



Кафедра информатики и
информационных технологий

Б1.О.18 БАЗЫ ДАННЫХ

Методические указания к лабораторным работам

РАЗРАБОТКА БАЗЫ ДАННЫХ В ORACLE SQL Developer

для направления подготовки 09.03.03 Прикладная информатика
профиль подготовки Прикладная информатика цифровой экономики

Квалификация выпускника
бакалавр

Уфа 2021

Рекомендовано к изданию методической комиссией экономического факультета
(протокол № 8 от 25.03.2021 г.)

Составитель: старший преподаватель Иванова Г.Р.

Ответственный за выпуск:

Заведующий кафедрой информатики и информационных технологий, д.т.н.
доцент Беляева А.С.

г. Уфа, ФГБОУ ВО Башкирский ГАУ, кафедра информатики и
информационных технологий

Содержание

Лабораторная работа №1. Создание основных объектов базы данных в в ORACLE SQL Developer.....	4
Лабораторная работа №2. Создание последовательностей, триггеров и организация заполнения таблиц данными	14
Лабораторная работа №3. Разработка функций и процедур для базы данных.....	24
Лабораторная работа №4 . Создание пакетов	40

Лабораторная работа №1. Создание основных объектов базы данных в в ORACLE SQL Developer

Цель работы - овладеть навыками работы в ORACLE SQL Developer, научиться создавать пользователя и таблицы базы данных.

1. Основные положения

Oracle SQL Developer является программным средством (IDE) для разработки объектов баз данных СУБД Oracle. Данное средство позволяет осуществлять полноценную разработку и сопровождение базы данных, а так же администрирование СУБД.

В Oracle понятия "схема" и "пользователь" используются для обозначения одного и того же: некоторого логического контейнера для хранения объектов базы данных, принадлежащих конкретному пользователю.

Таблицы (Table) используются для физического хранения пользовательских данных. В свойствах таблиц помимо описания полей, хранится описание ограничений целостности СУБД, созданных в соответствии с логической моделью данных разрабатываемой базы данных. Особое внимание при проектировании следует уделить параметрам **Storage**, которые описывают, каким образом будет храниться таблицы и как они будут изменяться после добавления в них данных.

Таблица 1.1.- Параметры Storage

Наименование параметра	Описание
Pct Free	Задаёт долю (в процентах) свободного пространства, выделяемого в каждом блоке данных таблицы для будущих обновлений строк. Если значение PCTFREE будет равно 0, то новые вставляемые строки будут заполнять блоки данных целиком. По умолчанию стоит значение 10. Это значит, что 10% каждого блока будет выделяться под обновление существующих строк, а вставляемые строки будут занимать в блоке максимум

	90%.
Pct Used	Указывает минимальный процент использования пространства каждого блока данных. Каждый блок становится кандидатом на вставку следующей новой строки, если процент использованного в нем пространства меньше указанного значения PCTUSED. Сумма значений PCTFREE и PCTUSED должна быть меньше 100!
Ini Trans	Задаёт начальное количество входов транзакций, выделяемых внутри каждого блока, выделенного для таблицы. Каждая транзакция, обновляющая данные блока, требует создания входа транзакции в этом блоке. Размер этого входа зависит от операционной системы. Использование этого параметра обеспечивает возможность обновления блока некоторым минимальным количеством параллельных транзакций и помогает исключить задержки динамического размещения входов транзакций.
Max Trans	Задаёт максимальное количество параллельных транзакций, обновляющих блок данных, выделенных для таблицы. Это ограничение не относится к запросам. Если количество транзакций, параллельно обрабатывающих данные блока, превышает значение параметра Ini Trans, СУБД начинает динамически размещать входы транзакций в блоке, пока их количество не превысит Max Trans или в блоке не останется свободного пространства.

Таблица 1.2.- Параметры экстенгов (Extents)

Наименование параметра	Описание
------------------------	----------

Initial	Указывает размер в байтах первого экстента объекта (таблицы). СУБД выделяет пространство для этого экстента, когда этот объект создается. Размер по умолчанию – размер 5-ти блоков данных.
Next	Указывает размер в байтах следующего экстента, который будет выделен для объекта.
Pct Increase	Указывает процент, на который каждый следующий экстент после второго увеличивается по сравнению с предыдущим.
Min	Указывает суммарное количество экстентов, выделяемых при создании данного объекта.
Max	указывает суммарное количество экстентов, включая и первый, которое может быть выделено для данного объекта.

Обзоры или представления (View) являются хранимыми запросами. Служат для обеспечения удобства при работе с сложными выборками из базы данных, так как обращаться к обзору и к его полям можно тем же способом как и обычным таблицам в SQL запросах. Результатом выполнения обзора является таблица. Таким образом, можно сказать, что обзор — это виртуальная таблица данных.

2 Содержание работы

1. На рабочем столе найти и запустить ярлык под названием Oracle SQL Developer. В появившемся рабочем окне на закладке **Connections** (соединения) создать новое соединение с сервером Oracle (рис. 1.1). Необходимо указать название соединения (**ORACLE_XE**), ввести имя пользователя **system** с паролем **manager**. Указать IP адрес компьютера, где установлена БД Oracle.

2. Далее нажать кнопку **Test** для проверки соединения. Затем, если соединение прошло успешно, нажать кнопку **Connect**.

3. Пользователь с именем **system** обладает привилегией DBA (Database Administrator). Эта привилегия дает пользователю право создавать других пользователей, табличные пространства и прочие системные объекты в СУБД Oracle. В схеме **system** нельзя создавать таблицы и другие объекты конкретных

пользователей. Поэтому для каждого пользователя необходимо создавать отдельную схему.

