXECUTION_MODE, BATCH);
SET_REPORT_OBJECT_PROPERTY(rep_id, REPORT_COMM_MODE, SYNCHRONOUS);
SET_REPORT_OBJECT_PROPERTY(rep_id, REPORT_DESTYPE, FILE);
Rep:=run_report_object(rep_id);
END;
```

Вы можете использовать объект «Отчет» для совместной работы с процедурами WEB.SHOW_DOCUMENT и RUN_PRODUCT.

Передача последнего условия Where Clause из формы в отчет

В Oracle Reports существуют лексические параметры, которые позволяют динамически изменять условия запроса, формирующего запрос, то есть вы можете определить фразу Where Clause с различными условиями ограничения выборки и подставить ее во фразу FROM. Мы с вами выполним пример, который продемонстрирует получение фразы WHERE_CLAUSE блока и ее последующую передачу в Oracle Reports; для этого разобьем пример на две части:

1. получение фразы WHERE_CLAUSE формы;
2. передача фразы WHERE_CLAUSE в Oracle Reports.

«Получение фразы WHERE_CLAUSE формы»

```
FUNCTION RETURN_WHERE_CLAUSE RETURN VARCHAR2
IS
BEGIN
IF INSTR(:System.Last_Query,'WHERE')> 0 THEN
RETURN (SUBSTR(:System.Last_Query,
INSTR(:System.Last_Query,'WHERE')+ 6));
END IF;
EXCEPTION
```

```
WHEN OTHERS THEN
RETURN NULL;
END;

PROCEDURE Report_With_Form_Where_Clause
IS
pl ParamList;
where_cl VARCHAR2(2000);
BEGIN
pl := Create_Parameter_List('plist');
IF RETURN_WHERE_CLAUSE IS NOT NULL THEN
Add_Parameter(pl, 'Form_Where_Clause', TEXT_PARAMETER,
RETURN_WHERE_CLAUSE);
END IF;
Run_Product(REPORTS, 'Form_Clause.rdf', SYNCHRONOUS, BATCH,
FILESYSTEM, pl );
Destroy_Parameter_List(pl);
END;
```

Пример. «Передача фразы WHERE_CLAUSE в Oracle Reports»

```
PROCEDURE Report_With_Form_Where_Clause
IS
pl ParamList;
where_cl VARCHAR2(2000);
BEGIN
pl := Create_Parameter_List('plist');
IF RETURN_WHERE_CLAUSE IS NOT NULL THEN
Add_Parameter(pl, 'Form_Where_Clause', TEXT_PARAMETER,
RETURN_WHERE_CLAUSE);
END IF;
/*Если вы работаете с Web-приложениями, то замените процедуру
Run_Product на WEB.Show_Document*/
Run_Product(REPORTS, 'Form_Clause.rdf', SYNCHRONOUS, BATCH,
FILESYSTEM, pl );
Destroy_Parameter_List(pl);
END;
```

Ни в одном из приведенных примеров нет привязки к каким-либо конкретным объектам, поэтому для того, чтобы посмотреть их в действии, вам достаточно скопировать приведенные листинги в свое приложение.

Скрытие URL отчета

Как уже было сказано выше, при запуске веб-отчета в строке гиперссылки браузера отображается вся строка передаваемого URL, что недопустимо с точки зрения безопасности. Ниже приведена функция скрытия URL отчета.

Листинг 9. Скрытие URL отчета

```
FUNCTION CRYPT(URL_PARAMS_IN Varchar2) RETURN VARCHAR2 IS
v_url VARCHAR2(2000) := URL_PARAMS_IN; -- Url string
v_url_temp VARCHAR2(4000) :=''; -- Temp URL string
v_a VARCHAR2(10);
-- conversion variable
v_b VARCHAR2(10);
-- conversion variable
c CHAR;
i NUMBER(10);
BEGIN
FOR i IN 1..LENGTH(v_url) LOOP
c:= substr(v_url,i,1);
IF c in (';', '/','?',':','@','+','$',',',' ') THEN
v_a := ltrim(to_char(trunc(ascii(substr(v_url,i,1))/16)));
IF v_a = '10' THEN v_a := 'A';
ELSIF v_a = '11' THEN v_a := 'B';
ELSIF v_a = '12' THEN v_a := 'C';
ELSIF v_a = '13' THEN v_a := 'D';
ELSIF v_a = '14' THEN v_a := 'E';
ELSIF v_a = '15' THEN v_a := 'F';
END IF;
v_b := ltrim(to_char(mod(ascii(substr(v_url,i,1)),16)));
IF v_b = '10' THEN v_b := 'A';
ELSIF v_b = '11' THEN v_b := 'B';
ELSIF v_b = '12' THEN v_b := 'C';
ELSIF v_b = '13' THEN v_b := 'D';
ELSIF v_b = '14' THEN v_b := 'E';
ELSIF v_b = '15' THEN v_b := 'F';
END IF;
v_url_temp := v_url_temp||'%'||v_a||v_b;
ELSE
v_url_temp :=v_url_temp||c;
END IF;
END LOOP;
```

```
return v_url_temp;
END;
```

В следующем листинге показан пример использования этой функции.

Листинг 10. Скрытие URL при помощи функции CRYPT

```
DECLARE
rep_id Report_Object;
Rep VARCHAR2(100);
BEGIN
rep_id:= find_report_object('My_Rep');
SET_REPORT_OBJECT_PROPERTY(rep_id, REPORT_EXECUTION_MODE, BATCH);
SET_REPORT_OBJECT_PROPERTY(rep_id, REPORT_COMM_MODE, SYNCHRONOUS);
SET_REPORT_OBJECT_PROPERTY(rep_id, REPORT_DESTYPE, FILE);
Rep:=run_report_object(rep_id);
END;
```

В этой главе мы научились интегрировать отчеты в приложения различными способами, а также запускать отчеты в GUI, Web и контролировать выполнение и формирование запускаемого модуля.

Лекция 22. Меню в Oracle Forms

В этой лекции слушатель научится создавать меню, определять его структуру, создавать подменю и всплывающее меню. Также в этой лекции слушатель ознакомится с различными типами команд меню, способами обращения к элементам меню и их программированием.

Ключевые слова: создание меню, удаление меню, пункт меню, магические элементы, команды меню, типы меню, срываемые меню, программирование меню, separate, магические элементы.

Цель лекции: слушатели научатся создавать различные виды меню, ознакомятся со структурой меню, методами его использования и программирования отдельных элементов.

Меню – это набор элементов, позволяющих пользователю осуществлять быстрый доступ к наиболее часто используемым командам: Execute_Query, Exit_Form, Next_Record, Enter_Query или Previous_Item. Если рассматривать стандартное меню – меню (рис. 22.1) по умолчанию, то большинство его команд ассоциировано с функциональными клавишами и встроенными процедурами. Меню Oracle Forms представлено отдельным модулем, который может быть использован совместно и одновременно любым количеством форм. В Forms вы можете не только применять меню по умолчанию, но и создавать свое собственное.

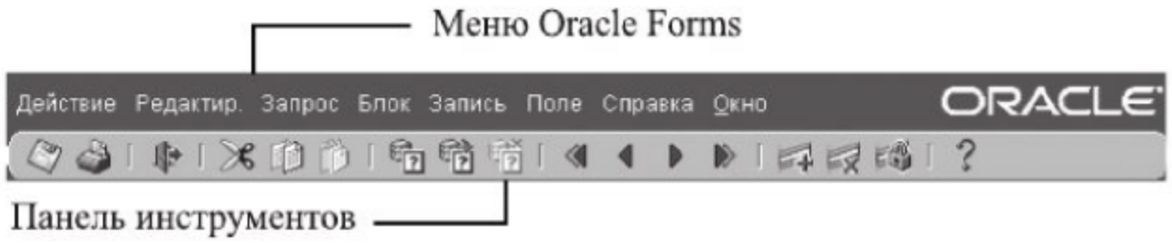

Рис. 22.1. Меню Oracle Forms.

Использование меню по умолчанию

Как уже было сказано ранее, Oracle Forms имеет свое стандартное меню команд, которое еще называется «меню по умолчанию». Вы можете управлять отображением стандартного меню с помощью параметра DEFAULT&SMARTBAR свойства «Menu Module» формы (рис. 22.2). Если вы сотрете это слово, то при последующем запуске формы меню и

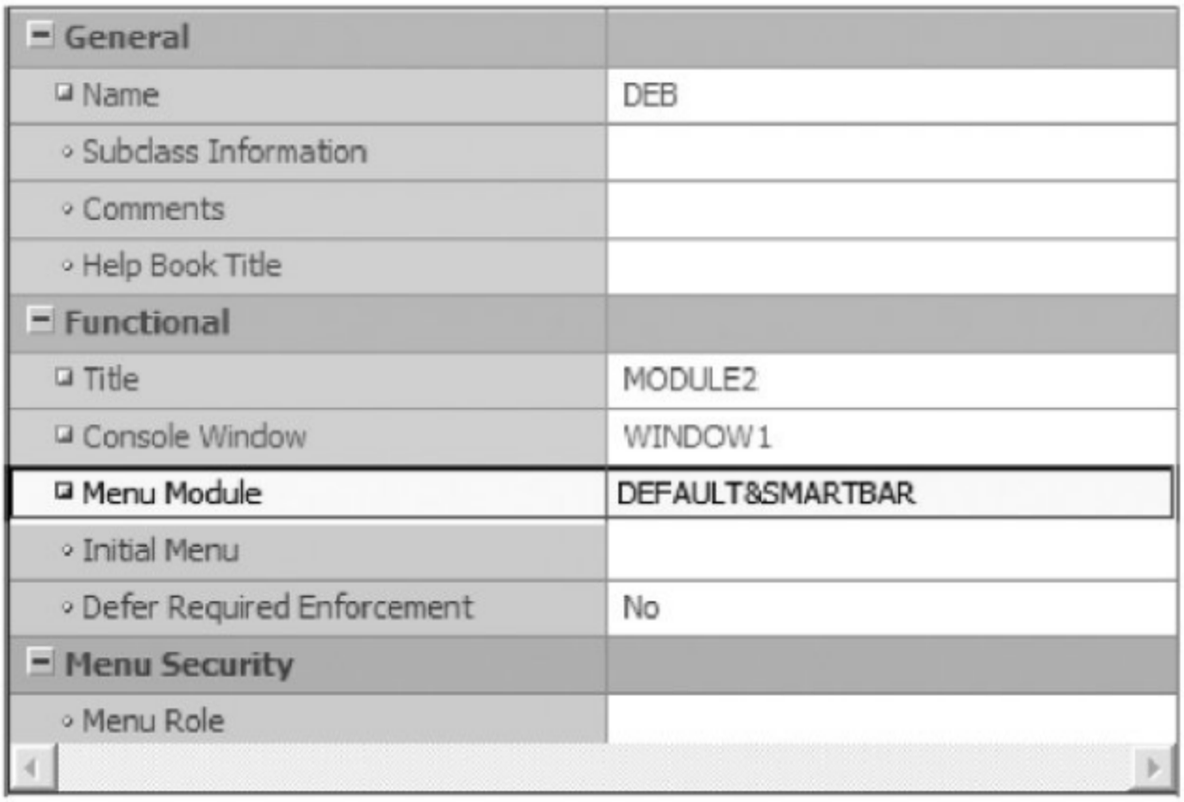

Рис. 22.2. Свойство «Модуль меню» формы

панель инструментов исчезнет. Для того чтобы отобразить только меню, без панели инструментов, сотрите сочетание «&SMARTBAR», оставив слово DEFAULT.

Создание специальных меню

Несмотря на то что меню по умолчанию содержит солидный набор необходимых команд, все же в большинстве случаев оказывается, что было бы неплохо добавить ту или иную команду, например, запуск отчета, другой формы или выполнение команды операционной системы. Вы не можете изменить стандартное меню (рис. 22.3), поэтому для того чтобы добавить в меню новые команды, вам придется создать собственное меню. Чтобы не делать лишнюю работу и повторно не воспроизводить команды стандартного меню, вы можете править существующий шаблон – OFGMNUT.MMB, который находится в папке DevSuiteHome_1\cgenf61\ADMIN\. Для подключения шаблона выполните следующие действия:

1. Найдите файл DevSuiteHome_1\cgenf61\ADMIN\OFGMNUT.MMB.
2. Откройте шаблон в форме.

> ***Примечание:*** *когда вы подключите к форме собственное меню, панель инструментов исчезнет и вам придется создавать собственную панель.*

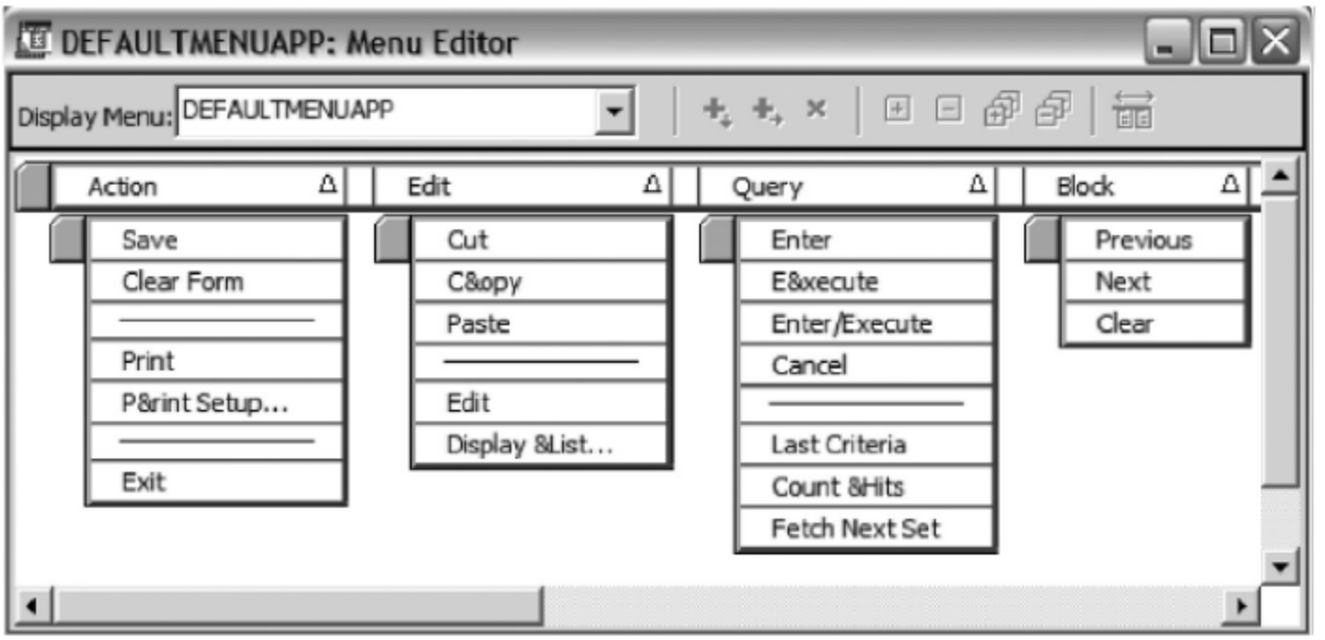

Рис. 22.3. Шаблон меню OFGMNUT.MMB

Создание меню, пунктов меню и подменю

Для того чтобы научится создавать меню, необходимо знать его структуру, а именно основные объекты, которые его составляют:

- *меню (menu)* – это главное меню и подменю. Главное меню – это меню типа «Файл», «Правка», а подменю – это иерархическое меню, ассоциированное с определенным пунктом меню и возникающее при его выборе;
- *пункт, или элемент меню (menu item)* – это элементы главного меню или подменю. Вы можете создавать программные единицы для элементов меню.

В зависимости от состояния меню могут быть динамическими и статическими. Правильнее, конечно же, пользоваться статическим меню, так как меню должно быть неизменным и не должно сбивать пользователя с толку. Когда нам нужно, чтобы элементы меню изменялись в зависимости от текущего состояния приложения или действий пользователя, мы используем динамическое меню. Примером такого меню может служить меню «Запрос» стандартного меню, в котором при выборе элемента «Enter Query» остальные пункты подменю становятся недоступными. Такой способ проектирования меню имеет преимущество над статическим, так как исключается выполнение пользователем команд, недопустимых в режиме запроса. Элемент меню, как и любой другой элемент Oracle Forms, имеет свое имя, которое уникально в пределах модуля и используется для обращения к нему в программных единицах.

У каждого меню и пункта меню есть имя, которое однозначно определяет его в модуле меню.

Примечание: *не создавайте длинных меню, так как короткие меню более приятны и удобны для восприятия. Также не следует создавать меню с большим уровнем вложенности или со сложной иерархической структурой, так как в таком меню легко потеряться и пользователю будет проще выполнить требуемую команду иным способом.*

Редактор меню

Проектировать меню в Forms не составляет особой сложности, так как все действия выполняются с помощью единственного инструмента – Menu Editor («Редактор меню»), который имеет небольшой набор команд, позволяющих быстро создать меню. Поэтому прежде чем приступить к созданию своего первого меню, ознакомимся с основными характеристиками редактора (рис. 22.4).

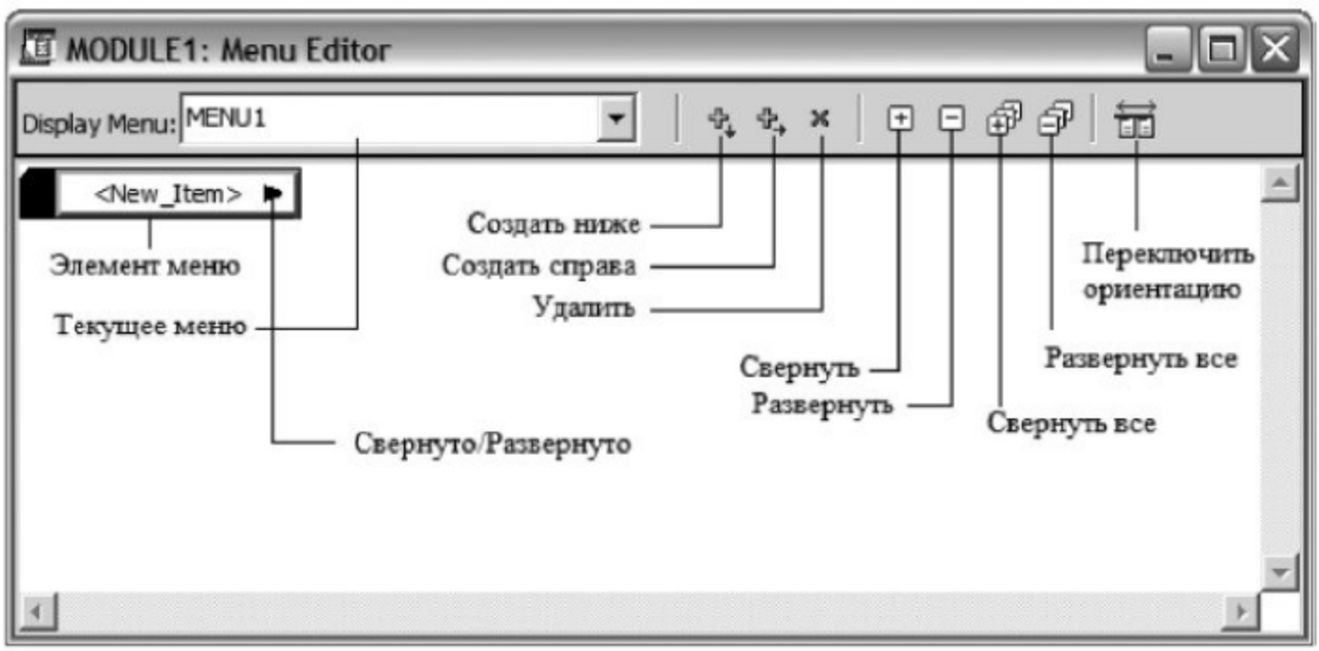

Рис. 22.4. Редактор меню

Рассмотрим более подробно управляющие элементы редактора, отображенные на рисунке.

- Создать ниже (Create Down) – это специальная команда, которая добавляет подменю ниже выбранного пункта меню или подменю. Эта команда аналогична команде главного меню Forms – Edit | Create Down или комбинации клавиш Ctrl+Down.
- Создать справа (Create Right) – это специальная команда, которая добавляет подменю справа от выбранного пункта меню или подменю. Она аналогична команде главного меню Forms – Edit | Create Right или комбинации клавиш Ctrl+Right.

- Удалить (Delete) – это специальная команда, которая удаляет выбранный пункт меню. Если вы удаляете пункт меню, который содержит другие подменю, то они будут автоматически удалены вместе с элементом меню, к которому они принадлежат. Эта команда аналогична команде главного меню Forms – Edit | Delete или клавише Delete.
- Развернуть (Expand) – это специальная команда, которая предназначена для развертывания (отображения) всех элементов подменю, относящихся к текущему элементу меню или подменю. Эта команда аналогична команде главного меню Forms —View | Expand.
- Свернуть (Collapse) – это специальная команда, которая предназначена для свертывания (скрытия) всех элементов подменю, относящихся к текущему элементу меню или подменю. Эта команда аналогична команде главного меню Forms – View | Collapse.
- Развернуть все (Expand All) – это специальная команда, которая предназначена для развертывания (отображения) всех элементов меню. Она аналогична команде главного меню Forms – View | Expand All.
- Свернуть все (Collapse All) – это специальная команда, которая предназначена для свертывания (скрытия) всех элементов меню. Она аналогична команде главного меню Forms – View | Collapse All.
- Переключить ориентацию (Switch Orientation) – это специальная команда, которая предназначена для изменения ориентации меню. Она позволяет превратить горизонтальное меню в вертикальное и наоборот одним нажатием кнопки.
- Текущее меню (Display Menu) – это элемент список, который отражает текущее меню, отображаемое в редакторе меню. При выборе элемента появляется список доступных меню модуля меню. Этот элемент предназначен для быстрого переключения между меню модуля.

Создание меню

Теперь, когда мы ознакомились с инструментарием редактора меню и структурой меню, можно приступить к созданию меню. Создайте простое меню с небольшим набором команд, для этого выполните следующие действия:

1. Создайте модуль меню с помощью команды главного меню File | New | Menu или командой «Create» навигатора объектов, находясь на узле Menus (Меню). Сохраните модуль как FMENU.MMB. По умолчанию после сохранения модуля его название в Навигаторе Объектов изменится со стандартного названия MODULE1 на имя, присвоенное при сохранении. Вы можете изменить имя модуля двойным щелчком левой кнопки мыши по названию модуля и ввести новое.

Изменение имени модуля в Навигаторе Объектов не влияет на имя файла модуля меню. Модулей меню, как и модулей форм, можно создавать неограниченное количество.

2. Для того чтобы создать меню, выберите узел Menus модуля FMENU и нажмите кнопку «Create» навигатора объектов. Вы также можете создать меню командой главного меню Edit | Create или нажатием комбинации клавиш Ctrl+Insert. Так же как и имя модуля меню, вы можете изменить имя меню, поэтому, повторяя ранее описанные действия, переименуйте меню с MENU1 на MyMenu. В пределах одного модуля можно создавать неограниченное количество меню.
3. Щелкните правой кнопкой мыши по названию меню и выберите пункт «Menu Editor» из всплывающего меню для запуска Menu Editor (рис. 22.5).

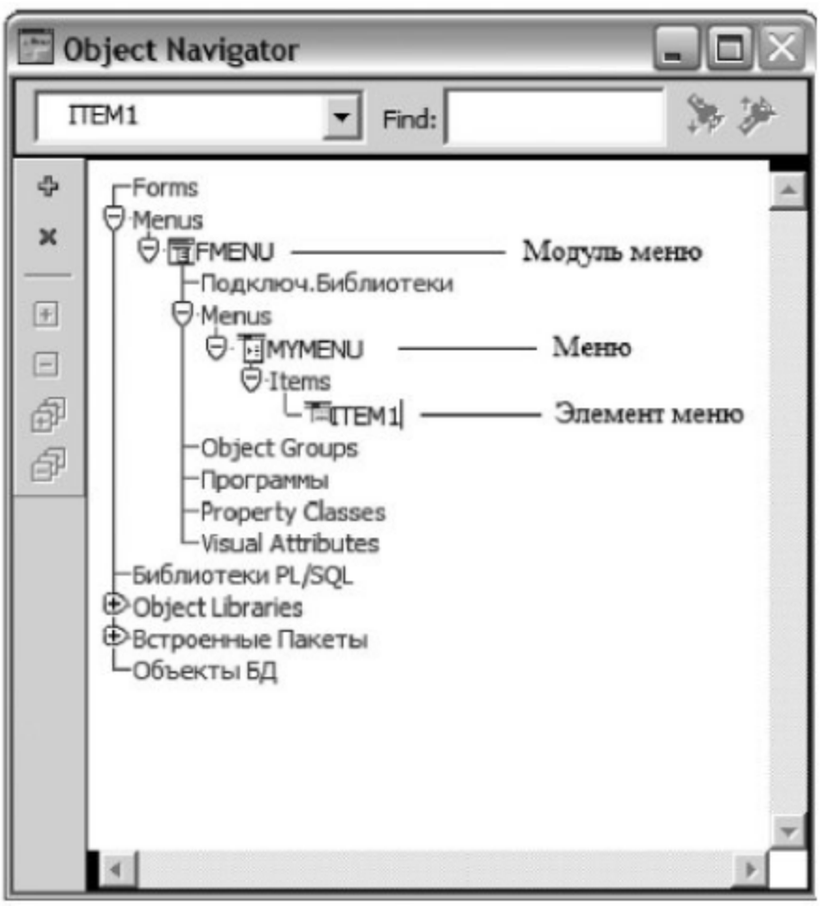

Рис. 22.5. Структура меню

4. Выберите элемент <New_Item> в окне рисования редактора меню и назовите его «Форма». Нажатием кнопки «Create Right» для создания новых элементов меню справа от текущего элемента выберите (рис. 22.6):
   - Блок;
   - Запись.

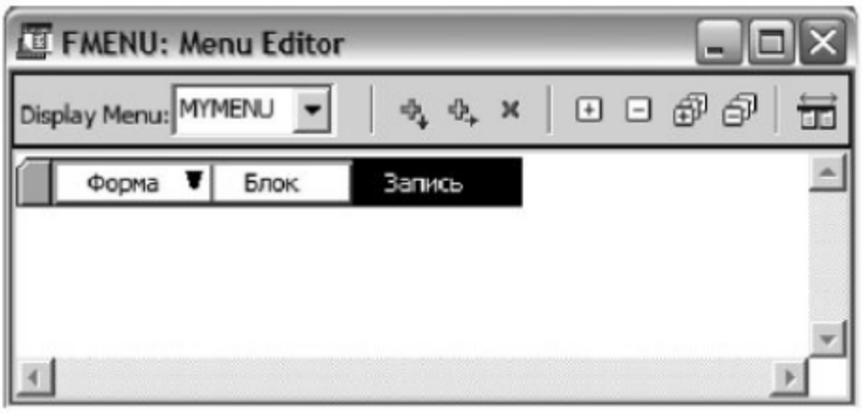

Рис. 22.6. Создание главных элементов меню

5. Вернитесь к элементу «Форма» и нажатием кнопки «Create Down» создайте подменю, состоящее из элементов:
   - Открыть;
   - Сохранить;
   - Отменить;
   - Вызвать отчет;
   - Выход.
6. Перейдите к элементу «Блок» и, повторяя действия, описанные в предыдущем примере, создайте подменю из элементов:
   - Следующий блок;
   - Предыдущий блок;
   - Очистить блок.
7. После создания подменю «Блока» перейдите на следующий элемент – «Запись» и создайте для него подменю следующего содержания:
   - Первая;
   - Последняя;
   - Следующая;
   - Предыдущая;
   - Удалить.
8. Перейдите в пункт подменю «Выход» элемента «Форма» и нажмите кнопку «Create Right» для создания еще одного подменю. Назовите новый элемент «Сохранить изменения», нажмите «Create Down» для создания пункта подменю ниже текущего и назовите его «Без сохранения изменений». На рис. 22.7 показана итоговая структура меню, которая должна была у вас получиться после выполнения упражнения.
9. Вы можете изменить ориентацию меню, то есть превратить его в вертикальное, нажав кнопку «Switch Orientation» (рис. 22.8).

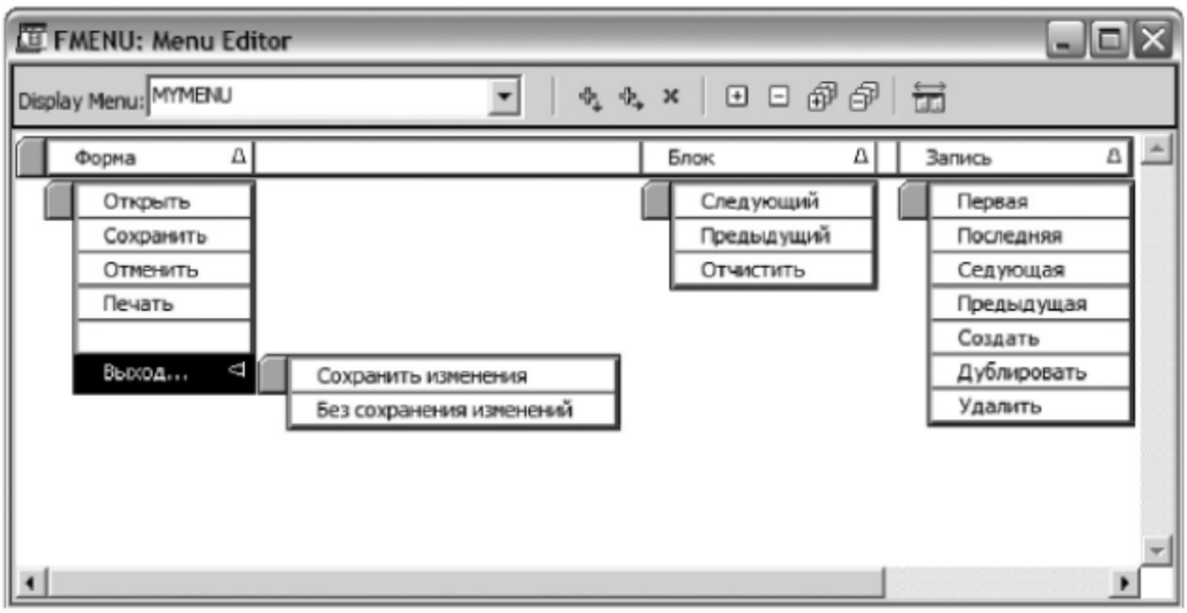

Рис. 22.7. Структура меню FMENU.

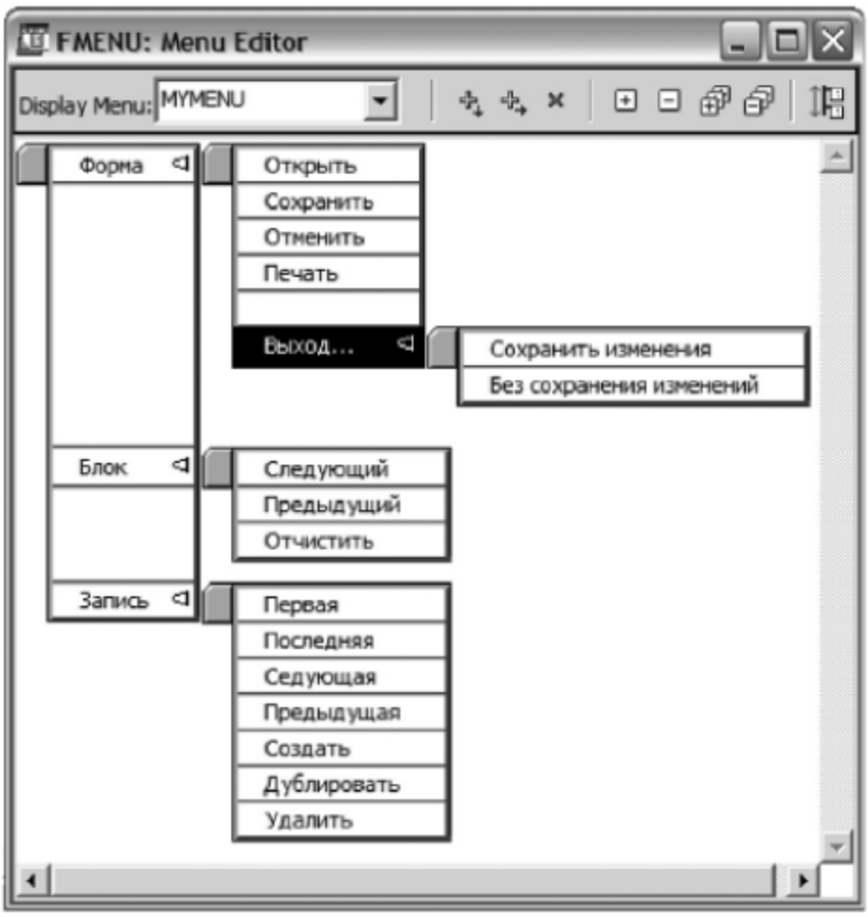

Рис. 22.8. Вертикальное меню.

Удаление меню

Удалять меню так же просто, как и создавать. Для этого достаточно выбрать нужный элемент и выполнить команду удаления одним из перечисленных способов:

1. Выбрать нужный элемент и нажать кнопку Del.
2. Выполнить команду главного меню Edit | Delete.
3. Нажать кнопку Delete навигатора объектов.

При удалении меню важно помнить, что если меню, которое вы собираетесь удалить, содержит подменю, то при удалении главного меню вложенное удалится автоматически.

Свойства меню

Вы можете управлять поведением меню, изменяя значения атрибутов модуля меню, объекта меню и его элементов. В Oracle Forms вы можете очень быстро создавать функциональные меню, не прилагая особых усилий для написания лишнего кода. В большинстве случаев какое бы меню мы ни создавали, в нем обязательно будут присутствовать стандартные или часто используемые команды, программирование которых будет отнимать у вас время. В меню Oracle Form существуют определенные параметры, устанавливая значения которых, можно назначить элементу меню выполнения стандартных команд. В Oracle Forms вы можете назначить элементу меню выполнение стандартной операции, просто установив значение определенного параметра. При работе с меню в Oracle Forms вам доступны свойства модуля, меню и элемента меню.

Свойства модуля меню

Свойства модуля меню позволяют разработчикам управлять параметрами инициализации и защиты меню.

- Главное меню (Main Menu) – имя главного меню, которое будет отображено при запуске.
- Директория меню (Menu Directory) – место размещения исполняемого файла меню .MMX. Если директория не указана, то Forms при попытке запуска меню будет обращаться к той директории, из которой была запущена форма.
- Имя файла меню (Menu File Name) – имя исполняемого файла меню .MMX, которое Oracle Forms будет искать при запуске формы.
- Код запуска (Startup Code) – блок PL/SQL, который будет выполнен после загрузки модуля меню в память при запуске формы.
- Использовать защиту (Use Security) – если значение этого свойства – «Yes», то Oracle Forms будет использовать схему защиты, определенную для меню в свойстве «Роли меню».
- Роли модуля (Module Roles) – роль в базе данных, доступная для элементов этого меню.

Рис. 22.9. Свойства модуля меню.

В отличие от модуля объект меню имеет единственный функциональный атрибут – Tear-Off Menu (рис. 22.10). Меню в отдельном окне (Tear-Off Menu) – если свойство этого атрибута установлено в «Yes», то меню будет отрывным.

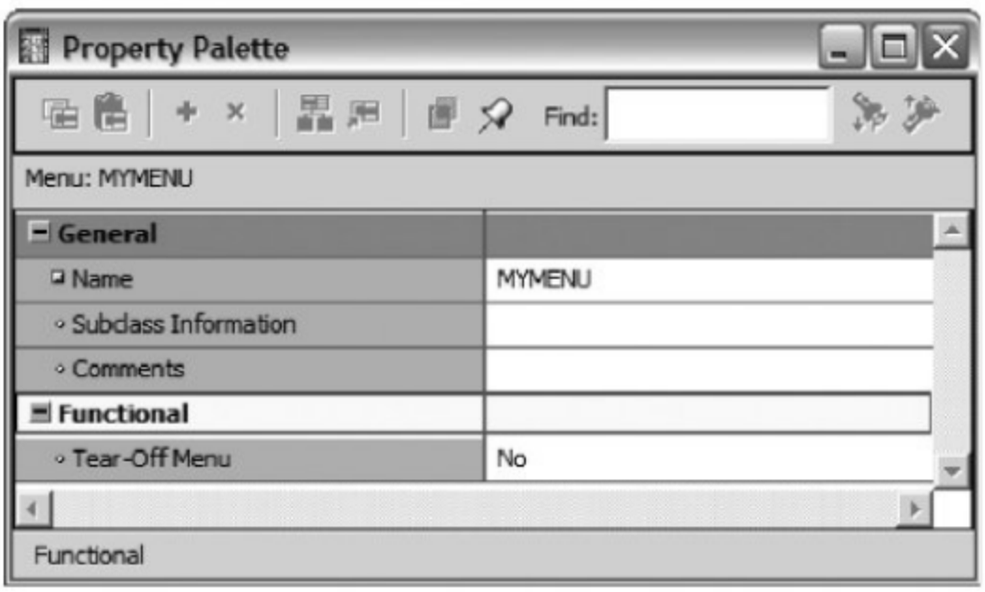

Рис. 22.10. Свойства меню.

Свойства элемента меню

Элементы меню являются функциональными объектами меню и, соответственно, имеют намного больше атрибутов, чем предыдущие объекты. Значения атрибутов элемента меню можно менять (рис. 22.11) как на этапе проектирования, так и во время выполнения формы.

- Включен (Enabled) – это свойство управляет состоянием элемента. Если значение атрибута установлено на «Yes», то элемент включен и доступен для манипулирования мышью.
- Метка (Label) – текстовая метка, выводимая для элемента меню. Старайтесь именовать пункты меню как можно короче и содержательнее, так, чтобы даже самый неосведомленный пользователь, прочитав наименование пункта меню, смог понять, какая команда с ним связана.

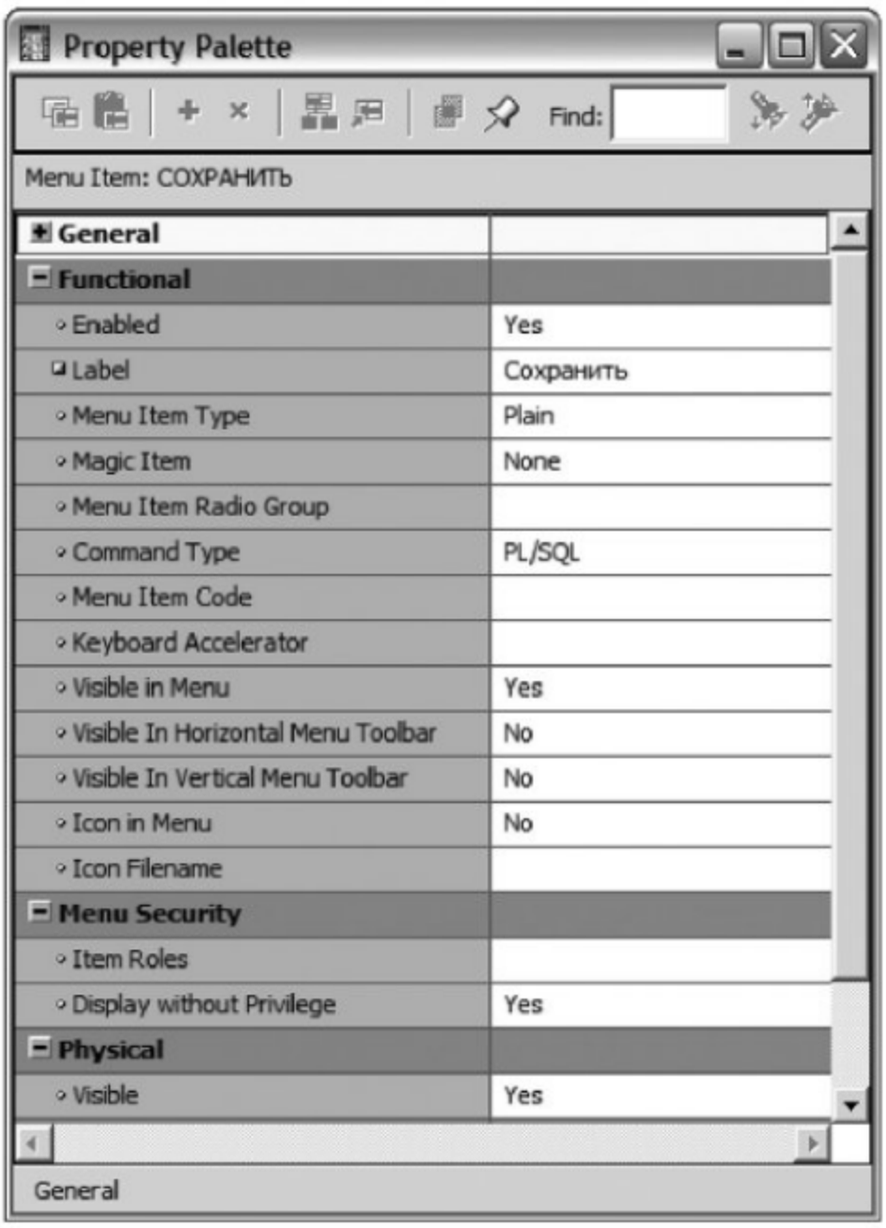

Рис. 22.11. Свойства элемента меню.

- Тип элемента меню (Menu Item Type) – это свойство определяет тип элемента меню.
- Магический элемент (Magic Item) – это свойство позволяет предопределить пункт меню, ассоциируя его с одной из предоставленных команд:
  – Вырезать;
  – Копировать;
  – Вставить;
  – Очистить;
  – Отменить;
  – Справка;
  – О (About);
  – Окно.
- Радиогруппа элементов меню (Menu Item Radio Group) – имя радиогруппы, с которой ассоциирован данный пункт меню.
- Тип команды (Command Type) – тип команды меню. Для элемента меню вы можете выбрать один из трех типов команд:
  – NULL;
  – Меню;
  – PL/SQL.
- Код пункта меню – это блок PL/SQL, исполняемый при выборе элемента меню.
- Имя подменю (Subclass Menu) – имя подменю, ассоциированное с элементом. Это свойство доступно, если свойство «Тип команды» имеет значение «PL/SQL».
- Горячие клавиши (Keyboard Accelerator) – это свойство определяет «горячую» функциональную клавишу, которая будет соответствовать пункту меню. Это свойство доступно только в том случае, если значение атрибута «Тип команды» имеет значение «NULL».
- Виден в меню (Visible in Menu) – это свойство управляет отображением пункта меню, делая его невидимым во время выполнения, если значение установлено на «No».
- Виден в горизонтальном меню (Visible in Horizontal Menu Bar) – если значение этого свойства - «Yes», то пункт меню появится в горизонтальной инструментальной панели меню.
- Виден в вертикальном меню (Visible in Vertical Menu Bar) – если значение этого свойства - «Yes», то пункт меню появится в вертикальной инструментальной панели меню.
- Иконка в меню (Icon in Menu) – если значение этого свойства установлено на «Yes», то иконка будет выводиться в раскрывающемся меню.

- Имя файла иконки (Icon FileName) – имя файла картинки для отображения иконки в меню.
- Роли меню (Item Roles) – это свойство определяет, какие роли меню имеют доступ к данному элементу меню.
- Показывать без привилегий (Display Without Privilege) – если значение этого свойства установлено на «Yes», то пользователь, не имеющий привилегии доступа, сможет увидеть элемент.
- Видимый (Visible) – если значение этого свойства установлено на «Yes», то элемент меню становится невидимым.

Программирование элементов меню

Прежде чем программировать меню, необходимо выбрать для элемента соответствующий тип исполняемой команды. Ниже в таблице 22.1 представлены возможные варианты свойств элемента меню «Тип команды».

Таблица 22.1.

Параметр	Описание
NULL	Этот параметр означает, что никакие операции не выполняются. Данный тип применяется как разделитель или указатель временного элемента меню.
PL/SQL	Этот параметр означает, что в качестве команды пункта меню будет выполнен блок PL/SQL, который может включать в себя процедуры и функции модуля формы. Помните, что, создавая PL/SQL-команду в контексте меню, вы должны будете обращаться к элементам модуля косвенно, то есть через команды COPY и Name_In.
Menu	Этот параметр означает, что при выборе элемента меню будет вызвано подменю. Назначайте этот тип команды всем элементам меню, имеющим подменю.

Для того чтобы приступить к выполнению следующего примера, загрузите модуль FMENU, который был создан в разделе «Создание меню».

1. Откройте меню FMENU.MMB.
2. Выберите элемент меню «Открыть». Наша задача – сделать так, чтобы при выборе этого элемента меню на экране появлялось окно выбора файлов. Находясь на элементе, вызовите всплывающее меню щелчком правой кнопки мыши и выберите пункт PL/SQL Editor.

В теле редактора наберите и скомпилируйте следующий код:

```
Вызов диалога «Открыть»
DECLARE
FileNM varchar2(300);
BEGIN
Filenm:=GET_FILENAME('');
…
END;
```

3. Далее выберите элемент «Сохранить». Наша задача – сделать так, чтобы при выборе этого элемента меню выполнялась фиксация всех изменений, сделанных в форме. Создайте PL/SQL-команду для этого элемента:

```
Команда «Сохранить»
BEGIN
DO_KEY('COMMIT_FORM');
END;
```

4. Выберите элемент «Отменить». Наша задача – сделать так, чтобы при выборе этого элемента меню выполнялся откат всех незафиксированных изменений в форме. Создайте PL/SQL-команду для элемента и наберите следующий код:

```
Команда «Отмена»
BEGIN
DO_KEY('ROLLBACK');
END;
```

5. Выберите элемент «Печать». Наша задача – сделать так, чтобы при выборе этого элемента вызывался отчет my_rep.rdf. Создайте PL/SQL-команду для элемента и наберите следующий код:

```
Команда «Печать»
DECLARE
rep_url varchar2(300);
BEGIN
rep_url:='http://<hostname><port>/reports/rwservlet?server='
||'Repsrv&report=my_rep.rdf&desformat=htmlcss&destype=cache '
||'&userid=user/pw@database&P_NAME ='||:parameter.param_
name||'&paramform
```

```
=no';
WEB.SHOW_DOCUMENT(rep_url,'_blank');
END;
```

6. Выберите элемент «Выход | Сохранить изменения». Наша задача – сделать так, чтобы при выборе этого элемента форма закрылась с сохранением изменений, но без проверки формы. Создайте PL/SQL-команду для элемента и наберите следующий код:

```
Команда «Выход|Сохранить изменения»
BEGIN
EXIT_FORM(do_commit, no_validate);
END;
```

7. Выберите элемент «Выход | Без Сохранения изменений». Наша задача – сделать так, чтобы при выборе этого элемента форма закрылась. Создайте PL/SQL-команду для элемента и наберите следующий код:

```
Команда «Выход|Без Сохранения изменений»
BEGIN
EXIT_FORM(no_commit, no_validate);
END;
```

8. Перейдем к следующему меню – «Блок». Выберите элемент меню «Блок | Следующий». Наша задача – сделать так, чтобы при выборе этого элемента выполнялся переход на предыдущий блок.

```
Команда «Блок | Предыдущий»
BEGIN
DO_KEY('NEXT_BLOCK');
END;
```

9. Выберите элемент меню «Блок | Предыдущий». Наша задача – сделать так, чтобы при выборе этого элемента выполнялся переход на предыдущий блок.

```
Команда «Блок | Предыдущий»
BEGIN
DO_KEY('PREVIOUS_BLOCK');
END;
```

10. Выберите элемент меню «Блок|Отчистить». Наша задача – сделать так, чтобы при выборе этого элемента выполнялась очистка блока.

```
Команда «Блок | Отчистить»
BEGIN
DO_KEY('CLEAR_BLOCK');
END;
```

11. Перейдем к следующему, заключительному, меню – «Запись». Выберите элемент меню «Запись | Первая». Наша задача – сделать так, чтобы при выборе этого элемента выполнялся переход на первую запись в блоке.

```
Команда «Запись | Первая»
BEGIN
FIRST_RECORD;
END;
```

12. Выберите элемент меню «Запись | Последняя». Наша задача – сделать так, чтобы при выборе этого элемента выполнялся переход на последнюю запись в блоке.

```
Команда «Запись | Последняя»
BEGIN
LAST_RECORD;
END;
```

13. Выберите элемент меню «Запись | Следующая». Наша задача – сделать так, чтобы при выборе этого элемента выполнялся переход на следующую запись в блоке.

```
Команда «Запись | Следующая»
BEGIN
DO_KEY('NEXT_RECORD');
END;
```

14. Выберите элемент меню «Запись | Предыдущая». Наша задача – сделать так, чтобы при выборе этого элемента выполнялся переход на предыдущую запись в блоке.

```
Команда «Запись | Предыдущая»
BEGIN
```

```
DO_KEY('PREVIOUS_RECORD');
END;
```

15. Выберите элемент меню «Запись | Создать». Наша задача – сделать так, чтобы при выборе этого элемента выполнялась вставка новой записи в блок.

```
Команда «Запись | Создать»
BEGIN
DO_KEY('CREATE_RECORD');
END;
```

16. Выберите элемент меню «Запись | Дублировать». Наша задача – сделать так, чтобы при выборе этого элемента выполнялось дублирование записи.

```
Команда «Запись | Дублировать»
BEGIN
DUPLICATE_RECORD;
END;
```

17. Выберите элемент меню «Запись | Удалить». Наша задача – сделать так, чтобы при выборе этого элемента запись была удалена.