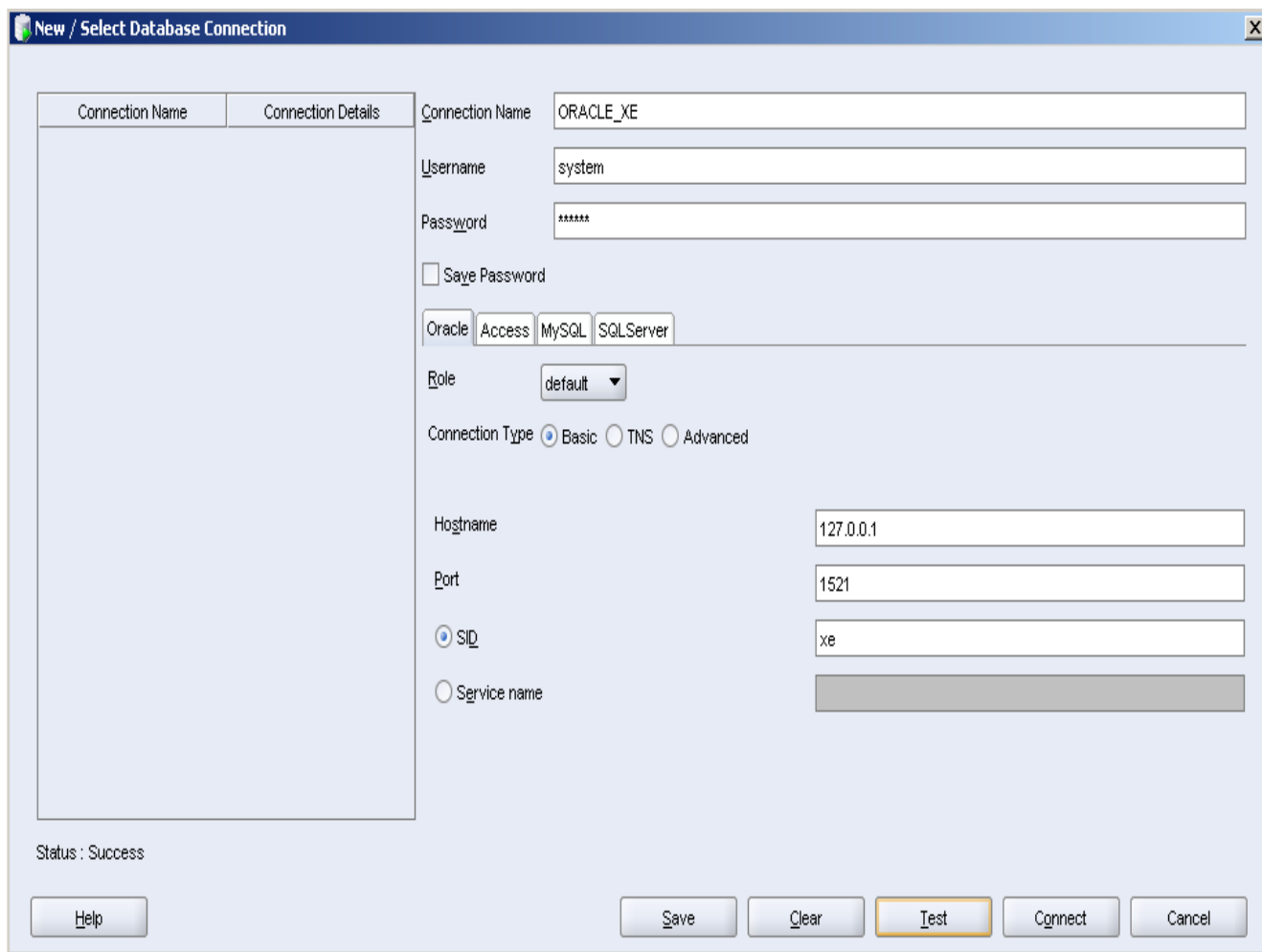


Рисунок 1.1 - Создание подключения

4. После успешного соединения с сервером Oracle и появления главного рабочего окна необходимо в навигаторе (в левой части окна) выбрать пункт **Other Users** и правой клавишей мыши вызвать для него контекстное меню. В контекстном меню выбрать элемент **Create User** (рис. 1.2).