```
Команда «Запись | Удалить»
BEGIN
DUPLICATE_RECORD;
END;
```

Если вы запрограммировали все элементы меню, можно приступать к подключению меню, предварительно выполнив компиляцию PL/SQL-кода – Program | Compile PL/SQL, а затем и компиляцию самого модуля. Скомпилировать модуль меню можно командой Program | Compile Module.

Использование PL/SQL в меню

Как и в случае с формой, вы можете управлять поведением меню и объектами, которые его составляют, во время выполнения, используя PL/SQL и встроенные подпрограммы Forms Builder. Основным отличием PL/SQL-программы, написанной в меню, от программы, написанной

в модуле формы, является возможность исключительно косвенного обращения к элементам меню.

Когда вы проектируете и программируете меню, вы должны не забывать о том, что меню – это прежде всего средство быстрого доступа к наиболее часто выполняемым операциям, а не средство для реализации сложной логики приложения: проверки введенных значений, создания блокировок и так далее. Например, если вы создаете элемент меню, выполняющий команду COMMIT, то различные проверки перед сохранением изменений лучше выполнить в соответствующих триггерах модуля формы. Так, выполнение команды DO_KEY('COMMIT') меню аналогично срабатыванию триггера «KEY-COMMIT» формы, поэтому необходимые проверки вычисления можно выполнить в нем. Ниже перечислены основные операции, которые лучше всего отображать в меню:

- взаимодействие с БД Oracle (операции DML и DDL);
- выполнение предыдущих действий;
- выполнение сортировки в блоке, то есть сортировка в блоке по убыванию или возрастанию;
- исполнение команд операционной системы;
- вызов другого продукта Developer (Oracle Reports, Oracle Graphics);
- вызов встроенной подпрограммы DO_KEY для исполнения триггера клавиши;
- вызов пользовательских триггеров встроенной подпрограммой EXECUTE_TRIGGER.

В этом случае использование пользовательских триггеров оправдано, так как в них вы можете выполнять анонимные PL/SQL-блоки с прямым обращением к элементам. Рассмотрим пример программирования элементов меню:

1. Взаимодействие с БД Oracle. Например, создайте пункт меню «Новая запись» и напишите для него следующий код:

```
Команда «Новая запись»
if :system.block_status!='NEW' then
   create_record;
ELSE
...
   END IF;
```

2. Вызовите пользовательский триггер:

```
Команда «Описание действия вашего триггера»
EXECUTE_TRIGGER('my_trg');
```

3. Вызовите другие продукты Oracle Developer:

```
Команда «Печать»
DECLARE
rep_url varchar2(300);
BEGIN
rep_url:='http://<hostname><port>/reports/rwservlet?server='
||'Repsrv&report=my_rep.rdf&desformat=htmlcss&destype=cache '
||'&userid=user/pw@database&P_NAME ='||:parameter.param_
name||'&paramform
=no';
WEB.SHOW_DOCUMENT(rep_url,'_blank');
END;
```

4. Выполните команду операционной системы, например, запустите калькулятор:

```
Команда «Калькулятор»
HOST('calc.exe');
```

5. Выполнение сортировки в блоке. Например, выполните сортировку по возрастанию:

```
Команда «Сортировка по возрастанию»
set_block_property ('block_name', ORDER_BY, :system.current_item
||' asc');
```

«Магические» элементы

Ранее в этой же главе мы уже затрагивали тему о магических элементах, теперь остановимся на ней подробнее. «Магические», или Мастер-элементы — это элементы, имеющие функциональность по умолчанию и не требующие дополнительной программной обработки. Мастер-элементы полностью соответствуют командам, выполняемым из стандартного меню, и не требуют дополнительного подключения клавиш акселератов и команд PL/SQL, так как Forms Builder выполняет все это автоматически. Ниже приведена таблица 22.2 всех типов Мастер-элементов.

Таблица 22.2. Типы Мастер-элементов

Мастер-элемент	Тип команды	Функциональность
ABOUT, UNDO	Все кроме MENU	Для того чтобы элемент выполнял нужную функцию, к нему обязательно и необходимо присоединить какую-либо команду, следовательно, функциональность по умолчанию отсутствует.
Clear, Copy, Cut, Paste	Null	Все перечисленные команды выполняют функции, определенные их именами, и при этом не требуется присваивать команду элементу меню.
Help	Menu	Для этого Мастер-элемента функциональность по умолчанию недоступна, поэтому вам необходимо назначить команду этого типа подменю.
Exit (Quit)	Null	Все перечисленные команды выполняют функции, определенные их именами, и при этом не требуется присваивать команду.
Window	Null или Menu	Элемент Window вызывает подменю по умолчанию со списком всех открытых окон. Операторы могут активизировать окно, выбрав его из подменю. Этот элемент может работать с двумя типами команд: Null, Menu.

Подключение меню к форме

Для того чтобы подключить меню к форме, необходимо настроить соответствующие параметры модуля формы: имя исполняемого файла меню, место размещения меню, имя меню. Выполните следующее упражнение.

1. Создайте новую форму.

2. Разместите хотя бы один элемент на форме, чтобы ее можно было запустить. Вы также можете открыть любую существующую форму для выполнения примера.
3. Вызовите палитру свойств модуля формы и установите значение свойств, как показано ниже:
   - Menu Module – FMENU;
   - Initial Menu – MYMENY.

   Примечание: *даже если файл меню расположен в той же директории, что и вызывающая форма, указывайте не только имя меню, но и путь, например, C:\FMENU.*

4. Откройте меню FMENU и выполните генерацию модуля для создания исполняемого файла FMENU.MMX. Если меню не будет сгенерировано, то при запуске формы вы получите ошибку чтения меню.
5. Откройте палитру свойств модуля меню FMENU и установите значение свойств, как показано ниже:
   - Main Menu – MYMENU;
   - Menu Directory – c:\forms\example\menu.

   Запустите форму на выполнение и попробуйте поработать с меню.

Если при изменении свойств вы допустили ошибку или какой-либо элемент меню не будет содержать PL/SQL-команду, то при попытке запуска меню вы получите сообщение об ошибке (рис. 22.12) – «Невозможно читать файл FMENU».

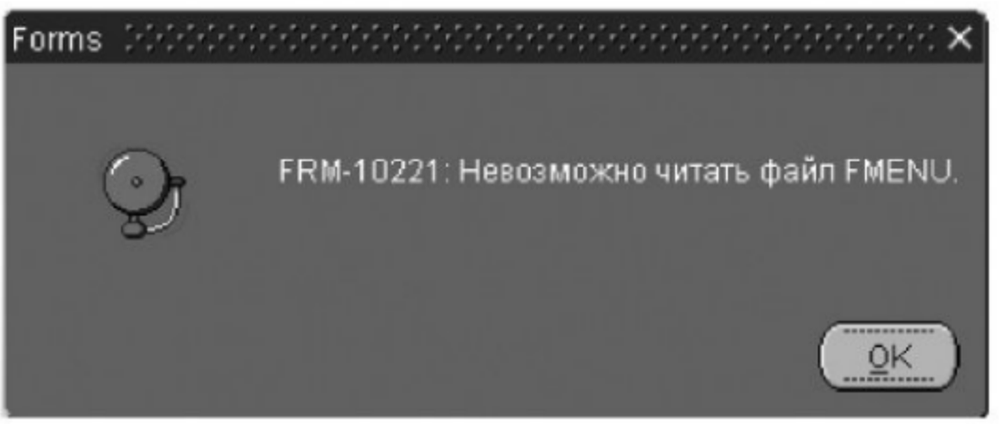

Рис. 22.12. Сообщение об ошибке при попытке запуска меню

Если вы работаете с многомодульным приложением, то можете назначить вызываемой форме текущее меню, используя параметр DO_REPLACE процедуры CALL_FORM при вызове формы процедурой CALL_FORM. По умолчанию при вызове формы процедурой CALL_FORM применяется параметр NO_REPLACE, который означает, что вызываемая форма не будет наследовать меню родителя.

Типы меню

В предыдущих разделах мы с вами научились создавать стандартное меню, элементы которого имели вид текстовых ярлыков. В Oracle Forms вы можете изменять вид отображаемого элемента и его функциональность, изменяя его тип. Свойство «Тип элемента меню» используется для изменения типа отображаемого пункта меню. Создавая элемент меню, вы можете выбрать один из четырех типов (табл. 22.3).

Таблица 22.3. Типы элементов меню

Значение	Описание
Простой (Plain)	Стандартный тип меню – просто ярлык с текстовой меткой.
Выбор (Check)	Рядом с текстовой меткой отображается элемент CheckBox в виде галочки. Такой тип меню может находиться в одном из двух состояний: True или False.
Радио (Radio)	Элемент меню превращается в одну из радиокнопок радиогруппы, также принимая значения True или False.
Разделитель (Separator)	Элемент меню превращается в линию-разделитель, которая визуально разделяет группу элементов меню на группы.
Магический (Magic)	Элемент меню, для которого предусмотрены встроенные функциональные средства реализации. Такой элемент может превращаться в одну из часто используемых команд: Вырезать, Копировать, Вставить и т. д. Вид команды для магического элемента можно выбрать в свойстве «Магический элемент». Магические элементы позволяют избежать лишнего кодирования часто используемых команд. Внешний вид такого элемента зависит от платформы, на которой производится запуск меню.

Для того чтобы установить новый тип элемента, достаточно изменить значение свойства «Тип элемента меню» (рис. 22.12).

Рис.22.12. Типы элементов меню

Радиокнопки и выключатели работают по такому же принципу, как и одноименные элементы модуля формы.

Примечание: *используйте элементы Check и Radio как переключатели каких-либо состояний, а не инициаторы каких-либо действий.*

Обращение к элементам меню

Для обращения к элементам меню используйте синтаксис:
имя_меню.имя_элемента
Для получения значения состояния радиокнопки или выключателя (выбор) меню используйте функцию GET_MENU_ITEM_PROPERTY, а для установки значения – функцию SET_MENU_ITEM_PROPERTY.

Создание срываемых меню

Срываемое меню – это меню, которое пользователь может «оторвать» от линейки меню и расположить в любом месте. Для того чтобы создать срываемое меню, выполните следующие действия:

1. Находясь в Menu Editor или в навигаторе объектов, выберите меню, например, MYMENU.
2. Вызовите палитру свойств для этого меню и установите свойство Tear-off равным True.

Лекция 23. Отладка приложения

В этой лекции слушатель научится отлаживать свое приложение, используя различные способы отладки. В лекции будут рассмотрены интерфейс отладчика Forms, запуск формы в режиме отладки, а также дополнительные методы отладки с помощью процедур и функций встроенного пакета Debug.

Ключевые слова: отладка, исключительные ситуации, пакет Debug, установка точек останова, отладка формы, подавление системных переменных, стек вызова, Raise_application.

Цель лекции: ознакомить слушателя с отладчиком Oracle Forms, его инструментами и параметрами режима отладки. Также слушатель научится использовать пакеты Debug и DBMS_ERROR для обработки и подавления системных сообщений.

Отладка

Разрабатывая приложение, вы так или иначе сталкиваетесь с ошибками выполнения и проектирования формы, идентифицировать которые бывает иногда просто, а иногда очень проблематично. Каждый разработчик знает, что отладка кода может занять существенную часть времени, отведенного на создание приложения. Если вы новичок в Oracle Forms, то для вас выяснение источника ошибки может превратиться в очень долгий и нервный процесс. В этой главе мы познакомимся с представленными Oracle Forms инструментами и встроенными подпрограммами для поиска источника и причины возникновения той или иной ошибки.

Методы отладки

Под методами отладки подразумеваются различные способы выявления причин и источников возникновения ошибок. Мы рассмотрим следующие методы отладки:

- Отображение отладочных сообщений.
- Обработка исключительных ситуаций.
- Компиляция и корректировка кода в редакторе PL/SQL.
- Работа с Forms Debug Console.

Комментарии и сообщения

Комментарии помогают не только оставлять пояснения к тексту программы, но и комментировать фрагменты программы, помечая ее как невыполнимую часть кода. Комментарии относятся к очень простому методу отладки и используются чаще всего в том случае, когда вы наверняка знаете, где находится ошибка. Если вы не уверены в каком-либо фрагменте кода, вы можете закомментировать этот фрагмент и, вновь запустив программу, проверить, выполняется ли программа дальше или ошибка все же осталась. Вы можете отслеживать ошибку последовательно, комментируя строки программы шаг за шагом. Этот метод хоть и прост, но является довольно трудоемким, когда текст программы очень большой. Также недостатком этого метода является и то, что вам необходимо помнить о том, какие строки нужно обратно раскомментировать. Ниже для сравнения приведено два фрагмента программы, в одном из которых заведомо определена исключительная ситуация.

```
DECLARE
Val1 number;
Val2 number;
BEGIN
Val1:=1/0;
Val2:=2+3;
Message(val2);
END;
```

Если вы попробуете выполнить вышеописанный пример, то получите ошибку «ORA-01476». Для того чтобы узнать, какой оператор его вызвал, val1 или val2, вам достаточно закомментировать оператор val1 и попробовать выполнить программу снова.

```
DECLARE
Val1 number;
Val2 number;
BEGIN
--Val1:=1/0; -- место предполагаемой ошибки закомментировано.
Val2:=2+3;
Message(val2);
END;
```

Выполнив этот пример, вы увидите, как после комментирования оператора val1 ошибка исчезнет и на экране появится сообщение со зна-

чением val2. Вам станет сразу ясно, какая из строк возбуждает исключительную ситуацию.

Сообщение – это еще один способ отслеживания выполняемого кода программы. Встроенная процедура MESSAGE позволяет вам выводить любое сообщение на экран. Расположив процедуру под наблюдаемым оператором и передав его в качестве параметра, вы можете узнать его значение в ходе выполнения программы. Сообщения могут выводиться в статусной строке формы, для этого свойство «Console Window» формы должно быть правильно настроено, или на экран. В предыдущем примере мы использовали сообщение для вывода на экран значения переменной.

Сообщения – это одна из самых часто используемых процедур, так как применяется она не только для вывода тематических сообщений пользователю, но и ошибок. Процедура MESSAGE дает гораздо больший эффект, если используется совместно с процедурами SYNCHRONIZE и PAUSE.

SYNCHRONIZE синхронизирует экран терминала и внутреннее состояние формы, то есть SYNCHRONIZE обновляет вывод на экран (рис. 23.1) для отображения информации, которую Forms имеет в своем внутреннем представлении экрана. Другими словами, SYNCHRONIZE гарантирует вывод сообщения на экран в требуемой точке кода. Ниже приведен пример использования процедуры.

```
message('Сообщение 1');
synchronize;
message('Сообщение 2');
synchronize;
message('Сообщение 3');
synchronize;
```

PAUSE синхронизирует вывод на экран и приостанавливает работу приложения до тех пор, пока оператор не нажмет клавишу подтверждения. PAUSE выводит предупреждающее сообщение на экран (рис. 23.2).

Рис. 23.1. Вывод сообщения на экран.

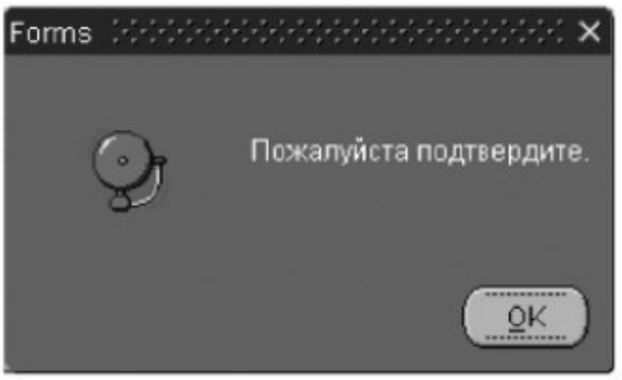

Рис. 23.2. Сообщение команды PAUSE

```
message('Сообщение 1');
PAUSE;
```

Что касается ограничений, то здесь нужно заметить, что длина выводимого сообщения не может превышать 200 символов. Не злоупотребляйте сообщениями и не забывайте комментировать отладочные сообщения. Так, например, сообщение, забытое в цикле, может привести к большим неудобствам. Чрезмерное количество сообщений опасно тем, что возникшая ошибка может быть перекрыта вашим сообщением, хотя этого можно избежать, если использовать MESSAGE совместно с командой SYNCHRONIZE.

Исключительные ситуации

Исключительные ситуации – это мощная конструкция языка PL/SQL, позволяющая разработчикам перехватывать и обрабатывать неожиданные и непредвиденные ошибки. Когда мы разрабатываем приложение, мы изначально и заведомо создаем идеальную модель, которая, как нам кажется, не содержит никаких ошибок. Если вы пишете простую программу, которая выдает результат деления двух чисел, вам и в голову не может прийти, что пользователь, работающий с вашей программой, не знает, что на 0 делить нельзя, и как следствие получает ошибку «FRM-40735: WHEN-BUTTON-PRESSED триггер вызвал необработанное исключение ORA - 01476». Ниже приведен рисунок, на котором изображен результат ошибочного действия пользователя, а также сам оператор, возбуждающий исключительную ситуацию (рис. 23.3).

```
BEGIN
:result:=:delimoe/:delitel;
END;
```

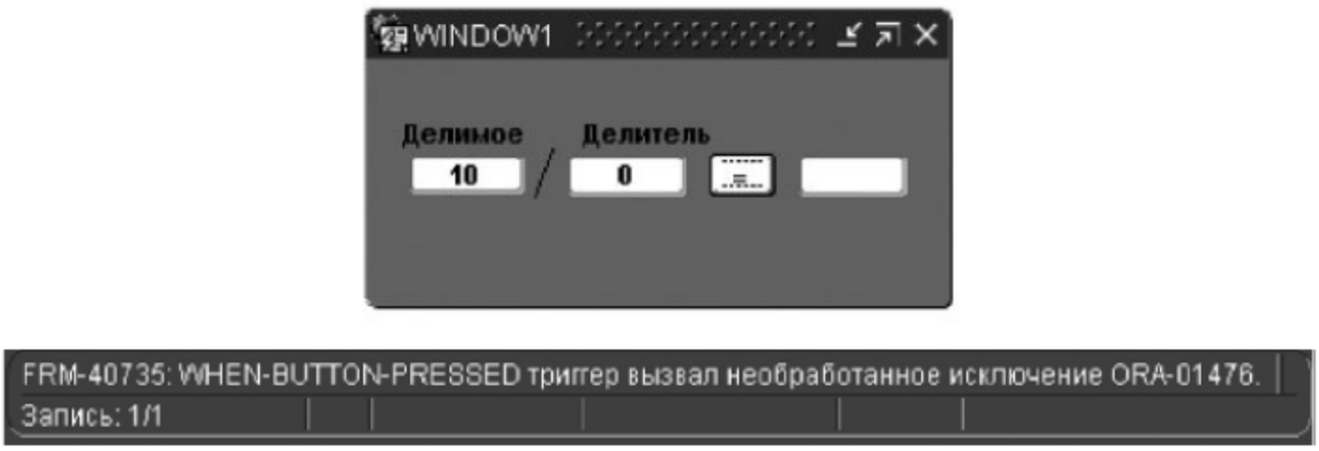

Рис. 23.3. Деление на ноль

Обработка исключительных ситуаций

В вышеописанном примере мы вызываем ошибку ZERO_DIVIDE делением 10 на 0. В этом случае мы получаем сообщение «ORA–01476» без описания причины. Для обработки этой исключительной ситуации добавьте в ваш блок секцию EXCEPTION. В случае возникновения исключительной ситуации будут выполнены операторы, описанные в этой секции, то есть при выполнении программы обработка завершается на операторе, возбудившем исключительную ситуацию, и управление сразу передается в секцию EXCEPTION. Исключительная ситуация может быть обработана в конце любого блока BEGIN-END.

```
BEGIN
:result:=:delimoe/:delitel;
EXCEPTION
   WHEN ZERO_DIVIDE THEN
   :delitel:=1;
END;
```

Вы также можете существенно сократить время на поиск причины ошибки, если обработаете исключительную ситуацию с помощью встроенных функций, которые отразят текст ошибки (рис. 23.4). Так, в предыдущем примере вместо текста «необработанное исключение» вы можете получить текст, дающий полное представление о причине ошибки, то есть «divisor is equal to zero».

```
BEGIN
:result:=:delimoe/:delitel;
```

```
EXCEPTION
   WHEN ZERO_DIVIDE THEN
   message(sqlerrm);
END;
```

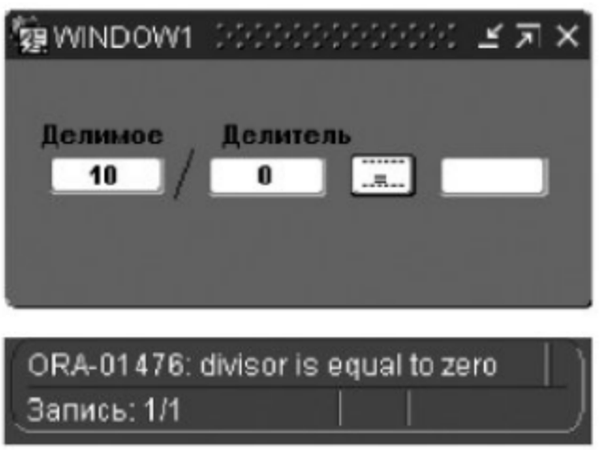

Рис. 23.4. Сообщение об ошибке

Пользовательские исключительные ситуации

Сообщение об ошибке типа «ORA-01476: divisor is equal to zero» тоже не является показательным, так как не каждый пользователь знает английский язык, да и фраза «ORA-01476» лишняя. Для того чтобы генерировать собственные сообщения, вы можете определить собственную исключительную ситуацию. Пользовательская исключительная ситуация может быть возбуждена в любом месте блока, в котором она объявлена. Чтобы создать пользовательское исключение, достаточно в секции DECLARE объявить переменную и присвоить ей тип exception. Ниже продемонстрирован пример (рис. 23.5), в котором мы объявляем собственную исключительную ситуацию и заменяем сообщение sqlerrm.

```
DECLARE
   myexcept exception;
BEGIN
IF :delitel =0 THEN
   RAISE myexcept; ELSE
:result:=:delimoe/:delitel;
END IF;
EXCEPTION
   WHEN myexcept THEN
message('Деление на ноль. Делитель должен иметь значение, отличное
от нуля');
END;
```

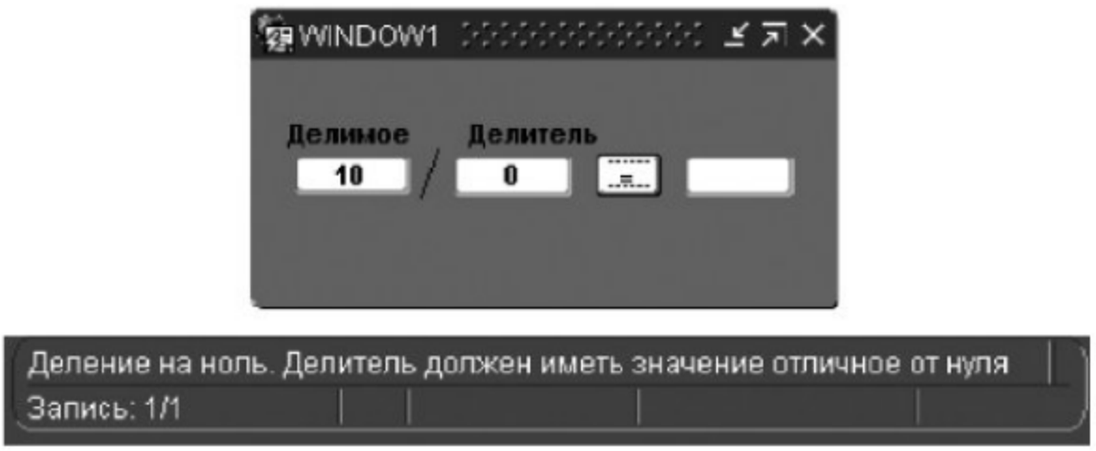

Рис. 23.5. Пользовательская исключительная ситуация

Связывание исключительной ситуации с кодом ошибки

Сервер Oracle может связывать определенные исключительные ситуации с кодами стандартных ошибок. Чтобы связать пользовательскую исключительную ситуацию, используйте прагму EXCEPTION_INIT. Ниже приведен пример, в котором показан пример объявления прагмы.

```
DECLARE
   myexcept exception;
   pragma exception_init(myexcept,-1476);
BEGIN
:result:=:delimoe/:delitel;
EXCEPTION
   WHEN myexcept THEN
message('Деление на ноль. Делитель должен иметь значение, отличное
от нуля');
END;
```

Как вы уже успели заметить, прагма EXCEPTION_INIT в качестве параметров принимает имя исключительной ситуации и код ошибки.

Процедура RAISE_APPLICATION_ERROR ()

Процедура RAISE_APPLICATION_ERROR – это еще один способ возбуждения исключительной ситуации, который не требует предварительного объявления и может быть вызван в любом месте блока. Вы можете связать эту исключительную ситуацию с пользовательской исключительной ситуацией, используя прагму EXCEPTION_INIT. Ниже приведен синтаксис процедуры и пример ее использования.

RAISE_APPLICATION_ERROR (номер_ошибки, сообщение, [TRUE|FALSE]);

Номер_ошибки – любое отрицательное число, которое лежит в диапазоне от [–20999, –20000].

Сообщение – это сообщение, выводимое при возбуждении исключительной ситуации. Максимальный размер текста для выводимого сообщения не должен превышать 2048 символов.

TRUE | FALSE – этот параметр является необязательным; если его значение истинно, то ошибка добавляется к стеку ошибок, иначе все предыдущие ошибки будут заменены.

Неименованные исключительные ситуации

Неименованные исключительные ситуации – это такой вид ситуации, который невозможно предугадать, или ситуация, которая должна произойти, но не представляет для вас особого интереса. Для обработки таких ситуаций существует ключевое слово OTHERS. OTHERS – это обработчик исключительных ситуаций, предназначенный для обработки всевозможных исключительных ситуаций в блоке. Ниже приведен рисунок 23.6, демонстрирующий работу обработчика OTHERS.

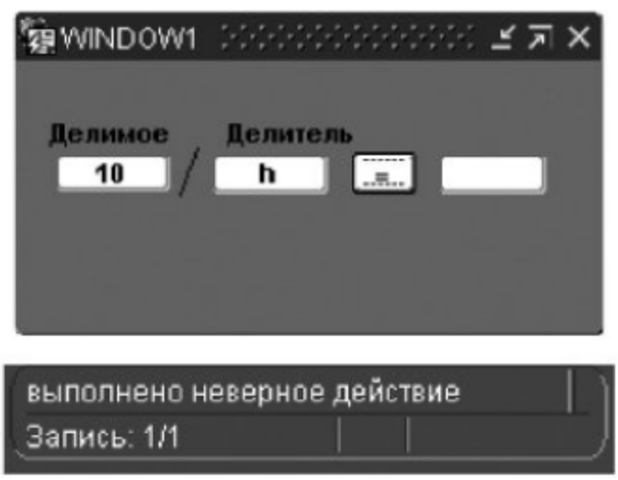

Рис. 23.6. Обработчик OTHERS

Функции SQLERRM и SQLCODE

Встроенные функции SQLERRM и SQLCODE возвращают текст и код ошибки для последней исключительной ситуации (рис. 23.7). Эти функции полезно использовать в обработчике OTHERS, так как они позволяют установить причину возникновения ошибки.

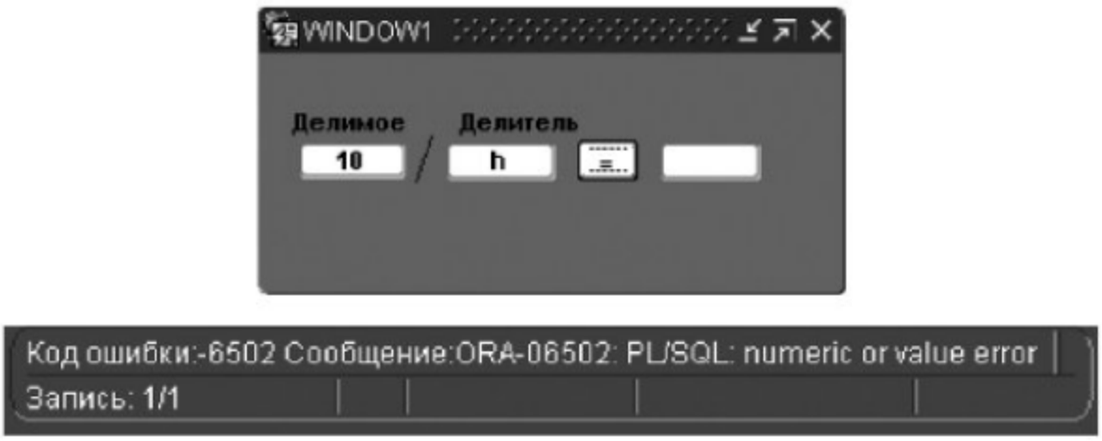

Рис. 23.7. Сообщение SQLERRM и SQLCODE

DBMS_ERROR_CODE

Эта функция возвращает номер последней ошибки, возникшей в базе данных. Эта функция не имеет ограничений в использовании. Для рекурсивных ошибок DBMS_ERROR_CODE возвращает код первого сообщения в стеке. Рассмотрим пример, в котором обработаем ошибку ORA-01438 is «value too large for column», но прежде чем приступить к выполнению примера, обратите внимание на синтаксис функции.

Пример 1. Триггер ON-ERROR. Обработка ошибки

```
DECLARE
ecode NUMBER := ERROR_CODE;
dbms_ecode NUMBER;
dbms_etext VARCHAR2(200);
BEGIN
IF ecode = 40508 THEN
Dbms_ecode := DBMS_ERROR_CODE;
Dbms_etext := DBMS_ERROR_TEXT;
IF dbms_ecode = -1438 THEN
Message('Введенное значение превышает допустимое! Введите заново!);
ELSE
Message('Вставка не выполнена, так как '||dbmserrtext);
END IF;
END IF;
```

Обработка NULL - значений

Несмотря на то что обработчик OTHERS ловит все типичные исключительные ситуации, такую ситуацию, как «невведение данных», он обработать не может. Попробуйте в предыдущем примере оставить поле

«Делитель» пустым, и вы увидите, что ничего не происходит. Обычно такие ситуации закрываются с помощью настройки определенных свойств элемента. Вы можете установить для поля делитель свойство «Required» равным «True», делая элемент обязательным для ввода. Но можно поступить и по-другому, воспользовавшись функцией NVL, как показано ниже (рис. 23.8).

```
DECLARE
   myexcept exception;
   pragma exception_init(myexcept,-1476);
BEGIN
:result:=:delimoe/NVL(:delitel,5);
EXCEPTION
   WHEN myexcept THEN
message('Деление на ноль. Делитель должен иметь значение, отличное
от нуля');
END;
```

Рис. 23.8. Обработка NULL-значений

Правильная обработка исключительных ситуаций имеет большое значение для разработчика, сокращая его время на поиск и обработку ошибок, ускоряя отладку приложения.

Forms Debugger (Debug Console)

В этом разделе мы познакомимся с более мощным инструментом отладки Oracle Forms – Forms Debugger. Forms Debug Console – это инструмент, который дает разработчику контроль над приложением в режиме отладки и полную информацию о текущем состоянии формы и всех выполняющихся внутри нее операций. Отладчик позволяет вам отслеживать выполнение операторов триггеров и программных модулей пошагово, а также дает вам возможность изменять значение переменных. Исследуя среду выполнения отладчиком, вы получаете возможность ана-

лизировать переменные, параметры, системные переменные, значения отображаемых и неотображаемых элементов. Инструмент Debug Console (рис. 23.9) имеет простой интерфейс, представляющий собой рабочую область, на которой вы можете наблюдать ход выполнения программы и значения переменных окружения и внутреннего представления Forms. В Debug Console вы можете управлять видимостью разделов, перечисленных ниже, переключая кнопки на горизонтальной панели отладчика:

- Stack;
- Variables;
- Watch;
- Form Values;
- PL/SQL packages;
- Global/System Variables;
- Breakpoints.

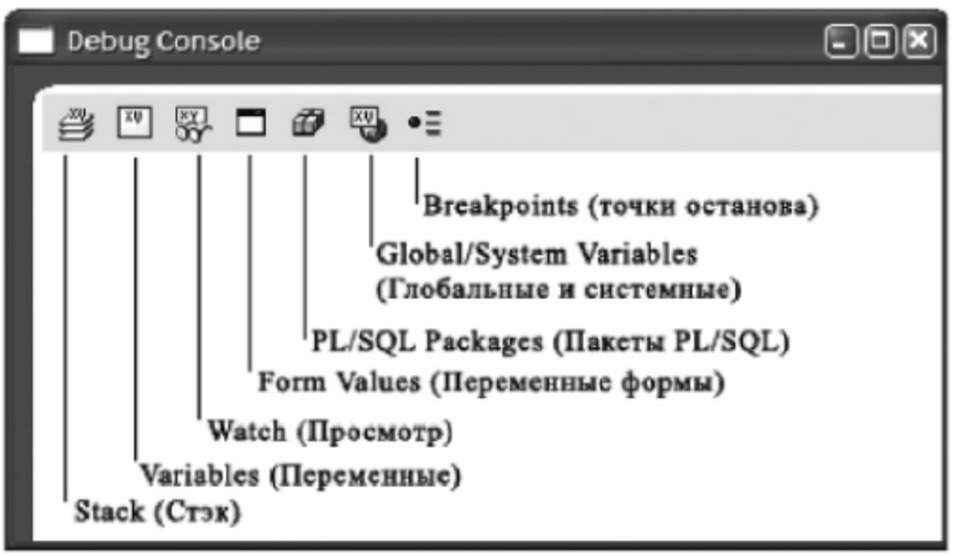

Рис. 23.9. Debug Console

Для того чтобы лучше понять принцип работы отладчика, рассмотрим все разделы на примере. Для выполнения примера выполните следующие действия:

1. Создайте форму и назовите ее DEB.fmb.
2. Создайте небазовый блок и разместите в нем одну кнопку.
3. Создайте параметр и установите его свойства: Name=result3, Type=number.
4. Создайте триггер WHEN-BUTTON-PRESSED для кнопки и напишите в нем нижеприведенный код:

```
WHEN-BUTTON-PRESSED
DECLARE
   value1 number;
```

```
    value2 number;
    result1  number;
    result2  number;
    result3  number;
BEGIN
    value1:=1;
    value2:=2;
    :global.result1:=value1+value2;
    result2:=1/0;
    :parameter.result3:=value2-value1;
    END;
```

Для того чтобы вызвать Forms Debugger, достаточно нажать кнопку  «Run Form Debug», которая находится справа от кнопки «Run Form» в верхней горизонтальной панели Forms. Когда вы попробуете запустить форму в режиме отладки, ничего особенного не произойдет, так как форма будет выполняться в привычном для нее режиме до тех пор, пока не встретит точку прерывания программы.

Breakpoints

Точка останова – это отметка, сообщающая форме, в каком месте нужно остановить выполнение программы и вызвать отладчик. Точки останова можно устанавливать только на выполняемых операторах, то есть точки останова, установленные на операторах типа begin, end, declare, if вызовут сообщение об ошибке. Точка останова имеет вид красной окружности и устанавливается слева от выполняемого оператора, как показано на рис. 23.10.

Для того чтобы установить точку останова, щелкните два раза левой кнопкой мыши на левой вертикальной панели PL/SQL-редактора напротив выполняемого оператора. Вы можете установить точку останова, используя один из перечисленных способов:

1. Нажатием клавиши F5. Если точка останова уже существует на этом операторе, то она удаляется.
2. Командой меню Debug | Insert/Remove Breakpoint.

Вы можете устанавливать точки останова как перед запуском формы в режиме отладки, так и после запуска. Установите точки останова на операторах value1, value2, :global.result и запустите форму в режиме отладки. Теперь, когда форма запущена и точки останова установлены, перейдем непосредственно к отладке.

Для того чтобы приступить к отладке, нажмите кнопку в ранее запущенной форме. После того как вы нажмете кнопку, выполнение кода пре-

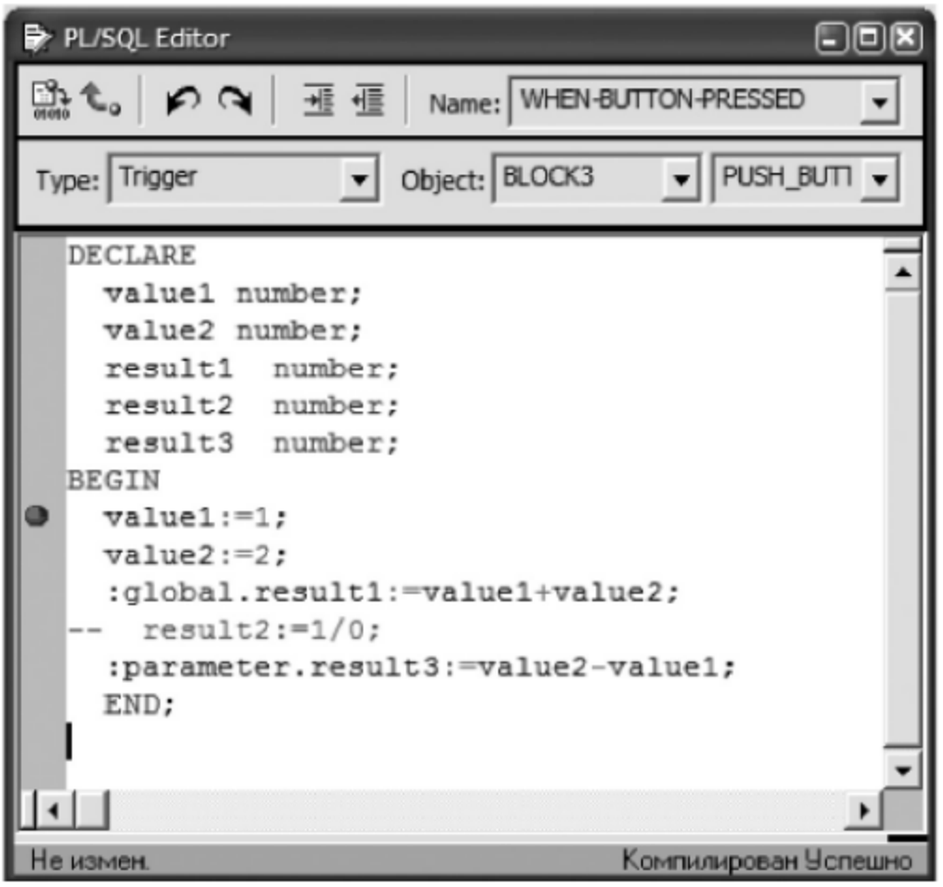

Рис. 23.10. Установка точки останова

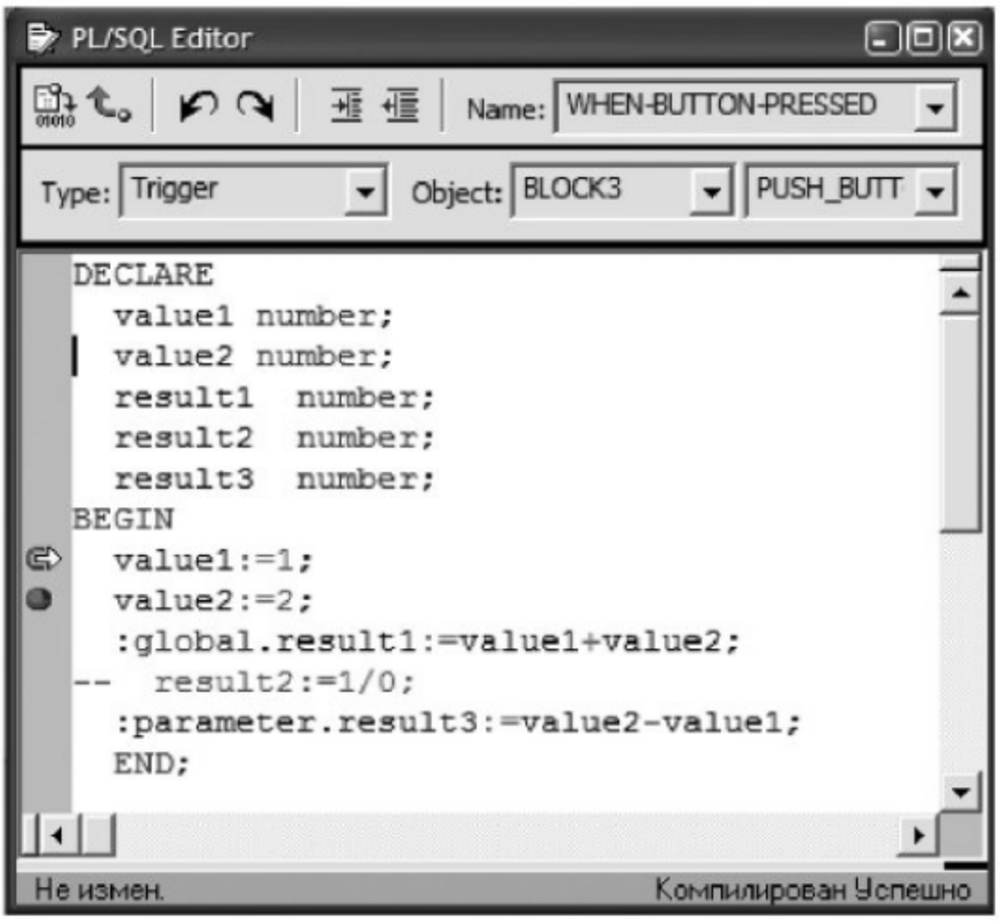

Рис. 23.11. Текущая точка останова

рвется и на самой первой точке останова появится стрелка, означающая текущую точку входа, как показано на рис. 23.11.

Приступим к рассмотрению разделов Debug Console и начнем с раздела Stack.

Стек вызова (Call Stack)

Стек вызова представляет собой цепочку подпрограмм, которая начинается с точки вызова и заканчивается выполняющейся в настоящее время программой. Для того чтобы активировать окно стека, выполните команду Debug | Debug Windows Stack|Stack или нажмите кнопку Stack на панели инструментов Debug Console (рис. 23.12).

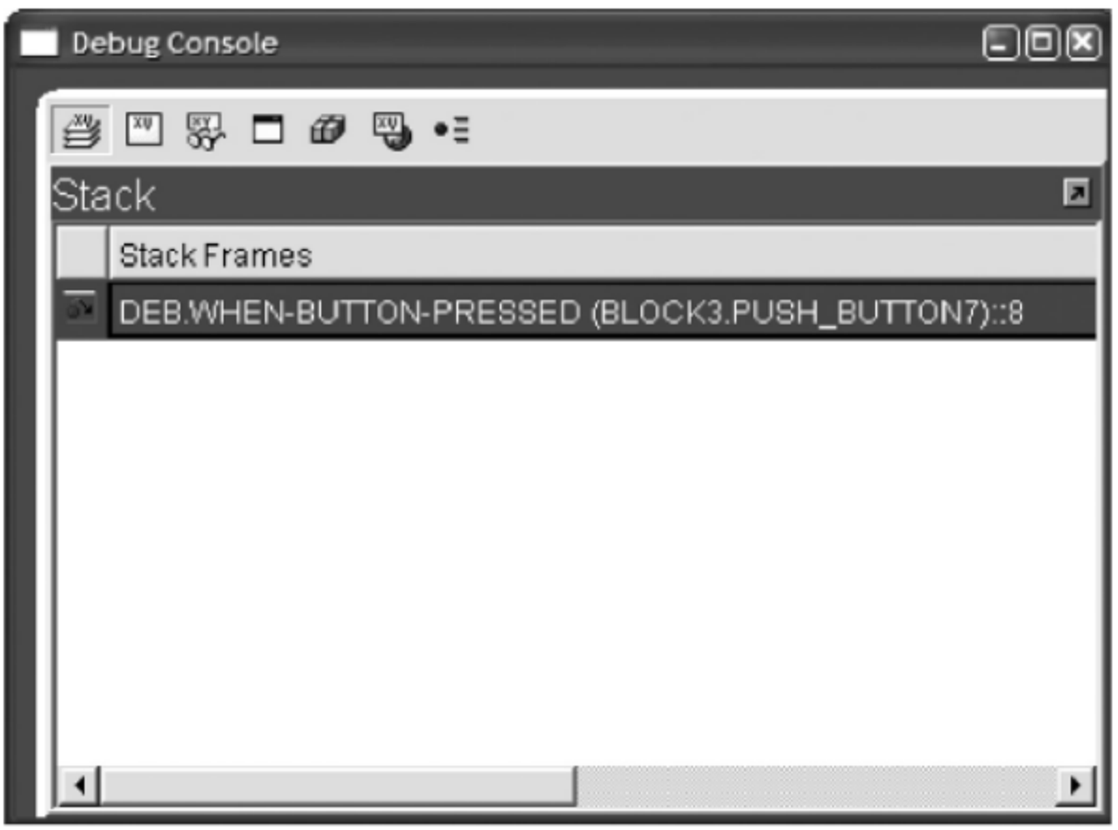

Рис. 23.12. Окно «Стек»

Нажатием кнопки мы инициировали событие WHEN-BUTTON-PRESSED, которое является точкой входа в стек вызовов. Как вы сможете далее увидеть, кадры стека размещаются в порядке их вызова, то есть самый первый вызов будет находиться в нижней части стека, а последний, соответственно, будет самым первым. Название стека кадра состоит из имени модуля, события, элемента и строки вызова.