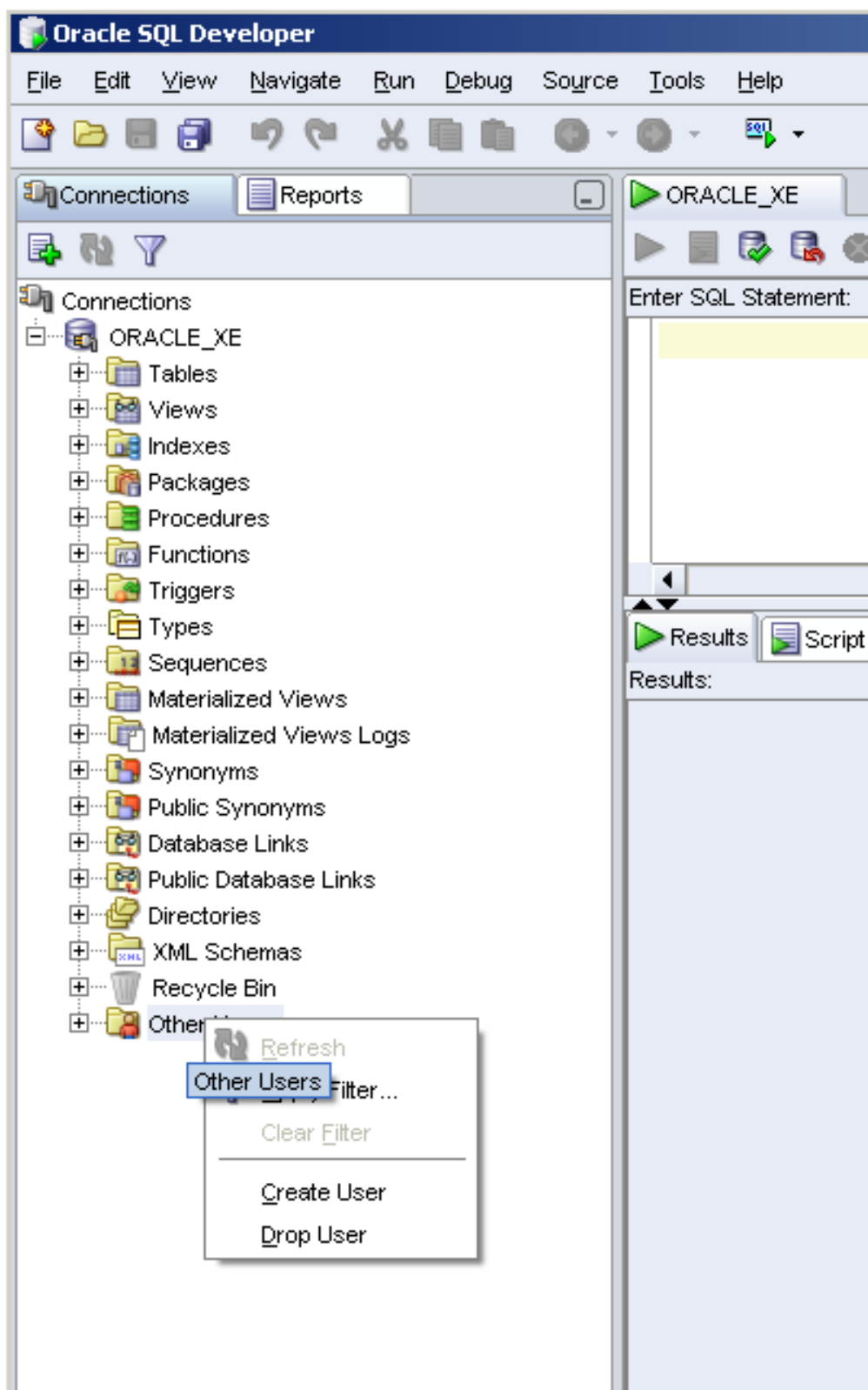


Рисунок 1.2 - Первый этап создания пользователя

5. В появившемся диалоговом окне (рис. 1.3) следует ввести имя пользователя в следующем формате: **AITnИмя** (без пробелов), где **n** – номер группы, а **Имя** – произвольное имя. Пароль должен полностью повторять имя

ПОЛЬЗОВАТЕЛЯ.

The screenshot shows the 'User Dialog' window with the following details:

- Tab:** User
- User Name:** AIT3STUDENT
- New Password:** [masked]
- Confirm Password:** [masked]
- Options:**
 - ☐ Password expired (user must change next login)
 - ☐ Account is Locked
- Default Tablespace:** USERS
- Temporary Tablespace:** TEMP
- Buttons:** Help, Apply, Close

Рисунок 1.3 - Настройка пользователя

6. Далее необходимо указать табличные пространства для данных и временных объектов (**USER** и **TEMP**). Табличные пространства – это логические области, где хранятся различные объекты базы данных. Под табличным пространством Default понимается пространство, где будут храниться объекты, принадлежащие конкретному пользователю, если их размещение не будет заранее задано. Физически табличное пространство представляет собой бинарный файл.

7. Далее, щелкнув на закладке **Roles**, предоставить (**Granted**) две роли: **Connect** и **Resource**. В закладке **System Privileges** предоставить системные привилегии **Create View** и **Unlimited tablespace**.

8. После всех действий нажать клавишу **Apply**. Пользователь создан. Теперь, нажав на закладку **SQL**, можно посмотреть SQL-скрипт, который был сгенерирован программой и выполнен для создания пользователя. Нажать

клавишу **Close**. Закрывать соединение ORACLE_XE (выбрав в его контекстном меню **Disconnect**).

9. Изменить соединение ORACLE_XE. Для этого выбрать в его контекстном меню пункт **Properties....** Вписать вместо имени пользователя **system** имя нового пользователя и указать его пароль. Осуществить соединение с базой данных под новым пользователем.

Задание 2. Создание таблиц базы данных. Необходимо создать те же таблицы, которые были спроектированы в курсовом проекте по дисциплине “Базы данных”.

Порядок выполнения задания 2.

1. Для создания таблицы необходимо в навигаторе выбрать элемент **Tables** и правой клавишей мыши вызвать для него контекстное меню. В меню выбрать пункт **New Table....** В открывшемся диалоговом окне (рис. 1.4) установить галочку **Advanced**.

2. В параметре **Schema** указывается схема, в которой будет создана данная таблица. Необходимо убедиться, что в качестве схемы выбрана схема вашего пользователя.

3. Далее следует указать имя таблицы, перейти на закладку **Columns** и создать поля будущей таблицы с **комментариями!** Особое внимание уделить галочке **Cannot be NULL**, отвечающей за обязательность полей.

4. На закладке **Primary Key** создать первичный ключ с именем в формате: **PK_Имя таблицы**. Для таблиц связи также необходимо создавать первичный ключ.