```
Имя_модуля.программная_единица.[имя_блока.имя_элемента]::номер_строки
```

Переменные (Variables)

Окно Variables (рис. 23.13) предназначено для вывода значений переменных кадра стека. Для того чтобы активировать окно переменных, выполните команду Debug | Debug Windows | Variables или нажмите кнопку Variables на панели инструментов Debug Console.

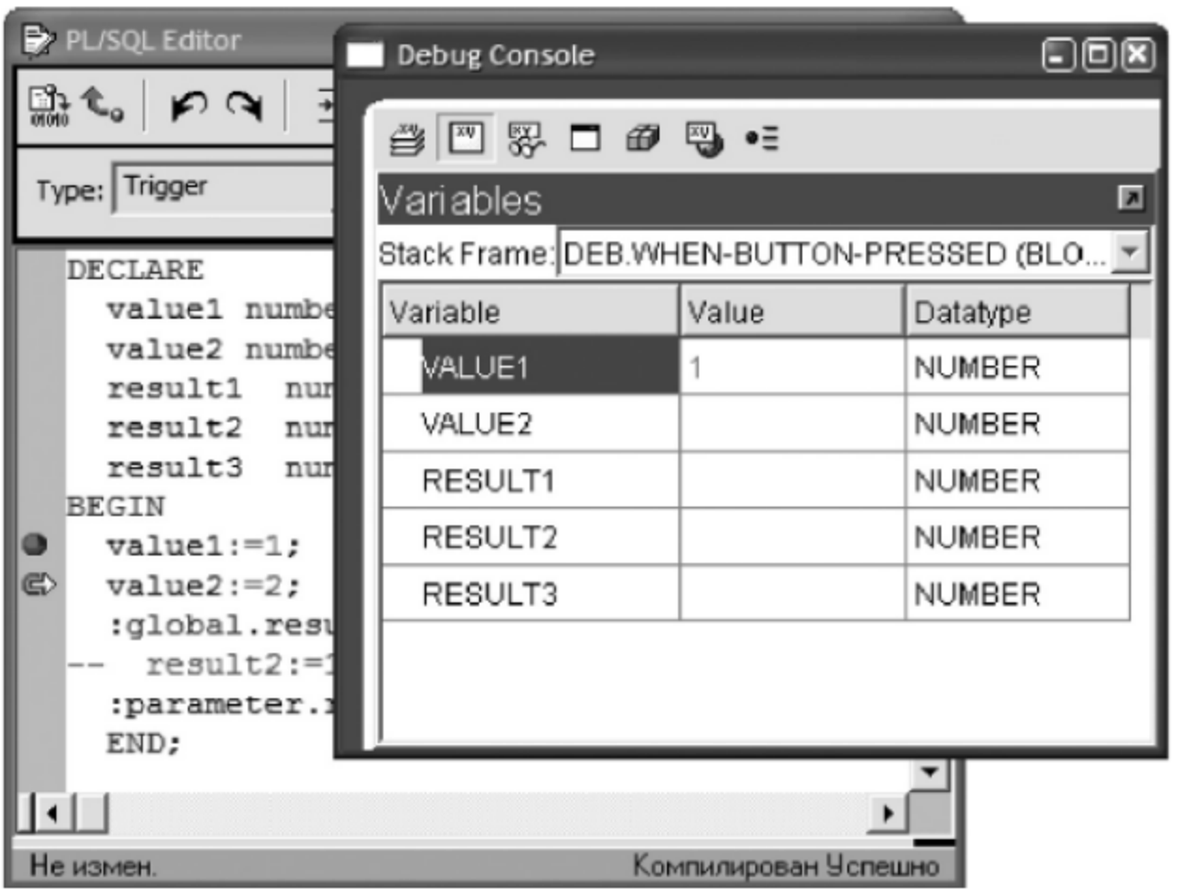

Рис. 23.13. Окно «Переменные»

Как видно на рисунке, окно отражает имя, значение и тип переменных стека, а также имя текущего кадра стека в поле «Stack Frame». Чтобы понять, как вам может помочь этот раздел, нажмите кнопку «Step Into» для вхождения в следующую точку. После перехода в новую точку останова вы увидите, как в поле «Value» отразится значение переменной, то есть 1. Вы можете изменить это значение на любое другое допустимое значение. Модифицирование переменных на этапе отладки дает вам возможность определить, как изменится поведение программы или как повлияет на результат изменение значения той или иной переменной. Исключением являются параметры, значения которых не редактируются и считаются read only. Значения всех не модифицируемых переменных подсвечиваются при попытке изменения.

Watch (Просмотр)

Для того чтобы добавить переменную в окно просмотра, выполните следующие действия:

1. Откройте окно «Variables» или «Form Value» и выберите переменную, которую хотите добавить.
2. Нажмите правую кнопку мыши для вызова всплывающего меню и выберите пункт «Add to Watch». После этого в окне просмотра появится выбранная вами переменная, как показано на рис. 23.14.

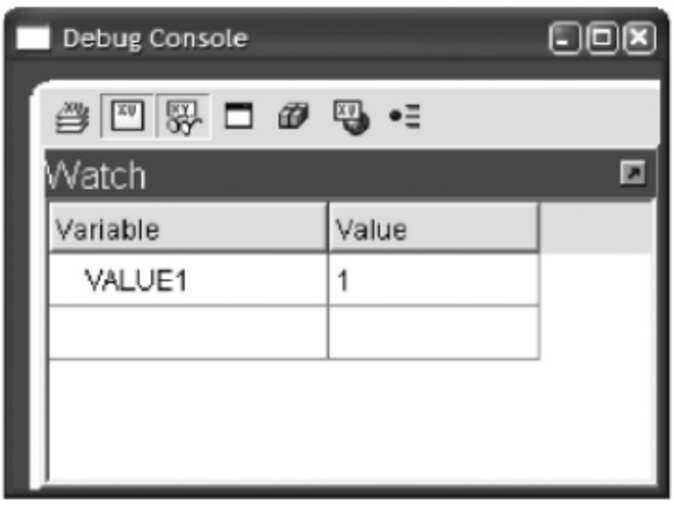

Рис. 23.14. Окно «Watch»

Для того чтобы удалить переменную из окна просмотра, выполните следующие действия:

1. Откройте окно «Watch» и выберите переменную, которую хотите удалить.
2. Вызовите всплывающее меню правым щелчком мыши и выберите пункт «Remove», после чего переменная исчезнет из списка. Чтобы очистить все окно «Watch», выберите пункт «Remove All» всплывающего меню.

Form Values (Значения формы)

Окно «Form Values» отражает значения всех элементов и параметров текущей формы. Для того чтобы активировать окно переменных, выполните команду Debug | Debug Windows | Form Values или нажмите кнопку Form Values на панели инструментов Debug Console.

Как видно на рисунке 23.15, окно содержит две вкладки – «Items» и «Parameters», а также имя текущего модуля в поле «Module». Вкладка «Items» отображает значение элементов блока, а вкладка «Parameters» отображает имя параметра и его значение. Вы можете изменять значения

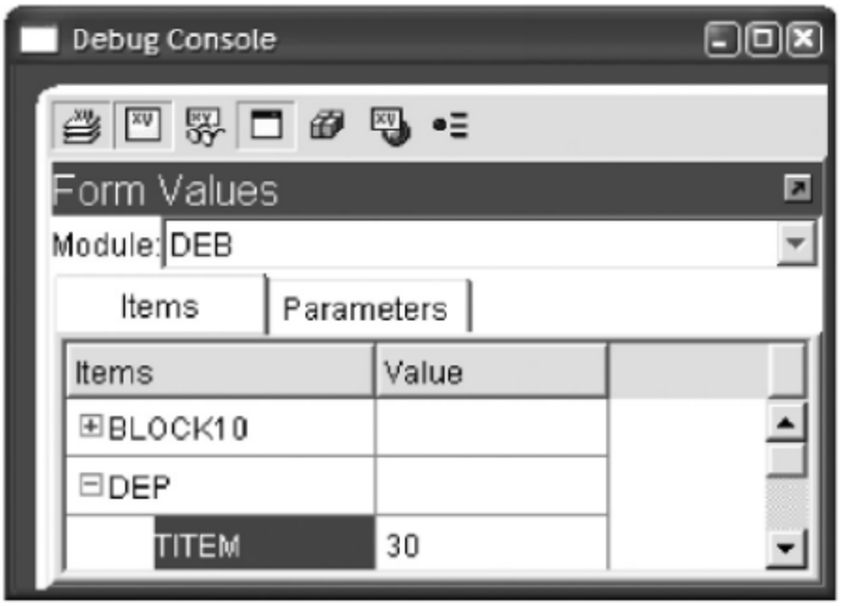

Рис. 23.15. Окно «Form Values»

элементов, если они не относятся к числу read only, для тестирования возможных вариантов поведения программы. Например, значение элемента display item нельзя модифицировать.

PL/SQL Packages (Пакеты PL/SQL)

Вы можете использовать окно «PL/SQL Packages» для просмотра выполняемого PL/SQL-пакета. В этом окне вы можете отслеживать только переменные пакета, все остальные значения, такие как возвращаемое значение функции и переменные, задействованные в методах тела пакета, будут отображены в окне переменных кадра стека DEB.SPECK_PACK.

Как показано на рисунке 23.16, окно «PL/SQL Packages» состоит из двух столбцов, отображающих имя и значение переменной, и радиогруппы переключателей, управляющих отображением пакетов. Вы можете выбирать, какие пакеты отображать, а какие нет; так, например, выбирая переключатель «Client», вы получите возможность просматривать только клиентские пакеты, то есть те, которые определены непосредственно в модуле формы. Для того чтобы отобразить серверные и клиентские пакеты, нужно выбрать переключатель «All». В поле «Packages» отображается имя текущего пакета, значения переменных которого отображаются в области окна.

Для того чтобы посмотреть, как отображаются данные в этом окне, создайте простой пакет, скрипт которого приведен в листинге 23.1.

Листинг 23.1. Program Unit

```
PACKAGE speck_pack IS
  X number:=2;
```

```
  FUNCTION a RETURN NUMBER;
END;
 PACKAGE BODY speck_pack IS
  FUNCTION A RETURN NUMBER
  IS
  val number;
  BEGIN
   val:=1+2;
   RETURN X;
   END;
END;
```

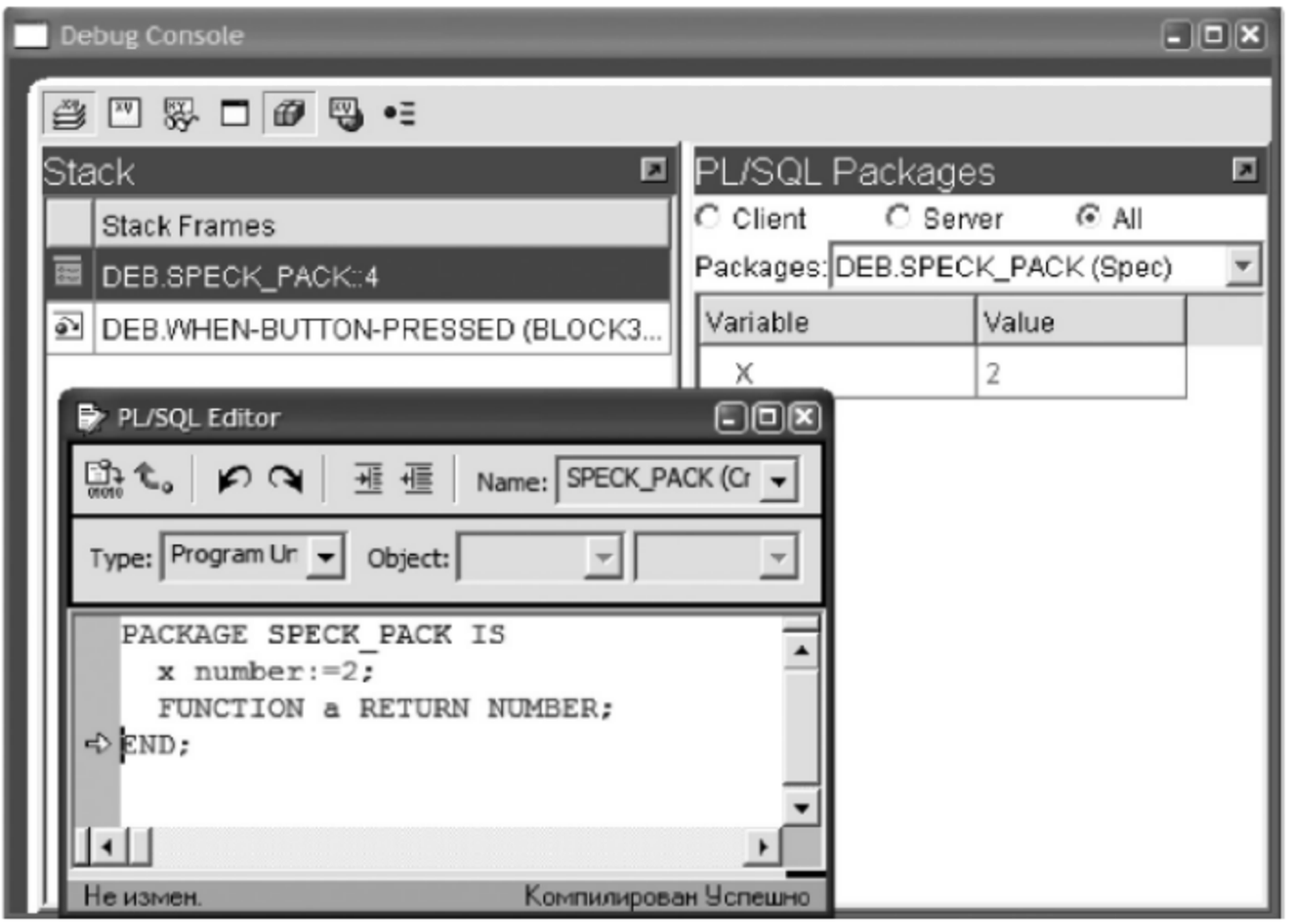

Рис. 23.16. Окно «PL/SQL Packages»

Добавьте в триггер WHEN-BUTTON-PRESSED новую переменную для вызова функции пакета. Ниже приведен листинг триггера, в котором новые строки выделены жирным шрифтом.

```
WHEN-BUTTON-PRESSED
DECLARE
   value1 number;
```

```
    value2 number;
    result1  number;
    result2  number;
result4  number;
    BEGIN
    value1:=1;
    value2:=2;
result4:=speck_pack.a;
    :global.result1:=value1+value2;
    --result2:=1/0;
    :parameter.result3:=value2-value1;
    END;
```

Теперь для того, чтобы просмотреть значения переменных пакета во время выполнения формы в режиме отладки, повторите действия, которые мы совершали, рассматривая предыдущие примеры. Для этого запустите форму в режиме отладки, предварительно установив точки останова на операторах value1 и value2, и нажмите кнопку. После того как выполнение кода прервется на операторе value1, продолжите пошаговое выполнение программы, нажимая кнопку «Step Into» до тех пор, пока курсор не попадет на оператор *:global.result1*. Как только точка входа устанавливается для оператора result4, отладчик перемещает вас в программный модуль пакета и помещает в стек вызовов новый кадр, двигаясь по строкам которого, вы получаете возможность отслеживать состояние глобальных переменных пакета и операторов, задействованных в теле пакета. Заметьте, что новый кадр помещается в верхнюю часть стека.

Global/System Variables (Глобальные и системные переменные)

Используйте окно «Global/System Variables» для просмотра значений переменных командной строки, системных и глобальных переменных. Как показано на рис. 23.17, окно «Global/System Variables» состоит из трех вкладок – Global, System и Command Line.

Для того чтобы просмотреть значение глобальной переменной, нажимайте кнопку «Step Into» до тех пор, пока курсор не покинет оператор *:global.result1*. Значения глобальных переменных можно изменять, а вот значения переменных Command Line изменять нельзя, так как все они имеют статус read only. Все системные переменные за исключением четырех также имеют статус read only. Системные переменные DATE_TREASHOLD, EFFECTIVE_DATE, MESSAGE_LEVEL и SUPRESS_WORKING являются теми самыми исключениями, значения которых можно изменять.

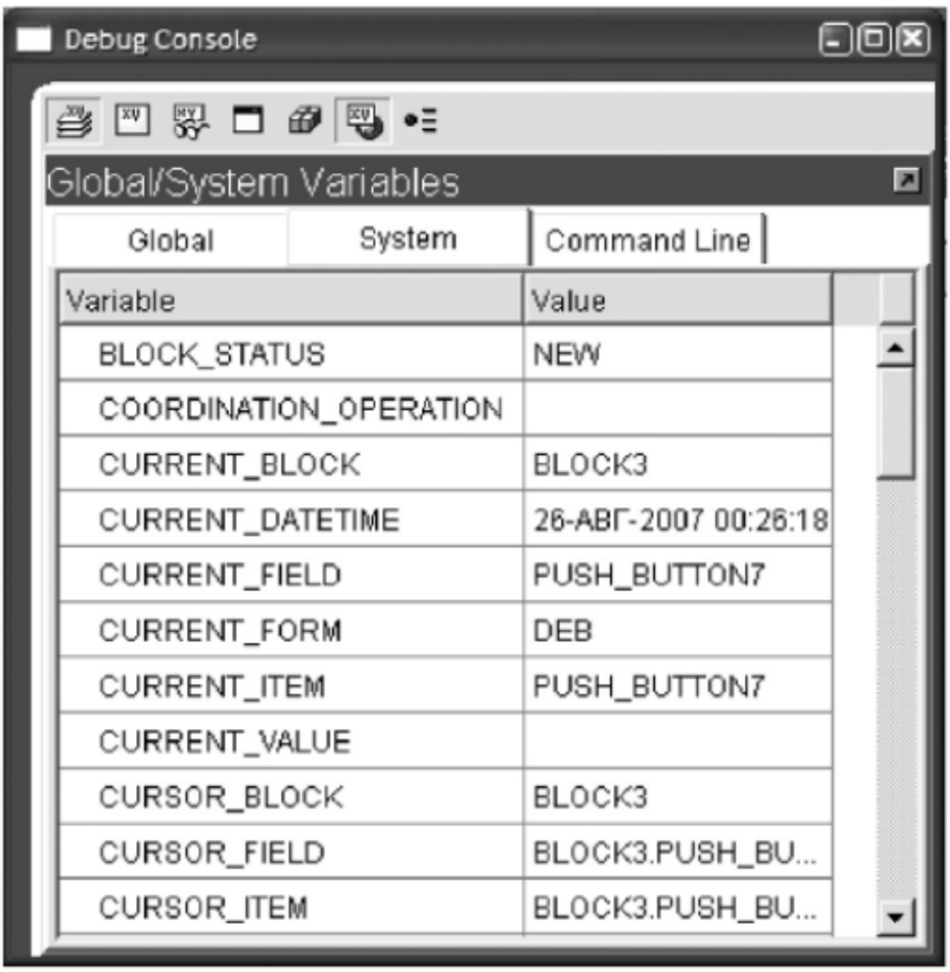

Рис. 23.17. Окно «Global/System Variables»

Breakpoints (Точки останова)

Окно «Breakpoints» позволяет не только видеть установленные точки останова, но и управлять ими.

Как показано на рисунке 23.18, вкладка «Breakpoints» отображает три столбца: Source, Enable и Line. Значение, приведенное в столбце «Line», указывает на номер строки, на которой установлена точка останова. Столбец «Source» – это локация точки останова. Вы можете управлять точками останова, делать их активными или неактивными, сбрасывая флаг выключателя. Если флажок сброшен, то точка останова считается неактивной и имеет серый цвет.

Вкладка «Break On Exceptions» отображает типичные исключительные ситуации, которые могут возникнуть в вашем приложении. Вкладка имеет три столбца, первый из которых позволяет включить или выключить исключительную ситуацию на выбранной точке останова (рис.23.19). Столбец «Number» отображает код ошибки для исключительной ситуации столбца «Exception Name».

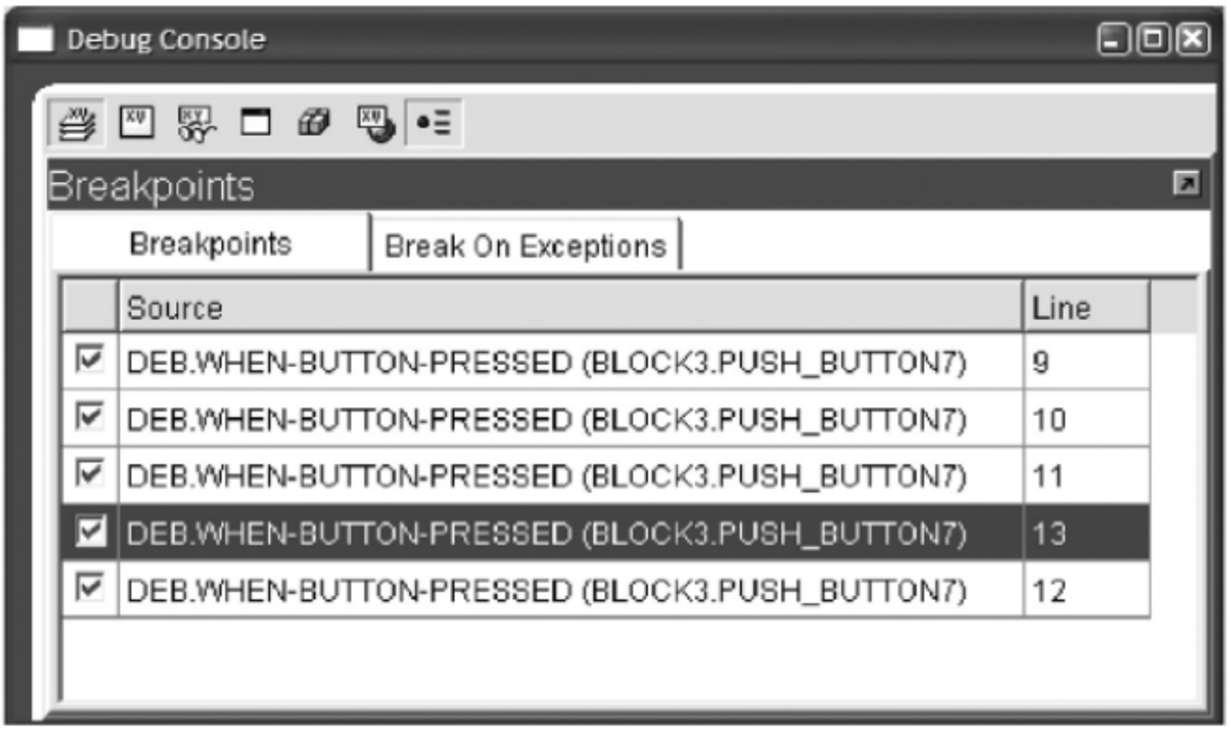

Рис. 23.18. Окно «Breakpoints»

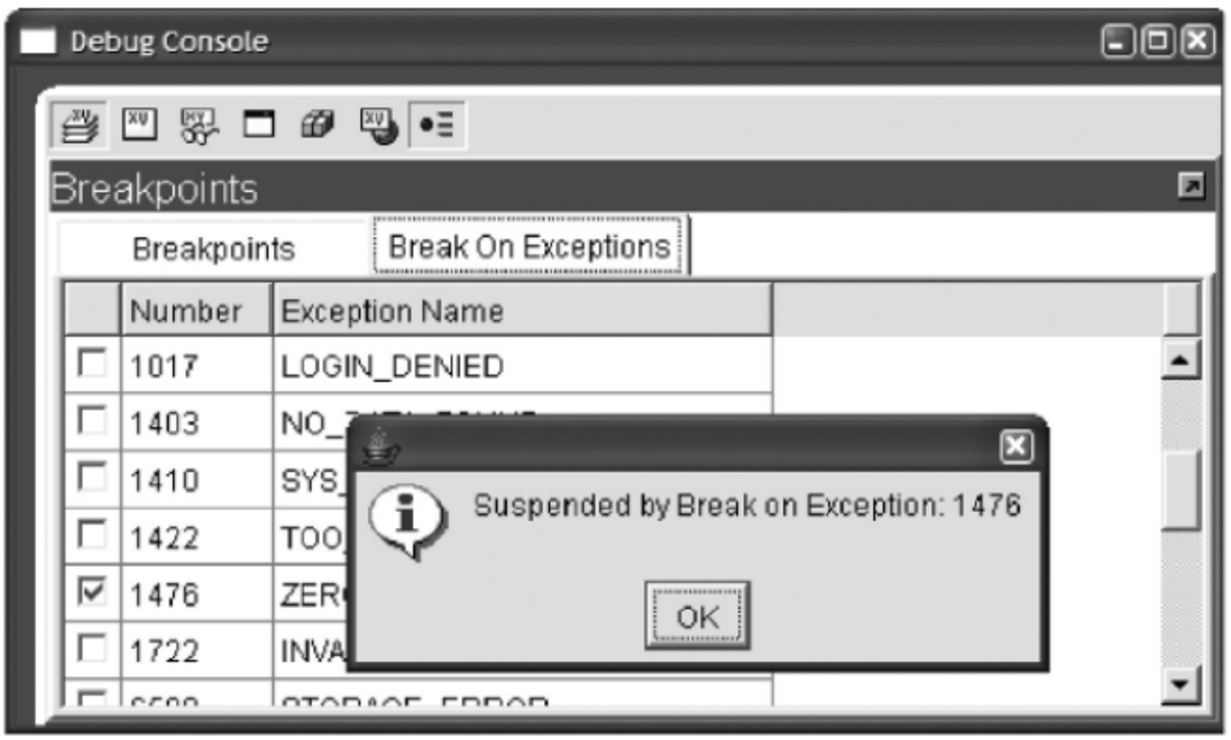

Рис. 23.19. Вкладка «Break On Exceptions».

Для того чтобы понять, как установить флажок и вывести на экран сообщение, показанное на рисунке, требуется выбрать точку останова, а затем, перейдя на вкладку «Break On Exceptions», установить флажок на исключительной ситуации с названием «ZERO_DIVIDE». Теперь, когда вы выполнили вышеописанные действия, попробуйте снова пошагово проанализировать код, и вы увидите, что после того как точка входа пре-

рывания покинет оператор, возбуждающий исключительную ситуацию, на экран выводится сообщение. Чтобы не отслеживать такого рода ошибки, используйте конструкцию EXCEPTION.

Использование пакета Debug

Вы также можете управлять отладкой программно, используя встроенный пакет Debug. Рассмотрим некоторые функции этого пакета на конкретных примерах. Для того чтобы установить точку останова программно, добавьте в ваш код строки, приведенные в листинге. Новые строки выделены жирным шрифтом.

```
WHEN-BUTTON-PRESSED
DECLARE
   value1 number;
   value2 number;
   result1  number;
   result2  number;
result4  number;
   BEGIN
   value1:=1;
   value2:=2;
result4:=speck_pack.a;
   :global.result1:=value1+value2;
   IF Debug.Getn(value2) > 0 THEN
raise Debug.Break;
   :parameter.result3:=value2-value1;
   END;
```

В этом случае, если значение переменой value2 окажется большим 0, выполнится прерывание.

Используя процедуру Attach (рис. 23.20) пакета Debug, вы можете перевести форму из нормального режима в режим отладки. После того как вы запустите форму в обычном режиме, нажмите кнопку, и на экране появится сообщение с описанием дальнейших действий.

```
WHEN-BUTTON-PRESSED
DECLARE
   value1 number;
   value2 number;
   BEGIN
   value1:=1;
```

```
    value2:=2;
DEBUG.ATTACH;
    :global.result1:=value1+value2;
    :parameter.result3:=value2-value1;
    END;
```

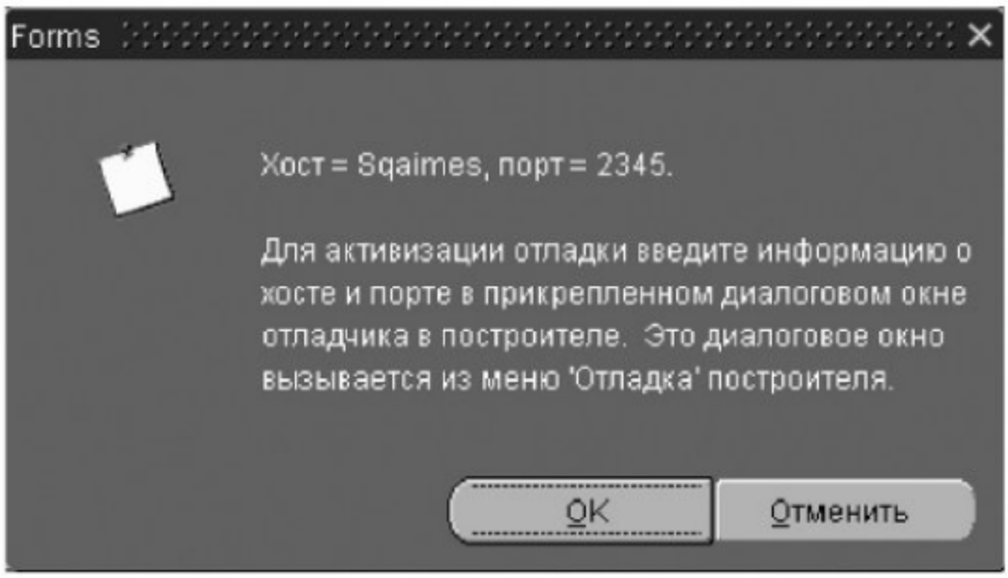

Рис. 23.20. Сообщение Debug.Attach

Для того чтобы активизировать отладку, выберите пункт главного меню Debug | Attach Debug. В появившемся окне «Attach Debug to» (рис. 23.21) введите имя хоста и номер порта, после чего подтвердите свои действия.

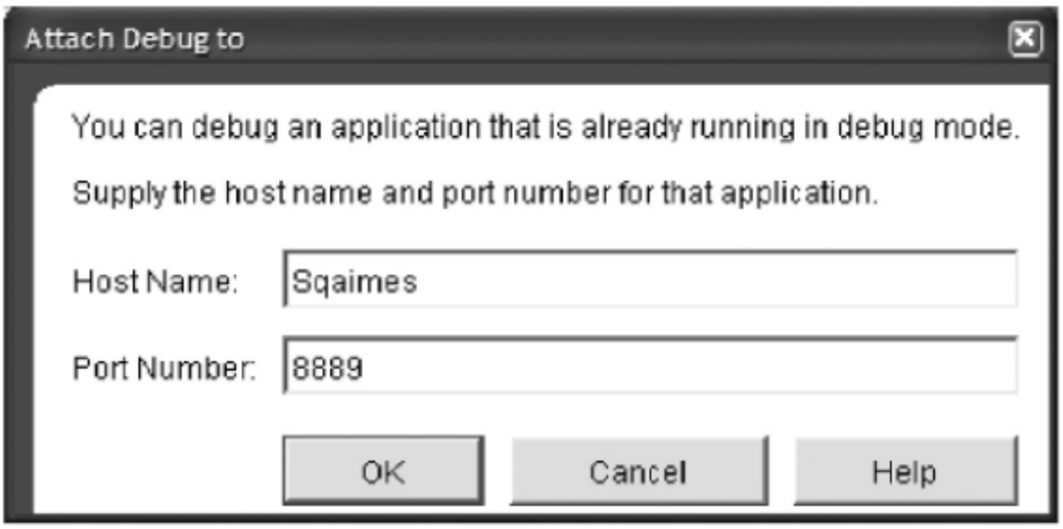

Рис. 23.21. Окно «Attach Debug to»

Вы можете передать выполнение кода отладчику, подобно тому, как это делает точка останова, используя процедуру Suspend пакета Debug (рис. 23.22).

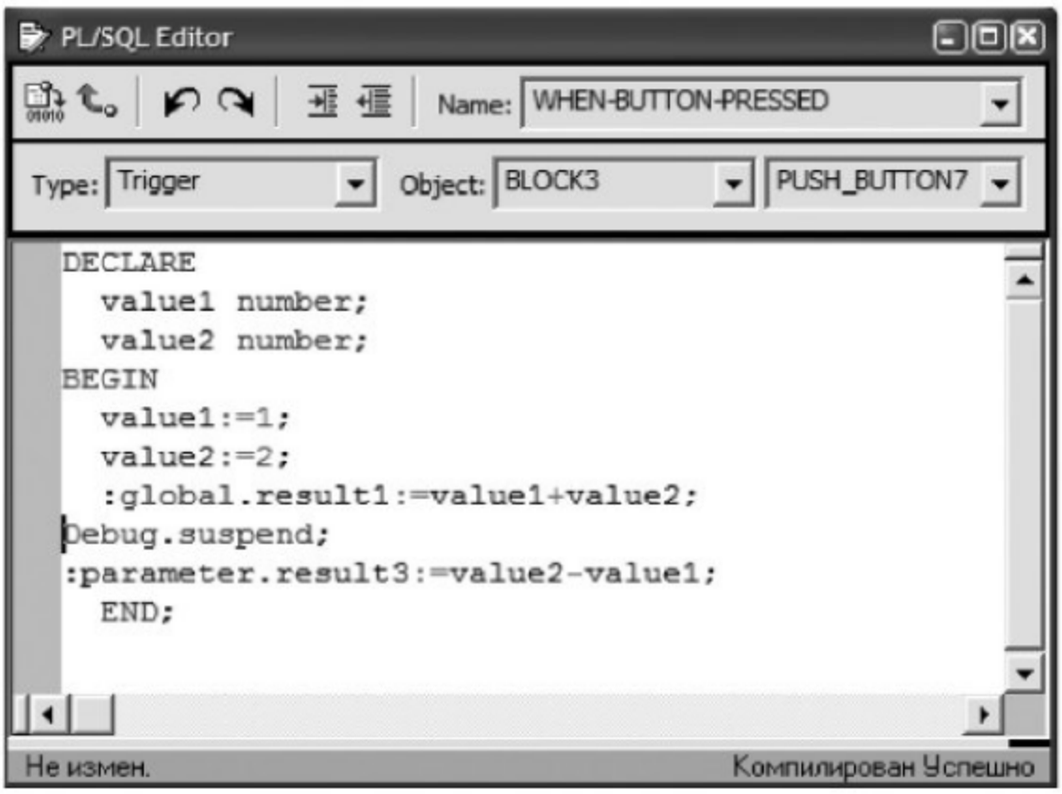

Рис. 23.22. Процедура Debug.Suspend

Управление отладкой

Когда мы нажимали кнопку «Step Into», мы переходили к выполнению следующей строки кода, то есть отлаживали код построчно. Помимо кнопки «Step Into» на панели инструментов Debug имеется еще несколько кнопок для управления ходом отладки. Вы можете воспользоваться пунктами меню Debug (рис. 23.23), действия которых идентичны кнопкам панели управления.

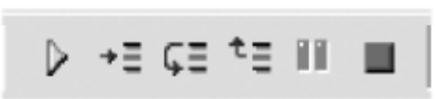

Рис.23.23. Панель Debug

- **Step Into** — шаг внутрь, выполняет следующую строку программы. Построчная отладка полезна в тех случаях, когда вы хотите индивидуально отследить значение каждой строки и попасть в вызываемые подпрограммы.
- **Step Over** — пропустить, выполняет следующую строку программы, пропуская при этом вызываемые подпрограммы, что позволяет выполнить подпрограмму, при этом не выполняя ни одной ее строки. Это существенно сокращает время по сравнению с предыдущей

командой, так как в программе с большим количеством подпрограмм иногда приходится долго ходить по строкам кода.

- **Step Out** — выйти, после выполнения подпрограммы возвращает управление отладчику, останавливаясь в месте вызова подпрограммы. Это действие поможет вам завершить пошаговое выполнение программы.
- **Go** — продолжить, выполняет следующую точку останова. После того как все точки останова будут пройдены, нормальное выполнение формы будет возобновлено.
- **Pause** — пауза, устанавливает небольшой интервал, после чего снова передает управление отладчику.
- **Stop** — остановить, выходит из формы, завершая работу отладчика.

Лекция 24. Шифрование данных в Oracle Forms

В этой лекции слушатель ознакомится с возможностями шифрования данных в Oracle, а именно с пакетом dbms_obfuscation_toolkit, для шифрования данных в своем приложении.

Ключевые слова: шифрование данных, скрытие текста, пакет dbms_obfuscation_toolkit.

Цель лекции: ознакомить пользователя с возможными методами защиты информации.

Потребность в шифровании данных существует всегда, так как информация, особенно в наши дни, – это довольно большая ценность. В современных промышленных и корпоративных системах уделяют большое внимание безопасности и конфиденциальности данных. Пакет DBMS_OBFUSCATION_TOOLKIT – это мощный и в то же время простой инструмент для шифрования данных. Этот пакет поддерживается в Oracle начиная с версии 8.1.6. Он состоит из двух основных процедур:

- DESENCRYPT – эта процедура предназначена для шифрования данных;
- DESDECRYPT – эта процедура предназначена для расшифровки данных.

Использование пакета dbms_obfuscation_toolkit

1. Выполните скрипт, как показано в листинге 24.1.

Листинг 24.1. Создание таблицы и наполнение данными

```
SQL> CREATE TABLE TCRYPT
  2     (val_num number,
  3             val_str varchar2(100))
  4  /
Таблица создана.

SQL> INSERT INTO TCRYPT VALUES (1, 'Oracle Forms')
  2  /
1 строка создана.

SQL> INSERT INTO TCRYPT VALUES (2, 'Oracle Reports')
  2  /
```

```
1 строка создана.

SQL> INSERT INTO TCRYPT VALUES (3, 'Oracle Graphics')
  2  /
1 строка создана.

SQL> INSERT INTO TCRYPT VALUES (4, 'Oracle Portal')
  2  /
1 строка создана.

SQL> INSERT INTO TCRYPT VALUES (5, 'JDeveloper')
  2  /
1 строка создана
```

2. Создайте новую форму и сохраните ее как CRYPT.
3. Создайте блок данных на таблицу TCRYPT и две кнопки. Расположите элементы на канве, как это показано на рисунке. Подпишите элементы и установите для первой кнопки метку «Зашифровать» и для второй – «Расшифровать» (рис. 24.1).

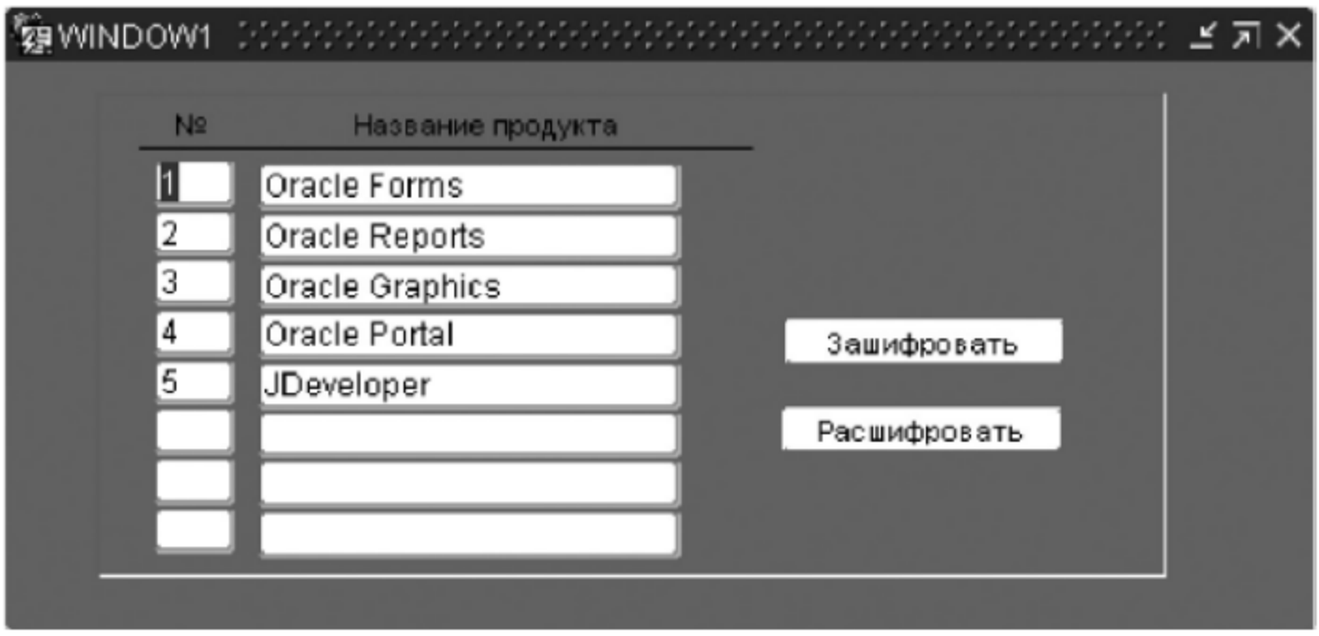

Рис. 24.1. Форма «CRYPT»

4. Находясь в Навигаторе Объектов, перейдите в узел «Программы» и создайте последовательно две функции: ENCRYPT (листинг 24.2) и DECRYPT (листинг 24.3). ENCRYPT – это функция, которая зашифровывает переданную в нее строку, а DECRYPT – это функция, которая расшифровывает указанную строку. Для шифровки и расшифровки данных необходимо знать и использовать ключ.

Листинг 24.2. Функция ENCRYPT

```
FUNCTION encrypt (
 p_text varchar2
 ,p_key varchar2
)
RETURN varchar2
IS
v_text varchar2(2000);
v_enc varchar2(2000);
BEGIN
IF p_text IS NULL THEN RETURN NULL;
END IF;
 if p_key IS NULL THEN
RETURN p_text;
END IF;
 v_text:=rpad(p_text,(trunc(length(p_text)/8)+1)*8,chr(0));
 sys.dbms_obfuscation_toolkit.desencrypt(
  input_string=>v_text,
  key_string=>rpad(p_key,8,p_key),
  encrypted_string=>v_enc);
 return v_enc;
END ENCRYPT;
```

Функция ENCRYPT принимает два параметра, первый из которых - имя шифруемой строки, а второй — это ключ шифрования.

Листинг 24.3. Функция DECRYPT

```
FUNCTION decrypt (
 p_text varchar2
 ,p_key varchar2
)
 RETURN varchar2
IS
 v_text varchar2(2000);
BEGIN
 IF p_text IS NULL
THEN RETURN NULL;
END IF;
 IF p_key IS NULL
THEN
RETURN p_text;
END IF;
 sys.dbms_obfuscation_toolkit.desdecrypt(
```

```
    input_string=>p_text,
    key_string=>rpad(p_key,8,p_key),
    decrypted_string=>v_text);
  RETURN rtrim(v_text,chr(0));
END DECRYPT;
```

5. Создайте триггер WHEN-BUTTON-PRESSED для кнопки «Зашифровать».

```
WHEN-BUTTON-PRESSED
BEGIN
:val_str:=encrypt(:val_str,'KeyCrypt');
END;
```

6. Создайте триггер WHEN-BUTTON-PRESSED для кнопки «Расшифровать».

```
WHEN-BUTTON-PRESSED
BEGIN
:val_str:=decrypt(:val_str,'KeyCrypt');
END;
```

7. Запустите форму на выполнение. Выберите данные и попробуйте их зашифровывать и расшифровывать. Вы можете шифровать данные по какому-либо условию, например, в триггерах POST-QUERY или WHEN-NEW-RECORD[ITEM]-INSTANCE. Ниже на рис. 24.2 изображен пример работы формы.

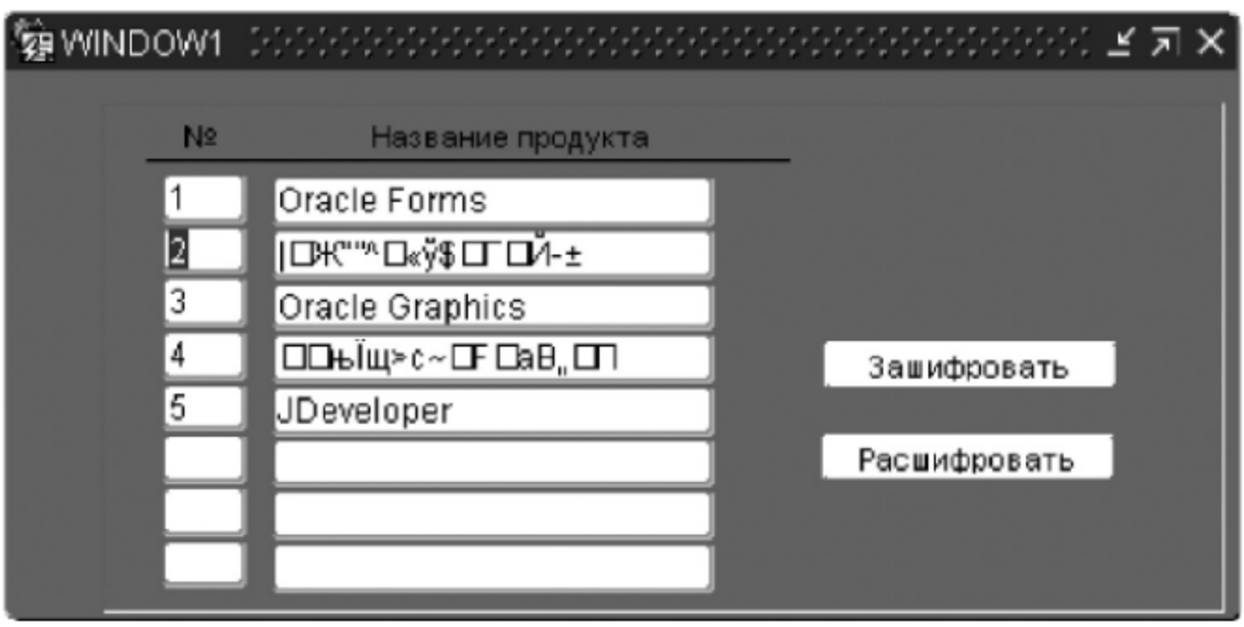

Рис. 24.2. Пример работы формы

Лекция 25. Oracle Forms и Excel

В этой лекции слушатель ознакомится с возможностями экспорта и импорта данных из приложения в Excel и обратно, используя различные интерфейсы обработки и передачи данных.

Ключевые слова: чтение данных из Microsoft Office Excel с помощью OLE2, выгрузка данных в Microsoft Office Excel с помощью OLE2, Text_IO, OLE2, OO4O, ячейка, формула, столбец.

Цель лекции: ознакомить пользователя с различными методами загрузки данных в Excel и чтения электронных таблиц.

Чтение данных из Excel

Несмотря на то что в Oracle есть свой построитель отчетов, иногда возникает потребность в нерегламентированных отчетах, то есть в тех отчетах, которые были не предусмотрены в ходе проектирования системы. Существует необходимость в загрузке данных из различных внешних источников, что обусловлено многообразием форм представления и хранения данных. Каждая среда разработки предоставляет различные интерфейсы экспорта и импорта данных; Oracle Forms, в свою очередь, поддерживает достаточно много технологий для осуществления загрузки и выгрузки данных из внешних источников.

В этой главе мы рассмотрим возможные способы экспорта и импорта данных между приложениями Excel и Oracle Forms с помощью пакета OLE2. В конце главы будет приведен пример экспорта и импорта данных с помощью технологии OLE2.

Oracle Objects for OLE (OO4O)

Oracle Objects for OLE (OO4O) – это разработанный компанией Oracle набор COM-компонентов, совместимых с OLE Automation, который позволяет работать с Oracle из клиентских приложений с использованием скриптовых языков. Этот метод лучше подходит для работы с Oracle, чем Microsoft Open Database Connectivity (ODBC), к тому же OO4O предоставляют полную поддержку PL/SQL.

Oracle Objects состоят из трех компонент:

- **OLE 2.0 Automation (InProcess) Server** – эта компонента предоставляет интерфейс OLE Automation для приложений, поддерживающих OLE automation scripting (например, Visual Basic).

- **Oracle Data Control** – управляющий элемент для Visual Basic.
- **Two C++ Class Libraries** – C-библиотеки для Microsoft Foundation Classes (MFC) и Borland (OWL).

Особенно хорошо то, что вы можете использовать OO4O из любых приложений MS Office, применяющих VB-подобный язык макросов.

Ниже приведен простой пример создания соединения с базой данных Oracle на Visual Basic:

```
Sub Form_Load ()
  Dim OraSession  As Object     'Declare variables as OLE Objects
  Dim OraDatabase As Object
  Dim OraDynaset  As Object

  Set OraSession = CreateObject(«OracleInProcServer.XOraSession»)
  Set OraDatabase = OraSession.DbOpenDatabase(«SQL*Net_Connect_
  String», «scott/tiger», 0&)
  MsgBox «Connected to « & OraDatabase.Connect & «@» &
  OraDatabase.DatabaseName

  'Create the OraDynaset Object and display the first value
  Set OraDynaset = OraDatabase.DbCreateDynaset(«select empno,
  ename from emp», 0&)
  MsgBox «Employee « & OraDynaset.Fields(«empno»).value & «, #» &
  OraDynaset.Fields(«ename»).value
End Sub
```

Технология OO40 имеет солидное преимущество над другим подходом – **VB/JET/ODBC**:

- Доступ к данным осуществляется через идентификатор строки – rowid, а не через первичный ключ или уникальный индекс в подходе VB/ODBC. Если rowid доступен, то данные доступны для корректировки.
- Возможность создавать связываемые переменных в SQL-предложениях (т. е. «SELECT * FROM EMP WHERE ENAME = :name»). PL/SQL также поддерживает связывание переменных.

Пакет TEXT_IO

Пакет TEXT_IO в Oracle Forms обеспечивает доступ к файлам операционной системы.

Пакет TEXT_IO позволяет приложению, разработанному на Oracle Forms, как читать из файлов операционной системы, так и писать в них.

Возможность использовать пакет Text_IO появилась в Oracle Developer 2000 (Forms 4.5). Рассмотрим пример выгрузки данных с помощью набора процедур и функций пакета TEXT_IO:

```
declare
out_file Text_IO.File_Type;
flnm varchar2(200);
begin
flnm := GET_FILE_NAME ('H:\', 'file_name.xls', 'XLS Files
(*.xls)|*.xls|', NULL, SAVE_FILE, TRUE);
out_file:=Text_IO.Fopen (flnm, 'w');
LOOP
Text_IO.New_Line (out_file);
Text_IO.Put (out_file, :eid);
Text_IO.Put (out_file, CHR(9));
Text_IO.Put (out_file, :NM);
IF :system.last_record = 'TRUE' THEN
EXIT;
ELSE
next_record;
END IF;
Text_IO.Fclose (out_file);
end;
```

В этом разделе мы рассмотрели и проработали возможные варианты работы с Excel-файлами. Такое количество подходов дает вам возможность выбрать для себя наиболее удобный способ импорта и экспорта данных. Мы же с вами рассмотрим пример с использованием технологии OLE2.

Object Linking and Embedding

Мы уже изучали подробно эту технологию, когда применяли OLE-контейнер в нашем приложении. Сейчас же мы рассмотрим несколько иной вариант, а именно – встроенный пакет OLE2, который поддерживается во всех последних версиях Forms. Пакет OLE2 сопровождает PL/SQL API для создания, манипулирования и доступа к атрибутам пакета OLE2.

Запуск и установка соединения с EXCEL

Для того чтобы манипулировать объектами OLE2, необходимо использовать определенный тип данных, который предназначен исклю-

чительно для работы с пакетом OLE2. В листинге 25.1 в секции DECLARE показан пример объявления переменных пакета.

Листинг 25.1 Объявление типов данных OLE2

```
DECLARE
 App ole2.obj_type;
 Book ole2.obj_type;
Cell ole2.obj_type;
Args ole2.list_type;
 …<другие переменные>…
BEGIN
Null;
END;
```

Для вызова методов пакета нужно указывать ключевое слово – *ole2.* [метод, тип]. Показанный в примере тип данных ole2.obj_type служит для объявления объектов OLE2, а ole2.list_type используется для объявления списка аргументов. Если провести аналогию, то тип данных ole2.list_type очень похож на тип данных ParamList, который также содержит список параметров.

Следующим нашим шагом в изучении пакета OLE2 будет создание объектов OLE2 (листинг 25.2).

Листинг 25.2 Создание объекта OLE2

```
DECLARE
 App ole2.obj_type;
BEGIN
App:=OLE2.Create_Obj('Excel.Application');
END;
```

Для того чтобы проверить вышеописанный код, создайте произвольную форму с одной кнопкой. Создайте для кнопки обработчик WHEN-BUTTON-PRESSED и выполните в нем приведенный код. Когда в режиме выполнения формы вы нажмете на кнопку, ничего не произойдет, но это на первый взгляд, на самом же деле если вы откроете «Диспетчер задач», то увидите процесс EXCEL.EXE. Теперь, для того чтобы увидеть электронную таблицу в рабочей области, а не в диспетчере задач, добавьте в тело триггера код, приведенный в листинге 25.3, который не только отображает приложение EXCEL, но и загружает указанный файл.

Листинг 25.3 Запуск и отображение объекта OLE2

```
DECLARE
App ole2.obj_type;
```

```
Book ole2.obj_type;
DocName varchar2(100);
WorkbooksCollection ole2.obj_type;
Sheets ole2.obj_type;
Sheet ole2.obj_type;
args ole2.list_type;
BEGIN
docname:=GET_FILE_NAME('');
App:=OLE2.Create_Obj('Excel.Application');
OLE2.Set_Property(App,'VISIBLE', 1 );
WorkbooksCollection:=OLE2.GET_OBJ_PROPERTY(App, 'Workbooks' );
args:=OLE2.CREATE_ARGLIST;
OLE2.ADD_ARG(args, docname);
Book:=OLE2.Invoke_Obj(WorkbooksCollection,'Open', args);
OLE2.DESTROY_ARGLIST(args);
END;
```

Для того чтобы вам было легко разобраться с примерами, которые будут приведены далее, ознакомьтесь с синтаксисом некоторых методов пакета (табл. 25.1).

- OLE2.Add_Arg – добавляет аргумент в список аргументов.
- OLE2.Add_Arg_Obj – добавляет объект-аргумент в список аргументов.
- OLE2.Create_arglist – создает список аргументов, который может быть передан OLE2.
- OLE2.Create_Obj – создает объект OLE2 Automation, например, Microsoft Word, Microsoft Excel.
- OLE2.Destroy_Arglist – уничтожает список аргументов.
- OLE2.Get_Char_Property – возвращает свойство объекта OLE2 Automation.
- OLE2.Get_Num_Property – возвращает идентификатор значения объекта OLE2 Automation.
- OLE2.Get_Obj_Property – возвращает значение объектного типа объекта OLE2 Automation.
- OLE2.Invoke – выполняет метод OLE2.
- OLE2.Invoke_Num – возвращает числовое (number) значение объекта OLE2 Automation, используя указанный метод.
- OLE2.Invoke_Char – возвращает символьное (character) значение объекта OLE2 Automation, используя указанный метод.
- OLE2.Invoke_Obj – возвращает значение типа объекта OLE2.
- OLE2.IsSupported – возвращает TRUE, если OLE2 поддерживается на текущей платформе.

- OLE2.Last_Exception – возвращает идентификатор последнего возбужденного исключения.
- OLE2.Release_Obj – сигнализирует объекту OLE2 Automation, что PL/SQL-клиент больше не нуждается в нем.
- OLE2.Set_Property – устанавливает значение указанного свойства объекта OLE2 Automation.

Таблица 25.1. Методы пакета OLE2

Метод	Параметр	Тип	Значение
PROCEDURE OLE2.ADD_ARG (параметры)	list	LIST_TYPE	Имя списка аргументов, созданного функцией OLE2.Create_Arglist
	value	VARCHAR2 [NUMBER]	Значение добавляемого аргумента
PROCEDURE OLE2.ADD_ARG_OBJ (параметры)	list	LIST_TYPE	Ссылка на объект списка
	value	OBJ_TYPE	Значение аргумента obj_type, передаваемое серверу OLE2
FUNCTION OLE2.CREATE _ARGLIST RETURN LIST_TYPE	Возвращает ссылку на объект список		
FUNCTION OLE2.CREATE_OBJ (параметры) RETURN OBJECT_TYPE	object	VARCHAR2	Создает указанный объект OLE2 Automation
PROCEDURE (параметры) OLE2.DESTROY_ARGLIST	list	LIST_TYPE	Имя списка аргументов, созданного функцией OLE2.Create_Arglist
FUNCTION OLE2.GET_NUM _PROPERTY (параметры) RETURN NUMBER	object	OBJ_TYPE	Объект OLE2 Automation
	property	VARCHAR2	Название свойства объекта OLE2, значение которого нужно изменить
	arglist	LIST_TYPE:=0	Имя списка аргументов, созданного функцией OLE2.Create_Arglist

FUNCTION OLE2.GET_OBJ_PROPERTY (параметры) RETURN OBJ_TYPE	object	OBJ_TYPE	Объект OLE2 Automation
	property	VARCHAR2	Название свойства объекта OLE2, значение которого нужно изменить
	arglist	LIST_TYPE:=0	Имя списка аргументов, созданного функцией OLE2.Create_Arglist
PROCEDURE OLE2.INVOKE (параметры)	object	OBJ_TYPE	Объект OLE2 Automation
	method	VARCHAR2	Метод объекта OLE2
	list	LIST_TYPE:=0	Имя списка аргументов, созданного функцией OLE2.Create_Arglist
FUNCTION OLE2.INVOKE_NUM (параметры) RETURN NUMBER	object	OBJ_TYPE	Объект OLE2 Automation
	method	VARCHAR2	Метод объекта OLE2
	arglist	LIST_TYPE:=0	Имя списка аргументов, созданного функцией OLE2.Create_Arglist
FUNCTION OLE2.INVOKE_CHAR (параметры) RETURN VARCHAR2	object	OBJ_TYPE	Объект OLE2 Automation
	method	VARCHAR2	Метод объекта OLE2
	arglist	LIST_TYPE:=0	Имя списка аргументов, созданного функцией OLE2.Create_Arglist
FUNCTION OLE2.INVOKE_OBJ (параметры) RETURN OBJ_TYPE	object	OBJ_TYPE	Объект OLE2 Automation
	method	VARCHAR2	Метод объекта OLE2
	arglist	LIST_TYPE:=0	Имя списка аргументов, созданного функцией OLE2.Create_Arglist
FUNCTION OLE2.ISSUPPORTED RETURN BOOLEAN	Возвращает TRUE, если OLE2 поддерживается на текущей платформе		
FUNCTION OLE2.LAST_EXCEPTION RETURN NUMBER	message OUT	VARCHAR2	Содержит текст ошибки OLE2

PROCEDURE OLE2.RELEASE_OBJ (параметры)	object	OBJ_TYPE	Объект OLE2 Automation
PROCEDURE OLE2.SET_PROPERTY (параметры)	object	OBJ_TYPE	Объект OLE2 Automation
	property	VARCHAR2	Название свойства объекта OLE2, значение которого нужно изменить
	value	NUMBER [VARCHAR2]	Значение свойства
	arglist	LIST_TYPE	Имя списка аргументов, созданного функцией OLE2.Create_Arglist

В листинге 25.3 для отображения приложения EXCEL на экране применяется процедура OLE2.SET_PROPERTY. В этом примере мы также использовали тип данных ole2.list_type для создания списка аргументов. Для того чтобы загрузить в EXCEL нужный нам файл, мы добавили аргумент docname, в который сохраняли имя запускаемого файла, а затем активировали его с помощью параметра «Open» процедуры Invoke_obj.

Выгрузка данных в Excel

Теперь, когда мы ознакомились со всеми методами пакета OLE2, можно перейти к следующему шагу – дополним пример возможностью передавать значение в ячейку листа электронной таблицы (листинг 25.4).

Листинг 25.4. Объявление типов данных пакета OLE2

```
PROCEDURE Into_xls (val_xls varchar2) IS
App ole2.obj_type;
Book ole2.obj_type;
DocName varchar2(100);
WorkbooksCollection ole2.obj_type;
Sheets ole2.obj_type;
Sheet ole2.obj_type;
cell ole2.obj_type;
args ole2.list_type;
begin
docname:=GET_FILE_NAME('');
App:=OLE2.Create_Obj('Excel.Application');
```

```
OLE2.Set_Property(App,'VISIBLE', 1 );
WorkbooksCollection:=OLE2.GET_OBJ_PROPERTY(App, 'Workbooks' );
args:=OLE2.CREATE_ARGLIST;
OLE2.ADD_ARG(args, docname);
Book:=OLE2.Invoke_Obj(WorkbooksCollection,'Open', args);
OLE2.DESTROY_ARGLIST(args);
Sheets:=OLE2.GET_OBJ_PROPERTY(Book,'Worksheets');
Sheet:= OLE2.GET_OBJ_PROPERTY(Book, 'ActiveSheet');
args := ole2.create_arglist;
ole2.add_arg(args,4);
ole2.add_arg(args,10);
cell := ole2.get_obj_property(sheet,'Cells',args);
ole2.destroy_arglist(args);
ole2.set_property(cell,'Value',val_xls);
ole2.release_obj(cell);
end;
```

Если вы в точности выполнили код, приведенный в листинге 25.4, то после выбора файла значение «Value from Forms» загрузится в ячейку с координатой 10:4. Завершим этот пример добавлением в тело триггера кода, который позволяет выйти из приложения EXCEL и очистить переменные (листинг 25.5).

Листинг 25.5 Объявление типов данных пакета OLE2

```
...<листинг 25.4>
args := OLE2.CREATE_ARGLIST;
 OLE2.ADD_ARG(args, 0);
 OLE2.INVOKE(book, 'Close', args);
 OLE2.DESTROY_ARGLIST(args);
OLE2.INVOKE(app,'Quit');
ole2.release_obj(sheet);
ole2.release_obj(sheets);
ole2.release_obj(book);
ole2.release_obj(WorkbooksCollection);
ole2.release_obj(app);
END;
```

На этом наше знакомство с возможностями управления приложением Excel не закончено, так как в конце главы будет показано несколько полезных примеров, которые научат вас управлять шрифтами и цветами, а пока приступим к рассмотрению примера загрузки данных из листа EXCEL в Forms.

Загрузка данных вида Мастер-Деталь

Когда мы работаем с приложением Excel, то реже всего мы сталкиваемся с обратной проблемой, а именно с загрузкой данных в форму из электронной таблицы. В листинге 25.6 приведен код функции, которая возвращает значение ячейки Excel. Параметры функции col_number и rec_number – это координаты ячейки, значение которой требуется получить.

Листинг 25.6 Функция From_xls. Загрузка данных из Excel

```
FUNCTION From_xls (col_number number, rec_number number) RETURN
varchar2 IS
App ole2.obj_type;
Book ole2.obj_type;
DocName varchar2(100):=get_file_name('');
WorkbooksCollection ole2.obj_type;
Sheets ole2.obj_type;
Sheet ole2.obj_type;
cell ole2.obj_type;
args ole2.list_type;
val_xls varchar(100);
BEGIN
/* Read Excel document */
App:=OLE2.Create_Obj('Excel.Application');
--OLE2.Set_Property(App,'VISIBLE', 1 );
WorkbooksCollection:=OLE2.GET_OBJ_PROPERTY(App, 'Workbooks' );
args:=OLE2.CREATE_ARGLIST;
OLE2.ADD_ARG(args, DocName);
Book:=OLE2.Invoke_Obj(WorkbooksCollection,'Open', args);
OLE2.DESTROY_ARGLIST(args);
Sheets:=OLE2.GET_OBJ_PROPERTY(Book,'Worksheets');
Sheet:= OLE2.GET_OBJ_PROPERTY(Book, 'ActiveSheet');
args:=OLE2.CREATE_ARGLIST;
ole2.add_arg(args, col_number);
ole2.add_arg(args, rec_number);
cell:= ole2.get_obj_property(Sheet, 'Cells', args);
ole2.destroy_arglist(args);
val_xls:=ole2.get_char_property(cell, 'Value');
args := OLE2.CREATE_ARGLIST;
 OLE2.ADD_ARG(args, 0);
 OLE2.INVOKE(book, 'Close', args);
 OLE2.DESTROY_ARGLIST(args);
OLE2.INVOKE(app,'Quit');
```

```
ole2.release_obj(sheet);
ole2.release_obj(sheets);
ole2.release_obj(book);
ole2.release_obj(WorkbooksCollection);
ole2.release_obj(app);
RETURN val_xls;
END;
```

Основным отличием функции From_xls от предыдущего примера является строка ole2.get_char_property(cell, 'Value'), которая получает значение указанной ячейки.

Создайте кнопку и в триггере WHEN-BUTTON-PRESSED вызовите функцию From_xls, как показано в листинге 25.7.

Листинг 25.7 WHEN-BUTTON-PRESSED

```
Message('Value from Excel: '||from_xls(1,1));
```

Загрузка документа в форму

Мы с вами рассмотрели простые примеры, в которых передавали значение из Excel в Forms и наоборот, но иногда требуется выгружать целые таблицы из Forms в Excel или, наоборот, загружать сложные документы Excel в форму. Рассмотрим пример, когда необходимо разработать систему приема заказов, причем все заказы приходят в виде Excel-заявок. Для рассмотрения примера мы будем использовать форму Master_Detail, созданную в главе «Блоки». На рис. 25.1 показан файл Excel – Inquiry.xls, «который мы будем называть «Заказ» и который мы будем загружать в нашу БД».

Как видите, файл «Заказ» имеет сложную структуру и поэтому состоит из трех частей:

1. Описание письма и заказчика.
2. Товар.
3. Характеристика товара.

Перед нами стоит задача загрузки документа в форму по одному нажатию кнопки, то есть мы должны проследить, чтобы данные были не только загружены, но и сохранены в том порядке и соотношении, в котором они находятся. Приведенный пример не является оптимальным, поэтому в ваших силах написать более производительное и функциональное решение.

Перед тем как вставлять полученные данные в рабочие таблицы, выгрузим данные документа во временные таблицы Excel1 и Excel2.

В таблицу Excel1 мы будем выгружать информацию о товаре, а в таблицу Excel2 – информацию о свойствах и количестве товара. В листинге 25.8 приведен пример загрузки документа во временные таблицы. Для упрощения примера процедура разбита на две части, где первая часть – это выгрузка информации о заказчике, а вторая часть – выгрузка информации о товаре и его характеристиках.

Украина			**ФИРМА СИДОРОВА**
Ukraine			**SIDOROV FIRM**
г . Мариуполь ул. Громовой 62.20			Gromovoi 62, Mariupol
Запрос-анкета / Inquiry-questionnaire №			

Письмо заказчика №	102/5025	Дата	06.06.06
Letter of the customer №		Date	
Заказчик / Customer	Фирма «IVANOV EUROPA NV»		
Срок поставки / Term of delivery	Четвертый квартал 2007		
Условия поставки / Delivery specification	Мариупольский торговый порт		
Толеранс по количеству, % / Admission by quantity, %	+ 5 - 5		
Марка стали / Grade of steel	AH36		
Приемка / Acceptance	Bureau Veritas		
Дополнительные требования / Additional requirements	УЗК-1		
Дополнительная маркировка / Additional marks	TRADE MARK, MADE IN UKRAINE		
ЛОТ/LOT	1		

№ п/п Item №	Толщина, мм Thickness, mm	Ширина, мм Width, mm	Длина, мм Length, mm	Количество, т Weight, t	Стандарт, класс Standard, class
1	1	2500	8000	100	EN 10029-91(A)
2	25	2500	10000	120	EN 10029-91(A)
3	10	2500	8000	125	EN 10029-91(A)
4	10	2000	12000	100	EN 10029-91(A)
5	25	2500	6000	100011	EN 10029-91(A)
6	14,5	2250	8200	80031,8	EN 10029-91(A)
7	13	2200	8000	100012	EN 10029-91(A)

Рис. 25.1. Макет файла Excel

Информацию о заказчике мы будем выгружать непосредственно в форму, а именно в элементы блока данных «PZAK». В листинге 8 приведен пример процедуры Excel_to_Forms, которая заполняет блок «PZAK» информацией из файла Excel.

Листинг 25.8 Процедура Excel_to_Forms

```
PROCEDURE EXCEL_to_form (DocName varchar2) IS
App ole2.obj_type;
Book ole2.obj_type;
DocName varchar2(100):=get_file_name('');
WorkbooksCollection ole2.obj_type;
Sheets ole2.obj_type;
Sheet ole2.obj_type;
cell ole2.obj_type;
args ole2.list_type;
BEGIN
App:=OLE2.Create_Obj('Excel.Application');
OLE2.Set_Property(App,'VISIBLE', 1 );
WorkbooksCollection:=OLE2.GET_OBJ_PROPERTY(App, 'Workbooks' );
args:=OLE2.CREATE_ARGLIST;
OLE2.ADD_ARG(args,docname);
Book:=OLE2.Invoke_Obj(WorkbooksCollection,'Open', args);
OLE2.DESTROY_ARGLIST(args);

Sheets:=OLE2.GET_OBJ_PROPERTY(Book,'Worksheets');
Sheet:= OLE2.GET_OBJ_PROPERTY( Book, 'ActiveSheet');
args:=OLE2.CREATE_ARGLIST;
ole2.add_arg(args, 6);
ole2.add_arg(args, 5);
cell:= ole2.get_obj_property(Sheet, 'Cells', args);
ole2.destroy_arglist(args);
:nlet :=ole2.get_char_property(cell, 'Value'); -- получение номера
письма из заявки
args:=OLE2.CREATE_ARGLIST;
:sdat:=sysdate; --запись текущей даты
ole2.add_arg(args, 8);
ole2.add_arg(args, 5);
cell:= ole2.get_obj_property(Sheet, 'Cells', args);
ole2.destroy_arglist(args);
:nfirm:=ole2.get_char_property(cell, 'Value'); -- наименование
заказчика
args:=OLE2.CREATE_ARGLIST;
ole2.add_arg(args, 10);
ole2.add_arg(args, 5);
cell:= ole2.get_obj_property(Sheet, 'Cells', args);
ole2.destroy_arglist(args);
:srpost :=ole2.get_char_property(cell, 'Value'); --срок поставки
```

```
args:=OLE2.CREATE_ARGLIST;
ole2.add_arg(args, 12);
ole2.add_arg(args, 5);
cell:= ole2.get_obj_property(Sheet, 'Cells', args);
ole2.destroy_arglist(args);
:yslp :=ole2.get_char_property(cell, 'Value');
args:=OLE2.CREATE_ARGLIST;
ole2.add_arg(args, 6);
ole2.add_arg(args, 15);
cell:=ole2.get_obj_property(Sheet, 'Cells', args);
ole2.destroy_arglist(args);
:pdat:=ole2.get_char_property(cell, 'Value');
args:=OLE2.CREATE_ARGLIST;
ole2.add_arg(args, 14);
ole2.add_arg(args, 5);
cell:=ole2.get_obj_property(Sheet, 'Cells', args);
ole2.destroy_arglist(args);
:tpus:=ole2.get_num_property(cell, 'Value');
args:=OLE2.CREATE_ARGLIST;
ole2.add_arg(args, 14);
ole2.add_arg(args, 6);
cell:=ole2.get_obj_property(Sheet, 'Cells', args);
ole2.destroy_arglist(args);
tmin:=ole2.get_num_property(cell, 'Value');
:tmin:=tlm;
args := OLE2.CREATE_ARGLIST;
 OLE2.ADD_ARG(args, 0);
 OLE2.INVOKE(book, 'Close', args);
 OLE2.DESTROY_ARGLIST(args);
OLE2.INVOKE(app,'Quit');
ole2.release_obj(sheet);
ole2.release_obj(sheets);
ole2.release_obj(book);
ole2.release_obj(WorkbooksCollection);
ole2.release_obj(app);
END;
```

Как видите, процедура получается довольно-таки объемной, хотя мы загрузили только первую часть документа. Перейдем ко второй части – загрузка информации о товаре. Для получения таблицы Excel2 и ее заполнения информацией из заявки, модифицируйте процедуру Excel_To_Forms, добавив в нее код, приведенный в листинге 9.

Листинг 9 Процедура Excel_to_Forms

```
PROCEDURE Excel_to_Forms (DocName varchar2) IS
App ole2.obj_type;
Book ole2.obj_type;
DocName varchar2(100):=get_file_name('');
WorkbooksCollection ole2.obj_type;
Sheets ole2.obj_type;
Sheet ole2.obj_type;
cell ole2.obj_type;
args ole2.list_type;
BEGIN
App:=OLE2.Create_Obj('Excel.Application');
OLE2.Set_Property(App,'VISIBLE', 1 );
WorkbooksCollection:=OLE2.GET_OBJ_PROPERTY(App, 'Workbooks' );
args:=OLE2.CREATE_ARGLIST;
OLE2.ADD_ARG(args,docname);
Book:=OLE2.Invoke_Obj(WorkbooksCollection,'Open', args);
OLE2.DESTROY_ARGLIST(args);

Sheets:=OLE2.GET_OBJ_PROPERTY(Book,'Worksheets');
Sheet:= OLE2.GET_OBJ_PROPERTY( Book, 'ActiveSheet');
args:=OLE2.CREATE_ARGLIST;
ole2.add_arg(args, 6);
ole2.add_arg(args, 5);
cell:= ole2.get_obj_property(Sheet, 'Cells', args);
ole2.destroy_arglist(args);
:nlet :=ole2.get_char_property(cell, 'Value'); -- получение номера
письма из заявки
args:=OLE2.CREATE_ARGLIST;
:sdat:=sysdate; --запись текущей даты
ole2.add_arg(args, 8);
ole2.add_arg(args, 5);
cell:= ole2.get_obj_property(Sheet, 'Cells', args);
ole2.destroy_arglist(args);
:nfirm:=ole2.get_char_property(cell, 'Value'); -- наименование
заказчика
args:=OLE2.CREATE_ARGLIST;
ole2.add_arg(args, 10);
ole2.add_arg(args, 5);
cell:= ole2.get_obj_property(Sheet, 'Cells', args);
ole2.destroy_arglist(args);
:srpost :=ole2.get_char_property(cell, 'Value'); --срок поставки
```

```
args:=OLE2.CREATE_ARGLIST;
ole2.add_arg(args, 12);
ole2.add_arg(args, 5);
cell:= ole2.get_obj_property(Sheet, 'Cells', args);
ole2.destroy_arglist(args);
:yslp :=ole2.get_char_property(cell, 'Value');
args:=OLE2.CREATE_ARGLIST;
ole2.add_arg(args, 6);
ole2.add_arg(args, 15);
cell:=ole2.get_obj_property(Sheet, 'Cells', args);
ole2.destroy_arglist(args);
:pdat:=ole2.get_char_property(cell, 'Value');
args:=OLE2.CREATE_ARGLIST;
ole2.add_arg(args, 14);
ole2.add_arg(args, 5);
cell:=ole2.get_obj_property(Sheet, 'Cells', args);
ole2.destroy_arglist(args);
:tpus:=ole2.get_num_property(cell, 'Value');
args:=OLE2.CREATE_ARGLIST;
ole2.add_arg(args, 14);
ole2.add_arg(args, 6);
cell:=ole2.get_obj_property(Sheet, 'Cells', args);
ole2.destroy_arglist(args);
tmin:=ole2.get_num_property(cell, 'Value');
:tmin:=tlm;

------------Получение данных о товаре
for i in 1..2000 loop
args:=OLE2.CREATE_ARGLIST;
               ole2.add_arg(args, i);
               ole2.add_arg(args, 1);
cell:= ole2.get_obj_property(Sheet, 'Cells', args);
               ole2.destroy_arglist(args);
if ole2.get_char_property(cell, 'Value')='Grade of steel' then

            args:=OLE2.CREATE_ARGLIST;
ole2.add_arg(args, i-1);
ole2.add_arg(args, 5);
cell:= ole2.get_obj_property(Sheet, 'Cells', args);
ole2.destroy_arglist(args);
sm:=ole2.get_char_property(cell, 'Value');

args:=OLE2.CREATE_ARGLIST;
```

```
ole2.add_arg(args, (i-1)+2);
ole2.add_arg(args, 5);
cell:= ole2.get_obj_property(Sheet, 'Cells', args);
ole2.destroy_arglist(args);
priemka :=ole2.get_char_property(cell, 'Value');

args:=OLE2.CREATE_ARGLIST;
ole2.add_arg(args, (i-1)+4);
ole2.add_arg(args, 5);
cell:= ole2.get_obj_property(Sheet, 'Cells', args);
ole2.destroy_arglist(args);
doptr:=ole2.get_char_property(cell, 'Value');

args:=OLE2.CREATE_ARGLIST;
ole2.add_arg(args, (i-1)+6);
ole2.add_arg(args, 5);
cell:= ole2.get_obj_property(Sheet, 'Cells', args);
ole2.destroy_arglist(args);
dopmr:=ole2.get_char_property(cell, 'Value');

args:=OLE2.CREATE_ARGLIST;
ole2.add_arg(args, (i-1)+8);
ole2.add_arg(args, 5);
cell:= ole2.get_obj_property(Sheet, 'Cells', args);
ole2.destroy_arglist(args);
lot:=ole2.get_char_property(cell, 'Value');
---------------------------------
insert into excel2 (smarka, priem, doptr, dopls, lot) values (sm,
priemka, substr(doptr,0,300), substr(dopmr,0,300), lot);
 forms_ddl('commit');
end if;
end loop;
args := OLE2.CREATE_ARGLIST;
 OLE2.ADD_ARG(args, 0);
 OLE2.INVOKE(book, 'Close', args);
 OLE2.DESTROY_ARGLIST(args);
OLE2.INVOKE(app,'Quit');
ole2.release_obj(sheet);
ole2.release_obj(sheets);
ole2.release_obj(book);
ole2.release_obj(WorkbooksCollection);
ole2.release_obj(app);
END;
```

Перейдем к третьей части – получению информации о характеристических данных товара. Для выгрузки данных напишем новую процедуру – Excel_To_Form2 (листинг 10), потому что, если добавить новый код в предыдущую процедуру, она будет довольно тяжелой для восприятия.

Листинг 10 Процедура Excel_to_Forms2

```
PROCEDURE Excel_to_Forms2 (docname varchar2) IS
App ole2.obj_type;
Book ole2.obj_type;
cell ole2.obj_type;
MyChar varchar2(30);
GOST varchar2(2000);
mk varchar2(2000);
NP NUMBER;
DLN NUMBER;
SHR NUMBER;
TLS NUMBER;
KOL NUMBER;
WorkbooksCollection ole2.obj_type;
Sheets ole2.obj_type;
Sheet ole2.obj_type;
args ole2.list_type;
BEGIN
/* Read Excel document */
App:=OLE2.Create_Obj('Excel.Application');
WorkbooksCollection:=OLE2.GET_OBJ_PROPERTY(App, 'Workbooks' );
args:=OLE2.CREATE_ARGLIST;
OLE2.ADD_ARG(args, docname);
Book:=OLE2.Invoke_Obj(WorkbooksCollection,'Open', args);
OLE2.DESTROY_ARGLIST(args);

Sheets:=OLE2.GET_OBJ_PROPERTY(Book,'Worksheets');
Sheet:= OLE2.GET_OBJ_PROPERTY(Book, 'ActiveSheet');

for i in 1..2000 loop
args:=OLE2.CREATE_ARGLIST;
                ole2.add_arg(args, i);
                ole2.add_arg(args, 1);
cell:= ole2.get_obj_property(Sheet, 'Cells', args);
                ole2.destroy_arglist(args);
if ole2.get_NUM_property(cell, 'Value')!=0 then
args:=OLE2.CREATE_ARGLIST;
```

```
                ole2.add_arg(args, i-3);
                ole2.add_arg(args, 1);
cell:= ole2.get_obj_property(Sheet, 'Cells', args);
ole2.destroy_arglist(args);
if ole2.get_char_property(cell, 'Value')='ЛОТ/LOT' then

   args:=OLE2.CREATE_ARGLIST;
ole2.add_arg(args, i-3);
ole2.add_arg(args, 5);
cell:= ole2.get_obj_property(Sheet, 'Cells', args);
ole2.destroy_arglist(args);
mk :=ole2.get_char_property(cell, 'Value');
                                    end if;

args:=OLE2.CREATE_ARGLIST;
ole2.add_arg(args, i);
ole2.add_arg(args, 1);
cell:= ole2.get_obj_property(Sheet, 'Cells', args);
ole2.destroy_arglist(args);
NP :=ole2.get_NUM_property(cell, 'Value');
--:smarka:=sm;

args:=OLE2.CREATE_ARGLIST;
ole2.add_arg(args, i);
ole2.add_arg(args, 2);
cell:= ole2.get_obj_property(Sheet, 'Cells', args);
ole2.destroy_arglist(args);
TLS :=ole2.get_NUM_property(cell, 'Value');

args:=OLE2.CREATE_ARGLIST;
ole2.add_arg(args, i);
ole2.add_arg(args, 3);
cell:= ole2.get_obj_property(Sheet, 'Cells', args);
ole2.destroy_arglist(args);
SHR:=ole2.get_NUM_property(cell, 'Value');

args:=OLE2.CREATE_ARGLIST;
ole2.add_arg(args, i);
ole2.add_arg(args, 4);
cell:= ole2.get_obj_property(Sheet, 'Cells', args);
ole2.destroy_arglist(args);
DLN:=ole2.get_NUM_property(cell, 'Value');
```

```
args:=OLE2.CREATE_ARGLIST;
ole2.add_arg(args, i);
ole2.add_arg(args, 5);
cell:= ole2.get_obj_property(Sheet, 'Cells', args);
ole2.destroy_arglist(args);
KOL:=ole2.get_NUM_property(cell, 'Value');

args:=OLE2.CREATE_ARGLIST;
ole2.add_arg(args, i);
ole2.add_arg(args, 6);
cell:= ole2.get_obj_property(Sheet, 'Cells', args);
ole2.destroy_arglist(args);
GOST:=ole2.get_char_property(cell, 'Value');
INSERT INTO excel3 (mk,NP,DLN,SHR,TLS,COL,GOST)
VALUES(mk,NP, DLN,SHR,TLS,KOL,GOST);
 FORMS_DDL('commit');
end if;
end loop;
FORMS_DDL('commit');
OLE2.RELEASE_OBJ(cell);
OLE2.RELEASE_OBJ(Sheet);
OLE2.RELEASE_OBJ(Sheets);
OLE2.RELEASE_OBJ(Book);
OLE2.INVOKE (App,'Quit');
OLE2.RELEASE_OBJ(App);
END;
```

Информация получена, и документ в раздробленном состоянии находится в базе. Наша дальнейшая задача – это связывание данных, находящихся в таблицах Excel1 и Excel2, между собой и с данными в форме. Для этого модифицируйте процедуру Excel_To_Forms, как показано в листинге 11 где приведен конечный вид процедуры для загрузки документа Excel.

Листинг 11 Excel_to_Forms

```
PROCEDURE EXCEL_to_form (DocName varchar2) IS
App ole2.obj_type;
Book ole2.obj_type;
DocName varchar2(100):=get_file_name('');
WorkbooksCollection ole2.obj_type;
Sheets ole2.obj_type;
```

```
Sheet ole2.obj_type;
cell ole2.obj_type;
args ole2.list_type;
CURSOR EX1 IS SELECT * FROM asunz.EXCEL2;
EX2 EX1%ROWTYPE;
CURSOR c1 IS SELECT * FROM asunz.EXCEL3 order by np asc;
c2 c1%ROWTYPE;
BEGIN
App:=OLE2.Create_Obj('Excel.Application');
OLE2.Set_Property(App,'VISIBLE', 1 );
WorkbooksCollection:=OLE2.GET_OBJ_PROPERTY(App, 'Workbooks' );
args:=OLE2.CREATE_ARGLIST;
OLE2.ADD_ARG(args,docname);
Book:=OLE2.Invoke_Obj(WorkbooksCollection,'Open', args);
OLE2.DESTROY_ARGLIST(args);

Sheets:=OLE2.GET_OBJ_PROPERTY(Book,'Worksheets');
Sheet:= OLE2.GET_OBJ_PROPERTY( Book, 'ActiveSheet');
args:=OLE2.CREATE_ARGLIST;
ole2.add_arg(args, 6);
ole2.add_arg(args, 5);
cell:= ole2.get_obj_property(Sheet, 'Cells', args);
ole2.destroy_arglist(args);
:nlet :=ole2.get_char_property(cell, 'Value'); -- получение номера
письма из заявки
args:=OLE2.CREATE_ARGLIST;
:sdat:=sysdate; --запись текущей даты
ole2.add_arg(args, 8);
ole2.add_arg(args, 5);
cell:= ole2.get_obj_property(Sheet, 'Cells', args);
ole2.destroy_arglist(args);
:nfirm:=ole2.get_char_property(cell, 'Value'); -- наименование
заказчика
args:=OLE2.CREATE_ARGLIST;
ole2.add_arg(args, 10);
ole2.add_arg(args, 5);
cell:= ole2.get_obj_property(Sheet, 'Cells', args);
ole2.destroy_arglist(args);
:srpost :=ole2.get_char_property(cell, 'Value'); --срок поставки
args:=OLE2.CREATE_ARGLIST;
ole2.add_arg(args, 12);
ole2.add_arg(args, 5);
```

```
cell:= ole2.get_obj_property(Sheet, 'Cells', args);
ole2.destroy_arglist(args);
:yslp :=ole2.get_char_property(cell, 'Value');
args:=OLE2.CREATE_ARGLIST;
ole2.add_arg(args, 6);
ole2.add_arg(args, 15);
cell:=ole2.get_obj_property(Sheet, 'Cells', args);
ole2.destroy_arglist(args);
:pdat:=ole2.get_char_property(cell, 'Value');
args:=OLE2.CREATE_ARGLIST;
ole2.add_arg(args, 14);
ole2.add_arg(args, 5);
cell:=ole2.get_obj_property(Sheet, 'Cells', args);
ole2.destroy_arglist(args);
:tpus:=ole2.get_num_property(cell, 'Value');
args:=OLE2.CREATE_ARGLIST;
ole2.add_arg(args, 14);
ole2.add_arg(args, 6);
cell:=ole2.get_obj_property(Sheet, 'Cells', args);
ole2.destroy_arglist(args);
tmin:=ole2.get_num_property(cell, 'Value');
:tmin:=tlm;

------------Получение данных о товаре
for i in 1..2000 loop
args:=OLE2.CREATE_ARGLIST;
               ole2.add_arg(args, i);
               ole2.add_arg(args, 1);
cell:= ole2.get_obj_property(Sheet, 'Cells', args);
               ole2.destroy_arglist(args);
if ole2.get_char_property(cell, 'Value')='Grade of steel' then

            args:=OLE2.CREATE_ARGLIST;
ole2.add_arg(args, i-1);
ole2.add_arg(args, 5);
cell:= ole2.get_obj_property(Sheet, 'Cells', args);
ole2.destroy_arglist(args);
sm:=ole2.get_char_property(cell, 'Value');

args:=OLE2.CREATE_ARGLIST;
ole2.add_arg(args, (i-1)+2);
ole2.add_arg(args, 5);
```

```
cell:= ole2.get_obj_property(Sheet, 'Cells', args);
ole2.destroy_arglist(args);
priemka :=ole2.get_char_property(cell, 'Value');

args:=OLE2.CREATE_ARGLIST;
ole2.add_arg(args, (i-1)+4);
ole2.add_arg(args, 5);
cell:= ole2.get_obj_property(Sheet, 'Cells', args);
ole2.destroy_arglist(args);
doptr:=ole2.get_char_property(cell, 'Value');

args:=OLE2.CREATE_ARGLIST;
ole2.add_arg(args, (i-1)+6);
ole2.add_arg(args, 5);
cell:= ole2.get_obj_property(Sheet, 'Cells', args);
ole2.destroy_arglist(args);
dopmr:=ole2.get_char_property(cell, 'Value');

args:=OLE2.CREATE_ARGLIST;
ole2.add_arg(args, (i-1)+8);
ole2.add_arg(args, 5);
cell:= ole2.get_obj_property(Sheet, 'Cells', args);
ole2.destroy_arglist(args);
lot:=ole2.get_char_property(cell, 'Value');
---------------------------------
insert into excel2 (smarka, priem, doptr, dopls, lot) values (sm,
priemka, substr(doptr,0,300), substr(dopmr,0,300), lot);
 forms_ddl('commit');
end if;
end loop;
args := OLE2.CREATE_ARGLIST;
 OLE2.ADD_ARG(args, 0);
 OLE2.INVOKE(book, 'Close', args);
 OLE2.DESTROY_ARGLIST(args);
OLE2.INVOKE(app,'Quit');
ole2.release_obj(sheet);
ole2.release_obj(sheets);
ole2.release_obj(book);
ole2.release_obj(WorkbooksCollection);
ole2.release_obj(app);

Excel_To_Form (docname);
```

```
OPEN EX1;
select count(*)  into j from asunz.excel2;
select count(*)  into k from asunz.excel3;
      for i in 1..j
      loop
            fetch ex1 into ex2;
      :ppzak.smarka:=ex2.smarka;
            :ppzak.priem:=ex2.priem;
            :ppzak.doptr:=ex2.doptr;
            :ppzak.dopls:=ex2.dopls;
            :ppzak.lot:=ex2.lot;
commit;
open c1;
for v in 1..k
      loop
            fetch c1 into c2;
            if ex2.lot=c2.mk or (ex2.lot is null and c2.mk is
null ) then
:dln=c2.dln;
:shr:=c2.shr;
:tls:=c2.tls;
:ton:=c2.col;
:PZAK_ATR.gost:=c2.gost;
:num:=c2.np;
go_block('pzak_atr');
            next_record;
end if;
end loop;
close c1;
commit;
            go_block('ppzak');
            next_record;
      end loop;
CLOSE EX1;
delete from excel2;
delete from excel3;
forms_ddl('commit');
END;
```

В этом листинге мы объявили два курсора, чтобы считать данные с таблиц Excel1 и Excel2, причем общим критерием для связывания таблиц служит номер лота.

Управление параметрами Excel

Теперь, когда мы научились обмениваться данными между приложениями Excel и Forms, можно перейти к не менее важной части – управлению функциональными возможностями электронной таблицы. Ниже приведены небольшие примеры, которые позволят вам научиться управлять форматированием ячеек и основными командами меню «Файл» Excel.

- В листинге 12 показан пример добавления листа (Sheet) Excel.

Листинг 12 Добавление листа

```
Declare
   app OLE2.OBJ_TYPE;
   workbooks OLE2.OBJ_TYPE;
   workbook OLE2.OBJ_TYPE;
   worksheet OLE2.OBJ_TYPE;
   Sheets OLE2.OBJ_TYPE;
   args OLE2.LIST_TYPE;
begin
   Excel := OLE2.CREATE_OBJ('Excel.Application');
   OlE2.SET_PROPERTY(app,'Visible', TRUE);
   workbooks := OLE2.GET_OBJ_PROPERTY(Excel, 'Workbooks');
   workbook := OLE2.INVOKE_OBJ(workbooks,'Add');

   Sheets := OLE2.GET_OBJ_PROPERTY(workbook, 'Sheets');
         OLE2.INVOKE(Sheets, 'Add');
...
exception
   when others then
            Message('Ошибка OLE');
end;
```

- Опираясь на пример, приведенный в листинге 13, вы можете управлять свойствами столбца, такими как высота, ширина, цвет шрифта и другие.

Листинг 13. Форматирование ячейки и столбца

```
Declare
  app    OLE2.OBJ_TYPE;
   workbook         OLE2.OBJ_TYPE;
   workbooks        OLE2.OBJ_TYPE;
   worksheet        OLE2.OBJ_TYPE;
```

```
    worksheets      OLE2.OBJ_TYPE;
    col             OLE2.OBJ_TYPE;
    font            OLE2.OBJ_TYPE;
    args            OLE2.List_Type;
BEGIN
    application := OLE2.CREATE_OBJ('Excel.Application');
    OLE2.SET_PROPERTY(app, 'Visible', 'True');
    workbooks := OLE2.GET_OBJ_PROPERTY(app, 'Workbooks');
    workbook := OLE2.INVOKE_OBJ(workbooks,'Add');

    worksheets := OLE2.GET_OBJ_PROPERTY(workbook, 'Worksheets');
    worksheet := OLE2.INVOKE_OBJ(worksheets,'Add');

-- Устанавливаем ширину столбцов в диапазоне G:J
    args := OLE2.CREATE_ARGLIST;
    OLE2.ADD_ARG(args, 'G:J');  -- Устанавливаем диапазон столбцов
    col:= OLE2.GET_OBJ_PROPERTY(worksheet, 'Columns', args);
    OLE2.DESTROY_ARGLIST(args);
    font := OLE2.GET_OBJ_PROPERTY(col, 'Font', args);
-- Устанавливаем ширину ячеек в диапазоне G:J
    OLE2.SET_PROPERTY(col, 'ColumnWidth', 100); -- Установка ширины
столбца
    OLE2.RELEASE_OBJ(col);
-- Установка значений ячеек
    OLE2.SET_PROPERTY(font, 'Name', 'Arial'); -- Устанавливаем
наименование шрифта
    OLE2.SET_PROPERTY(font, 'Size', 20); -- Устанавливаем размер
шрифта равным 20
    OLE2.SET_PROPERTY(font, 'Strikethrough', 'True');
    OLE2.SET_PROPERTY(font, 'Bold', 'True'); -- Устанавливаем
жирный шрифт
    OLE2.SET_PROPERTY(font, 'ColorIndex', 3);-- Устанавливаем
красный цвет шрифта
    OLE2.RELEASE_OBJ(font);
OLE2.RELEASE_OBJ(worksheet);
    OLE2.RELEASE_OBJ(worksheets);
    OLE2.RELEASE_OBJ(workbook);
    OLE2.RELEASE_OBJ(workbooks);
    OLE2.RELEASE_OBJ(app);
EXCEPTION
     WHEN others THEN
         OLE2.Release_Obj( app );
```

```
        message('Ошибка OLE');
END;
```

- **Пример, приведенный в листинге 14, показывает, как можно управлять свойствами ячейками, такими как высота и ширина.**

Листинг 14. Размер ячейки

```
DECLARE
  application     OLE2.OBJ_TYPE;
   workbook        OLE2.OBJ_TYPE;
   workbooks       OLE2.OBJ_TYPE;
   worksheet       OLE2.OBJ_TYPE;
   worksheets      OLE2.OBJ_TYPE;
   row             OLE2.OBJ_TYPE;
   args            OLE2.List_Type;
BEGIN
   application := OLE2.CREATE_OBJ('Excel.Application');
   OLE2.SET_PROPERTY(application, 'Visible', 'True');
   workbooks := OLE2.GET_OBJ_PROPERTY(app, 'Workbooks');
   workbook := OLE2.INVOKE_OBJ(workbooks,'Add');
   worksheets := OLE2.GET_OBJ_PROPERTY(workbook, 'Worksheets');
   worksheet := OLE2.INVOKE_OBJ(worksheets,'Add');
  args := OLE2.CREATE_ARGLIST;
   OLE2.ADD_ARG(args, '1:10');  -- Задаем диапазон ячеек с первой
по десятую
   row := OLE2.GET_OBJ_PROPERTY(worksheet, 'Rows', args);
   OLE2.DESTROY_ARGLIST(args);
   OLE2.SET_PROPERTY(row , 'RowHeight', 20); -- Устанавливаем
высоту ячейки
   OLE2.SET_PROPERTY(row, 'Value', 4); -- Утанавливаем значение
ячеек в диапазоне 1:10 равным четырем
   OLE2.RELEASE_OBJ(row);
OLE2.RELEASE_OBJ(worksheet);
   OLE2.RELEASE_OBJ(worksheets);
   OLE2.RELEASE_OBJ(workbook);
   OLE2.RELEASE_OBJ(workbooks);
   OLE2.RELEASE_OBJ(app);
EXCEPTION
    WHEN others THEN
       OLE2.Release_Obj( app );
       message('Ошибка OLE');
END;
```

- Вы можете применять функции Excel (листинг 15), используя для их вызова такие же имена, как и в Excel. Название этих функций можно просмотреть в окне «Мастер функций», которое вызывается командой меню Вставка|Функции.

Листинг 15 Использование формул Excel

```
Declare
   app              OLE2.OBJ_TYPE;
   workbook         OLE2.OBJ_TYPE;
   workbooks        OLE2.OBJ_TYPE;
   worksheet        OLE2.OBJ_TYPE;
   worksheets       OLE2.OBJ_TYPE;
   cell             OLE2.OBJ_TYPE;
   args             OLE2.List_Type;
BEGIN
   application := OLE2.CREATE_OBJ('Excel.Application');
   OLE2.SET_PROPERTY(app, 'Visible', 'True');
   workbooks := OLE2.GET_OBJ_PROPERTY(app, 'Workbooks');
   workbook := OLE2.INVOKE_OBJ(workbooks,'Add');

   -- Устанавливаем значение ячейки G7 равным 100

   args := OLE2.CREATE_ARGLIST;
   OLE2.ADD_ARG(args, 7);
   OLE2.ADD_ARG(args, 'G');
   cell := OLE2.GET_OBJ_PROPERTY(worksheet, 'Cells', args);
   OLE2.DESTROY_ARGLIST(args);
   OLE2.SET_PROPERTY(cell, 'Value', 100);
   OLE2.RELEASE_OBJ(cell);

   --Устанавливаем значение ячейки G8 равным 44

   args := OLE2.CREATE_ARGLIST;
   OLE2.ADD_ARG(args, 8);
   OLE2.ADD_ARG(args, 'G');
   cell := OLE2.GET_OBJ_PROPERTY(worksheet, 'Cells', args);
   OLE2.DESTROY_ARGLIST(args);
   OLE2.SET_PROPERTY(cell, 'Value', 44);
   OLE2.RELEASE_OBJ(cell);

   --Суммируем значение ячеек G7 и G8
```

```
    args := OLE2.CREATE_ARGLIST;
    OLE2.ADD_ARG(args, 9);
    OLE2.ADD_ARG(args, 'G');
    cell := OLE2.GET_OBJ_PROPERTY(worksheet, 'Cells', args);
    OLE2.DESTROY_ARGLIST(args);
    OLE2.SET_PROPERTY(cell, 'Formula', '=СУММ(G7;G8)');
    OLE2.RELEASE_OBJ(cell);
    -- Извлекаем корень из значения ячейки G9
    args := OLE2.CREATE_ARGLIST;
    OLE2.ADD_ARG(args, 10);
    OLE2.ADD_ARG(args, 'G');
    cell := OLE2.GET_OBJ_PROPERTY(worksheet, 'Cells', args);
    OLE2.DESTROY_ARGLIST(args);
    OLE2.SET_PROPERTY(cell, 'Formula', '=КОРЕНЬ(G9)');
    OLE2.RELEASE_OBJ(cell);

OLE2.RELEASE_OBJ(worksheet);
    OLE2.RELEASE_OBJ(worksheets);
    OLE2.RELEASE_OBJ(workbook);
    OLE2.RELEASE_OBJ(workbooks);
    OLE2.RELEASE_OBJ(app);
EXCEPTION
     WHEN others THEN
        OLE2.Release_Obj( app );
        message('Ошибка OLE');
END;
```

Лекция 26. Многомодульные приложения и библиотеки объектов

В этой лекции слушатель ознакомится с возможностями запуска нескольких модулей из одного приложения, совместного использования данных и способами передачи параметров. Также в этой лекции слушатель научится создавать библиотеки PL/SQL-кода и объектов, узнает, в чем преимущество многократного использования объектов и кода.

Ключевые слова: многомодульное приложение, мультиформа, дочернее приложение, глобальные группы записей, независимая форма, модальная форма, передача параметров между формами, Object Library, библиотека PL/SQL.

Цель лекции: ознакомить пользователя с различными методами запуска нескольких форм и глобальной передачей параметров. Научить слушателя создавать объектные библиотеки.

В Oracle Forms вы можете разрабатывать приложения, состоящие из одного и более модулей. В этой главе будет рассмотрено создание приложений с многомодульной структурой и обмен данными между составляющими этой структуры (рис. 26.1).

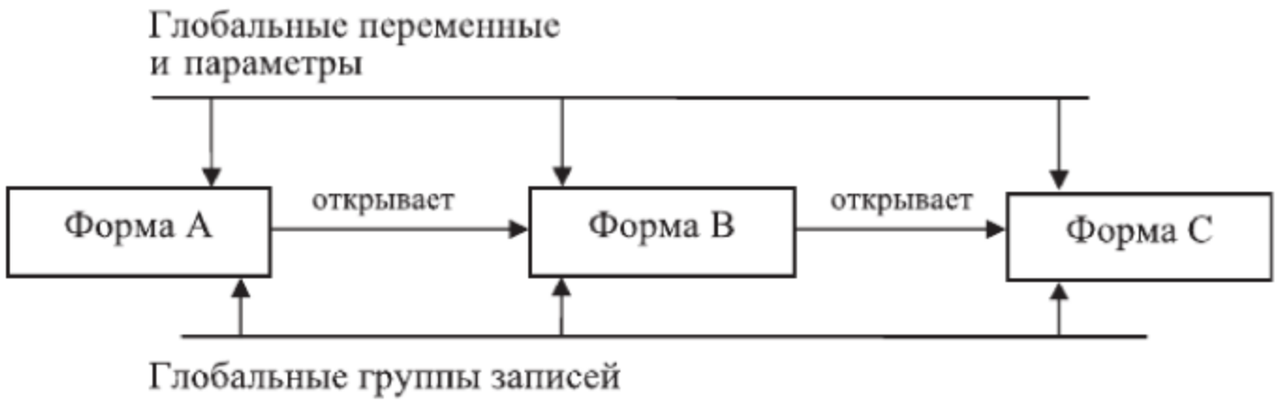

Рис. 26.1. Организация многомодульного приложения

Основной вопрос, который возникает, – это какова потребность в таком приложении и какой выигрыш от использования такой архитектуры. Для ответа перечислим основные преимущества использования многомодульного приложения:

- Отладка.
- Логика приложения.
- Масштабируемость и производительность.

Остановимся на каждом из пунктов более подробно.

Отладка – многомодульное приложение проще отладить, так как намного проще разбираться в небольших фрагментах кода и не ставить точки останова по всему приложению в поисках ошибки, а отлаживать конкретный модуль. Если у вас, к примеру, имеется готовое отлаженное приложение, то почему бы не интегрировать его в основной модуль? Это значительно проще и эффективнее, чем дописывать существующее приложение.

Логика приложения – многомодульное приложение позволяет разбить вашу задачу на логические единицы, тем самым давая возможность дорабатывать и вносить изменения не во все приложение, а лишь в отдельный модуль. Запуская отдельные модули приложения, вы можете определять тип открытия нового модуля, указывая Forms, открывать новую форму в той же сессии, что и родительская, или создать для него отдельную сессию. Логику такого приложения всегда проще понять и модернизировать.

Производительность – большие формы всегда намного дольше загружаются; а зачем загружать те части приложения, которые в ходе работы оказываются ненужными? В многомодульных приложениях все намного проще, так как необходимые структуры можно загружать по мере их надобности. Приложения, имеющие такую архитектуру, расходуют намного меньше данных.

Запуск другой формы

В Oracle Forms для запуска другой формы существуют встроенные подпрограммы, которые перечислены ниже:

- OPEN_FORM – открывает независимую форму;
- CALL_FORM – вызывает модальную форму;
- NEW_FORM – заменяет текущую форму.

Для более подробного ознакомления с этими процедурами выполните примеры, которые будут приведены далее.

CALL_FORM – запускает указанную форму, оставляя порождающую форму активной; Forms запускает вызываемую форму с теми же опциями Run Form, что и у порождающей формы. Когда вызываемая форма завершается – по функции EXIT или в результате неправильной передачи управления – обработка возвращается в порождающую форму в точку, откуда исходил вызов CALL. По умолчанию CALL использует параметры HIDE и NO_REPLACE:

- HIDE заставляет Forms стирать вызывающую форму с экрана перед рисованием вызываемой формы. Если вы используете параметр NO_HIDE, Forms не стирает вызывающую форму перед рисованием вызываемой формы. Это означает, что если вы вызываете форму с параметром NO_HIDE и некоторая страница вызываемой формы меньше области экрана, вызывающая форма выдается фоном.

- NO_REPLACE заставляет Forms поддерживать характеристику Default Menu Application вызывающей формы. Если вы используете параметр REPLACE, Forms заменяет характеристику Default Menu Application вызывающей формы на характеристику Default Menu Application вызываемой формы.

Процедура CALL_FORM имеет много описаний, которые вы можете найти в Online Help Forms.

Рассмотрим несколько основных способов написаний этих процедур, а также перечислим возможные параметры.

Синтаксис процедуры CALL_FORM

```
CALL_FORM (formmodule_name  VARCHAR2);
CALL_FORM (formmodule_name  VARCHAR2, display NUMBER, switch_menu
NUMBER,   query_mode NUMBER, data_mode NUMBER, paramlist_name
VARCHAR2);
...
```

- formmodule_name – определяет имя модуля. Тип данных VARCHAR2.
- display (HIDE установлен по умолчанию) – Form Builder скрывает вызывающее приложение перед прорисовкой вызванного модуля:
  - NO_HIDE – Form Builder не стирает вызывающую форму перед рисованием вызываемой формы;
  - HIDE – Form Builder скрывает вызывающее приложение перед прорисовкой вызванного модуля.
- switch_menu (NO_REPLACE по умолчанию) – заставляет Forms поддерживать характеристику Default Menu Application вызывающей формы:
  - DO_REPLACE – заставляет Form Builder заменить меню по умолчанию на модуль меню вызываемой формы;
  - NO_REPLACE – заставляет Form Builder поддерживать характеристику Default Menu Application вызывающей формы.
- query_mode (NO_QUERY_ONLY по умолчанию) – заставляет Form Builder запускать целевой модуль в нормальном режиме, разрешая выполнять операции DML:
  - QUERY_ONLY – заставляет Form Builder запускать формы в режиме RED_ONLY (только чтение), а также запрещает пользователю выполнять операции DML;
  - NO_QUERY_ONLY – заставляет Form Builder запускать формы в нормальном режиме.
- data_mode (NO_SHARE_LIBRARY_DATA по умолчанию) – в режиме выполнения Построитель форм не открывает общий доступ к данным библиотеки между формами:

  - SHARE_LIBRARY_DATA – в режиме выполнения построитель форм открывает общий доступ между формами;
  - NO_SHARE_LIBRARY_DATA – в режиме выполнения построитель форм не открывает общий доступ к данным библиотеки между формами.
- paramlist_name [ID] – имя списка параметров или его идентификатор.

С синтаксисом процедуры мы ознакомились, теперь можно перейти к выполнению примеров. Далее приведено два листинга, демонстрирующих вызов формы с параметрами (листинг 26.1) и без (листинг 26.2).

Листинг 26.1 Вызов формы с помощью процедуры CALL_FORM

```
BEGIN
  CALL_FORM('FormB', no_hide, no_replace, query_only);
END;
```

Ниже на рисунке приведен пример из листинга 26.1, где вызывающая форма не скрывается после запуска другой формы, но, несмотря на это, остается неактивной (рис. 26.2).

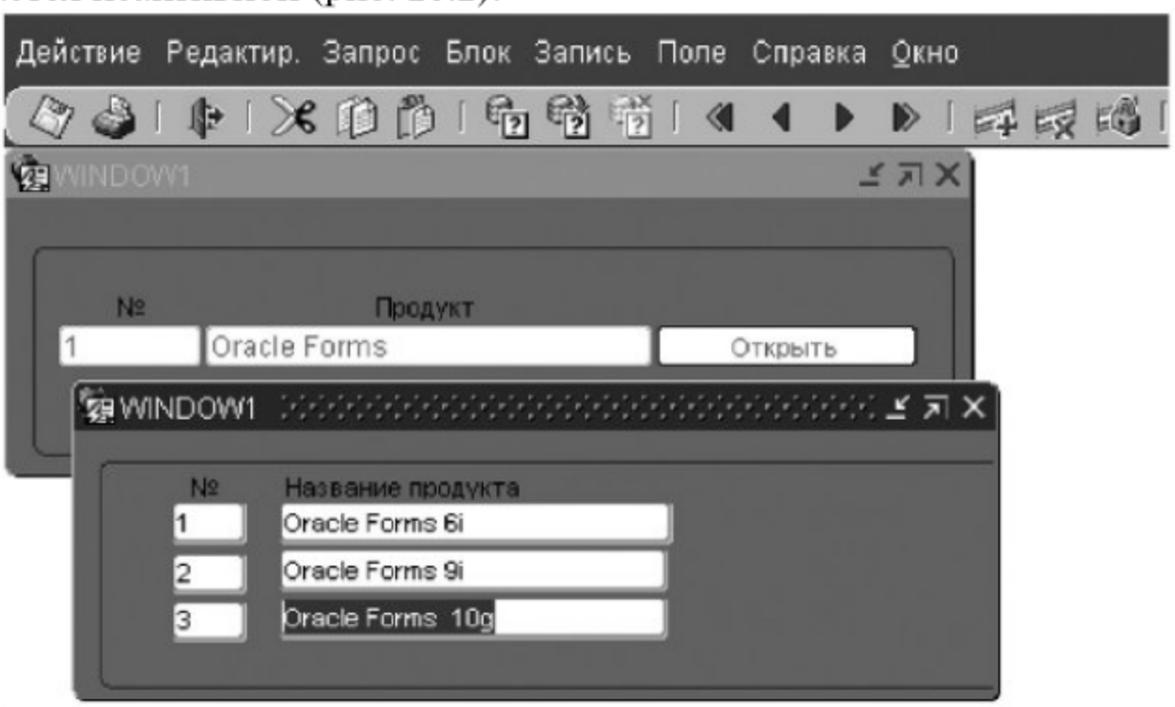

Рис. 26.2. Пример вызова формы с параметром NO_HIDE

Листинг 26.2 Вызов формы с передачей параметров

```
DECLARE
     pl_id   PARAMLIST;
BEGIN
   pl_id := Get_Parameter_List('list_data');
     IF NOT Id_Null(pl_id) THEN
       Destroy_Parameter_List( pl_id );
```

```
        END IF;
        pl_id := Create_Parameter_List('list_data');
      Add_Parameter(pl_id,'l_param',TEXT_PARAMETER,:item.param);
CALL_FORM('C:\exemple\Formb',NO_HIDE, no_replace, no_query_only,
pl_id);
END;
```

Предыдущий пример показывает, как с помощью списка параметров (рис. 26.3) можно передать значение переменной. Вы также можете использовать глобальные переменные и пользовательские параметры для передачи значений.

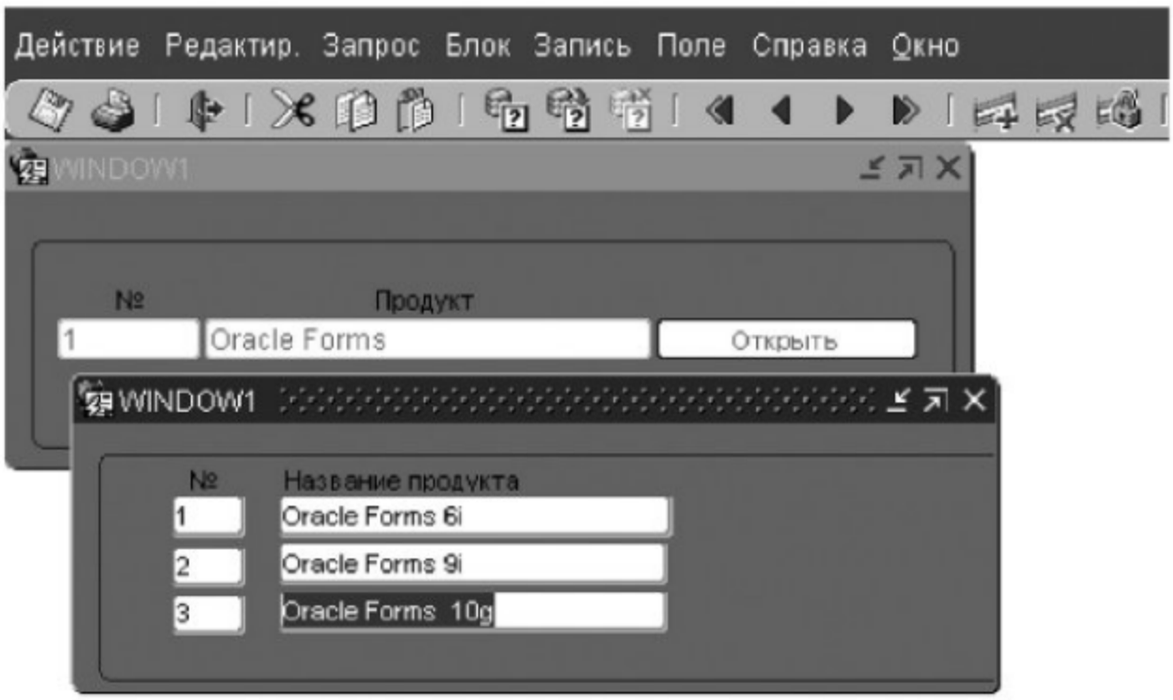

Рис. 26.3. Передача параметров через процедуру CALL_FORM

Ограничения в использовании CALL_FORM:

1. Передаваемый список параметров может содержать параметры типа TEXT_PARAMETER, но не DATA_PARAMETER.
2. После окончания сессии память, занятая процедурой CALL_FORM, не освобождается и образует стек вызываемых форм, поэтому не стоит злоупотреблять этой процедурой.

Запуск формы с помощью процедуры OPEN_FORM

OPEN_FORM – открывает указанную форму. Эта процедура в основном используется в многомодульных приложениях. Процедура OPEN_FORM может принимать различное число параметров, поэтому существует довольно много способов ее оформления. Ниже приведены некоторые из возможных вариантов написания этой процедуры.