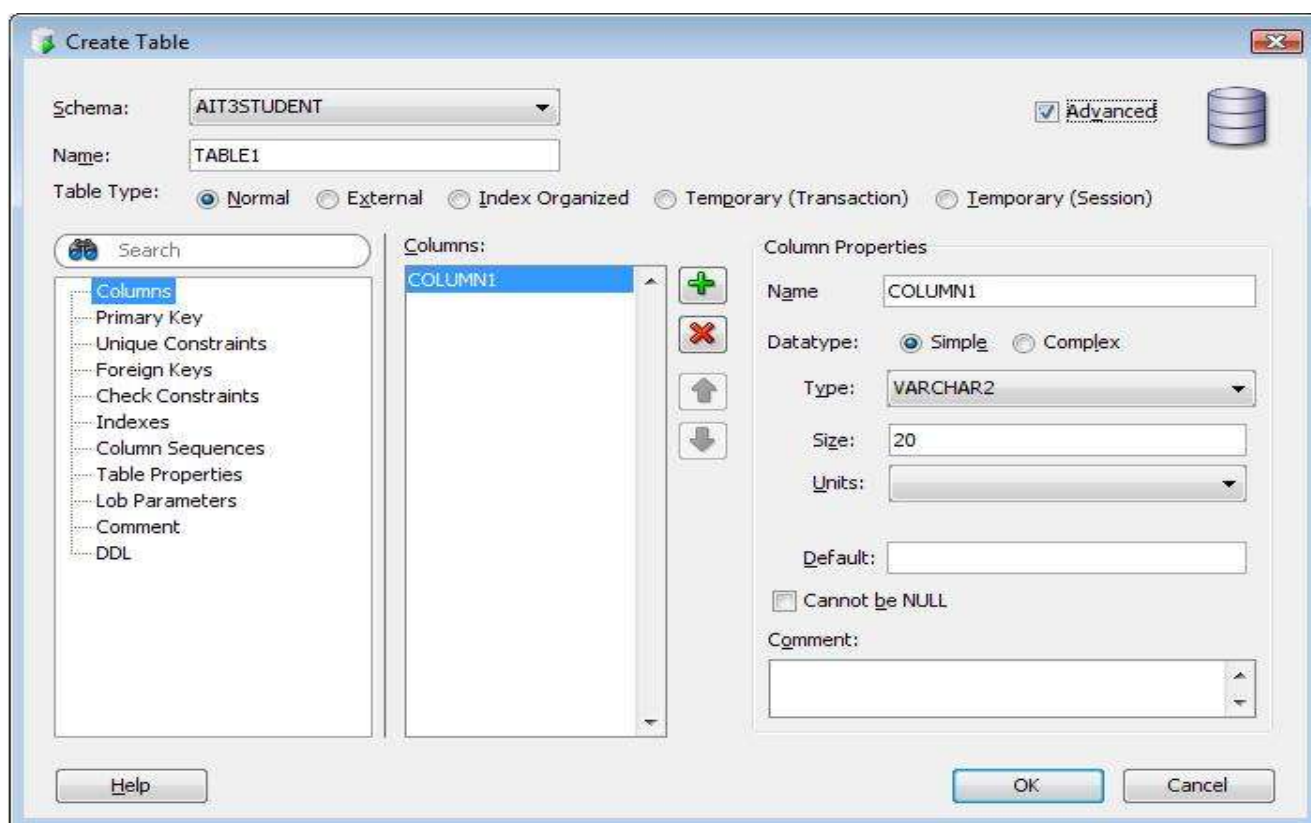


Рисунок 1.4 - Создание таблицы

5. На закладке **Foreign Keys** создать (если это необходимо для данной таблицы) внешние ключи с именами в формате **FK_Имя дочерней таблицы_Имя мастер-таблицы**. Мастер-таблица при этом должна быть уже создана. Если она еще не создана, то данный пункт пропускается, и внешние ключи создаются путем редактирования таблицы. Для этого необходимо вызвать контекстное меню, щелкнув по названию таблицы в навигаторе, и выбрать первый пункт **Edit**.

6. Необходимо следить за тем, чтобы имена внешних ключей не повторялись в пределах всей схемы. В параметре **Referenced table** указать таблицу, на которую ссылается внешний ключ (мастер-таблица). В колонке **Local Column** указать поле, которое будет внешним ключом. В колонке **Referenced Column** указать поле мастер-таблицы, на которое ссылается внешний ключ. Если необходимо, в параметре **On Delete** можно задать значение **CASCADE** для указания СУБД удалять связанные записи.

7. Перейти на закладку **DDL** и просмотреть скрипт, который автоматически был сгенерирован программой для создания таблицы. Нажать кнопку **ОК**. Таблица создана.

8. После создания таблицы рекомендуется открыть ее заново, вызвав меню правой клавишей мыши и выбрав пункт **Edit**, и просмотреть параметры хранения. Для этого перейти на закладку **Table Properties** и нажать кнопку **Storage Options...**

9. Задание необходимо проделать для всех таблиц, созданных в курсовом проекте.

Задание 3. Для каждой таблицы схемы создать обзоры.

Порядок выполнения задания 3.

1. Следует в навигаторе выбрать элемент **Views** и правой клавишей мыши вызвать для него контекстное меню. В меню выбрать пункт **New View...** В появившемся диалоговом окне задать имя обзора в следующем формате: **V_имя таблицы**.

2. В скрипте обзора (область **SQL Query**) написать запрос на выборку всех полей (с поименным указанием) из таблицы, для которой создается обзор. Если в таблице есть внешние ключи, то запрос должен быть сложным (по двум или более таблицам). Вместо значений внешнего ключа должны отображаться смысловые значения из связанной мастер-таблицы. Для полей типа **DATE** применять маску: 'dd.mm.yyyy'. Для наложения маски использовать оператор **TO_CHAR(D1,mask)** где D1 – переменная типа DATE, mask – строковая маска.

3. Можно протестировать правильность синтаксиса запроса нажатием клавиши **Test Syntax**. После успешной проверки зайти на закладку **DDL** и просмотреть автоматически сгенерированный скрипт создания обзора. Нажатием клавиши **ОК** создать обзор в базе данных.

4. Также для создания обзора можно воспользоваться интерактивным мастером построения, в который можно переключиться, включив галочку **Advanced** (в правом верхнем углу окна).

3 Контрольные вопросы

1. Для чего предназначены табличные пространства **USER (default tablespace)** и **TEMP (temporary tablespace)**?
2. В чем заключается назначение оператора **GRANT**?
3. Почему сумма значений **Pct Free** и **Pct Used** не должна быть меньше 100%?
4. Что произойдет, если количество параллельных транзакций превысит значение **Ini Trans**?
5. Объяснить назначение в скрипте по созданию таблицы параметров структуры **Storage Options**.
6. Объяснить, какие действия выполняет оператор **alter table** в скрипте по созданию таблицы.
7. Какая таблица называется мастер-таблицей? Что такое каскадное удаление? Описать действия, которые могут выполняться при удалении записи из мастер-таблицы в СУБД Oracle.

Лабораторная работа №2. Создание последовательностей, триггеров и организация заполнения таблиц данными

Цель работы - научиться создавать последовательности (Sequence), триггеры (Trigger) и заполнять таблицы данными, пользуясь оператором Insert и созданной системой триггеров.

1. Основные положения

Последовательность – объект базы данных, посредством которого пользователи могут генерировать уникальные целые значения. Как правило, последовательности используют для автоматической генерации уникальных ключей.

Значения для каждой последовательности автоматически генерируются специальными подпрограммами ORACLE и, следовательно, позволяют избежать неизбежных потерь в производительности в результате реализации последовательностей на уровне приложений. Например, обычным способом такой реализации является блокирование таблицы, хранящей номер, в каждой транзакции в момент приращения последовательности. В результате только один номер последовательности может быть сгенерирован в каждый момент времени. Последовательности же ORACLE позволяют одновременное генерирование нескольких номеров последовательности, гарантируя при этом, что каждый номер последовательности будет уникальным.

Когда генерируются номера последовательности, осуществляется приращение этой последовательности независимо от того, завершается эта транзакция или отменяется. Если два пользователя параллельно осуществляют приращение одной и той же последовательности, номера последовательности, видимые каждым из этих пользователей, могут иметь пропуски значений, поскольку эти значения были сгенерированы другим пользователем. Одному пользователю никогда не может встретиться номер последовательности, сгенерированный другим пользователем. Если значение последовательности

сгенерировано каким-либо пользователем, этот пользователь может использовать это значение независимо от того, осуществляется ли приращение последовательности другими пользователями.

Так как номера последовательности генерируются независимо от таблиц, одна и та же последовательность может использоваться для одной или более таблиц. Возможно, что отдельные номера последовательности будут вообще пропущены из-за того, что они могли быть сгенерированы и использованы в транзакции, которая, в конечном счете, была отменена.

Если в операторе **CREATE SEQUENCE** кроме имени не указать ни одного предложения, то по умолчанию создается возрастающая последовательность, которая начинается с 1, имеет шаг 1 и не имеет верхнего предела. Если указать только предложение **INCREMENT BY -1**, то создается убывающая последовательность, которая начинается с -1 и не имеет нижнего предела.

Можно создать последовательность так, чтобы приращение ее значений происходило одним из следующих способов:

- приращение значений последовательности продолжается неограниченно;
- приращение значений последовательности происходит до определенного предела и затем прекращается;
- приращение значений последовательности происходит до определенного предела и затем начинается заново.

Чтобы создать последовательность, приращение значений которой продолжается неограниченно, необходимо опустить параметр **MAXVALUE** или указать параметр **NOMAXVALUE** для возрастающей последовательности. Для убывающей последовательности опустить параметр **MINVALUE** или указать параметр **NOMINVALUE**.

Чтобы создать последовательность, приращение значений которой прекращается на определенном значении, необходимо указать значение для параметра **MAXVALUE** (если последовательность возрастающая) или значение для параметра **MINVALUE** (если последовательность убывающая). При этом необходимо указать также параметр **NOCYCLE**. Любая попытка сгенерировать

номер последовательности, если последовательность уже достигла своего предела, приведет к ошибке.

Чтобы создать последовательность, приращение значений которой начинается заново после достижения определенного значения, необходимо указать значения для обоих параметров, **MAXVALUE** и **MINVALUE**, а также параметр **CYCLE**.

Значение параметра **START WITH** устанавливает начальное значение, генерируемое последовательностью, после того как ее создали. Следует обратить внимание, что это не обязательно именно то значение, с которого циклическая последовательность начинается заново после достижения своего максимального или минимального значения.

Количество кэшируемых значений для последовательности определяется значением параметра **CACHE**. Кэшированные последовательности обеспечивают быструю генерацию номеров последовательности. Кеш для данной последовательности заполняется при первом запросе номера этой последовательности. Кеш регенерируется после количества запросов, указанного в параметре **CACHE**. В случае сбоя системы все копированные номера последовательности, которые не были использованы в завершенных операторах языка манипулирования данными, теряются. Возможное количество потерянных значений равно величине параметра **CACHE**. По умолчанию **ORACLE** кэширует 20 номеров последовательности.