```
OPEN_FORM
  (form_name  VARCHAR2);
OPEN_FORM
  (form_name       VARCHAR2,
   activate_mode   NUMBER,
   session_mode    NUMBER,
   data_mode       NUMBER,
   paramlist_id    PARAMLIST);
...
```

Принимаемые параметры:

- form_name – имя открываемой формы;
- activate_mode – управляет передачей фокуса. Это свойство может принимать два значения:
  - ACTIVE – после открытия формы передает фокус управления вызванной форме, делая ее активной;
  - NO_ACTIVE – после открытия фокус управления остается в вызывающем модуле;
- session_mode – определяет, в какой сессии будет открыта форма. Для этого вам доступны следующие константы:
  - SESSION – указывает, что открываемая форма должна использовать текущую сессию;
  - NO_SESSION – указывает, что для открываемой формы нужно создать новую сессию;
- data_mode – в режиме выполнения Построитель форм не открывает общий доступ к данным библиотеки между формами;
  - SHARE_LIBRARY_DATA – в режиме выполнения построитель форм открывает общий доступ между формами;
  - NO_SHARE_LIBRARY_DATA – в режиме выполнения построитель форм не открывает общий доступ к данным библиотеки между формами;
  - paramlist_name [ID] – имя списка параметров или его идентификатор.

Замечания:

1. Когда вы будете определять параметр ACTIVE или NO_ACTIVE, учтите, что в первом случае все выполняемые операторы, определенные после вызова процедуры, будут игнорироваться, так как фокус немедленно покидает вызывающее окно по завершении открытия новой формы. Если вы выбираете параметр NO_ACTIVE, то все исполняемые операторы, определенные ниже процедуры OPEN_FORM, будут выполнены по окончании загрузки открываемой формы в память.

2. Если вызывающая форма находится в режиме QUERY_ONLY, то и открываемая форма будет находиться в режиме QUERY_ONLY.

Ограничения:

1. Если вы используете параметр SESSION, то определение параметра data_mode как SHARE_LIBRARY_DATA вызовет ошибку выполнения (runtime error).

Теперь, когда мы ознакомились со всеми особенностями процедуры, выполним примеры, демонстрирующие влияние вышеописанных параметров.

Листинг 26.3 Примеры вызова процедуры OPEN_FORM

```
1. OPEN_FORM('form_name'); -- открывает форму с параметром NO_
SESSION по умолчанию, то есть форма будет открыта в текущей сессии
(процессе).
2. OPEN_FORM('form_name', activate, no_session) - открывает форму
в текущей сессии и немедленно передает ей управление
3. OPEN_FORM('form_name', activate, session); -- открывает форму
и создает для нее новую сессию, после загрузки форма немедленно
получает управление.
```

Ниже на рис. 26.4 изображен пример № 2 из листинга 26.3, на котором четко видны две активные формы.

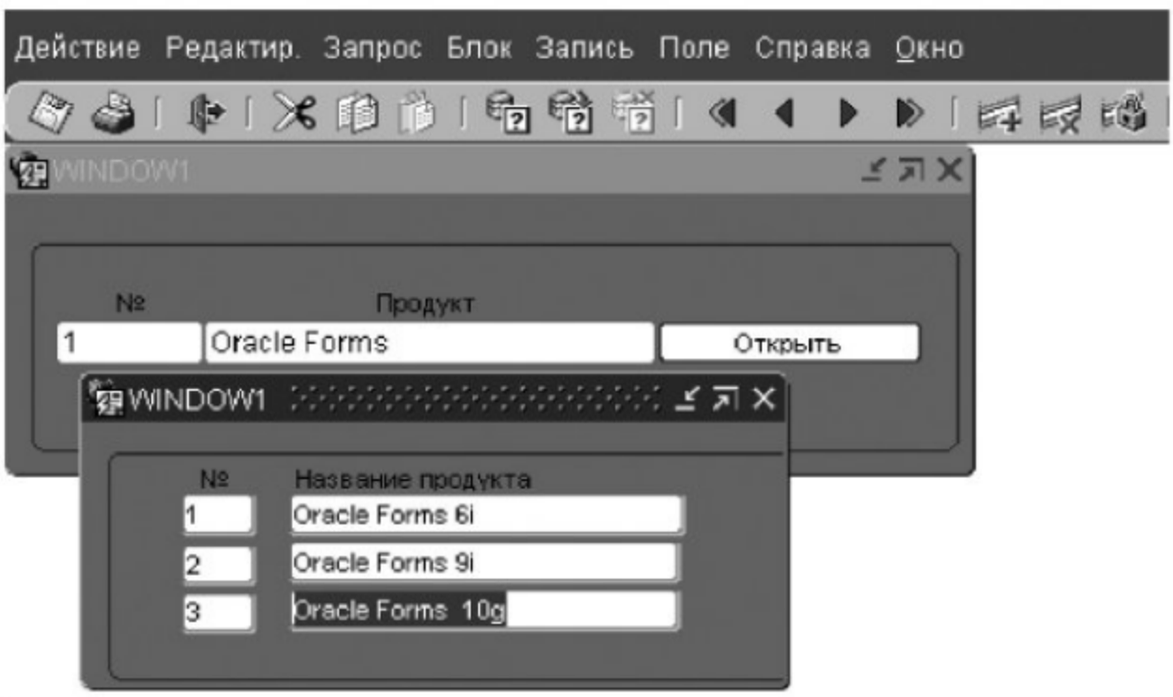

Рис. 26.4.. Вызов формы с помощью процедуры OPEN_FORM

Для того чтобы передать значение в открываемую форму, воспользуйтесь нижеприведенным примером, в котором показано, как передаются параметры через процедуру OPEN_FORM (рис. 26.5).

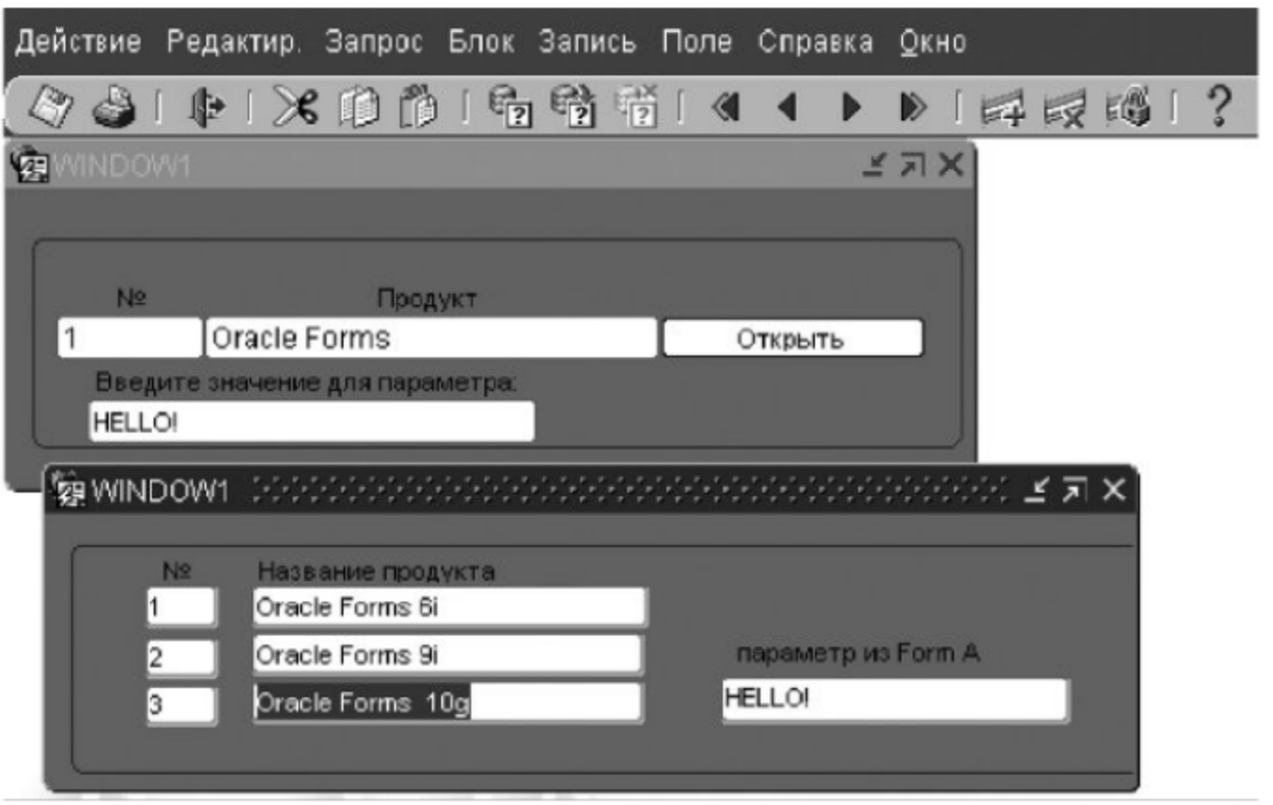

Рис. 26.5. Передача параметров через процедуру OPEN_FORM

Листинг 26.4. Передача параметров через процедуру OPEN_FORM

```
DECLARE
      pl_id    PARAMLIST;
      form_name  VARCHAR2(20);
BEGIN
    pl_id := Get_Parameter_List('list_data');
      IF NOT Id_Null(pl_id) THEN
        Destroy_Parameter_List( pl_id );
      END IF;
      pl_id := Create_Parameter_List('list_data');
     Add_Parameter(pl_id,'l_param',TEXT_PARAMETER,:item.param);
OPEN_FORM(Formb',activate, no_session, pl_id);
END;
```

Объектные библиотеки

Объектная библиотека (object library) – это модуль, который может объединять в себе различные наборы объектов, предназначенных для многократного использования. Объектная библиотека – это очень мощ-

ная структура, позволяющая хранить в себе всевозможные объекты и программные единицы. Применяя это средство многократного использования объектов, вы получаете возможность хранить в одном объекте однажды созданные объекты и программные единицы для совместной работы в других модулях. Объектная библиотека не имеет редактируемых свойств, кроме имени. Для включения в библиотеку новых объектов используется метод «Drag-and-Drop» («перетащить и оставить»).

Создание библиотеки

Для того чтобы создать объектную библиотеку и включить в ней различные объекты, выполните следующие действия:

1. Находясь в навигаторе объектов, выделите узел «Объектные библиотеки» и нажмите кнопку «Создать», после чего появится узел с именем библиотеки и иконкой в виде книги справа от названия.
2. Вызовите палитру свойств библиотеки и назовите ее «My_lib».
3. Находясь в узле «Объектные библиотеки», вызовите всплывающее меню и выберите пункт «Объектная библиотека» для открытия окна библиотеки (рис. 26.6).

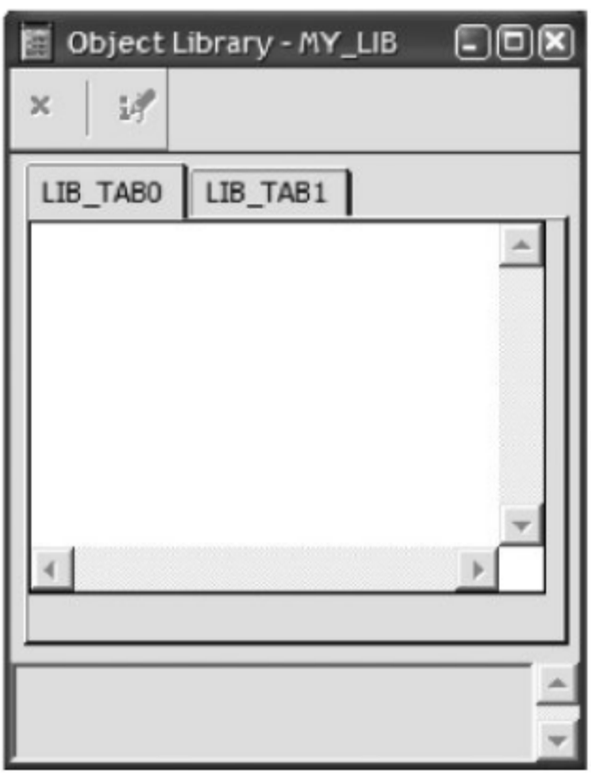

Рис. 26.6. Объектная библиотека

4. Вы можете включить в библиотеку любой объект модуля формы или программную единицу. Для этого выделите объект, который хотите поместить в библиотеку, и, не отпуская левую кнопку мыши, перетащите его на вкладку библиотеки.

5. Перетаскиваемый объект при попадании в объектную библиотеку не изменяет своих свойств, поэтому отображается в ней с тем же именем и типом, с которым вы его перетащили (рис. 26.7).

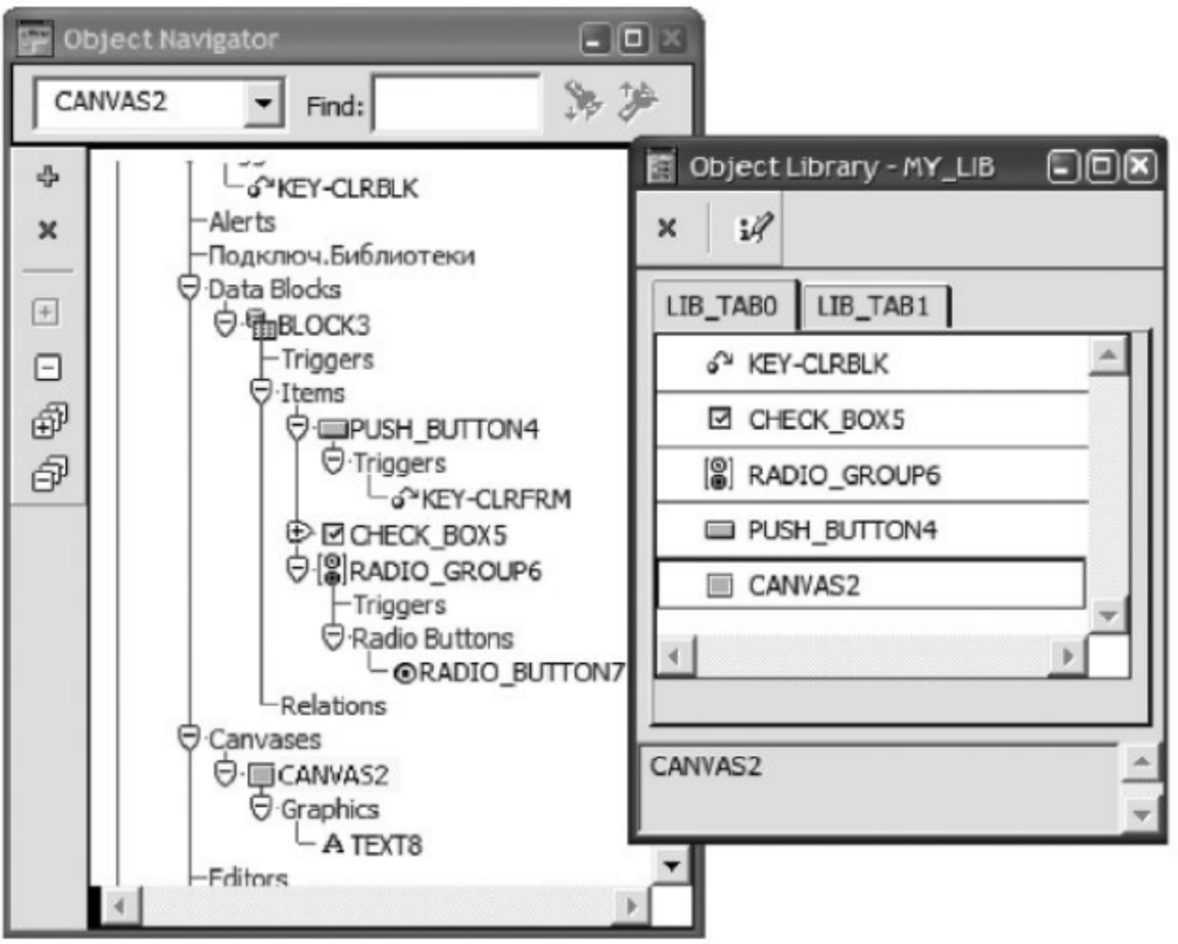

Рис. 26.7. Перетаскивание объектов

6. После того как вы собрали в объектную библиотеку все необходимые объекты, сохраните ее командой меню «Файл | Сохранить Как». Библиотека хранится в формате .OLB.

Вы не можете редактировать или изменять объекты, помещенные в объектную библиотеку, зато вы можете добавить комментарий к объекту, нажав кнопку «Edit Comment», располагающуюся на верхней палитре инструментов. Чтобы добавить комментарий к объекту, нажмите кнопку «Edit Comment» и в появившемся окне «Comment» введите свой комментарий. Чтобы удалить объект из объектной библиотеки, нажмите кнопку «Удалить объект».

Для повторного использования объекта библиотеки необходимо открыть ее в навигаторе объектов командой меню Файл | Открыть. Чтобы переместить объект в соответствующий модуль из объектной библиотеки, просто перетащите его, не отпуская левую кнопку мыши, в нужное место модуля.

Библиотека PL/SQL

Библиотека – это модуль, в котором объединены многократно использующиеся хранимые программы. Библиотеки позволяют однажды созданную программную единицу использовать совместно различными приложениями. Это средство удобно еще и потому, что имеет различные форматы файлов, специфичных своим содержанием. Ниже перечислены форматы фалов библиотеки и их описание:

- lib_name.PLL – этот файл хранит в себе исходный и специфический для каждой платформы исполняемый код (p-код).
- lib_name.PLX – этот файл хранит в себе только исполняемый код (скомпилированный).
- lib_name.PLD – этот файл хранит в себе исходный текст библиотеки.

Если перед вами стоит задача скрыть исходный код библиотеки, то вам, несомненно, нужно использовать библиотечный файл .PLX, который содержит только исполняемый код.

Создание библиотеки

Для того чтобы создать библиотеку, выполните следующие действия:

1. Находясь в Навигаторе Объектов, выделите узел «Библиотеки PL/SQL» и нажмите кнопку «Создать», после чего все в том же узле создастся новый узел с названием библиотеки.
2. Раскройте узел «Библиотеки PL/SQL» и выберите узел «Программы» для создания компонентов библиотеки. Находясь в узле «Программы», нажмите кнопку «Создать» для запуска окна «Новая программа», в котором определите имя и тип создаваемой программной единицы (функция, процедура, пакет). В запустившемся PL/SQL-редакторе наберите и скомпилируйте исходный текст программной единицы.
3. Вы также можете вставить уже готовую программную единицу в состав библиотеки, для этого достаточно просто скопировать целевую программу из узла «Программные единицы» модуля.

Когда вы пишете программные единицы для библиотек, вы не можете использовать глобальные и системные переменные, параметры и элементы, так как библиотека не компилируется отдельно от формы. Чтобы это обойти, нужно пользоваться встроенными подпрограммами NAME_IN и COPY для непрямого обращения к вышеперечисленным переменным.

Подключение библиотек к модулю

Теперь, когда мы научились создавать библиотеки, давайте попробуем их в работе. Для того чтобы подключить библиотеку, выполните следующие действия:

1. Находясь в навигаторе объектов, выделите узел «Attached Library» и нажмите кнопку «Создать» для запуска окна «Attached Library» (рис. 26.8).

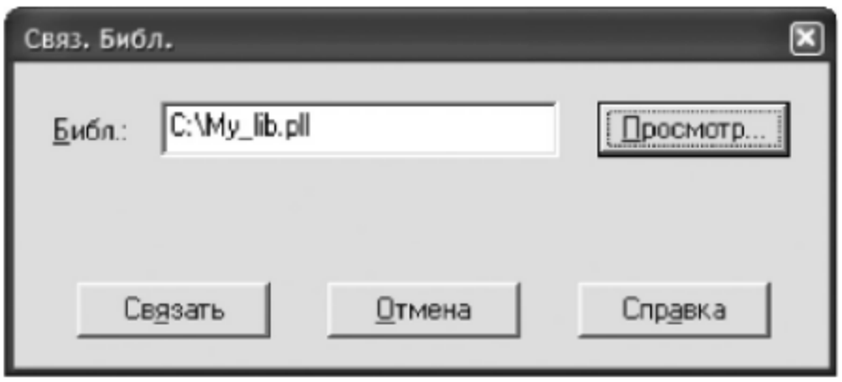

Рис. 26.8. Окно «Attached Library»

2. Находясь в окне «Связать библиотеку», нажмите кнопку «Просмотр» для выбора подсоединяемой библиотеки из файловой системы.
3. Выбранная библиотека отобразится в поле «Библиотека» с указанием полного пути ее расположения в файловой системе. Нажмите кнопку «Связать» для подключения библиотеки к модулю формы.
4. После подтверждения связывания библиотеки Forms выведет на экран предупреждение (рис. 26.9), в котором вам будет предложено сохранить или удалить путь, указанный при выборе библиотеки.

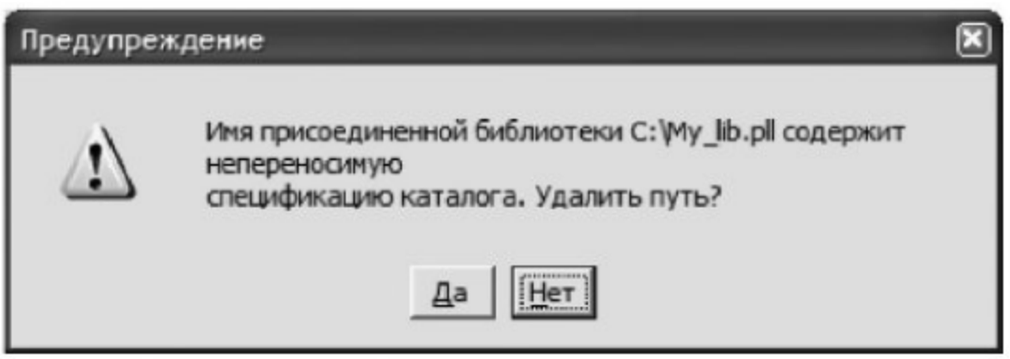

Рис.26.9. Предупреждение «Attached Library»

Варианты выбора:

- кнопка «Да» – в этом случае Forms удалит указанный путь к библиотеке и будет искать библиотеку в текущей директории, а затем в директориях, перечисленных в переменной среды PATH (FORMS%_PATH и ORACLE_PATH);

- кнопка «Нет» – в этом случае Forms сохранит путь и во время выполнения будет искать библиотеку строго по указанному пути.

Примечание: *с точки зрения переносимости нужно выбирать библиотеку без указания пути, так как в Forms определения пути для подключения библиотек хранятся внутренне.*

5. После того как вы подтвердили одно из вышеперечисленных действий, библиотека появляется в узле «Attached Library» и готова к использованию.

Подключенная библиотека доступна только для чтения, поэтому недоступна для редактирования. Если вы все же хотите отредактировать библиотеку, то откройте ее в навигаторе объектов в узле «Библиотеки PL/SQL», для этого выполните Файл | Открыть.

Для того чтобы обратится к хранимым программам или пакетам, определенным в присоединенной библиотеке, не нужно указывать никаких дополнительных префиксов, то есть все как обычно:

```
Var_name:=lib_function_name;
Procedure_lib_name;
```

Вызов функций и процедур из базы данных

Иногда не все можно сделать средствами одного только Oracle Forms, поэтому возникает необходимость в написании специфичных функций и процедур, т. к. PL/SQL в Forms не всегда позволяет использовать все расширения этого языка, да и зачем в Forms создавать функции или процедуры, которые уже есть в БД.

Можно привести еще один пример, суть которого заключается в том, что нужно узнать IP-адрес машины, на которой установлен клиент Forms'a. Как не ищите и не старайтесь, средствами Forms'a вы этого не сделаете, т. к. даже известная нам функция get_application_property(), которая может вывести пароль, строку связи, имя вашей ОС и т. д., не содержит нужной нам константы. Если бы такая задача встала перед вами в другом виде, например, узнать свой IP-адрес в SQL*Plus'e, то все было бы намного проще, т. к. можно ограничиться одной стандартной функцией:

```
SELECT SYS_CONTEXT ('USERENV','IP_ADDRESS') FROM Dual;
```

После ее выполнения мы получим желаемый результат. Зная это, мы можем написать функцию, которая будет возвращать нужное нам значение:

```
create or replace Function IP_ADDR return varchar2 is
  b varchar2(20);
begin
  select sys_context('USERENV','IP_ADDRESS') into b from dual;
  return (b);
end;
```

Теперь нам остается в Forms'е прочитать значение этой функции в какую-либо переменную Forms или элемент текста. Для демонстрации примера создадим кнопку и в триггере WHEN_BUTTON_PRESSED наберем код:

```
WHEN_BUTTON_PRESSED
DECLARE
a varchar2(20);
BEGIN
select имя_схемы.IP_ADDR into a from dual;
select a into :block_name.item_name from DUAL;
END;
 /*
А лучше всего выбрать эти данные не прямым запросом, причем у нас
их два, а просто сделать вот так:
*/
 BEGIN
   :block_name.item_name:=имя_схемы.IP_ADDR
END;
```

Из этого примера видно, что в Forms элементу формы можно присвоить значение функции и без выполнения двух запросов, а это намного экономичнее. С помощью такого подхода вы можете легко реализовать любую функцию трассировки БД и многое другое.

Выполнение команд операционной системы

В Oracle Forms вы можете выполнять команды операционной системы без особой сложности, используя встроенную процедуру HOST.

HOST – выполняет команду операционной системы.

Синтаксис:

```
HOST (SYSCMD IN varchar2, KWD IN number);
HOST (SYSCMD IN varchar2);
```

Параметры:

SYSCMD – команда операционной системы.

KWD – режим выполнения команды OS. Допустимые значения SCREEN и NO_SCREEN. Если вы используете параметр NO_SCREEN, Forms не очищает экран и не запрашивает у оператора выход из команды.

Если ни один из параметров не указан, то по умолчанию команда выполняется с параметром NO_SCREEN.

Выполнение команды операционной системы

```
HOST ('start notepad.exe', NO_SCREEN);
HOST ('start notepad.exe', NO_SCREEN);
HOST ('start notepad.exe', NO_SCREEN);
```

1. Создайте форму и сохраните ее как OS.fmb.
2. Создайте блок и вызовите редактор разметки для автоматического создания холста.
3. Начертите кнопку и элемент текста на холсте.
4. Назовите элемент текста «os_com», а кнопку – «Старт».
5. Создайте триггер WHEN-BUTTON-PRESSED и наберите в нем код, приведенный в листинге.

WHEN_BUTTON_PRESSED

```
HOST ('start '||:os_com);
```

6. Запустите форму на выполнение, введите в текстовый элемент команду notepad.exe и нажмите кнопку «Старт», после чего запустится блокнот.

Лекция 27. Использование ActiveX и OLE в Oracle Forms

В этой лекции рассказывается о компонентах ActiveX, о том, как их встраивать в форму и в другие приложения, такие как Windows Media Player, Internet Explorer и др. После этой лекции слушатель также научится импортировать библиотеки для работы с элементами управления ActiveX, в качестве примера будет рассмотрен пример отправки почты.

Ключевые слова: OLE, DCOM, ActiveX, WM Player, Internet Explorer, отправка почты, встраивание браузера, встраивание проигрывателя.

Цель лекции: научить пользователя встраивать в форму компоненту ActiveX и управлять ею посредством библиотеки Shell API.

ActiveX – это технология Microsoft, которая предоставляет программистам наборы стандартных библиотек, облегчающих процесс кодирования. ActiveX – это продвинутая версия OLE, основанная на распределенной компонентной объектной модели DCOM. Библиотеки ActiveX обеспечивают функциональность, достаточную для написания сетевых приложений, а также поддерживают совместимость с любыми компонентами OLE.

Примечание: *ActiveX не поддерживается в последних версиях Oracle Forms, и вы не можете использовать его в UNIX (Linux).*

Встраивание Windows Media Player

Мы с вами разберем несложный пример – создание WM Player, проигрывающего какой-либо файл.

1. Находясь в редакторе разметки, найдите на панели инструментов элемент «OCX» и начертите его на канве.
2. Расположите объект на канве так, чтобы была достаточная область видимости для проигрывания клипа.
3. Нажмите правую кнопку мыши, находясь в фокусе объекта ActiveX, вызовите всплывающее меню. Выберите пункт «Insert Objects», и на экране у вас появится одноименное окошко (рис. 27.1).
4. Выберите Windows Media Player из предложенного списка и подтвердите выбор.

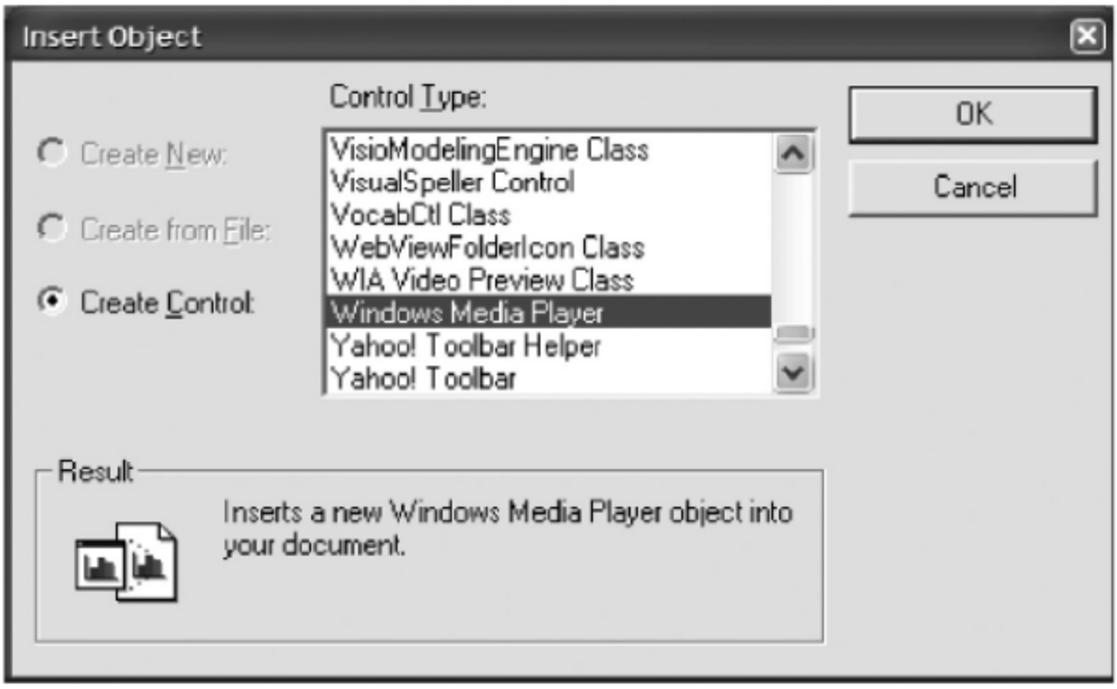

Рис. 27.1. Окно «Вставка объекта»

5. Объект WM Player загружен, и теперь ваш элемент имеет вид черного прямоугольника. Чтобы посмотреть, как он выглядит в режиме выполнения, запустите форму. В запущенной форме вы увидите проигрыватель Media Player, но не сможете выполнить никаких действий над ним. Закройте форму и перейдите обратно в свойства элемента MPlayer.
6. Найдите свойство «Свойство управление» и нажмите кнопку «Найти...» для запуска окна настроек WM Player (рис. 27.2).
7. В этом окне вы можете определить настройки и параметры запуска проигрывателя, выбрать компоновку элементов управления, а также указать файл воспроизведения. Укажите в источнике имя проигрываемого файла, им может быть как видео-, так и аудиофайл.
8. После того как вы выбрали файл проигрывания, запустите форму для просмотра результатов. По умолчанию в проигрывателе имеются все основные элементы управления, такие как звук, прокрутка и кнопки управления проигрыванием файла. В окне настроек проигрывателя вы можете выбрать режим отображения элементов управления:
   - None – при выборе этого режима будет отображаться только окно зрительных образов или окно видео. Такие элементы, как прокрутка, регулятор громкости и кнопки управления проигрыванием, будут невидимы.
   - Mini – в этом режиме помимо окна видео или зрительных образов также будут доступны регуляторы громкости.
   - Full (default) – в этом режиме доступен полный набор элементов управления.

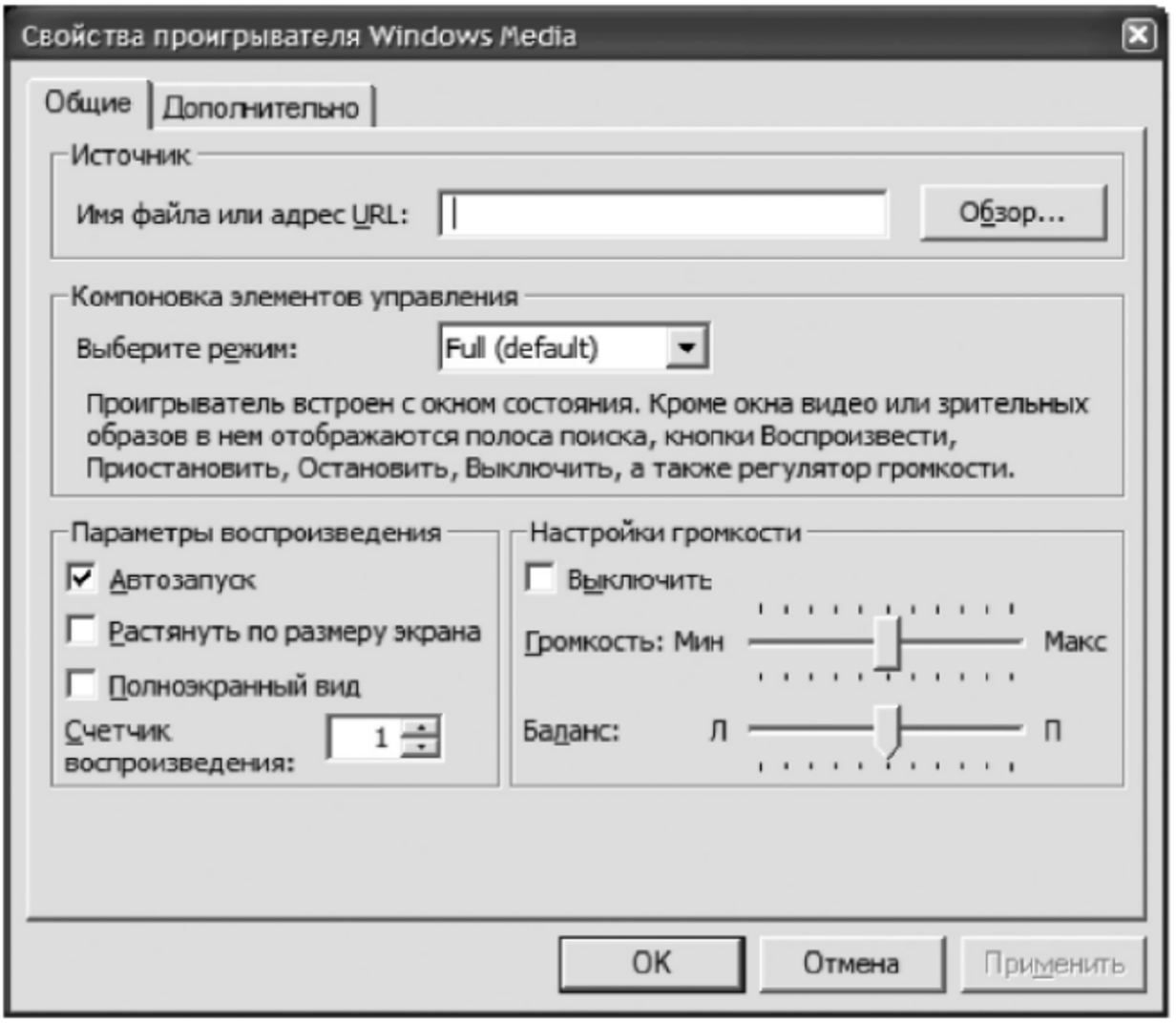

Рис. 27.2. Окно настроек проигрывателя

- Invisible – в этом режиме не видно никаких элементов, даже окна зрительных образов и видео. Этот режим подходит для проигрывания аудиофайлов, появляется ощущение фонового звука, и форма избавлена от отображения дополнительных элементов.

Мы добились желаемого результата – объект встроен и выполняет наши требования. Вы можете расширять функциональность этих элементов с помощью библиотек OLE, используя ее классы в своих программных модулях и триггерах.

Импорт интерфейса библиотеки OLE

Вы можете импортировать библиотеки OLE в свое приложение, используя инструмент «Импорт интерфейса библиотеки OLE». Пакеты методов OLE позволяют управлять встроенными элементами.

Рассмотрим пример, в котором создадим Oracle Sound Control Object и научимся им управлять с помощью библиотек OLE.

1. Находясь в редакторе разметки, найдите на панели инструментов элемент «ActiveX» и начертите его на канве. Установите для этого элемента следующие свойства:
   - Ширина: 210;
   - Высота: 100.
2. Правым щелчком мыши по элементу вызовите всплывающее меню и выберите пункт «Insert Object» для запуска окна объектов ActiveX. Найдите в списке Oracle Sound Control Object и подтвердите свой выбор.
3. Вызовите всплывающее меню элемента правым щелчком мыши и выберите пункт Oracle Sound Control Object | Свойства для запуска окна свойств проигрывателя (рис. 27.3).

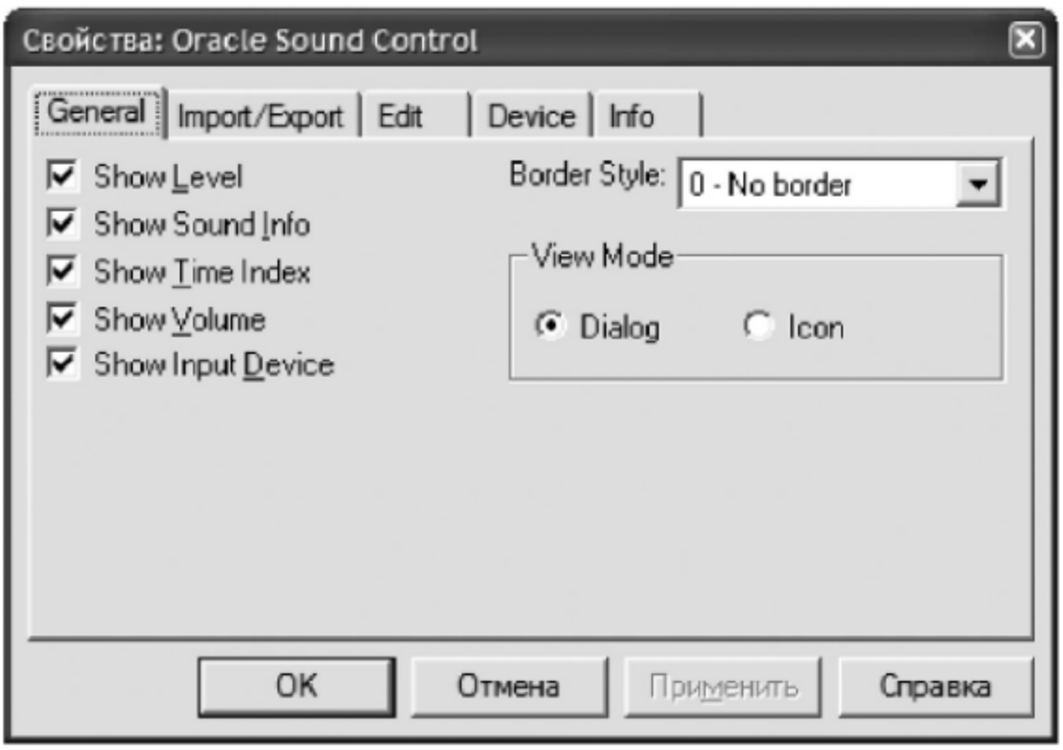

Рис. 27.3. Окно свойств «Oracle Sound Control»

4. В этом окне вы можете управлять настройками проигрывателя, изменять внешний вид, источник звука, а также скрывать регуляторы громкости, таймеры и другие элементы управления проигрыванием, выключая переключатели на вкладке «Общие». Перейдите на вкладку «Импорт/Экспорт» для выбора источника звука. Найдите в вашей файловой системе звуковой файл с расширением *.wav.
5. Перейдите на вкладку «Device» и выберите радиокнопку «Line-in». Нажмите кнопку «Применить» для подтверждения установленных настроек и запустите форму на выполнение. Убедитесь, что проигрыватель работает и воспроизводит звуковой файл после нажатия кнопки «Воспроизвести».

Элемент создан, и теперь, для того чтобы управлять им программно, нам необходимо импортировать соответствующие методы библиотеки OLE.

1. Находясь в редакторе разметки, выберите пункт главного меню Программа | Импорт интерфейса библиотеки OLE (рис. 27.4).

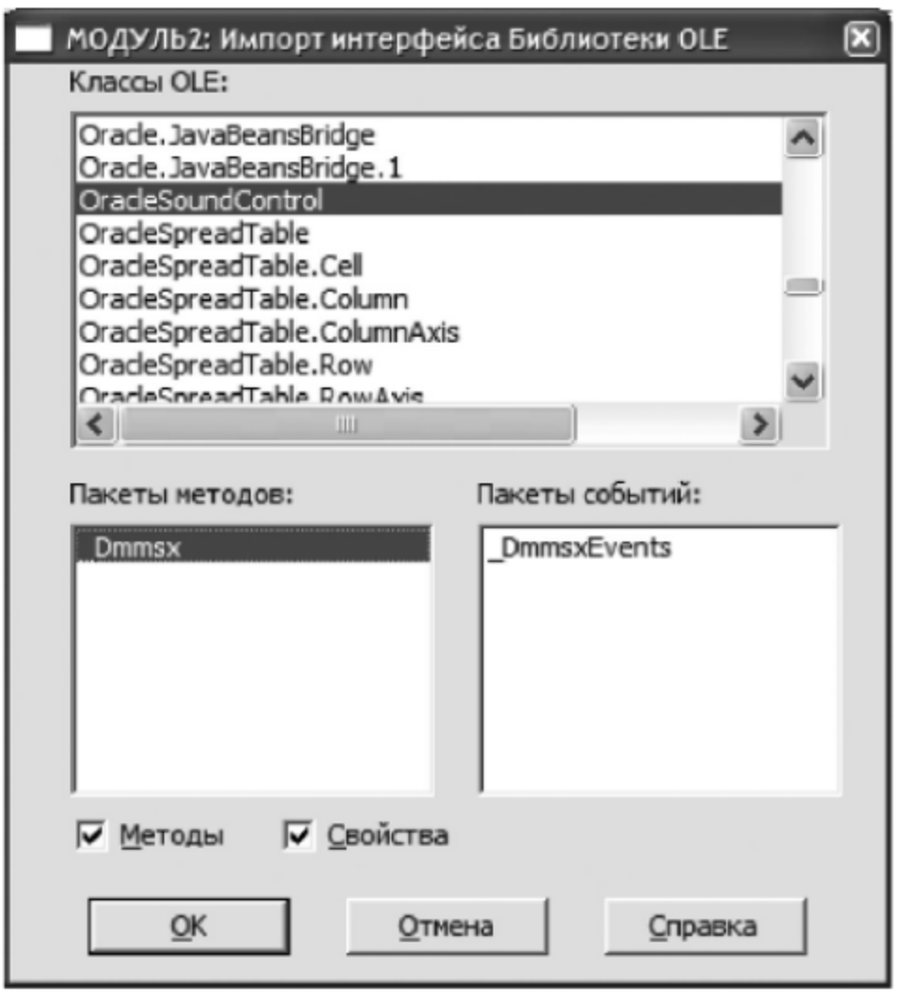

Рис. 27.4. Программа импорта

2. В этом окне содержится перечень доступных классов OLE и соответствующие им методы и события. Найдите класс OracleSoundControl в окне классов OLE. Выделите пакет методов _Dmmsx и нажмите кнопку «OK». Выбранный вами пакет появится в узле «Программы» навигатора объектов (рис. 27.5).
3. Создайте три кнопки и разместите их под элементом OSound. Установите для первой кнопки метку «Играть», для второй кнопки – «Стоп» и «Открыть файл» – для третьей.
4. Создайте триггер WHEN-BUTTON-PRESSED для кнопки «Воспроизвести» и напишите в нем следующий код:

```
WHEN_BUTTON_PRESSED
OracleSoundControl_DMMSX.Play(:item('Media.OSound').interface);
```

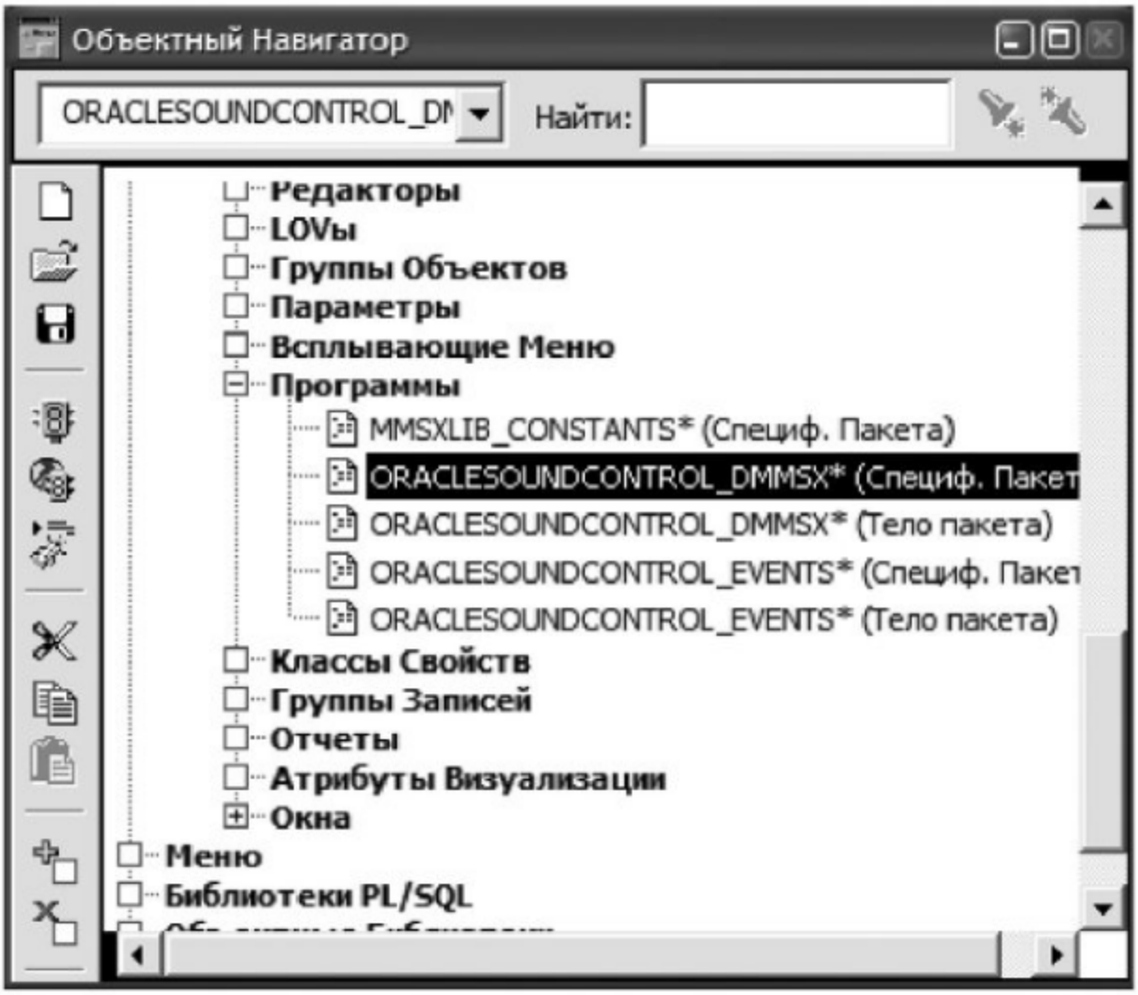

Рис. 27.5. Пакет «OracleSoundControl»

5. Создайте триггер WHEN-BUTTON-PRESSED для кнопки «Стоп» и напишите в нем следующий код:

```
WHEN_BUTTON_PRESSED
OracleSoundControl_DMMSX.Stop(:item('Media.OSound').interface);
```

6. Создайте триггер WHEN-BUTTON-PRESSED для кнопки «Открыть файл» и напишите в нем следующий код:

```
WHEN_BUTTON_PRESSED
OracleSoundControl_DMMSX.ImportFileAs (:item('Media.OSound').
interface);
```

Поэкспериментируйте с другими элементами, импортируя методы для управления их функциональностью.

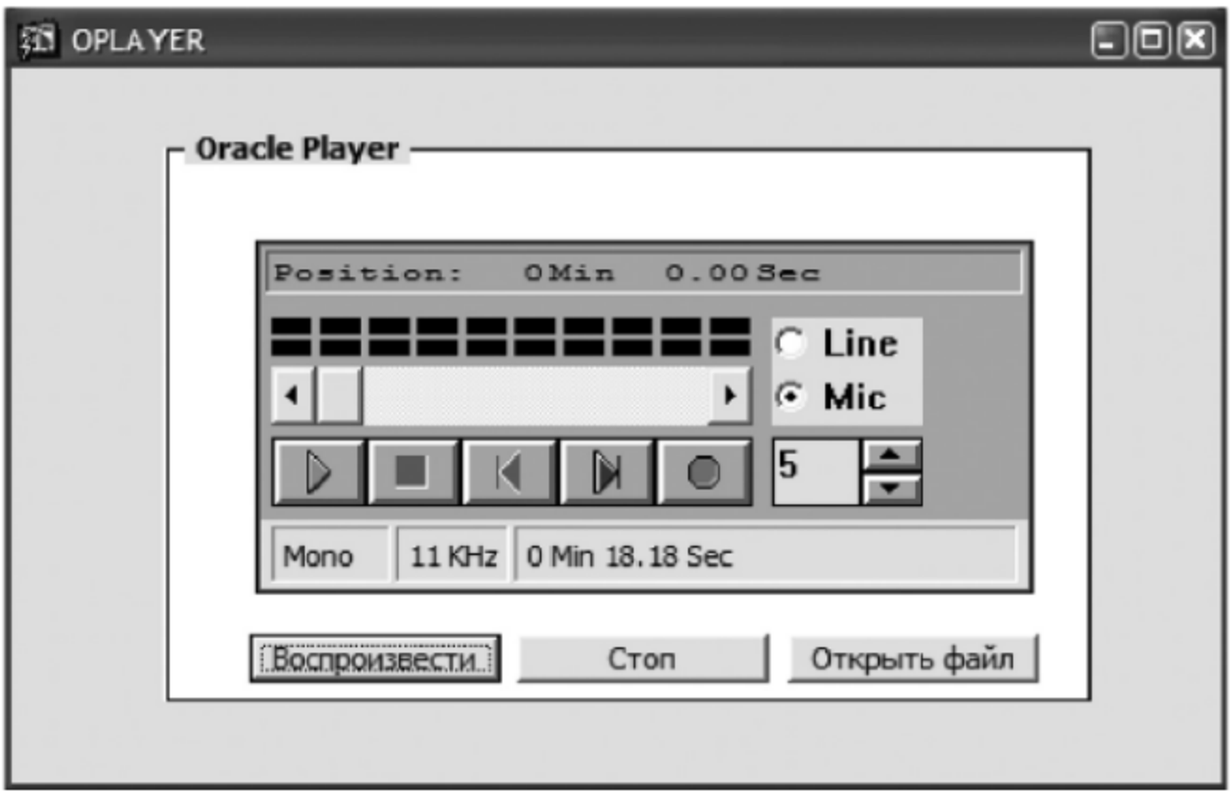

Рис. 27.6. Проигрыватель «Oracle Player»

Контейнер OLE

OLE (англ. *Object Linking and Embedding*) – это мощная технология связывания и внедрения объектов и протокол. Первая версия OLE была выпущена в 1990 году на основе технологии DDE. OLE позволяет передавать часть работы от одной программы редактирования к другой и возвращать результаты назад. Эта технология применяется для передачи данных между различными приложениями, внедрения одного типа документа в другой, работы с мультимедиа и для обработки составных документов. Следующая версия этой технологии уже начала развиваться в архитектуру COM (Component Object Model). OLE 2.0, в отличие от своего предшественника OLE 1.1, не только стала надстройкой над архитектурой COM, но и использует новые особенности автоматизации технологий **drag-and-drop**, **in-place activation** и **structured storage**.

Создание элемента

Используя OLE-контейнер, вы можете встраивать документы Microsoft Office, различные медиапроигрыватели и многое другое из списка объектов OLE, с которым вы ознакомитесь немного позже. Рассмотрим с вами пример, в котором, применяя компонент OLE, созда-

дим форму мониторинга вашей системы. Чтобы создать OLE-контейнер и вставить в него объект, необходимо выполнить следующие действия:

1. Находясь в редакторе разметки, найдите на панели инструментов элемент «OLE» и начертите его на канве. Установите для этого элемента следующие свойства:
   - Имя: SysMon;
   - Ширина: 340;
   - Высота: 220.
2. Для того чтобы вставить объект OLE, вызовите всплывающее меню элемента и выберите пункт «Insert Object». В появившемся окне (рис. 27.7) выберите объект «System Monitor Control».

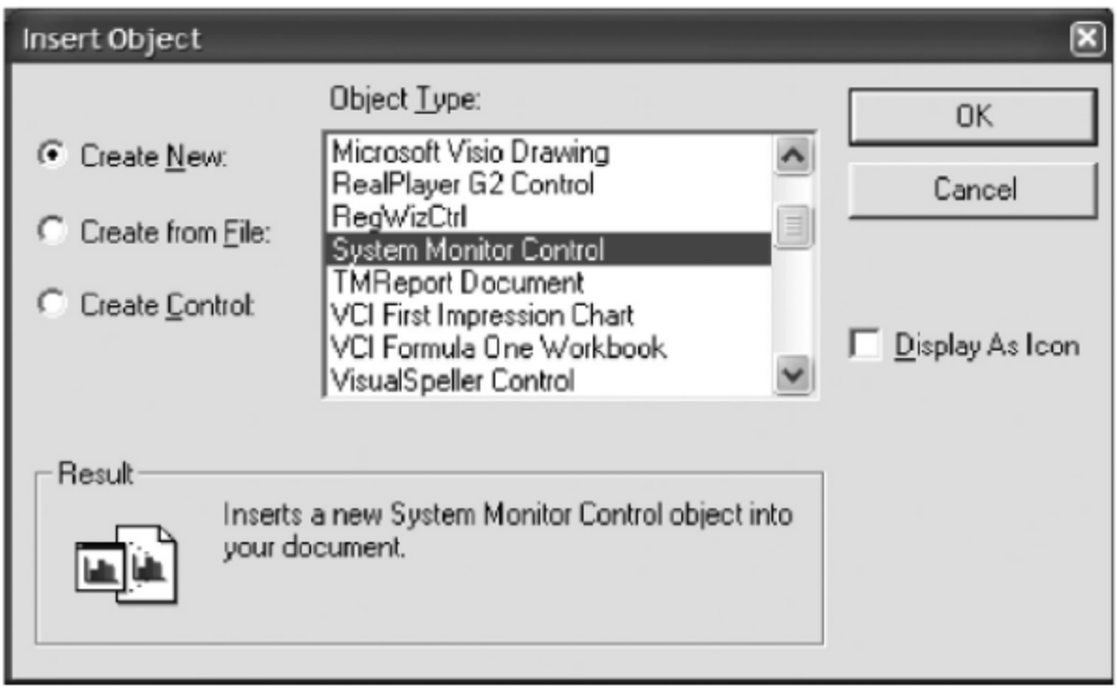

Рис. 27.7. Окно «Вставить объект OLE»

3. Запустите форму на выполнение. Вызовите всплывающее меню элемента и выберите пункт «Добавить счетчики» для запуска одноименного окна (рис. 27.8).
4. Для того чтобы добавить счетчик, выберите объект, за которым предполагается вести наблюдение из выпадающего списка «Объект». Выберите необходимые счетчики из списка, для этого каждый раз при выборе счетчика нажимайте кнопку «Добавить» (рис. 27.9).
5. Вы также можете управлять видом системного монитора, цветом, шрифтом и многими другими элементами, перечисленными в окне свойств монитора. Вы может следить не только за системой, но и за базой данных, указав ее в качестве источника. Для запуска окна свойств системного монитора вызовите всплывающее меню элемента и выберите пункт «Свойства». Пункт «Сохранить как» сохраняет текущие показания в файл с расширением *.html.

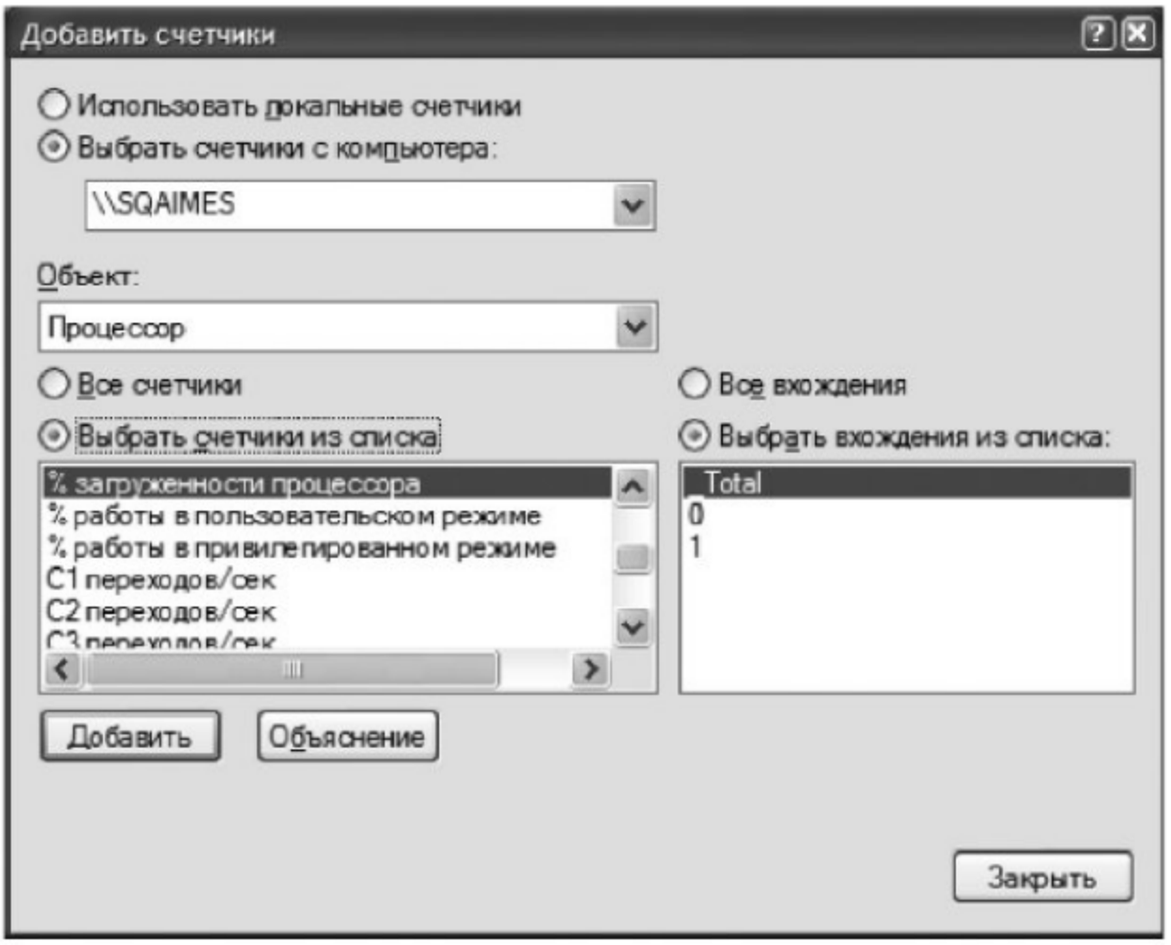

Рис. 27.8. Окно «Добавить счетчики»

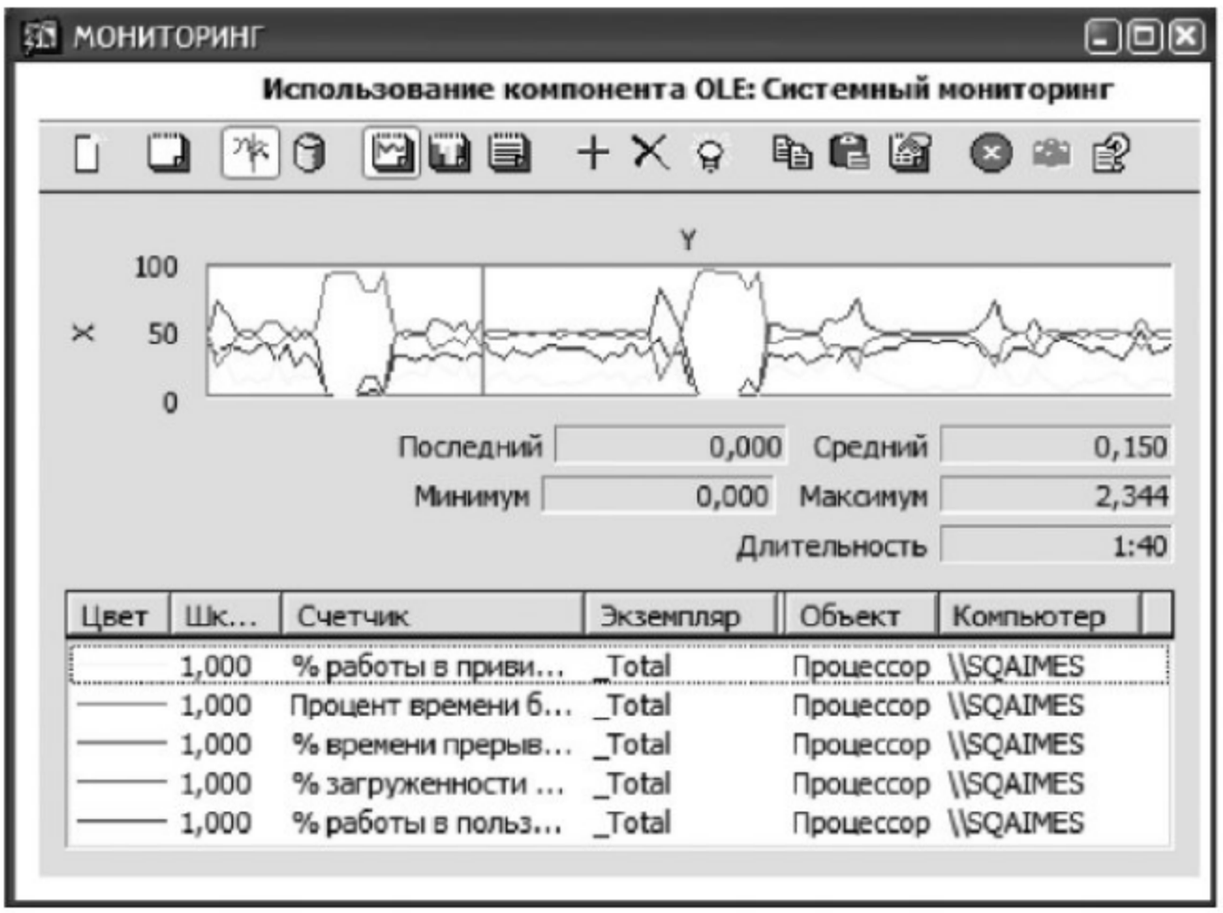

Рис. 27.9. Форма «МОНИТОРИНГ»

Попробуйте самостоятельно интегрировать различные объекты OLE в вашу форму, и вы обязательно найдете для себя что-то полезное.

Встраивание Internet Explorer

Как вы уже убедились, компоненты OLE и ActiveX позволяют за очень короткое время добиться хороших результатов, позволяя нам интегрировать в форму уже готовые объекты.

Выполним небольшое упражнение, в котором интегрируем Internet Explorer в нашу форму.

1. Разместите элемент ActiveX на канве и растяните его на всю область канвы.
2. Находясь в палитре свойств элемента, найдите свойство «Класс выполнения» и нажмите кнопку «Найти» для запуска окна «OLE-классы» (рис.27.10).

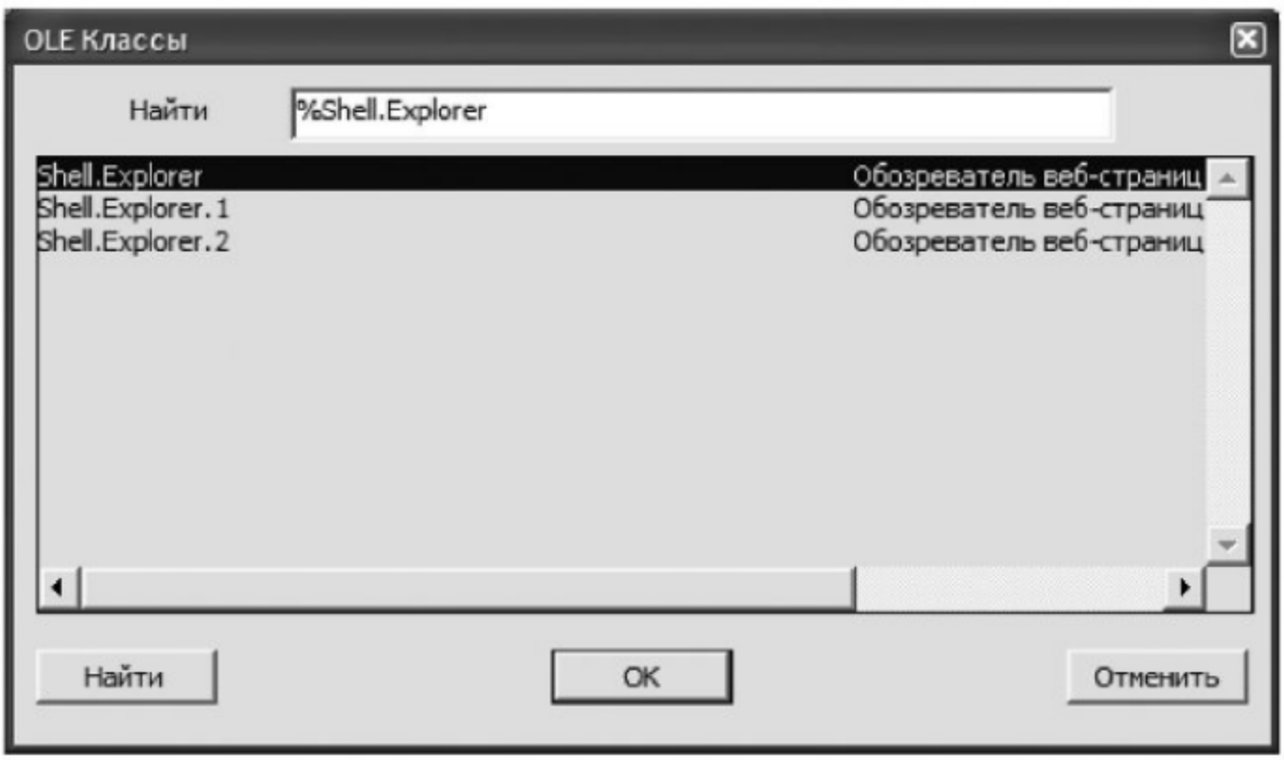

Рис. 27.10. Окно «OLE Классы»

3. Найдите класс Shell.Explorer и подтвердите выбор.
4. Вызовите всплывающее меню элемента и выберите пункт «Insert Object». Из доступного списка объектов выберите единственный доступный объект «Обозреватель веб-страниц Microsoft».
5. Находясь в навигаторе объектов, выделите элемент ActiveX и выберите пункт главного меню Программа | Импорт интерфейса библиотеки OLE. Выберите пакет методов IWebBrowser2.

6. Создайте триггер уровня формы WHEN-NEW-FORM-INSTANCE и напишите в нем следующий код:

```
WHEN-NEW-FORM-INSTANCE
Shell_IWebBrowser2.Navigate(:item('IEB.IE').interface,
'http://www.oracle.com');
```

Попробуйте запустить форму на выполнение, и вы увидите, как при запуске загрузится страница http://www.oracle.com.

Отправка почты с прикрепленным файлом

Отправляя письма своим друзьям, родным или близким, вы часто пользуетесь такой возможностью, как «прикрепление файла к письму», например, для того чтобы отправить письмо с фотографией, документом, архивом или еще каким-либо файлом. В этом разделе мы рассмотрим возможность отправки письма с прикрепленным файлом. Для выполнения поставленной задачи будем использовать встроенный пакет Oracle Forms – OLE2.

Пакет OLE2 был детально и неоднократно рассмотрен в предыдущих разделах, поэтому перейдем непосредственно к выполнению задачи.

1. Создайте новую форму и сохраните ее как MAIL_FILE.
2. Создайте небазовый блок данных и назовите его MAILBOX.
3. Создайте два текстовых элемента и одну кнопку (рис. 27.11). Для элементов текста и кнопок установите следующие свойства:

Элемент_текст 1:
Name – RECIPIENT
Data Type – Char
Maximum Length – 100
Database Item – No

Элемент_текст 2:
Name – attach_file
Data type – Char
Maximum Length – 300
Database item – No

Кнопка1:
Label – Отправить

Window:
Name – MAIL_BOX

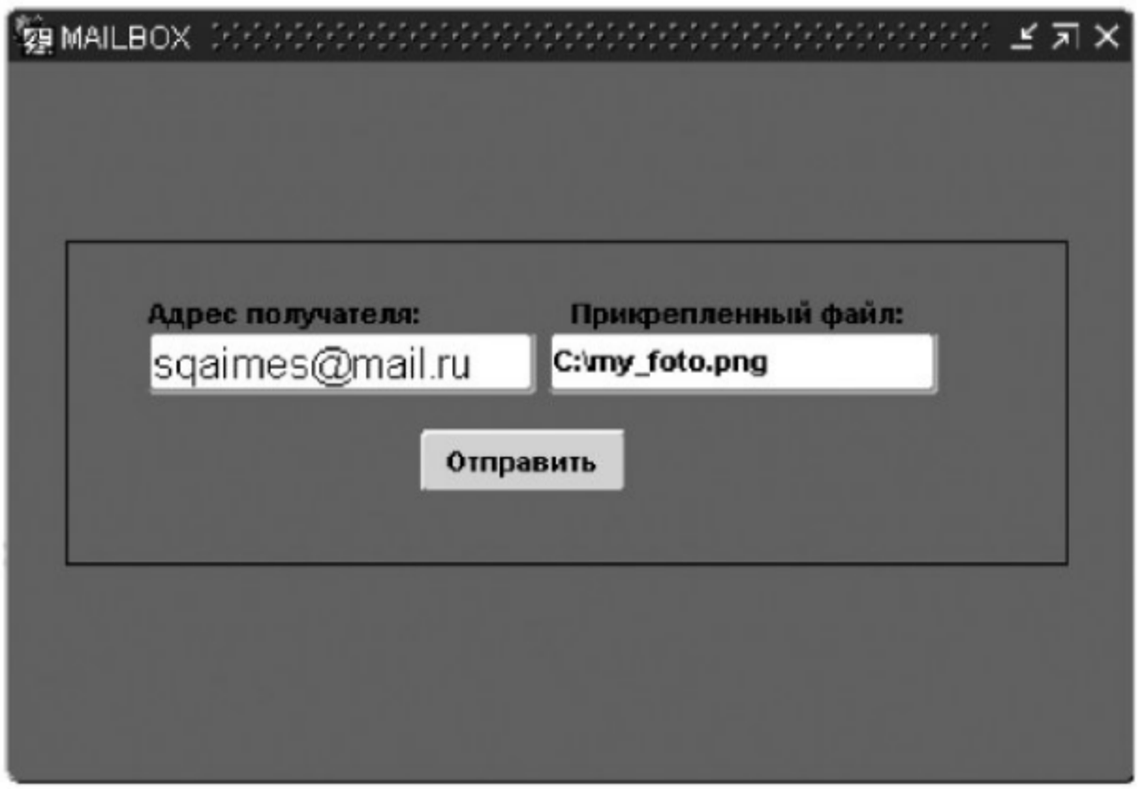

Рис. 27.11. Форма «MAILBOX»

4. Создайте процедуру с именем EMAIL. В теле процедуры напишите код, приведенный в листинге 27.1.

Листинг 27.1 Процедура EMAIL

```
PROCEDURE email(vmail varchar2,filename varchar2) IS
my_sqlerrm varchar2(200);
OutlookApp OLE2.OBJ_TYPE;
NameSpace OLE2.OBJ_TYPE;
MailItem OLE2.OBJ_TYPE;
OLEPARAM ole2.list_type;
Send OLE2.OBJ_TYPE;
Attachments OLE2.OBJ_TYPE;
Attachment_dummy OLE2.OBJ_TYPE;
v_to varchar2(50) := vmail ;
v_body varchar2(250);
v_subject varchar2(100):='PURCHASE ENQUIRY';
BEGIN
OutlookApp := OLE2.CREATE_OBJ('Outlook.Application');
OLEPARAM := OLE2.CREATE_ARGLIST;
OLE2.ADD_ARG(OLEPARAM,'MAPI');
NameSpace := OLE2.INVOKE_OBJ(OutlookApp,'GetNameSpace',OLEPARAM);
OLE2.DESTROY_ARGLIST( OLEPARAM );
OLEPARAM := OLE2.CREATE_ARGLIST;
```

```
OLE2.ADD_ARG(OLEPARAM,0);
MailItem := OLE2.INVOKE_OBJ(OutlookApp,'CreateItem',OLEPARAM);
OLE2.DESTROY_ARGLIST( OLEPARAM );
v_body:='HELLO! From Sergio Sergeenko V. (Sqaimes)';
ole2.set_property(MailItem,'To',v_to);
ole2.set_property(MailItem,'Subject',v_subject);
ole2.set_property(MailItem,'Body',v_body);
Attachments := OLE2.GET_OBJ_PROPERTY(MailItem,'Attachments');
-- Прикрепление файла
OLEPARAM := OLE2.CREATE_ARGLIST;
OLE2.ADD_ARG(OLEPARAM,filename);
Attachment_dummy := OLE2.INVOKE_OBJ(Attachments,'add',OLEPARAM);
OLE2.DESTROY_ARGLIST( OLEPARAM );
Send := OLE2.INVOKE_OBJ(Mailitem,'Send');
OLE2.RELEASE_OBJ( MailItem);
OLE2.RELEASE_OBJ( NameSpace );
OLE2.RELEASE_OBJ( OutlookApp );
EXCEPTION
   WHEN OTHERS THEN
   my_sqlerrm := SUBSTR(SQLERRM, 1, 150);
END;
```

5. Для кнопки «Кнопка1» создайте триггер WHEN_BUTTON_PRESSED. В теле триггера напишите код, приведенный в листинге 27.2.

Листинг 27.2 WHEN_BUTTON_PRESSED

```
/* проверим, чтобы поле «получатель» было заполнено*/
IF : recipient is not null and inst(:recipient,'@')>0 THEN
Email(:recipient, :attach_file);
END IF;
```

6. Скомпилируйте форму и запустите на выполнение. В поле recipient укажите почтовый адрес получателя, а в поле attach_file – имя файла с указанием пути. Нажмите кнопку «отправить».

Возможность отправки писем может существенно упростить вашу работу, если вы, к примеру, ведете учет сотрудников или клиентов и, соответственно, имеете в распоряжении их почтовые адреса. Расширив функциональность приведенного примера, вы сможете использовать процедуру EMAIL для рассылки однотипной информации одним нажатием кнопки.

Лекция 28. Файловый ввод/вывод в Oracle Forms

В этой лекции речь пойдет о пакете TEXT_IO, предназначенном для работы с файлами операционной системы. В лекции будут рассмотрены примеры чтения файла и запись в файл.

Ключевые слова: чтение файла, запись в файл, открытие файла, закрытие файла.

Цель лекции: научить пользователя встраивать в форму компоненту ActiveX и управлять ею посредствам библиотеки Shell API.

Файловый ввод/вывод в Oracle Forms

TEXT_IO – это пакет, который содержит набор процедур и функций для работы с файлами операционной системы. Нет смысла рассказывать о необходимости и полезности этого инструмента, поскольку потребность работать с файлами операционной системы была и будет всегда. Следует заметить, что TEXT_IO в отличие от пакетов VBX, OLE2, DDE поддерживается в последних версиях Forms, тогда как все вышеперечисленные закончили свое существование в Forms 6i. В этой главе мы рассмотрим различные примеры, которые продемонстрируют нам возможности записи и чтения данных.

Спецификация пакета

Пакет TEXT_IO включает в себя все необходимые методы (таблица 28.1) для работы с файлами операционной системы. Для обращения к процедурам или функциям этого пакета вы должны использовать слово «TEXT_IO».

Процедура FOpen может открывать файл в трех режимах:

- R – открывает файл только для чтения;
- W – открывает файл для чтения и записи, удаляя при этом существующие строки;
- A – открывает файл для чтения и вставки данных, но при этом существующие данные не удаляются, то есть, по сути, происходит операция вставки.

Таблица 28.1. Методы пакета TEXT_IO

Метод	Синтаксис	Назначение
TEXT_IO.Fclose	Fclose (file file_type);	Закрывает файл
TEXT_IO.File_Type	TYPE Text_IO.File_Type;	Тип данных для объявления переменной управления файлом (FILE_TYPE)
TEXT_IO.FOpen	Fopen (spec VARCHAR2, filemode VARCHAR2) RETURN Text_IO.File_Type	Открывает файл
TEXT_IO.Is_Open	Is_Open (file file_type) RETURN BOOLEAN;	Проверяет, открыт ли текущий файл
TEXT_IO.Get_Line	Get_Line (file file_type, item OUT VARCHAR2);	Чтение данных из файла. Возвращает следующую строку открытого файла
TEXT_IO.New_Line	New_Line (file file_type, n PLS_INTEGER := 1);	Создает новую строку
TEXT_IO.Put	Put (file file_type, item VARCHAR2);	Записывает строку в файл
TEXT_IO.PutF	Putf (arg VARCHAR2); (file file_type, arg VARCHAR2); (file file_type, format VARCHAR2, [arg1 [,..., arg5] VARCHAR2]); (format VARCHAR2, [arg1 [,..., arg5] VARCHAR2]);	Записывает строку в файл каждый раз начиная с новой строки
TEXT_IO.Put_Line	Put_Line (file file_type, item VARCHAR2);	Записывает строку в файл

Запись данных в файл

Для записи данных в файл необходимо придерживаться такой последовательности:

1. Создать ссылку на файл, которая в дальнейшем будет указателем на файл при обращении к нему в других методах.
2. Открыть файл для замены или для добавления текста.
3. Записать данные в файл.
4. Закрыть файл, вызвав Fclose.

Для того чтобы убедиться в вышесказанном, выполним пример, который покажет, как записать в Excel данные из формы:

```
WHEN_BUTTON_PRESSED
declare
   out_file Text_IO.File_Type;
   flnm varchar2(200);

   begin
   flnm := GET_FILE_NAME('H:\', 'file_name.xls', 'XLS Files
   (*.xls)|*.xls|', NULL, SAVE_FILE, TRUE);

   out_file:=Text_IO.Fopen(flnm, 'w');
   LOOP
   Text_IO.New_Line(out_file);
   Text_IO.Put(out_file, :ITEM1);
   Text_IO.Put(out_file, CHR(9));
   Text_IO.Put(out_file, :ITEM);
   IF :system.last_record = 'TRUE' THEN
   EXIT;
   ELSE
   next_record;
   END IF;
   Text_IO.Fclose (out_file);
   end;
```

Чтение данных из файла

Для чтения данных из файла придерживайтесь такой последовательности:

1. Создать ссылку на файл, которая в дальнейшем будет указателем на файл при обращении к нему в других методах.
2. Создать буфер для хранения строки данных из файла.
3. Открыть файл в режиме чтения.
4. Прочесть данные из буфера (Get_Line).
5. Закрыть файл.

Выполните небольшой пример, который наглядно продемонстрирует вам вышесказанное.

1. Создайте файл os_file.txt.
2. Создайте кнопку и определите для нее триггер WHEN-BUTTON-PRESSED. В теле триггера напишите следующий код:

```
WHEN_BUTTON_PRESSED
Declare
      file1 Text_io.file_type;
      z varchar2(2000);
       Begin
         file1 := text_io.fopen('c:\r.txt','r');
           text_io.get_line(file1,z);
             message(z);
              text_io.fclose(file1);
exception
when no_data_found then
text_io.fclose(file1);
end;
```

Если вы хотите прочесть все данные, находящиеся в файле, то выполните команду Get_Line цикле.

Справочник встроенных подпрограмм и функций ORACLE FORMS DEVELOPER

ABORT_QUERY – закрывает запрос, который был открыт в текущем блоке. Enter Query Mode yes

ACTIVATE_SERVER – активизирует OLE-сервер, ассоциированный с вашим OLE-контейнером.

Принимает один из параметров:

item_id специфицирует уникальный идентификатор, который Form Builder связывает с элементом при создании. Для нахождения идентификатора элемента пользуйтесь функцией FIND_ITEM. Тип данных – ID элемента.

item_name принимает значение – имя объекта, созданного на этапе проектирования, то есть в этом случае вам достаточно просто указать в качестве параметра имя элемента, для которого активизируется OLE-сервер. Тип данных параметра – VARCHAR2.

Примечание: *актуально только для Windows и Macintosh.*

ADD_GROUP_COLUMN – добавляет колонку (столбец) к группе записей.

Возвращает – GroupColumn.

Принимает следующие параметры, причем из двух параметров recordgroup_name и recordgroup_id вы выбираете один:

recordgroup_id – уникальный идентификатор, который Forms Builder связывает с группой записи при создании. Функция поиска идентификатора – FIND_RECORD_GROUP. Тип данных – ID группы записей.

recordgroup_name – имя группы записей. Тип данных – VARCHAR2.

column_type – специфицирует тип созданной колонки. Принимает следующие типы:

- CHAR_COLUMN – колонка, принимающая тип данных VARCHAR2
- DATE_COLUMN – колонка, принимающая тип данных DATE
- LONG_COLUMN – колонка, принимающая тип данных LONG
- NUMBER_COLUMN – колонка, принимающая тип данных NUMBER

column_width – этот параметр указывает на ширину столбца, является обязательным только в случае, если тип создаваемого вами столбца – CHAR_COLUMN, причем максимальный размер 2000.

Примечание*: неприменима к статическим группам записей.*

ADD_GROUP_ROW – добавляет ячейку к существующей группе записей.

Принимает следующие параметры:

recordgroup_id – уникальный идентификатор, который Forms Builder связывает с группой записи при создании. Функция поиска идентификатора – FIND_RECORD_GROUP. Тип данных идентификатора – RECORDGROUP группы записей.

recordgroup_name – имя группы записей. Тип данных – VARCHAR2.

row_number – номер, под которым будет ячейка добавлена в группу записей. Если вы хотите добавить ячейку в конец группы, то вам достаточно воспользоваться константой **END_OF_GROUP**.

ADD_LIST_ELEMENT – добавляет новый элемент в элемент списка LIST ITEM.

Принимает следующие параметры:

list_id – уникальный идентификатор, который Forms Builder связывает с элементом списка при создании. Функция поиска идентификатора – FIND_ITEM. Тип данных идентификатора – ITEM.

list_name – имя элемента списка. Тип данных – VARCHAR2.

list_index – специфицирует значение индекса элемента списка, то есть номер, под которым будет добавлен элемент в список элементов. Начинается с 1.

list_label – определяет то, как, точнее, под каким именем (меткой) будет отображаться элемент в списке элементов LIST ITEM. Тип данных – VARCHAR2.

list_value – определяет значение, которое будет содержать элемент списка.

ADD_OLEARGS – определяет тип и значение аргумента, который будет передан методу OLE объекта.

Newvar – значение этого аргумента. Это типы : NUMBER,VARCHAR2 или OLEVAR, если это типы FORMS или PL/SQL-данных.

Vtype – тип аргумента, понимаемый OLE-методом.

Для типа NUMBER аргумент по умолчанию – VT_TYPE :=VT_R8.

Для типа VARCHAR2 аргумент по умолчанию – VT_TYPE :=VT_BSTR.

Для типа OLEVAR аргумент по умолчанию – VT_TYPE :=VT_VARIANT.

ADD_PARAMETER – добавляет параметр в лист параметров.

Принимает следующие параметры:

[list] or [name] – определяет список параметров (лист параметров), с которым будет ассоциирован данный параметр. Принимает либо

идентификатор списка (тип PARAMLIST) параметров, либо его имя (тип VARCHAR2).

Key – Имя параметра. Имеет тип VARCHAR2.

Paramtype – определяет один из двух типов параметров:

- TEXT_PARAMETER A VARCHAR2 – строковый литерал.
- DATA_PARAMETER A VARCHAR2 – принимает в качестве значения имя группы записей текущей формы. Тип данных также VARCHAR2.

Value – значение, которые вы собираетесь передавать в вызываемый модуль. Если это текстовый (TEXT_PARAMETER) параметр, то максимальная длина – 64K символа. Тип данных – VARCHAR2.

ADD_TREE_DATA – добавляет набор данных под соответствующий узел (node) элемента TREE VIEW.

Параметры:

Iitem_name – определяет имя элемента, в данном случае, в качестве имени нужно указать block_name.tree_name типа VARCHAR2.

Item_id – уникальный идентификатор, который Forms Builder связывает с элементом списка при создании. Используйте также FIND_ITEM для поиска данного идентификатора.

Node – специфицирует элемент дерева.

Offset_type – специфицирует тип сдвига вершины. Возможные значения:

- PARENT_OFFSET – в этом случае добавляется так называемый «поднабор» данных под специфичным узлом, который размещен среди дочерних, идентифицируемых как сдвиги.
- SIBLING_OFFSET – добавляет новые данные к специфичному узлу.

Offset – идентифицирует позицию нового узла.

Если offset_type – PARENT_OFFSET, то сдвигом может быть каждый 1-n или LAST_CHILD.

Если offset_type – SIBLING_OFFSET, то сдвигом может быть NEXT_NODE или PREVIOUS_NODE.

Data_source – определяет тип источника данных. Возможные значения:

- RECORD_GROUP —указывает имя группы.
- QUERY_TEXT – здесь имеется в виду текст запроса.

ADD_TREE_NODE – добавляет элемент данных в TREE VIEW.

Параметры:

Iitem_name – определяет имя элемента, в данном случае в качестве имени нужно указать block_name.tree_name типа VARCHAR2.

Item_id – уникальный идентификатор, который Forms Builder связывает с элементом списка при создании. Используйте также FIND_ITEM для поиска данного идентификатора.
Node – специфицирует элемент дерева
Offset_type – специфицирует тип сдвига вершины. Возможные значения:
– PARENT_OFFSET – в этом случае добавляется так называемый «поднабор» данных под специфичным узлом, который размещен среди дочерних, идентифицируемых как сдвиги.
– SIBLING_OFFSET – добавляет новые данные к специфичному узлу.
Offset – идентифицирует позицию нового узла.
Если offset_type – PARENT_OFFSET, то сдвигом может быть каждый 1-n или LAST_CHILD.
Если offset_type – SIBLING_OFFSET, то сдвигом может быть NEXT_NODE или PREVIOUS_NODE.
State – определяет состояние узла. Возможные значения:
COLLAPSED_NODE – все ветки свернуты.
EXPANDED_NODE – развернутые ветки.
LEAF_NODE – развернута только текущая ветка.
label – отображает текст узла.
icon – имя файла, содержащего иконку для отображения слева от элемента дерева.
Value – определяет значения узла.

APPLICATION_PARAMETER – отображает все параметры, соответствующие текущему меню, и их текущее значение.

BELL – заставляет терминал выдавать звуковой сигнал в моменты изменения внутреннего состояния формы.

BLOCK_MENU – выводит меню блока и принимает операторский ввод. Если оператор делает правильный выбор, BLOCK_MENU передает управление в соответствующий блок.

BREAK – если текущая форма запущена в режиме отладки, BREAK останавливает выполнение формы, и выводит меню Debug Mode Options, если ничего не делает.

CALL_FORM – запускает указанную форму, оставляя порождающую форму активной. Forms запускает вызываемую форму с теми же опциями SQL*Forms(Run Form), что и у порождающей формы. Когда вызываемая

форма завершается – по функции EXIT или в результате неправильной передачи управления – обработка возвращается в порождающую форму в точку, откуда исходил вызов CALL. По умолчанию CALL использует параметры HIDE и NO_REPLACE. Параметр HIDE заставляет SQL*Forms стирать вызывающую форму с экрана перед рисованием вызываемой формы. Если вы используете параметр NO_HIDE, Forms не стирает вызывающую форму перед рисованием вызываемой формы. Это означает, что если вы вызываете форму с параметром NO_HIDE и некоторая страница вызываемой формы меньше области экрана, вызывающая форма выдается фоном. Параметр NO_REPLACE заставляет SQL*Forms поддерживать характеристику Default Menu Application вызывающей формы. Если вы используете параметр REPLACE, SQL*Forms заменяет характеристику Default Menu Application вызывающей формы на характеристику Default Menu Application вызываемой формы. Например: CALL (‘Firm.Firm_Name , HIDE , DO_REPLACE).

CALL_INPUT – используется лишь для совместимостями с прежними версиями. В новых приложениях желательно не использовать.

CALL_OLE – передает управление OLE-объекту. Принимаемые параметры:

obj – имя OLE объекта. Тип OLEOBJ.

Memberid – идентификатор метода запуска. Тип PLS_INTEGER. *Возможные значения:* FORM_OLE_FAILURE, LAST_OLE_EXCEPTION.

CALL_OLE_<returntype> – передает управление OLE-объекту, только в отличие от предыдущего случая здесь можно добавить возвращаемый тип:

```
FUNCTION CALL_OLE_CHAR
(obj OLEOBJ, memberid PLS_INTEGER)
RETURN returnval VARCHAR2;
...или...
FUNCTION CALL_OLE_NUM
(obj OLEOBJ, memberid PLS_INTEGER)
RETURN returnval NUMBER;
...или...
FUNCTION CALL_OLE_OBJ
(obj OLEOBJ, memberid PLS_INTEGER)
RETURN returnval OLEOBJ;
...или...
FUNCTION CALL_OLE_VAR
(obj OLEOBJ, memberid PLS_INTEGER)
RETURN returnval OLEVAR;
```

Принимаемые параметры:

obj – имя OLE объекта. Тип OLEOBJ.

Memberid – идентификатор метода запуска. Тип PLS_INTEGER. Возможные значения: FORM_OLE_FAILURE, LAST_OLE_EXCEPTION.

CANCEL_REPORT_OBJECT – отменяет долгую загрузку асинхронного отчета. Вы можете отменять загрузку отчета, используя REPORT_OBJECT_STATUS.

Параметры:

report_id – значение типа VARCHAR2, возвращаемое функцией RUN_REPORT_OBJECT, однозначно идентифицирует текущий запущенный отчет как на локали, так и на сервере.

Примечание: *вы не можете прервать загрузку отчета, если он у вас установлен в режим SYNCHRNOUS.*

CHECKBOX_CHECKED – при обращении к этой функции возвращается значение типа BOOLEAN, которое показывает состояние, заданное CHECK BOX. Если элемент, указанный в функции CHECKBOX_CHEKED, не CHECK BOX, тогда Forms вернет ошибку:

```
FRM-41038: Item <item_name> is not a check box.
```

Параметры:

item_id – уникальный идентификатор, который Forms Builder связывает с элементом при создании. Тип данных идентификатора – ITEM.

item_name – имя элемента, которое вы присвоили элементу на этапе проектирования. Тип данных – VARCHAR2.

Примечание: *предварительно проверяйте свойство элемента функцией GET_ITEM_PROPERTY.*

CHECK_RECORD_UNIQUENESS – когда срабатывает триггер On-Check-Unique, инициализируется проверка по умолчанию уникальности первичного ключа записи.

CLEAR_BLOCK – очищает текущий блок. Если в этом блоке есть изменения, которые не были посланы на сохранение (posted) или сохранены (commited), то поведение CLEAR_BLOCK зависит от параметров, которые вы укажете:

- **нет параметров** – Forms предложит оператору сохранить изменения при работе с CLEAR_BLOCK.
- ASK_COMMIT – Forms предложит оператору сохранить изменения при работе с CLEAR_BLOCK.
- DO_COMMIT – Forms протестирует изменения, выполнит сохранение и очистит текущий блок без вопроса оператору.

– NO_COMMIT – Forms протестирует изменения и очистит текущий блок без выполнения сохранения или вопроса оператору.
– NO_VALIDATE – Forms очистит текущий блок без тестирования изменений и их сохранения или вопроса оператору.

CLEAR_EOL – очищает текущее значение поля с текущей позиции курсора до конца строки или поля.

CLEAR_FORM – отменяет все изменения, которые не были сохранены, когда текущая форма обрабатывалась во время текущей сессии SQL*Forms(Run Form). Если текущая форма вызывала другие формы, которые содержали посланные на сохранение изменения, CLEAR_FORM отвергает и эти изменения.

Примечание: *если вы используете оператор ROLLBACK из PL/SQL в анонимном блоке или процедуре уровня формы, Forms интерпретирует его как упакованную процедуру CLEAR_FORM без параметров.*

CLEAR_ITEM – очищает значение текстового элемента, устанавливая его в NULL.

CLEAR_LIST – очищает элемент списка (list item). После того как Form Builder отчистит элемент списка, он будет содержать один «null list element».

Параметры (один из перечисленных):

list_id – уникальный идентификатор, который Forms Builder связывает с элементом списка при создании. Тип данных идентификатора – ITEM.

list_name – имя элемента, которое вы присвоили элементу списка на этапе проектирования. Тип данных – VARCHAR2.

CLEAR_MESSAGE – удаляет текущее сообщение с области экрана.

CLEAR_RECORD – очищает текущую запись в блоке. Если запрос открыт в этом блоке, то Forms вызывает запись для повторного заполнения блока.

CLOSE_FORM – в многомодульном приложении закрывает указанную форму. В случае если закрывается текущая форма, то действие производится аналогично EXIT_FORM.

Параметры:

form_name – специфицирует имя закрываемой формы. Тип данных – VARCHAR2.

form_id – уникальный идентификатор, привязываемый динамически форме при инициализации в режиме выполнения. Для поиска идентификатора используйте FIND_FORM с соответствующим типом. Тип данных идентификатора – FORMMODULE.

Примечание: *вы не можете закрывать вызывающую форму. Например, если Form_A вызывает Form_B, тогда Form_B не может закрыть Form_A.*

CLOSE_SERVER – деактивирует OLE-сервер, привязанный к вашему OLE-контейнеру.

item_id – уникальный идентификатор, который Forms Builder связывает с элементом списка при создании.

item_name – имя элемента, которое вы присвоили элементу на этапе проектирования. Тип данных – VARCHAR2.

Примечание: *только для Windows и Macintosh.*

COMMIT_FORM – обновляет данные в БД в соответствии с данными в форме. Forms вначале тестирует форму. Если есть изменения, которые нужно отразить в БД, то для каждого блока формы Forms посылает необходимые удаления, вставки и корректировки в БД и сохраняет обновленные данные формы в БД. Когда процедура COMMIT завершается, Forms освобождает все блокировки таблиц и строк, сделанные оператором. Если вы послали какие-нибудь данные на сохранение в БД во время текущей сессии Forms (Run Form), COMMIT_FORM сохраняет и эти данные в БД.

Примечание: *если вы используете оператор COMMIT из PL/SQL в анонимном блоке или процедуре уровня формы, Forms интерпретирует его как упакованную процедуру COMMIT_FORM.*

COPY (value_string , target_field_name) – записывает указанное значение в поле. COPY используется, в частности, для записи значения в поле, на которое ссылаются по упакованной процедуре NAME_IN. Это свойство существует потому, что вы не можете использовать стандартный синтаксис PL/SQL для установки поля, к которому идет доступ по ссылке, равным определенному значению.

COPY_REGION – копирует выделенный регион текстового элемента или картинки с экрана в буфер.

COPY_REPORT_OBJECT_OUTPUT – копирует выходные данные из отчета в файл.

Параметры:

report_id – значение типа VARCHAR2, возвращаемое функцией RUN_REPORT_OBJECT, однозначно идентифицирует текущий запущенный отчет как на локали, так и на сервере.

output_file – имя выходного файла.

COUNT_QUERY – определяет число строк, которые запрос отыщет для текущего блока. Если в блоке есть изменения, которые нужно сохранять в БД, Forms подсказывает оператору об этом при обработке COUNT_QUERY.

CREATE_GROUP – создает незапросную группу с присвоенным именем. Созданная группа не имеет ни ячеек, ни столбцов. Все последующие действия по заполнению и добавлению выполняются с помощью встроенных процедур и функций: ADD_GROUP_COLUMN, ADD_GROUP_ROW и POPULATE_GROUP_WITH_QUERY.

Параметры:

recordgroup_name – имя группы записей. Тип данных – VARCHAR2.

Scope – определяет вид использования группы записей, то есть будет ли группа глобальна – доступна всем формам, задействованным в приложении, – либо только текущей.

FORM_SCOPE – означает, что группа записей может быть использована только с текущей формой. Это значение является значением по умолчанию.

GLOBAL_SCOPE – означает, что группа записей будет глобальной и может быть использована всеми формами вашего многомодульного приложения. Причем однажды созданная глобальная группа записей будет действовать в течение всей сессии созданной runtime'ом формы.

array_fetch_size – определяет размер массива извлекаемых строк. По умолчанию этот параметр имеет значение 20.

CREATE_GROUP_FROM_QUERY – создает запросную группу с заданным именем. Созданная группа уже имеет столбцы, соответствующие названиям столбцам таблицы. Для добавления ячеек нужно использовать функцию POPULATE_GROUP.

Параметры:

recordgroup_name – имя группы записей.

Scope – определяет вид использования группы записей, то есть будет ли группа глобальна – доступна всем формам, задействованным в приложении, – либо только текущей.

FORM_SCOPE – означает, что группа записей может быть использована только с текущей формой. Это значение является значением по умолчанию.
GLOBAL_SCOPE – означает, что группа записей будет глобальной и может быть использована всеми формами вашего многомодульного приложения. Причем однажды созданная глобальная группа записей будет действовать в течение всей сессии созданной runtime'ом формы.
array_fetch_size – определяет размер массива извлекаемых строк. По умолчанию этот параметр имеет значение 20.

CREATE_OLEOBJ – создает OLE-объект, который даже в случае использования другой формы будет вызываться повторно, то есть не инициализируясь еще раз.
Name – программный идентификатор объектного OLE-сервера.
Localobject – указатель на объект OLE, статус которого выбран как настоящий или не настоящий.
persistence_boolean – значение типа boolean. Если подсоединен, то TRUE. По умолчанию определяется как подключенный. Это опциональный параметр. По умолчанию – TRUE.

CREATE_PARAMETER_LIST – создает лист параметров с заданным именем. После создания лист параметров не содержит никаких параметров. Параметры добавляются с помощью процедуры ADD_PARAMETER.
Парметры:
name – определяет имя листа параметров. После создания листа параметров Form Builder создаст уникальный идентификатор для типа PARAMLIST данного листа параметров. После этого вы можете вызывать лист параметров как по идентификатору, так и по имени.

CREATE_QUERIED_RECORD – когда вызывается триггер On-Fetch, создаются записи в листе ожидания блока. Лист ожидания имитирует буфер записей, состоящий из записей, которые должны будут быть извлечены из источника данных, но еще не извлечены как «активные» записи. Эта процедура в основном предназначена для случая использования транзакционных триггеров.

CREATE_RECORD – создает новую запись в текущем блоке после текущей записи. Затем Forms перейдет на эту новую запись.

CREATE_TIMER – создает новый таймер с заданным именем.
Параметры:
timer_name – определяет имя таймера, причем имя должно содержать не боле 30 символов. Тип данных – VARCHAR2.

Milliseconds – определяет действия таймера в миллисекундах. Предельное значение 2147483648. Тип данных — Number.
Iterate – определяет поведение таймера. Поведение определяется константами:
– REPEAT – указывает, что нужно повторять действие по истечении времени. Это значение стоит по умолчанию.
– NO_REPEAT – указывает, что действие нужно выполнить лишь однажды.

CREATE_VAR – создает неименованную переменную.
Бывает двух типов для скалярных величин и массивов:

```
FUNCTION CREATE_VAR
(persistence BOOLEAN)
RETURN newvar OLEVAR;
...or...
FUNCTION CREATE_VAR
(bounds OLE_SAFEARRAYBOUNDS,
vtype VT_TYPE,
persistence BOOLEAN)
RETURN newvar OLEVAR;
```

Параметры:
persistence – состояние переменной после создания. True, если существует.
Bounds – PL/SQL-таблица, в которой определены измерения, присваиваемые создаваемому массиву.
vtype – тип OLE variant (VT_TYPE) элементов в созданном массиве. Если массив состоит из смешанных типов, используйте тип VT_VARIANT.

CUT_REGION – удаляет выделенный регион текстового элемента или картинки, запоминая вырезанную часть в буфере.

DBMS_ERROR_CODE – возвращает номер последней ошибки базы данных.

DBMS_ERROR_TEXT – возвращает номер сообщения (типа ORA-14041) и текст ошибки БД.

DEBUG_MODE – переводит форму в режим отладки.

DEFAULT_VALUE – копирует указанное значение в указанную переменную, если текущее значение переменной NULL, иначе DEFAULT_VALUE не

делает ничего. (Поэтому для полей эта упакованная процедура работает идентично упакованной процедуре COPY для поля NULL.) Если переменная является неопределенной глобальной переменной, Forms создает эту переменную.

Параметры:

value_string – значение для копирования в переменную. Тип данных – VARCHAR2.

variable_name – имя переменной, в которую будем копировать значение. Тип данных – VARCHAR2.

Замечание: *упакованная процедура DEFAULT_VALUE не выполняет функции, аналогичные характеристике поля Default Value.*

DELETE_GROUP – удаляет выбранную группу записей.

Параметры:

recordgroup_id – уникальный идентификатор, который Forms Builder связывает с группой записи при создании. Функция поиска идентификатора – FIND_RECORD_GROUP. Тип данных идентификатора – RECORDGROUP группы записей.

recordgroup_name – имя группы записей. Тип данных – VARCHAR2.

DELETE_GROUP_ROW – удаляет записи в группе.

Параметры:

recordgroup_id – уникальный идентификатор, который Forms Builder связывает с группой записи при создании. Функция поиска идентификатора – FIND_RECORD_GROUP. Тип данных идентификатора – RECORDGROUP группы записей. Или имя группы записей, тогда тип данных VARCHAR2.

row_number – определяет номер удаляемой ячейки, соответствующий номеру ячейки в группе. Тип данных — NUMBER. ALL_ROWS – константа, которую можно использовать для удаления сразу всех записей.

DELETE_LIST_ELEMENT – удаляет элемент из элемента списка.

Парметры:

list_id – уникальный идентификатор, который Forms Builder связывает с элементом списка при создании. Функция поиска идентификатора – FIND_ITEM. Тип данных идентификатора – ITEM.

list_name – имя элемента списка. Тип данных – VARCHAR2.

list_index – специфицирует значение индекса элемента списка, то есть номер, под которым будет добавлен элемент в список элементов. Начинается с 1.

DELETE_PARAMETER – удаляет параметр и присвоенный ему ключ из списка параметров.

Параметры:
list or name – Specifies the parameter list, either by list ID or name. The actual parameter can be either a parameter list ID of type PARAMLIST, or the VARCHAR2 name of the parameter list.
Key – имя параметра. Тип данных ключа – VARCHAR2.

DELETE_RECORD – удаляет текущую запись из блока. Если запись соответствует строке в БД, Forms блокирует запись перед удалением ее и помечает ее как удаленную. Если в блоке открыт запрос, Forms вызывает запись для повторного заполнения блока при необходимости.

DELETE_TIMER – удаляет таймер.
Параметры:
timer_id – уникальный идентификатор, который Forms Builder связывает с таймером при его создании.
timer_name – имя элемента, которое вы присвоили элементу на этапе проектирования. Тип данных – VARCHAR2.

DELETE_TREE_NODE – удаляет элемент данных (узел) из дерева.
Параметры:
Iitem_name – определяет имя элемента, в данном случае в качестве имени нужно указать block_name.tree_name типа VARCHAR2.
Item_id – уникальный идентификатор, который Forms Builder связывает с элементом списка при создании. Используйте также FIND_ ITEM для поиска данного идентификатора.
Node – специфицирует элемент дерева.

DESTROY_PARAMETER_LIST – удаляет динамически созданный лист параметров, а также параметры, который он содержал.
Параметры:
name – определяет имя листа параметров. После создания листа параметров Form Builder создаст уникальный идентификатор для типа PARAMLIST данного листа параметров. После этого вы можете вызывать лист параметров как по идентификатору, так и по имени.

DESTROY_VARIANT – удаляет OLE-переменную, созданную функцией CREATE_VAR.
Параметры:
variant – OLE-переменная, которая должна быть удалена.

DISPLAY_ERROR – выводит экран Display Error, если существует зарегистрированная ошибка. Когда оператор нажимает любую функциональную клавишу, Forms перерисовывает форму.

DISPLAY_ITEM – используется только для совместимости с предыдущими версиями. В новых приложениях принято использовать SET_ITEM_INSTANCE_PROPERTY. DISPLAY_ITEM – отображает элемент с заданным визуальным атрибутом.

Параметры:

item_id or item_name – специфицирует в качестве параметра либо имя элемента, либо его идентификатор.

attribute – название визуального атрибута, который будет назначен указанному элементу.

DOWN – передает управление на представление текущего поля в записи со следующим более старшим последовательным номером. Если необходимо, Forms вызывает на экран эту запись, которой передается управление. Если Forms должен создать запись, которой передается управление, DOWN перемещает курсор на первое вводимое поле в этой новой записи.

DO_KEY – выполняет действия какого-либо ключа или оператора.

Принимаемые параметры:

Built-in	**Key trigger**	**Associated Function Key**
BLOCK_MENU	Key-MENU	[Block Menu]
CLEAR_BLOCK	Key-CLRBLK	[Clear Block]
CLEAR_FORM	Key-CLRFRM	[Clear Form]
CLEAR_RECORD	Key-CLRREC	[Clear Record]
COMMIT_FORM	Key-COMMIT	[Commit]
COUNT_QUERY	Key-CQUERY	[Count Query Hits]
CREATE_RECORD	Key-CREREC	[Insert Record]
DELETE_RECORD	Key-DELREC	[Delete Record]
DOWN	Key-DOWN	[Down]
DUPLICATE_ITEM	Key-DUP-ITEM	[Duplicate Item]
DUPLICATE_RECORD	Key-DUPREC	[Duplicate Record]
EDIT_TEXTITEM	Key-EDIT	[Edit]
ENTER	Key-ENTER	[Enter]
ENTER_QUERY	Key-ENTQRY	[Enter Query]
EXECUTE_QUERY	Key-EXEQRY	[Execute Query]
EXIT_FORM	Key-EXIT	[Exit/Cancel]
HELP	Key-HELP	[Help]
LIST_VALUES	Key-LISTVAL	[List]

LOCK_RECORD	Key-UPDREC	[Lock Record]
NEXT_BLOCK	Key-NXTBLK	[Next Block]
NEXT_ITEM	Key-NEXT-ITEM	[Next Item]
NEXT_KEY	Key-NXTKEY	[Next Primary Key Fld]
NEXT_RECORD	Key-NXTREC	[Next Record]
NEXT_SET	Key-NXTSET	[Next Set of Records]
PREVIOUS_BLOCK	Key-PRVBLK	[Previous Block]
PREVIOUS_ITEM	Key-PREV-ITEM	[Previous Item]
PREVIOUS_RECORD	Key-PRVREC	[Previous Record]
PRINT	Key-PRIN	[Print]
SCROLL_DOWN	Key-SCRDOWN	[Scroll Down]
SCROLL_UP	Key-SCRUP	[Scroll Up]
UP	Key-UP	[Up]

DUPLICATE_ITEM – назначает текущему полю то же значение, что имеет это поле в предыдущей записи.

DUPLICATE_RECORD – копирует значение каждого поля в записи с предыдущим меньшим номером в соответствующие поля текущей записи. Текущая запись уже не должна соответствовать никакой строке в БД, иначе возникает ошибка.

EDIT_TEXTITEM – запускает редактор для текущего элемента текста, а также переводит формы в режим редактирования.

Параметры:

x – определяет координату «x» на экране относительно левого края элемента.

Y – определяет координату «y» на экране относительно левого края элемента.

Width – определяет ширину редактора, включая кнопки.

Height – определяет высоту редактора, включая кнопки.

ENFORCE_COLUMN_SECURITY – обеспечивает по умолчанию проверку столбца. Данная процедура ограничения действительна только в соответствующем триггере On-Column-Security.

ENTER – тестирует данные в текущем проверяемом модуле.

ENTER_QUERY [({ ALL_RECORDS [, FOR_UPDATE [, NOWAIT]] | FOR_UPDATE [, NOWAIT] })] Поведение ENTER_QUERY меняется в зависимости от используемых параметров:

– ENTER_QUERY – завершает работу с текущим блоком и переводит форму в режим Enter Query. Если в этом блоке есть изменения,

которые нужно сохранить, Forms предлагает оператору сделать это во время события ENTER_QUERY.

- ENTER_QUERY(ALL_RECORDS) – выполняет те же действия, что и ENTER_QUERY, за исключением того, что когда вызывается EXECUTE_QUERY, Forms вызывает все выбранные записи.
- ENTER_QUERY(FOR_UPDATE) – выполняет те же действия, что и ENTER_QUERY, за исключением того, что когда вызывается EXECUTE_QUERY, Forms пытается немедленно заблокировать все выбранные записи.
- ENTER_QUERY(ALL_RECORDS,FOR_UPDATE) – выполняет те же действия, что и ENTER_QUERY, за исключением того, что когда вызывается EXECUTE_QUERY, Forms пытается немедленно заблокировать все выбранные записи и вызывает их.

Примечание: *рекомендуется использовать параметры ALL_RECORDS и FOR_UPDATE очень осторожно. Вызов очень большого числа строк может потребовать большой задержки. Одновременная блокировка большого числа строк требует много ресурсов.*

ERASE (global_variable_name) – удаляет указанную глобальную переменную так, что она больше не существует, и освобождает память, связанную с глобальной переменной.

Параметры:

global_variable_name – имя глобальной переменной, предназначенной для удаления.

ERROR_CODE – возвращает код ошибки Form Builder.

ERROR_TEXT – возвращает текст ошибки Form Builder.

ERROR_TYPE – возвращает тип ошибки Form Builder. Тип ошибки – ORA или FRM.

EXECUTE_QUERY [({ ALL_RECORDS [, FOR_UPDATE [, NOWAIT]] | FOR_UPDATE [, NOWAIT] })] – меняется в зависимости от используемых параметров:

- EXECUTE_QUERY – завершает работу с текущим блоком, открывает запрос и выводит некоторое число выбранных записей. Если в этом блоке есть изменения, которые нужно сохранить, Forms предлагает оператору сделать это во время обработки EXECUTE_QUERY.
- EXECUTE_QUERY(ALL_RECORDS) – выполняет те же действия, что и EXECUTE_QUERY, за исключением того, что SQL*Forms вызывает все выбранные записи.

– EXECUTE_QUERY(FOR_UPDATE) – выполняет те же действия, что и EXECUTE_QUERY, за исключением того, что Forms пытается немедленно заблокировать все выбранные записи.
– EXECUTE_QUERY(ALL_RECORDS, FOR_UPDATE) – выполняет те же действия, что EXECUTE_QUERY, за исключением того, что Forms пытается немедленно заблокировать все выбранные записи и вызывает их.

Примечание: *рекомендуется использовать параметры ALL_RECORDS и FOR_UPDATE очень осторожно. Вызов очень большого числа строк может потребовать большой задержки. Одновременная блокировка большого числа строк требует много ресурсов.*

ERASE (global_variable_name) – удаляет указанную глобальную переменную так, что она больше не существует, и освобождает память, связанную с глобальной переменной.

Параметры:

global_variable_name – имя глобальной переменной, предназначенной для удаления.

ERROR_CODE – возвращает код ошибки Form Builder.

ERROR_TEXT – возвращает текст ошибки Form Builder.

ERROR_TYPE – возвращает тип ошибки Form Builder. Тип ошибки – ORA или FRM.

EXECUTE_QUERY [({ ALL_RECORDS [, FOR_UPDATE [, NOWAIT]] | FOR_UPDATE [, NOWAIT] })] – меняется в зависимости от используемых параметров:

– EXECUTE_QUERY – завершает работу с текущим блоком, открывает запрос и выводит некоторое число выбранных записей. Если в этом блоке есть изменения, которые нужно сохранить, Forms предлагает оператору сделать это во время обработки EXECUTE_QUERY.
– EXECUTE_QUERY(ALL_RECORDS) – выполняет те же действия, что и EXECUTE_QUERY, за исключением того, что SQL*Forms вызывает все выбранные записи.
– EXECUTE_QUERY(FOR_UPDATE) – выполняет те же действия, что и EXECUTE_QUERY, за исключением того, что Forms пытается немедленно заблокировать все выбранные записи.
– EXECUTE_QUERY(ALL_RECORDS, FOR_UPDATE) – выполняет те же действия, что EXECUTE_QUERY, за исключением того, что Forms пытается немедленно заблокировать все выбранные записи и вызывает их.

Примечание: *рекомендуется использовать параметры ALL_RECORDS и FOR_UPDATE очень осторожно. Вызов очень большого числа строк*

может потребовать большой задержки. Одновременная блокировка большого числа строк требует много ресурсов.
По умолчанию же, если вы напишите просто execute_query, Forms извлечет все записи из текущего блока

EXECUTE_TRIGGER – выполняет указанный триггер, обычно это именованный триггер, то есть триггер, созданный пользователем.

Параметры:

trigger_name – определяет имя пользовательского (именованного) триггера.

EXIT_FORM [(ASK_COMMIT | DO_COMMIT | NO_COMMIT | NO_VALIDATE)] – завершает функцию CALL_INPUT во время ее работы. Во всех других контекстах EXIT_FORM передает управление «за» форму. Если в текущей форме есть изменения, которые не были посланы на сохранение (posted) или сохранены (committed), то поведение EXIT_FORM зависит от параметров:

- **нет параметров** – Forms предложит оператору сохранить изменения при работе с EXIT_FORM.
- ASK_COMMIT – Forms предложит оператору сохранить изменения при работе с EXIT_FORM.
- DO_COMMIT – Forms протестирует изменения, выполнит сохранение изменений и выйдет из текущей формы без вопроса оператору.
- NO_COMMIT – Forms протестирует изменения и выйдет из текущей формы без выполнения сохранения или вопроса оператору.
- NO_VALIDATE – Forms выйдет из текущей формы без тестирования изменений и их сохранения или вопроса оператору.

FETCH_RECORDS – вызванная из триггера On-Fetch, инициирует по умолчанию процесс для извлечения набора данных, определяя его как выборку (select).

FIND_ALERT – это функция поиска идентификатора «Предупреждения» (Alert). Возвращает тип – Alert.

Параметры:

alert_name – имя «Предупреждения». Тип данных – VARCHAR2.

FIND_BLOCK – это функция поиска идентификатора «Блока» (Block). Возвращает тип Block.

Параметры:

block_name – имя «Блока». Тип данных – VARCHAR2.

FIND_CANVAS – это функция поиска идентификатора «Вид-картинки» (Canvas). Возвращает тип Canvas.

Параметры:

canvas_name – имя «Вид-картинки» (Canvas). Тип данных – VARCHAR2.

FIND_COLUMN – это функция поиска идентификатора «столбца» (Column) группы записей. Возвращает тип GroupColumn.

Параметры:

recordgroup_name.column_name – имя «столбца» (Column). Тип данных – VARCHAR2.

FIND_EDITOR – это функция поиска идентификатора «Редактора» (Editor). Возвращает тип Editor.

Параметры:

canvas_name – имя «Редактор» (Editor). Тип данных – VARCHAR2.

FIND_FORM – это функция поиска идентификатора «Формы» (Form). Возвращает тип FORMMODULE.

Парамеры:

canvas_name – имя модуля формы. Тип данных – VARCHAR2.

FIND_GROUP – это функция поиска идентификатора «Группы записей» (RecordGroup). Возвращает тип RecordGroup.

Параметры:

group_name – имя группы записей. Тип данных – VARCHAR2.

FIND_ITEM – это функция поиска идентификатора элемента (ITEM). Возвращает тип ITEM.

Параметры:

block_name.item_name – имя элемента (Item). Тип данных – VARCHAR2.

FIND_LOV – это функция поиска идентификатора «Списка значений» (LOV). Возвращает тип LOV.

Параметры:

LOV_name – имя «Списка значений» (LOV). Тип данных – VARCHAR2.

FIND_MENU_ITEM – это функция поиска идентификатора элемента меню (MenuItem). Возвращает тип MenuItem.

Параметры:

menu_name.menuitem_name – имя элемента меню (MenuItem). Тип данных – VARCHAR2.

FIND_RELATION – это функция поиска идентификатора «Связи» (Relation). Возвращает тип Relation.

Параметры:

relation_name – имя «Связи» (Relation). Тип данных – VARCHAR2.

FIND_REPORT_OBJECT – возвращает идентификатор указанного отчета. Возвращаемый тип – Report_Object.

Параметры:

report_name – имя отчета, идентификатор (ID) которого мы собираемся получить.

FIND_TAB_PAGE – это функция возвращает идентификатор вкладки (Tab) вложенной вид-картинки (Tab-Canvas). Возвращаемый тип – TAB_PAGE.

Параметры:

tab_page_name – уникальное имя страницы вкладки. Тип данных – VARCHAR2.

Примечание: *если вложенная вид-картинка имеет страницы с идентичными именами, то в таком случае нужно писать следующим образом: MY_TAB_CVS.TAB_PAGE_1.*

FIND_TIMER – это функция возвращает идентификатор таймера. Тип данных – Timer.

Параметры:

timer_name – принимает в качестве значения имя таймера. Тип данных – VARCHAR2.

FIND_TREE_NODE – это функция возвращает идентификатор узла дерева. Тип данных – Node.

Параметры:

item_name – специфицирует имя объекта, данное ему на этапе проектирования. Тип данных – VARCHAR2.

Item_id – уникальный идентификатор, который Forms Builder связывает с элементом при создании. Функция поиска идентификатора – FIND_ITEM.

search_string – строка поиска. Тип данных – VARCHAR2.

search_type – определяет тип поиска (по дочернему элементу или узлу). Возможные значения: FIND_NEXT, FIND_NEXT_CHILD. Тип данных – NUMBER.

search_by – определяет тип поиска – по метке или значению элемента. Соответственно, принимаемые значения – NODE_LABEL и NODE_VALUE. Тип данных – NUMBER.
search_root – определяет корневой узел в указанном дереве. Например: FTREE.ROOT_NODE.
start_point – определяет стартовую точку поиска, к примеру, для поиска с начала дерева – FTREE.ROOT_NODE.

FIND_VA – это функция поиска визуального атрибута блока или элемента. Возвращаемый тип – Visual_Attribute.
Параметры:
va_name – определяет имя визуального атрибута.

FIND_VIEW – возвращает идентификатор отображения (View) вид-картинки (Canvas). Тип данных – ViewPort.
viewcanvas_name – имя вид-картинки. Тип данных – Varchar2.

FIND_WINDOW – эта функция возвращает идентификатор окна. Тип данных – Window.
Параметры:
window_name – имя окна. Тип данных – Varchar2.

FIRST_RECORD – переходит к первой записи.

FORM_FAILURE – возвращает идентификатор результата выполнения действия (Action). Другими словами, она дает вам возможность узнать статус выполнения вашей процедуры или триггера:
FORM_FAILURE – возвращает BOOLEAN:
– успешно (succes) FALSE
– неуспешно (failure) TRUE
– фатальная ошибка (fatal) FALSE
Если никакой процедуры не выполнялось во время текущей сессии Forms(RunForm), FORM_FAILURE возвращает FALSE. Существует также процедура FORM_FATAL, которая возвращает true, в случае фатальной ошибки, в остальных случаях – FALSE.

FORM_FATAL – в Runtime-сессии тестирует последнюю процедуру в триггере, стоящую перед FORM_FATAL, на фатальную ошибку, и в случае если:
– успешно (success) FALSE
– неуспешно (failure) FALSE
– фатальная ошибка (fatal error) TRUE

FORM_SUCCESS – проверяет предыдущее событие на успешное выполнение. Если:

- успешно (success) TRUE
- неуспешно (failure) FALSE
- фатальная ошибка (fatal error) FALSE

FORMS_DDL – выполняет команды DDL (язык определения данных: commit, create...), DML (язык описания данных: insert, update...) и PL/SQL в RunTime сессии.

Параметры:

statement – любая строка, не превышающая 32K. Тип данных – Varchar2.

GENERATE_SEQUENCE_NUMBER – инициализирует процесс создания последовательности по умолчанию средствами Forms Builder. Действителен только в триггере On-Sequence-Number.

GET_APPLICATION_PROPERTY – возвращает информацию о текущем Form Builder приложении. Возвращаемый тип – Varchar2.

Параметры:

property – здесь вы указываете одну из нижеперечисленных констант, типа NUMBER:

- APPLICATION_INSTANCE – получить значение указателя на ссылку экземпляра.
- BUILTIN_DATE_FORMAT – возвращает текущее значение формата даты Builtin.
- CALLING_FORM – возвращает имя текущей формы.
- CONNECT_STRING – возвращает строку подключения к БД.
- CURRENT_FORM – возвращает имя файла .FMX текущей формы.
- CURRENT_FORM_NAME – возвращает имя текущей формы, указанное в поле Name.
- CURSOR_STYLE – возвращает style property текущей формы, опции: BUSY, CROSSHAIR, DEFAULT, HELP, and INSERTION.
- DATE_FORMAT_COMPATIBILITY_MODE – возвращает compatibility установки для этого свойства.
- DISPLAY_HEIGHT – возвращает высоту дисплея в координатной системе модуля формы.
- DISPLAY_WIDTH – возвращает ширину дисплея в координатной системе модуля формы.
- FLAG_USER_VALUE_TOO_LONG – возвращает текущее значение свойства ...если True или False, которые управляются транзакцией пользователя (возвращает True или False).

– OPERATING_SYSTEM – возвращает имя текущей операционной системы.
– PASSWORD – возвращает пароль текущего оператора.
– PLSQL_DATE_FORMAT – возвращает маску формата даты PL/SQL.
– SAVEPOINT_NAME – возвращает имя последней точки сохранения.
– TIMER_NAME – возвращает время 'most recently expired timer'.
– USER_INTERFACE – возвращает имя интерфейса пользователя.
– USER_NLS_CHARACTER_SET – символьные характеристики пользовательского NLS.
– USER_NLS_LANG – возвращает полное текущее значение переменной окружения user_nls.
– USER_NLS_LANGUAGE – название языка в user_nls.
– USER_NLS_TERRITORY – название территории в user_nls.
– USERNAME – возвращает логин текущего пользователя Forms'a.

GET_BLOCK_PROPERTY – возвращает информацию об указанном блоке. Тип возвращаемых данных – Varchar2.

Параметры:

block_id – уникальный идентификатор, который Forms Builder связывает с блоком при создании.

block_name – имя блока, присвоенное ему при создании. Тип данных – VARCHAR2.

Property – название свойства, информацию о котором мы хотим получить. Ниже перечислены возможные значения этого свойства:

– ALL_RECORDS – это свойство вернет либо True, в случае если свойство блока ALL_RECORDS (все записи) имеет значение True, или False, что означает, что записи в блоке извлекаются наборами.
– BLOCKSCROLLBAR_X_POS – это свойство возвращает значение координаты позиции полосы прокрутки по «X».
– BLOCKSCROLLBAR_Y_POS – это свойство возвращает значение координаты позиции полосы прокрутки по «Y».
– COLUMN_SECURITY Returns the VARCHAR2 value of TRUE i column security is set to Yes, and the VARCHAR2 string FALSE if it is set to No.
– COORDINATION_STATUS – для блока, который является детальным (подчиненым) в связке master-detail (мастер-деталь). Возвращает два значения, типа Varchar2:
– COORDINATED – если блок связан со всеми главными для себя блоками, с которыми у него существует связь.

– NON_COORDINATED – в случае, когда блок связан не со всеми мастер-блоками.
– FIRST_DETAIL_RELATION – возвращает имя первой связи в мастер-детали, с которой связан детальный блок. Возвращает NULL, если таковой не существует.
– FIRST_ITEM – возвращает имя первого элемента в указанном блоке.
– FIRST_MASTER_RELATION – возвращает имя первой связки, в которой указанный блок выступает как master. Возвращает NULL, если таковой не существует.
– INSERT_ALLOWED – возвращает TRUE, если вставка в указанном блоке разрешена, и False, если запрещена.
– KEY_MODE – возвращает значение ключа, соответствующее режиму KEY-MODE в Property Pallete блока. Возвращает : UNIQUE_KEY, UPDATEABLE_PRIMARY_KEY, или NON_ UPDATEABLE_PRIMARY_KEY.
– LAST_ITEM – возвращает имя последнего элемента в блоке.
– LAST_QUERY – возвращает SQL- предложения последнего выполненного запроса в указанном блоке.
– LOCKING_MODE – возвращает способ режима блокировки : IMMEDIATE, DELAYED.
– MAX_QUERY_TIME – возвращает максимальное время выполнения запроса.
– MAX_RECORDS_FETCHED – возвращает число, представляющее максимальное кол-во записей, которое может быть выбрано. Работает только в случае, если свойство Query All Records установлено в Yes.
– NAVIGATION_STYLE – возвращает значение, определяющее стиль навигации: SAME_RECORD (остаться на той же записи), CHANGE_RECORD (переход на запись) или CHANGE_BLOCK (переход на блок).
– NEXTBLOCK – возвращает имя следующего блока.
– NEXT_NAVIGATION_BLOCK – возвращает имя следующего блока, в который будет осуществлен переход при навигации. По умолчанию если это свойство в Property Pallete блока установлено в NULL, то следующим блока будет тот блок, который в Навигаторе объектов следует за текущим.
– OPTIMIZER_HINT – возвращает подсказку оптимизатора, использованную при конструировании запросов.
– ORDER_BY – возвращает фразу ORDER BY, указанную в одноименном свойстве блока.
– PRECOMPUTE_SUMMARIES[Under Construction]

– PREVIOUSBLOCK – возвращает имя предыдущего блока, в который будет осуществлен переход при навигации. По умолчанию если это свойство в Property Pallete блока установлено в NULL, то следующим блоком будет тот блок, который в Навигаторе объектов стоит перед текущим.
– QUERY_ALLOWED – возвращает TRUE, если запросы в блоке разрешены, и, соответственно, False – если запросы запрещены.
– QUERY_DATA_SOURCE_NAME – возвращает имя источника запроса блока.
– QUERY_DATA_SOURCE_TYPE – возвращает тип источника запроса.
– QUERY_HITS – возвращает значение, аналогичное COUNT_QUERY, то есть количество записей в таблице.
– RECORDS_DISPLAYED – возвращает значение, определяющее кол-во отображаемых записей в блоке.
– RECORDS_TO_FETCH – возвращает значение, которое определяет кол-во записей, подготовленных к выборке (см. QUERIED RECORDS), то есть набор записей, извлекаемый за раз.
– STATUS – возвращает значение, определяющее статус блока.
– TOP_RECORD – возвращает значение, определяющее кол-во видимых записей в указанном блоке.
– UPDATE_ALLOWED – возвращает TRUE, если обновление в указанном блоке разрешено, и False, если запрещено.
– UPDATE_CHANGED_COLUMNS – возвращает TRUE, если обновляются только те столбцы, которые обновляются непосредственно пользователем, и False – соответственно. Если изменяется один столбец, то Forms по умолчанию обновляет все колонки, принадлежащие блоку.

GET_CANVAS_PROPERTY – возвращает информацию об указанной вид-картинке. Тип возвращаемых данных – Varchar2.

Параметры:

canvas_id – уникальный идентификатор, который Forms Builder связывает с вид-картинкой (Canvas) при создании.

canvas_name – имя вид-картинки (Canvas), присвоенное ей при создании. Тип данных – VARCHAR2.

Property – название свойства, информацию о котором мы хотим получить. Ниже перечислены возможные значения этого свойства:

– BACKGROUND_COLOR – возвращает цвет заднего фона.
– FILL_PATTERN – возвращает палитру, использованную для «заливки» региона.
– FONT_NAME – возвращает имя шрифта.

– FONT_SIZE – возвращает размер шрифта.
– FONT_SPACING – возвращает значение кернинга (расстояния между символами).
– FONT_STYLE – возвращает стиль шрифта.
– FONT_WEIGHT – возвращает вес шрифта.
– FOREGROUND_COLOR – возвращает цвет переднего плана региона. В элементах текста этим свойством определяется текст в элементе.
– HEIGHT – возвращает высоту канваса.
– TAB_PAGE_X_OFFSET – возвращает значение, определяющее расстояние между левым краем вложенной вид-картинки и левым краем страницы вкладки. Значение будет возвращено в соответствии с принятой системой координат.
– TAB_PAGE_Y_OFFSET – возвращает значение, определяющее расстояние между верхним краем вложенной вид-картинки и верхним краем страницы вкладки. Значение будет возвращено в соответствии с принятой системой координат.
– TOPMOST_TAB_PAGE – возвращает имя текущей страницы вкладки.
– WHITE_ON_BLACK – белое на черном. True – белое на черном.
– VISUAL_ATTRIBUTE – возвращает имя используемого атрибута.

GET_FILE_NAME – отображает стандартный диалог открытия файла.

Параметры:

directory_name – определяет директорию, содержащую файл для открытия. По умолчанию это значение NULL, поэтому если значение так и остается NULL, то Forms по умолчанию в качестве пути выбирает последнюю посещенную директорию.

file_name – определяет имя файла, предназначенный для открытия. По умолчанию значение и этого параметра NULL.

file_filter – определяет маску для файлов, по умолчанию NULL, которому соответствует маска All Files (*.*)|*.*| .

message – определяет тип файла, который будет выделен. Значение по умолчанию – NULL.

dialog_type – определяет тип диалога :OPEN_FILE (открытие файла) или SAVE_FILE (сохранение файла). Значение по умолчанию – OPEN_FILE.

select_file – определяет выбор файла или директории. Значение по умолчанию, как у OPEN_FILE, так и у SAVE_FILE, – True.

GET_FORM_PROPERTY – возвращает информацию об указанной форме. Тип возвращаемых данных – Varchar2.

Параметры:

form_name – специфицирует имя закрываемой формы. Тип данных – VARCHAR2.

form_id – уникальный идентификатор, привязываемый динамически форме при инициализации в режиме выполнения. Для поиска идентификатора используйте FIND_FORM с соответствующим типом. Тип данных идентификатора – FORMMODULE.

Property – название свойства, информацию о котором мы хотим получить. Ниже перечислены возможные значения этого свойства:

– COORDINATE_SYSTEM – возвращает текущую систему координат.
– CURRENT_RECORD_ATTRIBUTE – возвращает имя визуального атрибута, который был использован для текущей ячейки.
– FILE_NAME – возвращает имя файла, под которым сохранен модуль формы.
– FIRST_BLOCK – возвращает имя блока с наименьшим номером последовательности в индикаторе формы.
– FIRST_NAVIGATION_BLOCK – возвращает имя блока, в который Forms Builder переместится при запуске.
– FORM_NAME – возвращает имя формы.
– INTERACTION_MODE – возвращает значение режима, в котором находится форма: BLOCKING или NONBLOCKING.
– ISOLATION_MODE – возвращает значение, показывающее уровень изоляции: READ_COMMITTED или SERIALIZABLE.
– LAST_BLOCK – возвращает имя последнего блока в форме, то есть имя блока, имеющего наибольший последовательный номер.
– MAX_QUERY_TIME – возвращает значение текущей установки этого свойства. Это значение показывает, спустя какое время прервется запрос.
– MAX_RECORDS_FETCHED – возвращает максимальное значение кол-ва извлекаемых записей за один раз (при условии, что свойство All Records установлено в Yes).
– SAVEPOINT_MODE – возвращает PROPERTY_ON или PROPERTY_OFF, то есть определяет, включена ли поддержка точек сохранения (Savepoint).

GET_GROUP_CHAR_CELL – возвращает значение строки группы записей, находящейся на пересечении ячейки и столбца.

Параметры:

groupcolumn_id – уникальный идентификатор, который Forms Builder связывает со столбцом группы записей при создании. Для поиска идентификатора используйте вышеприведенную функцию FIND_COLUMN.
groupcolumn_name – определяет имя группы записей. Причем при описании этот параметр необходимо привести к такому виду: recordgroup_name.groupcolumn_name.
row_number – определяет ячейку, из которой продублируется значение строки.

GET_GROUP_DATE_CELL – возвращает значение типа DATE для группы записей, идентифицированное номером ячейки соответствующего столбца. Строка является пересечением ячейки и столбца.

Параметры:

groupcolumn_id – уникальный идентификатор, который Forms Builder связывает со столбцом группы записей при создании. Для поиска идентификатора используйте вышеприведенную функцию FIND_COLUMN.
groupcolumn_name – определяет имя группы записей. Причем при описании этот параметр необходимо привести к такому виду: recordgroup_name.groupcolumn_name.
row_number – определяет ячейку, из которой продублируется значение строки.

GET_GROUP_NUMBER_CELL – возвращает значение типа NUMBER для группы записей, идентифицированное номером ячейки соответствующего столбца. Строка является пересечением ячейки и столбца.

Параметры:

groupcolumn_id – уникальный идентификатор, который Forms Builder связывает со столбцом группы записей при создании. Для поиска идентификатора используйте вышеприведенную функцию FIND_COLUMN.
groupcolumn_name – определяет имя группы записей. Причем при описании этот параметр необходимо привести к такому виду: recordgroup_name.groupcolumn_name.
row_number – определяет ячейку, из которой продублируется значение строки.

GET_GROUP_RECORD_NUMBER – возвращает номер первой же совпавшей записи, значение которой соответствует значению параметра функции cell_value. В случае неудачи будет возвращен «0».

Параметры:

groupcolumn_id – уникальный идентификатор, который Forms Builder связывает со столбцом группы записей при создании. Для поиска идентификатора используйте вышеприведенную функцию FIND_COLUMN.

groupcolumn_name – определяет имя группы записей. Причем при описании этот параметр необходимо привести к такому виду: recordgroup_name.groupcolumn_name.
cell_value – определяет значение записи, по которому мы хотим найти ее номер в группе. Соответственно значения трех типов: VARCHAR2, NUMBER или DATE.

GET_GROUP_ROW_COUNT – определяет кол-во записей в созданной вами группе.
Параметры:
recordgroup_id – уникальный идентификатор, который Forms Builder связывает с группой записи при создании. Функция поиска идентификатора – FIND_RECORD_GROUP. Тип данных идентификатора – RECORDGROUP группы записей. Или берется имя группы записей, тогда тип данных – VARCHAR2.
group_name – определяет имя группы записей. Тип данных – Varchar2.

GET_GROUP_SELECTION – возвращает последовательный номер выделенной ячейке в указанной группе.
Параметры:
recordgroup_id – уникальный идентификатор, который Forms Builder связывает с группой записи при создании. Функция поиска идентификатора – FIND_RECORD_GROUP. Тип данных идентификатора – RECORDGROUP группы записей. Или берется имя группы записей, тогда тип данных – VARCHAR2.
group_name – определяет имя группы записей. Тип данных – Varchar2.
selection_number – идентифицирует выделенные ячейки и сортирует их относительно выделения. Например, если вы пометили как выделенные ячейки с номерами: 10, 23 и 3, то после выделения они будут в следующем порядке: 1, 2 и 3, то есть последовательно.

GET_GROUP_SELECTION_COUNT – возвращает число, которое определяет кол-во строк, помеченных в группе как выделенные.
Параметры:
recordgroup_id – уникальный идентификатор, который Forms Builder связывает с группой записи при создании. Функция поиска идентификатора – FIND_RECORD_GROUP. Тип данных идентификатора – RECORDGROUP группы записей. Или берется имя группы записей, тогда тип данных – VARCHAR2.
group_name – определяет имя группы записей. Тип данных – Varchar2.

GET_INTERFACE_POINTER – возвращает ссылку (handle) на OLE2 авто-объект. Возвращаемый тип данных — PLS_INTEGER.

Параметры:

Iitem_name – определяет имя объекта. Тип данных – VARCHAR2.

Item_id – уникальный идентификатор, который Forms Builder связывает с элементом списка при создании. Используйте также FIND_ITEM для поиска данного идентификатора.

GET_ITEM_INSTANCE_PROPERTY – возвращает значение свойства указанного экземпляра элемента. Возвращает либо инициализированное (определенное на этапе проектирования значение), либо последнее значение, установленное процедурой SET_ITEM_INSTANCE_PROPERTY.

Параметры:

Iitem_name – определяет имя элемента. Тип данных – VARCHAR2.

Item_id – уникальный идентификатор, который Forms Builder связывает с элементом списка при создании. Используйте также FIND_ITEM для поиска данного идентификатора.

record_number – номер записи или CURRENT_RECORD.

Property – название свойства, информацию о котором мы хотим получить. Ниже перечислены возможные значения этого свойства:

- BORDER_BEVEL – возвращает стиль отображения углов : RAISED, LOWERED или PLAIN.
- BORDER_BEVEL – устанавливает свойство в одно из значений: RAISED, LOWERED или PLAIN. Если данное значение не установлено, то это свойство в качестве значения вернет « ».
- INSERT_ALLOWED – возвращает True в случае, если вставка разрешена, и False, если вставка новых записей запрещена.
- NAVIGABLE – возвращает True, если свойство установлено в соответствующее значение. Возвращает тип данных Varchar2. По умолчанию True – то есть навигация с клавиатуры разрешена.
- NAVIGABLE – свойство установлено в True, возвращает False, если свойство установлено в False.
- REQUIRED – возвращает True, если свойство REQUIRED установлено в True, иначе – False.
- SELECTED_RADIO_BUTTON – возвращает метку выделенной радиокнопки. Возвращает NULL, если радиогруппа для определенной для нее записи не имеет выделенной радиокнопки.
- UPDATE_ALLOWED – возвращает True в случае, если вставка разрешена, и False, если вставка новых записей запрещена.
- VISUAL_ATTRIBUTE – возвращает имя текущего атрибута, связанного с элементом. В случае если это свойство элемента не

определено, возвращается DEFAULT-атрибут, если же элемент не поддерживает такового, будет возвращена пустая строка « ».

GET_ITEM_PROPERTY – возвращает значение свойства заданного элемента.

Примечание: *не все перечисленные параметры можно устанавливать программно, некоторые из них возможно лишь получать.*

Параметры:

Iitem_name – определяет имя элемента. Тип данных – VARCHAR2.

Item_id – уникальный идентификатор, который Forms Builder связывает с элементом списка при создании. Используйте также FIND_ITEM для поиска данного идентификатора.

record_number – номер записи или CURRENT_RECORD.

Property – название свойства, информацию о котором мы хотим получить. Ниже перечислены возможные значения этого свойства:

- **AUTO_HINT** – возвращает True, если свойство установлено в соответствующее значение. Возвращает тип данных Varchar2. По умолчанию False – то есть автоматическая подсказка не разрешена.
- AUTO_SKIP – возвращает True, если свойство установлено в соответствующее значение. Возвращает тип данных Varchar2. По умолчанию False – то есть при заполнении последнего символа в элементе курсор не перемещается на следующий элемент.
- BACKGROUND_COLOR – цвет заднего региона объекта.
- BLOCK_NAME – возвращает имя блока, которому принадлежит элемент.
- BORDER_BEVEL – возвращает стиль отображения углов: RAISED, LOWERED или PLAIN.
- BORDER_BEVEL – устанавливает свойство в одно из значений: RAISED, LOWERED или PLAIN. Если данное значение не установлено, то это свойство в качестве значения вернет « ».
- CASE_INSENSITIVE_QUERY – возвращает TRUE, если запрос без учета регистра, и False – значение по умолчанию, если с учетом регистра.
- CASE_RESTRICTION – возвращает одно из трех значений: UPPERCASE, если текст отображается в символах верхнего регистра; LOWERCASE, если текст отображается в символах нижнего регистра, или NONE, если нет ограничений на регистр.
- COLUMN_NAME – возвращает имя столбца в БД, с которым ассоциирован элемент блока данных.
- COMPRESS – возвращает значение (TRUE или FALSE), которое идентифицирует, каким образом звуковой объект будет прочтен

в форме из файла, предварительно должным образом обжатого в соответствующий формат для Oracle.
- CONCEAL_DATA – возвращает TRUE, если оператор спрятал данные, и False – значение по умолчанию, если не спрятал.
- CURRENT_RECORD_ATTRIBUTE – возвращает имя текущего атрибута элемента.
- CURRENT_ROW_BACKGROUND_COLOR – текущий цвет заднего фона ячейки.
- CURRENT_ROW_FILL_PATTERN – тип заливки, использованной для заливки фона ячейки.
- CURRENT_ROW_FONT_NAME – имя текущего шрифта ячейки.
- CURRENT_ROW_FONT_SIZE – размер текущего шрифта ячейки.
- CURRENT_ROW_FONT_SPACING – возвращает значение кернинга (расстояния между символами).
- CURRENT_ROW_FONT_STYLE – стиль шрифта.
- CURRENT_ROW_FONT_WEIGHT – вес шрифта.
- CURRENT_ROW_FOREGROUND_COLOR – цвет для области переднего плана.
- CURRENT_ROW_WHITE_ON_BLACK – определяет, как отображается текст на монохромном дисплее – как белый цвет на черном фоне.
- DATABASE_VALUE – для базового элемента возвращает значение, однозначно выбранное из БД.
- DATATYPE – возвращает тип данных элемента: ALPHA, CHAR, DATE, JDATE, EDATE, DATETIME, INT, RINT, MONEY, RMONEY, NUMBER, RNUMBER, TIME, LONG, GRAPHICS или IMAGE.
- DIRECTION – возвращает в качестве значения порядок чтения: RIGHT_TO_LEFT, LEFT_TO_RIGHT или DEFAULT (значение по умолчанию).
- DISPLAYED – возвращает TRUE, если элемент отображается, и False – если не отображается.
- ECHO
- EDITOR_NAME
- EDITOR_X_POS
- EDITOR_Y_POS
- ENFORCE_KEY
- ENABLED
- FILL_PATTERN
- FIXED_LENGTH
- FONT_NAME
- FONT_SIZE

- FONT_SPACING
- FONT_STYLE
- FONT_WEIGHT
- FOREGROUND_COLOR
- FORMAT_MASK
- HINT_TEXT
- ICON_NAME
- ICONIC_BUTTON
- IMAGE_DEPTH
- IMAGE_FORMAT
- INSERT_ALLOWED
- INSERT_ALLOWED
- ITEM_CANVAS
- ITEM_IS_VALID
- ITEM_NAME
- ITEM_TAB_PAGE
- ITEM_TYPE
- JUSTIFICATION
- KEEP_POSITION
- LABEL
- LIST
- LOCK_RECORD_ON_CHANGE
- LOV_NAME
- LOV_X_POS
- LOV_Y_POS
- MAX_LENGTH
- MERGE_CURRENT_ROW_VA
- MERGE_TOOLTIP_ATTRIBUTE
- MERGE_VISUAL_ATTRIBUTE
- MOUSE_NAVIGATE
- MULTI_LINE
- NAVIGABLE
- NAVIGABLE
- NEXTITEM
- NEXT_NAVIGATION_ITEM
- POPUPMENU_CONTENT_ITEM
- POPUPMENU_COPY_ITEM
- POPUPMENU_CUT_ITEM
- POPUPMENU_DELOBJ_ITEM
- POPUPMENU_INSOBJ_ITEM
- POPUPMENU_LINKS_ITEM
- POPUPMENU_OBJECT_ITEM

- POPUPMENU_PASTE_ITEM
- POPUPEMNU_PASTESPEC_ITEM
- PREVIOUSITEM
- PREVIOUS_NAVIGATION_ITEM
- PRIMARY_KEY
- PROMPT_ALIGNMENT_OFFSET
- PROMPT_BACKGROUND_COLOR
- PROMPT_DISPLAY_STYLE
- PROMPT_EDGE
- PROMPT_EDGE_ALIGNMENT
- PROMPT_EDGE_OFFSET
- PROMPT_FILL_PATTERN
- PROMPT_FONT_NAME
- PROMPT_FONT_SIZE
- PROMPT_FONT_SPACING
- PROMPT_FONT_STYLE
- PROMPT_FONT_WEIGHT
- PROMPT_FOREGROUND_COLOR
- PROMPT_TEXT
- PROMPT_TEXT_ALIGNMENT
- PROMPT_VISUAL_ATTRIBUTE
- PROMPT_WHITE_ON_BLACK
- QUERYABLE
- QUERY_LENGTH
- QUERY_ONLY
- RANGE_HIGH
- RANGE_LOW
- REQUIRED
- SCROLLBAR
- SHOW_FAST_FORWARD_BUTTON
- SHOW_PALETTE
- SHOW_PLAY_BUTTON
- SHOW_RECORD_BUTTON
- SHOW_REWIND_BUTTON
- SHOW_SLIDER
- SHOW_TIME_INDICATOR
- SHOW_VOLUME_CONTROL
- TOOLTIP_BACKGROUND_COLOR
- TOOLTIP_FILL_PATTERN
- TOOLTIP_FONT_NAME
- TOOLTIP_FONT_SIZE
- TOOLTIP_FONT_SPACING

– TOOLTIP_FONT_STYLE
– TOOLTIP_FONT_WEIGHT
– TOOLTIP_FOREGROUND_COLOR T
– TOOLTIP_WHITE_ON_BLACK
– TOOLTIP_TEXT
– UPDATE_ALLOWED
– UPDATE_ALLOWED
– UPDATE_COLUMN
– UPDATE_NULL
– UPDATE_PERMISSION
– UPDATEABLE
– VALIDATE_FROM_LIST
– VISIBLE
– VISUAL_ATTRIBUTE
– WHITE_ON_BLACK
– WINDOW_HANDLE
– WRAP_STYLE
– X_POS
– Y_POS

GET_LIST_ELEMENT_COUNT – возвращает значение, определяющее кол-во элементов в элементе списка, включая элементы, содержащие NULL-значения.

Параметры:

list_id – уникальный идентификатор, который Forms Builder связывает с элементом списка при создании. Функция поиска идентификатора – FIND_ITEM. Тип данных идентификатора – ITEM.

list_name – имя элемента списка. Тип данных – VARCHAR2.

GET_LIST_ELEMENT_LABEL – возвращает информацию о метке элемента.

Параметры:

list_id – уникальный идентификатор, который Forms Builder связывает с элементом списка при создании. Функция поиска идентификатора – FIND_ITEM. Тип данных идентификатора – ITEM.

list_name – имя элемента списка. Тип данных – VARCHAR2.

list_index – специфицирует значение индекса элемента списка, то есть номер, под которым будет добавлен элемент в список элементов. Начинается с 1.

GET_LIST_ELEMENT_VALUE – возвращает значение элемента в элементе списка.

Параметры:

list_id – уникальный идентификатор, который Forms Builder связывает с элементом списка при создании. Функция поиска идентификатора – FIND_ITEM. Тип данных идентификатора – ITEM.

list_name – имя элемента списка. Тип данных – VARCHAR2.

list_index – специфицирует значение индекса элемента списка, то есть номер, под которым будет добавлен элемент в список элементов. Начинается с 1.

GET_LOV_PROPERTY – возвращает в качестве значения информацию о свойстве LOV.

Параметры:

lov_id – уникальный идентификатор, который Forms Builder связывает со списком значений (LOV) при создании. Функция поиска идентификатора – FIND_LOV. Тип данных идентификатора – LOV.

lov_name – имя списка значений. Тип данных – Varchar2.

Property – название свойства, информацию о котором мы хотим получить. Ниже перечислены возможные значения этого свойства:

- AUTO_REFRESH – возвращает TRUE, если автоматическое обновление включено (выполнять запрос для заполнения LOV повторно), если False, то, соответственно, автообновление отключено.
- GROUP_NAME – возвращает имя группы записей, на которой базируется LOV.
- HEIGHT – возвращает высоту LOV.
- WIDTH – возвращает ширину LOV.
- X_POS – возвращает позицию по «X» положения LOV на вид-картинке или дисплее.
- Y_POS – возвращает позицию по «Y» положения LOV на вид-картинке или дисплее.

GET_MENU_ITEM_PROPERTY – возвращает состояние элемента меню относительно рассматриваемого свойства.

Параметры:

menuitem_id – уникальный идентификатор, который Forms Builder связывает с элементом меню при создании. Функция поиска идентификатора – FIND_MENU_ITEM. Тип данных – MenuItem.

menu_name.menuitem_name – имя элемента меню (MenuItem). Тип данных – VARCHAR2.

Property – название свойства, информацию о котором мы хотим получить. Ниже перечислены возможные значения этого свойства:

- CHECKED – возвращает True, если check box элемента меню помечен (checked), и False, если не помечен (unchecked).

– ENABLED – возвращает True, если элемент меню доступен, и False, если недоступен.
– ICON_NAME – возвращает имя файла, который содержит иконку, ассоциированную с элементом.
– LABEL – возвращает строку метки элемента меню.
– VISIBLE – возвращает True, если элемент виден, False – если не виден.

GET_MESSAGE – возвращает текущее сообщение соответствующего типа.

GET_PARAMETER_ATTR – возвращает текущее значение и тип выбранного параметра в выбранном листе параметров (parameter list).

Параметры:

list or name – определяет либо идентификатор листа параметров (parameter list), либо имя списка параметров. Тип данных для списка параметров – PARAMLIST.

Key – имя параметра символьного типа.

Paramtype – тип выходного параметра. Типы параметров:

– DATA_PARAMETER
– TEXT_PARAMETER

value – значение выходного параметра; если это, к примеру, текстовый параметр, то значением параметра будет актуальное ему значение.

GET_PARAMETER_LIST – функция поиска идентификатора (ID) списка параметров. Тип возвращаемых данных – ParamList.

Параметры:

name – имя списка параметров. Тип данных – Varchar2.

GET_RADIO_BUTTON_PROPERTY – возвращает информацию об указанной радиокнопке.

Параметры:

Iitem_name – определяет имя элемента. Тип данных – VARCHAR2.

Item_id – уникальный идентификатор, который Forms Builder связывает с элементом списка при создании. Используйте также FIND_ITEM для поиска данного идентификатора.

button_name – определяет имя радиокнопки, свойства которой нас интересуют.

Property – название свойства, информацию о котором мы хотим получить. Ниже перечислены возможные значения этого свойства:

– BACKGROUND_COLOR – цвет заднего фона.

– ENABLED – возвращает True, если элемент меню доступен, и False, если недоступен.
– FILL_PATTERN – тип заливки, использованной для заполнения региона.
– FONT_NAME – название шрифта.
– FONT_SIZE – размер шрифта.
– FONT_SPACING – расстояние между буквами – кернинг.
– FONT_STYLE – стиль шрифта.

GET_RECORD_PROPERTY – возвращает значение свойства существующей записи, то есть номер записи принимаемый параметром функции record_number должен существовать.

Параметры:

record_number – определяет запись в блоке, для которой нужно получить информацию.

block_name – определяет блок, содержащий целевую запись.

Property – определяет свойство, значение которого мы хотим проверить:

NEW, CHANGED, QUERY.

GET_TAB_PAGE_PROPERTY – возвращает значение свойств для указанной вкладки.

tab_page_id – уникальный идентификатор, который Forms Builder связывает со страницей вкладки при создании. Функция поиска уникального идентификатора – FIND_TAB_PAGE. Тип данных – TAB_PAGE.

tab page_name – имя страницы вкладки.

Property – название свойства, информацию о котором мы хотим получить. Ниже перечислены возможные значения этого свойства:

– BACKGROUND_COLOR – цвет заднего фона.
– ENABLED – возвращает True, если элемент меню доступен, и False, если недоступен.
– FILL_PATTERN – тип заливки, использованной для заполнения региона.
– FONT_NAME – название шрифта.
– FONT_SIZE – размер шрифта.
– FONT_SPACING – расстояние между буквами – кернинг.
– FONT_STYLE – стиль шрифта.
– FONT_WEIGHT – вес шрифта.

GET_TREE_NODE_PARENT – возвращает родителя указанного узла.

Параметры:

item_name – специфицирует имя объекта, данное ему на этапе проектирования. Тип данных – VARCHAR2.
Item_id – уникальный идентификатор, который Forms Builder связывает с элементом при создании. Функция поиска идентификатора – FIND_ITEM.
Node – определяет узел.

GET_TREE_PROPERTY – возвращает значение свойства дерева.
Параметры:
item_name – специфицирует имя объекта, данное ему на этапе проектирования. Тип данных – VARCHAR2.
Item_id – уникальный идентификатор, который Forms Builder связывает с элементом при создании. Функция поиска идентификатора – FIND_ITEM.
Property – название свойства, информацию о котором мы хотим получить. Ниже перечислены возможные значения этого свойства:
- DATASOURCE – возвращает источник, использованный для заполнения дерева. Возвращает EXTERNAL, если свойство установлено каким-либо другим способом.
- RECORD_GROUP – возвращает наименование группы записей, использованной для заполнения иерархического дерева, причем текст запроса может быть создан как на этапе проектирования, так и с помощью SET_TREE_PROPERTY.
- QUERY_TEXT – возвращает текст запроса, использованного для заполнения иерархического дерева, причем текст запроса может быть создан как на этапе проектирования, так и с помощью SET_TREE_PROPERTY.
- NODE_COUNT – возвращает число ячеек источника иерархического дерева.
- SELECTION_COUNT – возвращает число выделенных ячеек.
- ALLOW_EMPTY_BRANCHES – разрешено ли существование пустых узлов.
- ALLOW_MULTI-SELECT – разрешено ли множественное выделение.

GET_TREE_SELECTION – возвращает узел данных, помеченный как выделенный, причем выделение отмечается как индекс в списке индексов выделенных элементов.

Параметры:
item_name – специфицирует имя объекта, данное ему на этапе проектирования. Тип данных – VARCHAR2.

Item_id – уникальный идентификатор, который Forms Builder связывает с элементом при создании. Функция поиска идентификатора – FIND_ITEM.
Selection – определяет выделение единичного узла.

GET_WINDOW_PROPERTY – возвращает текущие установки помеченного вами окна.

Примечание: *в Microsoft Windows вы можете связывать MDI приложения окна с помощью константы FORMS_MDI_WINDOW.*

Параметры:

window_id – уникальный идентификатор, который Forms Builder связывает с окном при создании. Функция поиска идентификатора – FIND_WINDOW.
window_name – имя окна, заданное на этапе проектирования.
Property – название свойства, информацию о котором мы хотим получить. Ниже перечислены возможные значения этого свойства:

- BACKGROUND_COLOR – цвет заднего фона.
- ENABLED – возвращает True, если элемент меню доступен, и False, если недоступен.
- FILL_PATTERN – тип заливки, использованной для заполнения региона.
- FONT_NAME – название шрифта.
- FONT_SIZE – размер шрифта.
- FONT_SPACING – расстояние между буквами – кернинг.
- HEIGHT – возвращает высоту окна.
- HIDE_ON_EXIT – скрывается ли окно при выходе.
- ICON_NAME – возвращает имя файла картинки, ассоциированной с окном в минимизированном режиме.
- TITLE – возвращает название окна
- VISIBLE – возвращает True, если окно видимо, и False, если не видимо.
- WHITE_ON_BLACK WHITE_ON_BLACK – белое на черном. True – белое на черном.
- WIDTH – возвращает ширину окна.
- WINDOW_HANDLE – возвращает уникальную константу, которая используется для привязки к объекту. Актуально только для платформ Windows, иначе возвращает «0».
- WIDOW_SIZE – возвращает высоту и ширину окна.
- WINDOW_STATE – возвращает текущее состояние отображения окна: NORMAL, MAXIMIZE или MINIMIZE.
- X_POS – возвращает позицию по «X» положения окна на дисплее.
- Y_POS – возвращает позицию по «Y» положения окна на дисплее.

GO_BLOCK (block_name) – переходит к указанному блоку. Тип данных – Varchar2.

Параметры:

block_name – имя блока, к которому нужно перейти.

GO_FORM (form_name) – переходит к указанной форме. Тип данных – Varchar2.

Параметры:

form_name – имя формы, к которой надо перейти.

GO_ITEM (item_name) – переходит к указанному элементу.

item_name – имя элемента, к которому надо перейти.

GO_RECORD (record_number) – переходит к указанной записи по ее номеру.

Пример: TO_NUMBER(:SYSTEM.TRIGGER_RECORD+8), а также используя другие системные переменные: SYSTEM.CURSOR_RECORD и SYSTEM.TRIGGER_RECORD.

HELP – выдает поясняющее сообщение о текущем поле в строке сообщений. Если такое сообщение уже выведено, HELP выводит детальный help-экран для данного поля.

HIDE_MENU – заставляет текущее меню исчезать (но не выходить из него), если оно в данный момент было выведено на экран, открывая ту часть вывода формы, которая была закрыта этим меню. Меню снова выведется на экран при вызове упакованной процедуры SH0W_MENU или при нажатии оператором функциональной клавиши [Menu].

HIDE_VIEW – скрывает указанную вид-картинку (Canvas).

Параметры:

view_id – уникальный идентификатор, который Forms Builder связывает с вид-картинкой (Canvas) при создании. Функция поиска идентификатора – FIND_VIEW.

view_name – имя вид-картинки (Canvas).

HIDE_WINDOW – скрывает указанное окно. HIDE_WINDOW эквивалентно свойству окно VISIBLE, установленному в False (No).

Параметры:

window_id – уникальный идентификатор, который Forms Builder связывает с окном при создании. Функция поиска идентификатора – FIND_WINDOW.

window_name – имя окна, заданное на этапе проектирования.

HOST ('system_command' [, NOSCREEN]) – выполняет указанную команду ОС. Если вы используете параметр NO_SCREEN, Forms не очищает экран и не запрашивает у оператора выход из команды.

ID_NULL – возвращает булево (BOOLEAN) значение, если идентификатор (ID) объекта доступен.

Параметры:

object_id – идентификаторы нижеперечисленных объектов, состояние которых нужно проверить:

- Alert;
- Block;
- Canvas;
- Editor;
- FormModule;
- GroupColumn;
- Item;
- LOV;
- MenuItem;
- ParamList;
- RecordGroup;
- Relation;
- Timer;
- Viewport;
- Window.

INSERT_RECORD – вставка текущей записи в БД во время Post и Commit Transactions процесса.

Примечание: *процедура актуальна только в ON-INSERT триггере.*

ISSUE_ROLLBACK – при вызове из On-Rollback триггера инициирует процесс Form Buildera – откат до последней точки сохранения (savepoint).

Параметры:

savepoint name – имя точки сохранения (savepoint), до которой будет совершен откат. В случае, когда имя не указано, будет совершен полный откат.

ISSUE_SAVEPOINT – при вызове из On-Savepoint триггера инициирует процесс Form Buildera – установки точки сохранения (savepoint).

Параметры:

savepoint_name – имя точки сохранения.

ITEM_ENABLED – возвращает TRUE, если элемент меню активный, и FALSE, если неактивный.

Параметры:

mnunam – имя меню. Тип данных – Varchar2.

Itmnam – имя элемента. Тип данных – Varchar2.

LAST_RECORD – переходит на последнюю запись в списке блоков записей. Если в блоке открыт запрос, Forms вызывает оставшиеся выбранные записи в список блоков записей.

LIST_VALUES (NO_RESTRICT | RESTRICT) – выводит список значений для текущего поля. Список значений остается на экране до тех пор, пока оператор не нажмет [Exit/Cancel] или не выберет значение в поле. По умолчанию LIST_VALUES применяет параметр NO_RESTRICT, который заставляет Forms не использовать свойство автоматического поиска и завершения. Если вы установите параметр RESTRICT, Forms использует это свойство. Когда применяется автоматический поиск и завершение, список значений рассматривает текущее значение поля как значение поиска. Это значит, что если оператор нажмет [List] в поле, которое имеет список значений, Forms проверяет, содержит ли поле значение. Если поле содержит некоторое значение, Forms автоматически использует это значение так, как если бы оператор ввел это значение в поле поиска списка значений и нажал [List] для сужения списка. Если значение поля суживает список до одного значения, Forms не выдает такой список, а автоматически считывает правильное значение в поле.

LOCK_RECORD – пытается заблокировать строку в БД, которая соответствует текущей записи.

LOGON_SCREEN – отображает стандартное окно подсоединения к БД Oracle Form Builder.

LOGOUT – отсоединиться от БД.

MESSAGE (‘message’) – выводит указанный текст в строке сообщений. Вы можете выводить сообщение длиной до 78 символов. Если нужно вывести знак апострофа в сообщении, вводите его дважды в этой строке.

MESSAGE_CODE – возвращает код сообщения, которое Form Builder сгенерировал в текущей Runform сессии.

MESSAGE_TEXT – возвращает текст сообщения, которое Form Builder сгенерировал в текущей Runform сессии.

MESSAGE_TYPE – возвращает тип сообщения, которое Form Builder сгенерировал в текущей Runform сессии.

NAME_IN (argument_reference), где argument_reference может быть любым видом ссылки на аргумент (включая другую косвенную ссылку), которая выражает имя поля или переменную, чье значение является строкой символов, удовлетворяющей требованиям аргумента.

NEW_FORM ('form') – завершает работу с текущей формой и входит в указанную форму. Вызывающая форма завершается как форма-родитель. Если вызывающая форма сама вызывалась из формы более высокого уровня, Forms сохраняет вызов более высокого уровня активным и трактует его как вызов новой формы. Forms освобождает память (типа курсоров БД), которая была занята завершенной формой. Forms запускает новую форму с теми же опциями SQL*Forms(Run Form), что и форма-родитель. Если форма-родитель сама была вызываемой, Forms запускает новую форму с теми же параметрами (например, HIDE и NO_REPLACE), что и у формы-родителя.

NEXT_BLOCK – перейти к следующему блоку.

NEXT_FORM – перейти к следующей форме.

NEXT_ITEM – перейти к следующему элементу.

NEXT_KEY – передает управление на вводимое поле первичного ключа со следующим последовательным номером, большим номера текущего поля. Если такого поля нет, NEXT_KEY передает управление на вводимое поле первичного ключа с минимальным последовательным номером. Если нет поля первичного ключа в текущем блоке, возникает ошибка. Если модулем проверки является поле, NEXT_KEY тестирует все поля с последовательными номерами, большими, чем у текущего поля, или меньшими, чем у заданного поля.

NEXT_MENU_ITEM – перейти к следующему элементу меню.

NEXT_RECORD – перейти к следующей записи.

NEXT_SET – вызывает другую группу записей из БД и передает управление на первую вызванную запись. NEXT_SET успешно выполняется только тогда, когда запрос открыт на текущем блоке.

PASTE_REGION – вставляет данные из буфера. Действует только для элементов текста и картинок.

PAUSE – приостанавливает работу до тех пор, пока оператор не нажмет любую функциональную клавишу. pause может выводить предупреждающее сообщение.

PLAY_SOUND – проигрывает звук в указанном элементе звука.

POPULATE_GROUP – извлекает запрос, ассоциированный с указанной группой записей, и возвращает числовой индикатор выполнения запроса. При успешном запросе POPULATE_GROUP возвращает «0» . В случае невыполнения запроса генерируется ошибка. Ячейки, возвращенные как результат запроса, заменяют существующие, если таковые есть.

Параметры:

recordgroup_id – уникальный идентификатор, который Forms Builder связывает с группой записи при создании. Функция поиска идентификатора – FIND_RECORD_GROUP. Тип данных идентификатора – RECORDGROUP группы записей. Или берется имя группы записей, тогда тип данных – VARCHAR2.

group_name – определяет имя группы записей. Тип данных – Varchar2.

POPULATE_GROUP_FROM_TREE – формирует группу из данных иерархического дерева.

Параметры:

group_name – определяет имя группы.

group_id – определяет идентификатор, связанный с группой.

item_name – специфицирует имя объекта, данное ему на этапе проектирования. Тип данных – VARCHAR2.

Item_id – уникальный идентификатор, который Forms Builder связывает с элементом при создании. Функция поиска идентификатора – FIND_ITEM.

Node – определяет существующий узел. Если определен, то используется для заполнения группы, включая указанный узел.

POPULATE_GROUP_WITH_QUERY – формирует группу из указанного запроса.

Параметры:

recordgroup_id – уникальный идентификатор, который Forms Builder связывает с группой записи при создании. Функция поиска идентификатора – FIND_RECORD_GROUP. Тип данных идентифика-

тора – RECORDGROUP группы записей. Или берется имя группы записей, тогда тип данных – VARCHAR2.

group_name – определяет имя группы записей. Тип данных – Varchar2.

query – в качестве этого параметра вы указываете запрос, который будет формировать вашу группу. Любые результирующие столбцы, возвращенные этим запросом, получают такой же тип, как и в таблице БД. Тип данных запроса – VARCHAR2.

POPULATE_LIST – очищает список от текущих элементов и заполняет его данными из группы записей. Такая группа должна быть создана в режиме выполнения формы (Runtime) и содержать не менее двух столбцов (VARCHAR2) следующей структуры:

Column 1(столбец 1):	Column 2(столбец 2):
the list label (метка листа)	the list value (значение элемента)

POPULATE_TREE – заполняет иерархическое дерево данными из группы записей либо иерархического запроса, удаляя при этом существующие данные, если таковые есть.

Параметры:

item_name – специфицирует имя объекта, данное ему на этапе проектирования. Тип данных – VARCHAR2.

Item_id – уникальный идентификатор, который Forms Builder связывает с элементом при создании. Функция поиска идентификатора – FIND_ITEM.

POST – записывает данные из формы в БД, но не выполняет операции commit в БД. Forms вначале проверяет форму. Если существуют изменения, которые нужно записать в БД, то для каждого блока в форме Forms записывает удаления, вставки и корректировки в БД. Любые данные, которые вы записываете в БД, сохраняются в ней по следующей процедуре COMMIT_FORM, которая выполняется во время текущей сессии Forms (Run Form). Иначе эти данные отвергаются по следующей процедуре CLEAR_FORM.

- PREVIOUS_BLOCK – вернуться к предыдущему блоку.
- PREVIOUS_FIELD – вернуться к предыдущей записи.
- PREVIOUS_RECORD – переходит на запись с максимальным последовательным номером, меньшим, чем номер текущей записи.
- PREVIOUS_FORM – вернуться к предыдущей форме.
- PREVIOUS_ITEM – вернуться к предыдущему элементу.
- PREVIOUS_MENU – возвращает в предыдущее меню.
- PREVIOUS_MENU_ITEM – возвращает к предыдущему элементу меню.

– PRINT – записывает одну или все страницы формы в файл или выводит на печать. Forms запрашивает у оператора пометку страниц для записи, имя файла, куда записывать, и нужно ли посылать файл на системный принтер.

PTR_TO_VAR – First, creates an OLE variant of type VT_PTR that contains the supplied address. Then, passes that variant and type through the function VARPTR_TO_VAR.

QUERY_PARAMETER – отображает диалог Query Parameter (параметр запроса). Этот диалог показывает текущие значения определенных параметров. Конечный пользователь может устанавливать значение параметра, указанного в списке. Query Parameter (параметр запроса) – диалог модальный, и управление после подтверждения или отмены диалога не передается обратно в вызывающий триггер или процедуру.

Параметры:

parameter_string – определяет строку параметров для элементов меню. Синтаксис для указанной строки параметров объявляется с амперсандом : «&param_name». Параметры, использующиеся в PL/SQL, объявляются как связные переменные (bind variables) : «:param.name».

READ_IMAGE_FILE – читает картинку из файла указанного типа и отображает в Form Builder в элементе картинка (image item).

Параметры:

file_name – имя существующего файла. Имя файла должно содержать полный путь к нему.

file_type – существующий файл картинки типа: BMP, CALS, GIF, JFIF, JPG, PICT, RAS, TIFF или TPIC.

item_name – специфицирует имя объекта, данное ему на этапе проектирования. Тип данных – VARCHAR2.

Item_id – уникальный идентификатор, который Forms Builder связывает с элементом при создании. Функция поиска идентификатора – FIND_ITEM.

READ_SOUND_FILE – проигрывает звуковой объект из указанного файла в указанном элементе звука.

Параметры:

file_name – указание полного имени и пути к проигрываемому файлу.

file_type – тип звукового файла: AU, AIFF, AIFF-C и WAVE.

item_name – специфицирует имя объекта, данное ему на этапе проектирования. Тип данных – VARCHAR2.

Item_id – уникальный идентификатор, который Forms Builder связывает с элементом при создании. Функция поиска идентификатора – FIND_ITEM.

RECALCULATE – отмечает значение в указанном вычисляемом элементе для пересчета.

Параметры:

item_name – специфицирует имя объекта, данное ему на этапе проектирования. Тип данных – VARCHAR2.

Item_id – уникальный идентификатор, который Forms Builder связывает с элементом при создании. Функция поиска идентификатора – FIND_ITEM.

REDISPLAY – перерисовывает экран. Это уничтожает все посторонние системные сообщения.

RELEASE_OBJ (obj OLEOBJ, kill_persistence_boolean := NULL) – закрывает соединение с OLE-объектом.

REPLACE_CONTENT_VIEW – программно замещает текущий вид отображения окна.

Параметры:

window_id – уникальный идентификатор, который Forms Builder связывает с окном при создании. Функция поиска идентификатора – FIND_WINDOW.

window_name – имя окна, заданное на этапе проектирования.

view_id – уникальный идентификатор, который Forms Builder связывает с вид-картинкой (Canvas) при создании.

view_name – имя вид-картинки (Canvas), присвоенное ей при создании. Тип данных – VARCHAR2.

REPLACE_MENU ('menu_application' [, menu_type [, 'starting_menu' [, 'group_name']]]) – заменяет текущее меню на указанное меню. Эта процедура выводит новое меню на экран, но не делает это новое меню активным. REPLACE_MENU также позволяет вам изменить порядок вывода меню и группу защиты, которую использует при определении, какие опции меню выводить. Т. к. REPLACE_MENU не делает новое меню активным, Forms не разрешает меню закрывать любую часть активной страницы. Поэтому все меню или его часть не появляются на экране, если этот экран занимает активная страница.

Параметры:

menu_application – определяет приложение SQL*Menu, которое заменит текущее приложение меню. Этот параметр заменяет приложение, определенное в характеристике формы Default Menu Application.
menu_type { BAR |FULL_SCREEN |PULL_DOWN} – определяет тип вывода меню. Этот параметр заменяет тип меню по умолчанию (PULL_DOWN). Если вы не укажете тип меню, Forms будет использовать тип PULL_DOWN.
starting_menu_name – определяет меню в приложении меню, которое Forms будет использовать как начальное меню. Этот параметр заменяет меню, указанное в характеристике формы Starting Menu Name. Если вы не укажете начальное меню, будет использоваться меню с тем же именем, что и приложение меню.
group_name – определяет группу защиты, которое Menu будет использовать. Этот параметр заменяет характеристику формы Menu Security Group. Если вы не укажете имя группы, то меню использует текущее имя пользователя для определения группы.

REPORT_OBJECT_STATUS – проверяет статус объекта отчета, запущенного из формы функцией RUN_REPORT_OBJECT.
Параметры:
report_id – значение типа Varchar2, возвращаемое функцией RUN_REPORT_OBJECT. Это значение уникально идентифицирует текущий запущенный отчет локально либо в report server (сервер отчетов).

RESET_GROUP_SELECTION – снимает выделение, если таковое есть, в указанной группе.
Параметры:
recordgroup_id – уникальный идентификатор, который Forms Builder связывает с группой записи при создании. Функция поиска идентификатора – FIND_RECORD_GROUP. Тип данных идентификатора – RECORDGROUP группы записей. Или берется имя группы записей, тогда тип данных – VARCHAR2.
group_name – определяет имя группы записей. Тип данных – Varchar2.

RESIZE_WINDOW – изменяет размеры (ширину и высоту) указанного окна.
Параметры:
window_id – уникальный идентификатор, который Forms Builder связывает с окном при создании. Функция поиска идентификатора – FIND_WINDOW.
window_name – имя окна, заданное на этапе проектирования.
Width – определяет ширину окна.

Height – определяет высоту окна.

RETRIEVE_LIST – процедура создания так называемого «снимка» элемента списка с самого себя.

Параметры:

ITEM_NAME – этот параметр принимает имя списка, из которого мы собираемся извлекать содержимое.

RECGRP_ID – этот параметр принимает имя группы записей, в которую мы будем извлекать содержимое.

Примечание: *в этом случае условие извлечения содержимого элемента списка в группу записей такое же, как и при заполнении списка из Группы записей, то есть необходимо существование двух символьных столбцов для заполнения данных списка и их значений.*

RUN_PRODUCT

Run_Product(REPORTS, '/opt/MFORM/himanaliz/PH', SYNCHRONOUS, RUNTIME, FILESYSTEM, pl_id, NULL);

Параметры:

REPORTS – имя запускаемого продукта, это может быть и Graphics.

'/opt/MFORM/himanaliz/PH' – путь к вашему отчету.

SYNCHRONOUS (ASYNCHRONOUS) – режим запуска, запустить Reports синхронно или асинхронно.

RUNTIME – режим запуска.

FILESYSTEM – место размещения.

pl_id – список параметров, если их нет, можно написать null.

NULL(Display) – режим отображения.

RUN_REPORT_OBJECT – запускает объект-отчет.

SCROLL_UP – скроллирует список записей текущего блока вниз примерно на 80 % вывода блока. Такое действие выведет записи, находившиеся «выше» текущего вывода блока.

SCROLL_DOWN – скроллирует список записей текущего блока вверх примерно на 80% вывода блока. Такое действие выведет записи, находившиеся «ниже» текущего вывода блока.

SCROLL_VIEW – данная процедура используется для программной прокрутки вид-картинки.

Параметры:

view_id – уникальный идентификатор, который Forms Builder связывает с вид-картинкой (Canvas) при создании.

view_name – имя вид-картинки (Canvas), присвоенное ей при создании. Тип данных – VARCHAR2.
X – определяет координату прокрутки по «X».
У – определяет координату прокрутки по «У».

SELECT_ALL – выделяет весь текст в текущем элементе. Вызывайте эту процедуру перед выполнением CUT_REGION или COPY_REGION, когда вы хотите скопировать или вырезать текст.

SERVER_ACTIVE – идентифицирует связанный с контейнером сервер, в качестве идентификаторов выступают два состояния: сервер запущен (возвращается True), сервер не запущен (возвращается False). Функция возвращает – BOOLEAN.

Примечание: *данная функция актуальна только для платформ Microsoft Windows и Macintosh.*

Параметры:

item_name – специфицирует имя объекта, данное ему на этапе проектирования. Тип данных – VARCHAR2.
Item_id – уникальный идентификатор, который Forms Builder связывает с элементом при создании. Функция поиска идентификатора – FIND_ITEM.

SET_ALERT_BUTTON_PROPERTY – устанавливает метку на одну из кнопок предупреждения (Alert).

Параметры:

alert_id – уникальный идентификатор, который Forms Builder связывает с элементом при создании. Функция поиска идентификатора – FIND_ALERT.
alert_name – имя предупреждения. Тип данных – Varchar2.
ButtoA – константа, определяющая одну из кнопок: ALERT_BUTTON1, ALERT_BUTTON2 или ALERT_BUTTON3.
Property – метка, определяющая текст метки для кнопки предупреждения (alert button).
Value – определяет значение типа VARCHAR2, добавленное к свойству, которое вы определили.

SET_ALERT_PROPERTY – устанавливает сообщение для существующего предупреждения.

Параметры:

alert_id – уникальный идентификатор, который Forms Builder связывает с элементом при создании. Функция поиска идентификатора – FIND_ALERT.

alert_name – имя предупреждения. Тип данных – Varchar2.
Property – определяет свойство предупреждения, которое вы устанавливаете: ALERT_MESSAGE_TEXT – текст предупреждения.
TITLE – определяет заголовок предупреждения.
Message – определяет текст сообщения, который вы хотите отобразить в сообщении предупреждения.

SET_APPLICATION_PROPERTY – устанавливает или сбрасывает свойства текущего приложения.
Параметры:
property – определяет свойство, которое вы хотите изменить. Возможные варианты:
– BUILTIN_DATE_FORMAT
– CURSOR_STYLE
– DATE_FORMAT_COMPATIBILITY_MODE
– FLAG_USER_VALUE_TOO_LONG
– PLSQL_DATE_FORMAT
Value – новое значение, которое вы собираетесь присвоить выбранному свойству.

SET_CANVAS_PROPERTY – установить свойства вид-картинки

SET_FORM_PROPERTY – установить свойства формы.

SET_GROUP_SELECTION – установить выделение в группе

SET_ITEM_INSTANCE_PROPERTY – установить свойства экземпляра элемента.

SET_ITEM_PROPERTY – установить свойства элемента.

SET_LOV_COLUMN_PROPERTY – установить свойства столбца списка значений LOV.

SET_LOV_PROPERTY – устанавливает значение выбранного свойства LOV – списка значений.
Параметры:
lov_id – уникальный идентификатор, который Forms Builder связывает со списком значений (LOV) при создании. Функция поиска идентификатора – FIND_LOV. Тип данных идентификатора – LOV.
lov_name – имя списка значений. Тип данных – Varchar2.

Property – название свойства, информацию о котором мы хотим получить. Ниже перечислены возможные значения этого свойства:
AUTO_REFRESH – возвращает TRUE, если автоматическое обновление включено (выполнять запрос для заполнения LOV повторно), и False, то, соответственно, автообновление отключено.
GROUP_NAME – возвращает имя группы записей, на которой базируется LOV.
LOV_SIZE – определяет ширину и высоту LOV.
POSITION – определяет x, y координаты позиции LOV.
TITLE – определяет имя заголовка LOV.
Value – определяет одну из констант:
– PROPERTY_TRUE
– PROPERTY_FALSE
Recordgroup Name – имя группы записей.
X – определяет «x» координату ширины.
Y – определяет «y» координату высоты.

SET_MENU_ITEM_PROPERTY – устанавливает свойства элемента меню.

Параметры:
menuitem_id – уникальный идентификатор, который Forms Builder связывает с элементом меню при создании. Функция поиска идентификатора – FIND_MENU_ITEM. Тип данных – MenuItem.
menu_name.menuitem_name – имя элемента меню (MenuItem). Тип данных – VARCHAR2.
Property – название свойства, информацию о котором мы хотим получить. Ниже перечислены возможные значения этого свойства:
CHECKED – возвращает True, если check box элемента меню помечен (checked), и False, если не помечен (unchecked).
ENABLED – возвращает True, если элемент меню доступен, и False, если недоступен.
ICON_NAME – возвращает имя файла, который содержит иконку, ассоциированную с элементом.
LABEL – возвращает строку метки элемента меню.
VISIBLE – возвращает True, если элемент виден, False – если не виден.
Value – определяет новое значение для заданного свойства.
Label – определяет метку для элемента меню типа Varchar2.

SET_PARAMETER_ATTR – устанавливает тип и значения указанного параметра.

Параметры:

list or name – определяет лист параметров. Нужный лист параметров может быть указан как идентификатор с типом PARAMLIST или как имя с типом VARCHAR2.
Key – имя параметра. Тип данных – Varchar2.
Paramtype – тип параметра:
– DATA_PARAMETER
– TEXT_PARAMETER
value – значение параметра, заданное символьной строкой.

SET_RADIO_BUTTON_PROPERTY – устанавливает свойство радиокнопки.

Параметры:

item_name – специфицирует имя объекта, данное ему на этапе проектирования. Тип данных – VARCHAR2.
Item_id – уникальный идентификатор, который Forms Builder связывает с элементом при создании. Функция поиска идентификатора – FIND_ITEM.
button_name – определяет имя радиокнопки, свойство которой вы хотите установить. Тип данных – VARCHAR2.
Property – определяет свойство, которое вы хотите установить. Значения можно установить:
– BACKGROUND_COLOR
– ENABLED
– FILL_PATTERN
– FONT_NAME
– FONT_SIZE
– FONT_SPACING
– FONT_STYLE
– FONT_WEIGHT
– FOREGROUND_COLOR
– HEIGHT
– ITEM_SIZE
– LABEL
– POSITION
– PROMPT
– PROMPT_BACKGROUND_COLOR
– PROMPT_FILL_PATTERN
– PROMPT_FONT_NAME
– PROMPT_FONT_SIZE
– PROMPT_FONT_SPACING
– PROMPT_FONT_STYLE
– PROMPT_FONT_WEIGHT

– PROMPT_FOREGROUND_COLOR
– PROMPT_WHITE_ON_BLACK

SET_RECORD_PROPERTY – устанавливает свойства записи.
Параметры:
record_number – номер записи.
block_name – имя блока, содержащего целевую запись. Тип данных – VARCHAR2.
Property – свойство, значение которого собираемся изменить, – это константа STATUS.
Value – используйте возможное значение:
– CHANGED_STATUS
– INSERT_STATUS
– NEW_STATUS
– QUERY_STATUS

SET_RELATION_PROPERTY – устанавливает значения указанного свойства RELATION (отношения).
Параметры:
relation_id – идентификатор отношения. Тип данных – Relation.
relation_name – имя отношения. Тип данных – VARCHAR2.
Property – название свойства, значение которого мы хотим модифицировать. Ниже перечислены возможные значения этого параметра:
– AUTOQUERY
– DEFERRED_COORDINATION
– MASTER_DELETES
– PREVENT_MASTERLESS_OPERATION
Value – возможные значения:
– CASCADING
– ISOLATED
– NON_ISOLATED
– MASTER_DELETES
– TRUE
– FALSE

SET_REPORT_OBJECT_PROPERTY – устанавливает свойства объекта отчета.
Параметры:
report_id – уникальный идентификатор, который Forms Builder связывает с объектом отчет при создании. Функция поиска идентификатора – FIND_REPORT_OBJECT.
report_name – Specifies the unique name of the report.

Property – название свойства, значение которого мы хотим модифицировать. Ниже перечислены возможные значения этого параметра:
– REPORT_EXECUTION_MODE:
BATCH or RUNTIME
– REPORT_COMM_MODE:
SYNCHRONOUS or ASYNCHRONOUS
– REPORT_DESTYPE:
PREVIEW, FILE, PRINTER, MAIL, CACHE or SCREEN
– REPORT_FILENAME
– REPORT_SOURCE_BLOCK
– REPORT_QUERY_NAME

SET_TAB_PAGE_PROPERTY – устанавливает свойства указанной страницы вкладки.

Параметры:

tab_page_id – уникальный идентификатор, который Forms Builder связывает со страницей вкладки при создании. Функция поиска уникального идентификатора – FIND_TAB_PAGE. Тип данных – TAB_PAGE.

tab page_name – имя страницы вкладки.

Property – название свойства, значение которого мы хотим модифицировать. Ниже перечислены возможные значения этого параметра:

BACKGROUND_COLOR – цвет заднего фона.

ENABLED – True, если элемент меню доступен, и False, если недоступен.

FILL_PATTERN – тип заливки, использованной для заполнения региона.

FONT_NAME – название шрифта.

FONT_SIZE – размер шрифта.

FONT_SPACING – расстояние между буквами – кернинг.

FONT_STYLE – стиль шрифта.

FONT_WEIGHT – вес шрифта.

Value – новое значение заданного свойства.

SET_TIMER – устанавливает значения существующего таймера.

Параметры:

timer_id – уникальный идентификатор, который Forms Builder связывает с таймером при создании. Функция поиска уникального идентификатора – FIND_TIMER. Тип данных – TIMER.

timer_name – имя таймера. Тип данных – VARCHAR2.

Milliseconds – определяет значение таймера в миллисекундах. Максимально возможное значение от 1 до 2147483648; число, выхо-

дящее за границу, будет округлено до 2147483648. Также данный параметр может принимать в качестве значения константу NO_CHANGE, определяющую свойство текущей установки.

iterate – определяет тип итераций таймера: REPEAT (по умолчанию), NO_REPEAT, NO_CHANGE.

SET_TREE_NODE_PROPERTY – устанавливает свойство элемента данных (узла) иерархического дерева.

Параметры:

item_name – специфицирует имя объекта, данное ему на этапе проектирования. Тип данных – VARCHAR2.

Item_id – уникальный идентификатор, который Forms Builder связывает с элементом при создании. Функция поиска идентификатора – FIND_ITEM.

Node – определяет существующий элемент дерева (узел).

Property – определяет свойство, значение которого мы хотим модифицировать:

- NODE_STATE – состояние узла, возможные значения: EXPANDED_NODE, COLLAPSED_NODE и LEAF_NODE.
- NODE_LABEL – устанавливает метку узла дерева.
- NODE_ICON – устанавливает иконку для соответствующего узла.
- NODE_VALUE – устанавливает значение узла.

Value – новое значение заданного свойства.

SET_TREE_PROPERTY – устанавливает свойства указанного иерархического дерева.

Параметры:

item_name – специфицирует имя объекта, данное ему на этапе проектирования. Тип данных – VARCHAR2.

Item_id – уникальный идентификатор, который Forms Builder связывает с элементом при создании. Функция поиска идентификатора – FIND_ITEM.

Node – определяет существующий элемент дерева (узел).

Property – определяет свойство, значение которого мы хотим модифицировать:

- RECORD_GROUP – переназначить источник данных (задать новую группу записей).
- QUERY_TEXT – изменить текст запроса, заполняющего дерево.
- ALLOW_EMPTY_BRANCHES – разрешение существования пустых узлов. Принимает значение True или False.

Value – определяет новое значение для заданного свойства.

SET_TREE_SELECTION – установить выделение в иерархическом дереве.

Параметры:

item_name – специфицирует имя объекта, данное ему на этапе проектирования. Тип данных – VARCHAR2.

Item_id – уникальный идентификатор, который Forms Builder связывает с элементом при создании. Функция поиска идентификатора – FIND_ITEM.

Node – определяет существующий элемент дерева (узел).

selection_type – тип выделения:

– SELECT_ON
– SELECT_OFF
– SELECT_TOGGLE

SET_VIEW_PROPERTY – устанавливает свойства указанной вид-картинки (Canvas).

Параметры:

view_id – уникальный идентификатор, который Forms Builder связывает с вид-картинкой (Canvas) при создании.

view_name – имя вид-картинки (Canvas), присвоенное ей при создании. Тип данных – VARCHAR2.

Property – название свойства, значение которого хотим модифицировать. Ниже перечислены возможные значения этого свойства:

– DIRECTION – устанавливает значение свойства DIRECTION в: DIRECTION_DEFAULT, RIGHT_TO_LEFT, LEFT_TO_RIGHT.
– DISPLAY_POSITION – устанавливает позицию отображения для стековой вид-картинки (Canvas).
– HEIGHT – определяет высоту вида отображения.
– POSITION_ON_CANVAS – размещение отображения по координатам X и Y на вид-картинке (Canvas).
– VIEWPORT_X_POS – позиция отображения координаты X в стековой вид-картинке (stacked view).
– VIEWPORT_Y_POS – позиция отображения координаты Y в стековой вид-картинке (stacked view).
– VIEWPORT_X_POS_ON_CANVAS – размещение позиции обзора по «X» на вид-картинке (Canvas).
– VIEWPORT_Y_POS_ON_CANVAS – размещение позиции обзора по «Y» на вид-картинке (Canvas).
– VIEW_SIZE – для стековой вид-картинки размер отображения как по высоте, так и по ширине.
– VISIBLE – управляет отображением обзора.
– WIDTH – определяет ширину обзора.

Value – новое значение заданного свойства.
X – определяет «x» координату ширины.
Y – определяет «y» координату высоты.

SET_WINDOW_PROPERTY – устанавливает свойства указанного окна.

Параметры:

window_id – уникальный идентификатор, который Forms Builder связывает с окном при создании. Функция поиска идентификатора – FIND_WINDOW.

window_name – имя окна, заданное на этапе проектирования.

Property – название свойства, значение которого хотим модифицировать. Ниже перечислены возможные значения этого свойства:

- BACKGROUND_COLOR – цвет заднего фона.
- ENABLED – True, если элемент меню доступен, и False, если недоступен.
- FILL_PATTERN – тип заливки, использованной для заполнения региона.
- FONT_NAME – название шрифта.
- FONT_SIZE – размер шрифта.
- FONT_SPACING – расстояние между буквами – кернинг.
- HEIGHT – определяет высоту окна.
- HIDE_ON_EXIT – скрывается ли окно при выходе.
- ICON_NAME – определяет имя файла картинки, ассоциированной с окном, в минимизированном режиме.
- TITLE – определяет название окна.
- VISIBLE – определяет True, если окно видимо, и False, если не видимо.
- WHITE_ON_BLACK – белое на черном. True – белое на черном.
- WIDTH – определяет ширину окна.
- WIDOW_SIZE – определяет высоту и ширину окна.
- WINDOW_STATE – определяет текущее состояние отображения окна: NORMAL, MAXIMIZE или MINIMIZE.

Value – возможные значения – FALSE или TRUE, если это касается свойства отображения окна; если устанавливаете состояние окна (WINDOW_STATE) – NORMAL, MAXIMIZE, MINIMIZE.

X – определяет позицию по «X» положения окна на дисплее.
Y – определяет позицию по «Y» положения окна на дисплее.

SHOW_ALERT – отображает на экране предупреждение и возвращает целочисленное значение в зависимости от нажатой кнопки.

Параметры:

alert_id – уникальный идентификатор, который Forms Builder связывает с предупреждением при создании. Функция поиска идентификатора – FIND_ALERT. Тип данных – Alert.
alert_name – определяет имя предупреждения, данное ему на этапе проектирования. Тип данных – VARCHAR2.

SHOW_EDITOR – отображает текстовый редактор с указанными координатами. Причем если в элементе есть строка, то редактор сразу же при открытии отображает эту строку.
Параметры:
editor_id – уникальный идентификатор, который Forms Builder связывает с редактором при создании. Функция поиска идентификатора – FIND_EDITOR. Тип данных – Editor.
editor_name – определяет имя редактора. Тип данных – VARCHAR2.
message_i – определяет входной (IN) параметр типа VARCHAR2. Значение, присваиваемое этому параметру, должно быть NULL. Вы также можете связывать элемент текста или переменную.
X – определяет координату «x» для редактора.
Y – определяет координату «y» для редактора.
message_out – определяет выходной (OUT) параметр типа VARCHAR2. Вы можете связать его с элементом текста или переменной. Если оператор отменяет работу с редактором, то возвращается результат FALSE и message_out – NULL.
Result – определяет выходной параметр типа BOOLEAN. Если оператор подтверждает работу с редактором, то возвращается результат – True. Если отменяет работу с редактором, то результат вернет False, а message_out — NULL.

SHOW_KEYS – выводит экран Show Keys. Когда оператор нажимает функциональную клавишу, Forms вновь выводит форму, которая была на экране до вызова SHOW_KEYS.

SHOW_LOV – отображает на дисплее список значений LOV. Возвращает True, если оператор выбрал значение из списка, и False, если отменил работу со списком значений (LOV).
Параметры:
lov_id – уникальный идентификатор, который Forms Builder связывает со списком значений (LOV) при создании. Функция поиска идентификатора – FIND_LOV. Тип данных идентификатора – LOV.
lov_name – имя списка значений. Тип данных – Varchar2.
X – определяет координату «x» позиции LOV.
Y – определяет координату «y» позиции LOV.

SHOW_MENU – выводит текущее меню, если только оно уже не выведено. Эта процедура не делает меню активным. Поскольку SHOW_MENU не делает меню активным, Forms не позволяет меню закрывать любую часть активной страницы. Поэтому все меню или его часть не появляются на экране, если его занимает активная страница.

SHOW_VIEW – отображает вид-картинку в указанных координатах.
Параметры:
view_id – уникальный идентификатор, который Forms Builder связывает с вид-картинкой (Canvas) при создании.
view_name – имя вид-картинки (Canvas), присвоенное ей при создании. Тип данных – VARCHAR2.

SHOW_WINDOW – отображает указанное окно на экране в заданных координатах. Если выбранное окно модальное, то SHOW_WINDOW имитирует процедуру GO_ITEM, переходя к первому элементу в модальном окне.
Параметры:
window_id – уникальный идентификатор, который Forms Builder связывает с окном при создании. Функция поиска идентификатора – FIND_WINDOW.
window_name – имя окна, заданное на этапе проектирования.
X – определяет координату «x» позиции окна.
Y – определяет координату «y» позиции окна.

SYNCHRONIZE – синхронизирует экран терминала и внутреннее состояние формы. То есть, SYNCHRONIZE обновляет вывод на экран для отображения информации, которую SQL*Forms имеет в своем внутреннем представлении экрана. Очень удобно использовать совместно с функцией MESSAGES.

TERMINATE – работает аналогично команде [Accept] – подтверждения, в диалоге или форме.

UNSET_GROUP_SELECTION – снять выделение с заданной ячейки в группе записей.
Параметры:
recordgroup_id – уникальный идентификатор, который Forms Builder связывает с группой записи при создании. Функция поиска идентификатора – FIND_RECORD_GROUP. Тип данных идентификатора – RECORDGROUP группы записей. Или берется имя группы записей, тогда тип данных – VARCHAR2.

group_name – определяет имя группы записей. Тип данных – Varchar2.
row_number – номер ячейки, с которой хотим снять выделение.

UP – передает управление на представление текущего поля в записи со следующим минимальным текущим последовательным номером.

UPDATE_RECORD – обновление текущей записи в БД во время Post и Commit Transactions процесса.
Примечание: процедура актуальна только в ON-UPDATE триггере.

USER_EXIT('user_exit_name_&_any_parameters' [, 'error_string']), может принимать и один параметр (**'user_exit_name_&_any_parameters'**) – вызывает пользовательский выход по имени, указанному в user_exit_string.
user_exit_string – определяет имя пользовательского выхода, который вы хотите вызвать, и возможные параметры.
error_string – определяет сообщение об ошибке, которое Forms делает доступным при сбое пользовательского выхода.

WEB.SHOW_DOCUMENT – определяет URL и целевое окно Web-приложения.
Параметры:
url – тип данных – VARCHAR2. Определяет Uniform Resource Locator документа, который должен быть загружен.
target – тип данных – VARCHAR2. Определяет одно из возможных значений:
– _SELF
– _PARENT
– _TOP
– _BLANK

Учебное электронное издание

Сергеенко Сергей Васильевич

РАЗРАБОТКА И ПРОЕКТИРОВАНИЕ WEB-ПРИЛОЖЕНИЙ В ORACLE DEVELOPER

Учебное пособие

Литературный редактор С. Перепелкина
Корректор А. Ангелина
Компьютерная верстка Ю. Волшмид
Обложка М. Автономова