Триггер базы данных (database trigger) – это хранимый блок **PL/SQL**, который ассоциирован с таблицей. В триггере посредством **PL/SQL** описаны действия, которые необходимо выполнить при обработке операторов **SQL** (таких как **Insert**, **Update**, **Delete** и т.д.) для данной таблицы. **Oracle** автоматически выполняет триггер, во время обработки оператора **SQL**.

2 Содержание работы

Задание 1. Создать последовательности для автоматического заполнения ключевых полей при добавлении новых записей.

Порядок выполнения задания 1.

1. Для создания сиквенса необходимо в навигаторе найти пункт **Sequences** и правой клавишей мыши вызвать для него контекстное меню. В меню выбрать пункт **New Sequence....** В появившемся диалоговом окне (рис. 2.1) указать имя сиквенса в следующем формате: **S_ИМЯ**, где ИМЯ – это название таблицы, для которой создается сиквенс.

2. Далее необходимо заполнить остальные поля в окне (рис. 2.1).

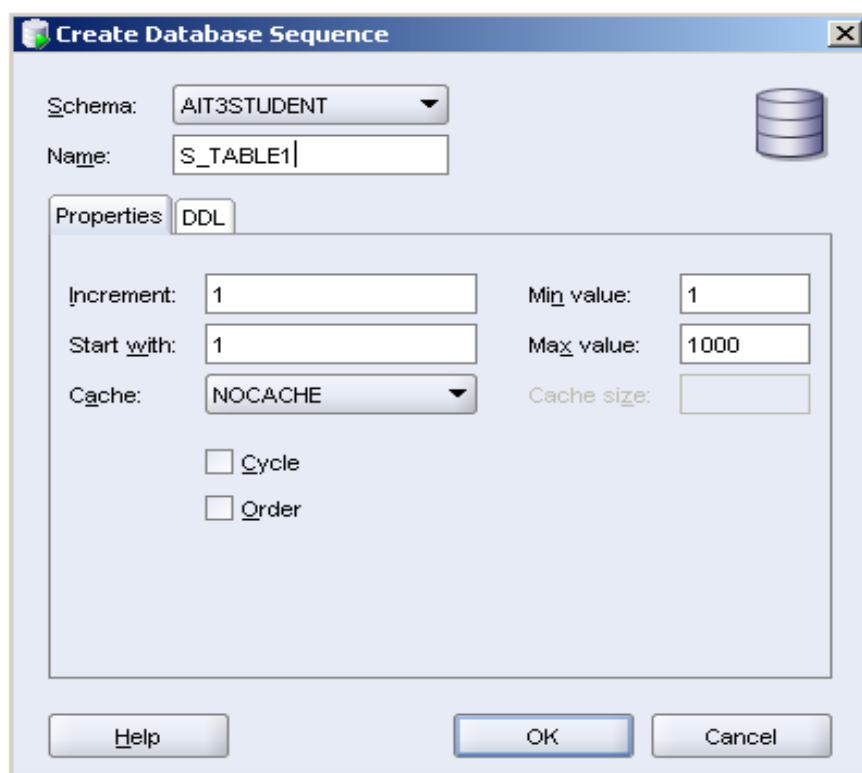


Рисунок 2.1- Создание последовательности

Min value (MINVALUE) – указывает минимальное значение последовательности (как правило, = 1).

Max value (MAXVALUE) – указывает максимальное значение последовательности, которое она может сгенерировать.

Start with (START WITH) – указывает значение первого генерируемого номера последовательности. Для возрастающей последовательности значением по умолчанию для этого параметра является ее минимальное значение, для убывающей последовательности – ее максимальное значение.

Increment (INCREMENT BY) – указывает интервал между числами последовательности. Это значение может быть любым положительным или отрицательным целым числом в стандарте Oracle, но не может быть нулем. Если значение отрицательное, это – убывающая последовательность, если положительное – возрастающая последовательность. Если в поле ничего не указывать, значение будет принято равным 1.

Cache size (CACHE) – указывает, как много значений последовательности Oracle генерирует заранее и держит в памяти для быстрого доступа. Минимальное значение для этого параметра – 2. Для циклических последовательностей это значение должно быть меньше, чем количество значений в цикле. Пример записи в скрипте: **CACHE 10**. Если в данном поле указать 0, то это будет означать отсутствие кэширования значений последовательности (пример записи в скрипте: **NOCACHE**). Можно также выбрать **NOCACHE** в поле Cache.

Cycle (CYCLE, NOCYCLE) – указывает, что последовательность продолжает циклически генерировать значения после достижения своего минимального или максимального значения. После того как возрастающая последовательность достигла своего максимального значения, она генерирует свое минимальное значение. Для убывающей последовательности – наоборот.

Order (ORDER, NOORDER) – гарантирует, что номера последовательности генерируются в порядке поступления запросов. Этот параметр может быть полезным при использовании номеров последовательности в качестве временных отметок. Гарантирование порядка, как правило, не является важным условием для последовательностей, используемых для генерирования первичных ключей (по умолчанию – **NOORDER**).

3. После заполнения соответствующих полей рекомендуется просмотреть автоматически созданный скрипт посредством перехода на закладку **DDL**. Далее необходимо нажать на кнопку **OK**, и автоматически сгенерированный скрипт по созданию последовательности выполнится. Задание необходимо повторить для всех таблиц, где есть первичные ключи!

Задание 2. Создать систему триггеров для таблиц базы данных.

Порядок выполнения задания 2.

1. Для создания триггера необходимо в навигаторе найти пункт **Triggers** и правой клавишей мыши вызвать для него контекстное меню. В меню выбрать пункт **New Trigger....** В появившемся диалоговом окне заполнить поля (рис. 2.2). Имя триггера должно быть в следующем формате: **T_Имя таблицы**.

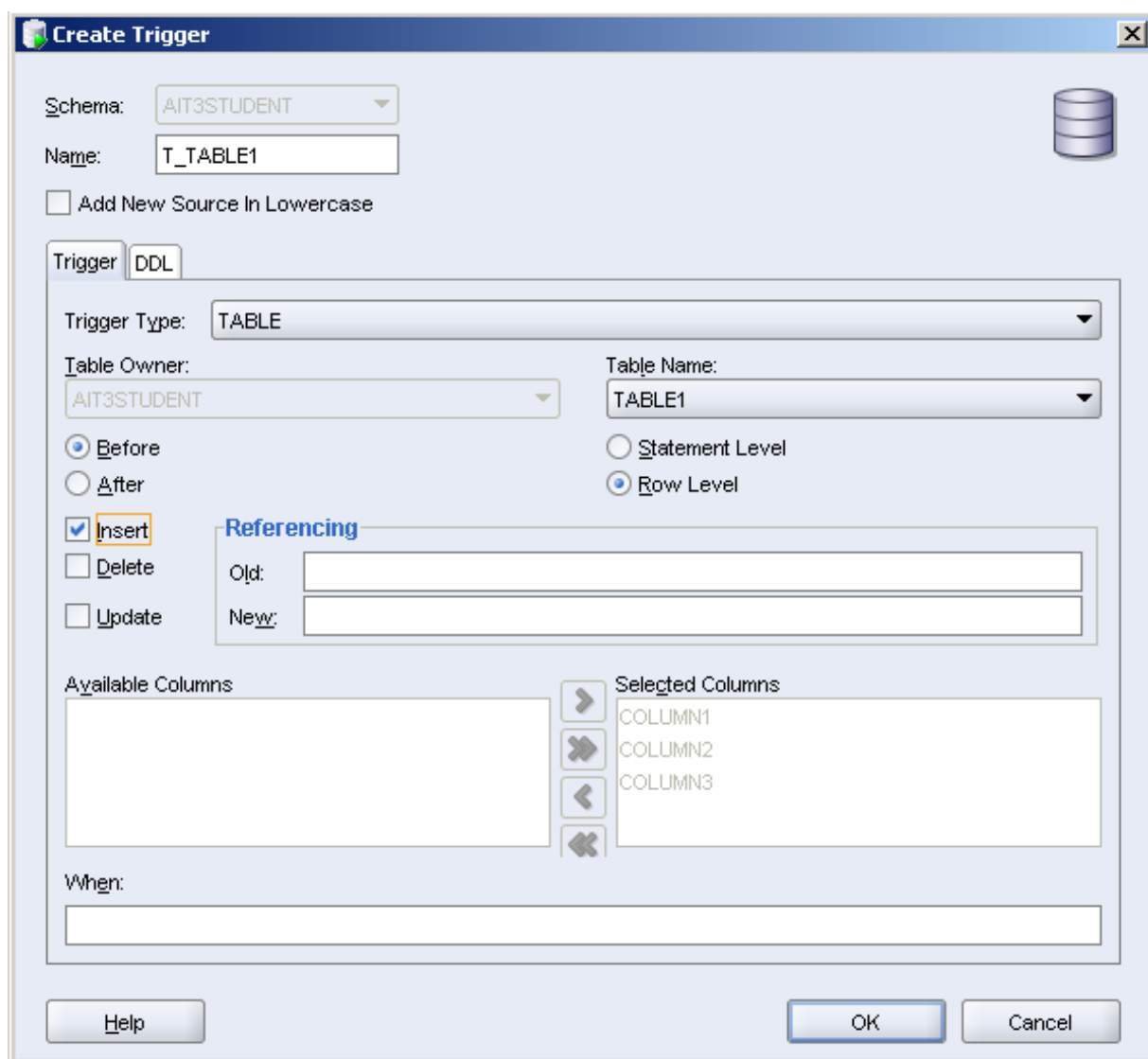


Рисунок 2.2 – Создание триггера

Trigger type – вид триггера. Триггер может быть на таблицу, обзор, схему или базу данных. В нашем случае необходимо указать TABLE, то есть табличный триггер.

Table Name – имя таблицы, с которой будет ассоциироваться триггер, т.е. таблица, для которой мы собственно и создаем триггер.

Before – указывает, что триггер срабатывает до выполнения оператора, вызывающего этот триггер.

After – указывает, что триггер срабатывает после выполнения оператора, вызывающего этот триггер.

Delete – указывает, что триггер срабатывает всегда, когда какой-либо оператор DELETE удаляет строки из ассоциированной с триггером таблицы.

Insert – указывает, что триггер срабатывает всегда, когда какой-либо оператор INSERT добавляет строки к ассоциированной с триггером таблице.

Update – указывает, что триггер срабатывает всегда, когда какой-либо оператор UPDATE изменяет значение какого-либо столбца ассоциированной с триггером таблицы.

В данном примере рассматривается следующая комбинация: Before Insert, т.е. мы будем обрабатывать событие перед вставкой новой записи.

Заполнив указанные поля, нажать клавишу **OK**.

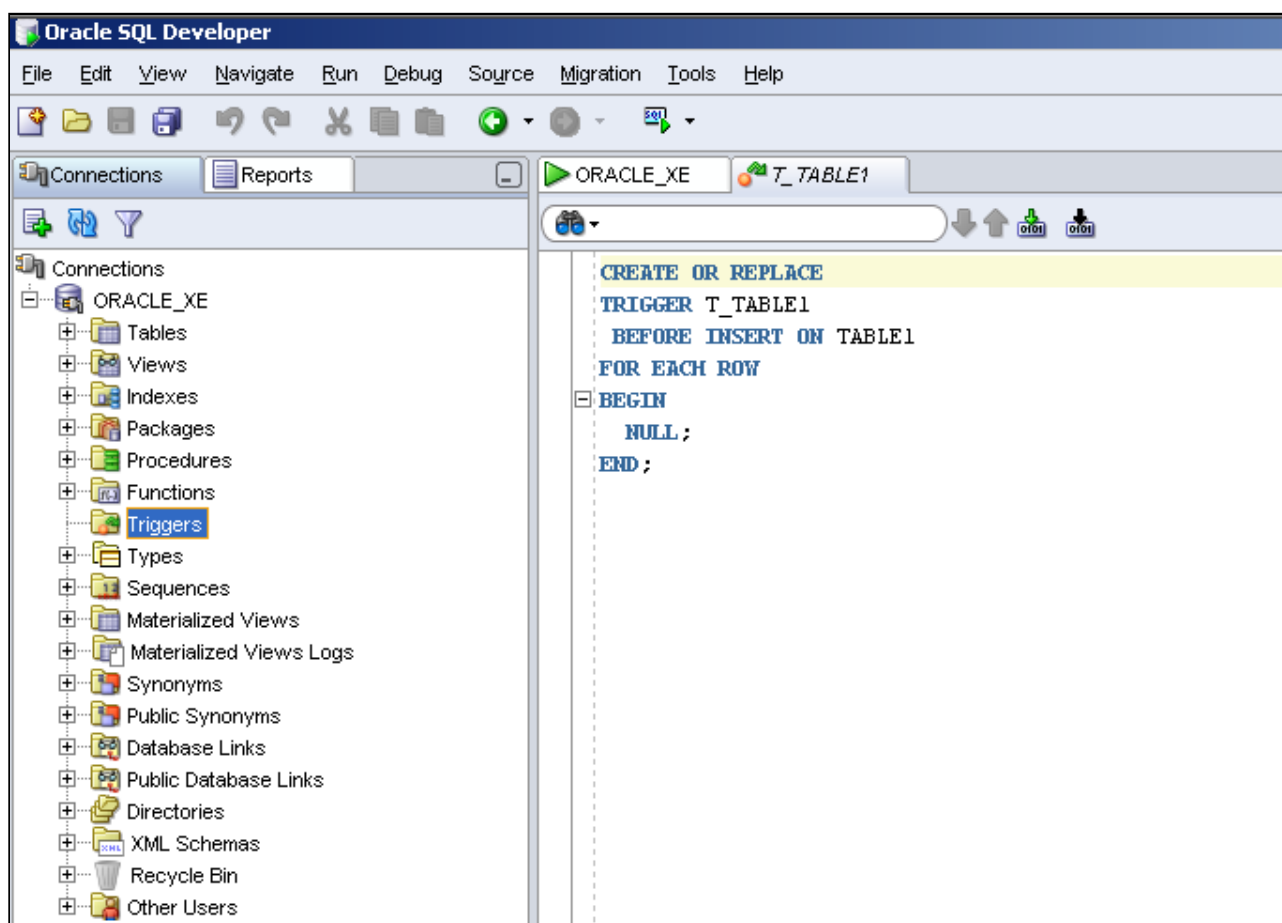


Рисунок 2.3 – Пример шаблона тела триггера

2. После этого откроется окно редактирования с загруженным в него шаблоном будущего триггера (рис. 2.3). Этот шаблон необходимо отредактировать, добавив туда текст PL/SQL.

Задание 3. Создать систему хранения данных аудита.

Порядок выполнения задания 3.

1. В каждой таблице необходимо создать два поля: ID_CRE (тип varchar2) и DAT_CRE (тип date). В эти поля при вставке каждой записи автоматически должны прописываться имя текущего пользователя и текущая дата. Также необходимо предусмотреть вставку значения первичного ключа для таблиц, где он присутствует. Вернемся теперь к редактору: ключевое слово **FOR EACH ROW** указывает, что данный триггер – строковый. Триггер срабатывает один раз для каждой строки (записи), на которую действует оператор, запускающий триггер, и которая удовлетворяет условиям ограничений, определенных для этого триггера в предложении **WHEN**. В нашем случае это предложение опущено, значит, триггер действует для каждой строки (записи).

При необходимости можно перед ключевым словом **BEGIN** вставить ключевое слово **DECLARE**, после которого можно декларировать локальные переменные (с типами данных Oracle). В данном случае нам это не потребуется.

2. Теперь в разделе **BEGIN – END** вместо оператора **NULL**; необходимо написать скрипт на PL/SQL, выполняющий нужные нам действия по вставке значений данных в поля. Рассмотрим, как это сделать на примере таблицы STUDENT. Предположим, для таблицы уже создан сиквенс с названием S_STUDENT. Скрипт PL/SQL, описывающий нашу задачу, будет выглядеть следующим образом:

```
select S_STUDENT.nextval into :new.STUDENT_ID from dual;
:new.ID_CRE:=user;
:new.DAT_CRE:=sysdate;
```


Первая строка заполняет значение ключевого поля, значение извлекается из последовательности S_STUDENT. **NEXTVAL** возвращает очередное значение последовательности а dual – это зарезервированное слово, означающее псевдотаблицу и необходимое для завершенности синтаксиса оператора Select.

NEW – это параметр (зарезервированное ключевое слово), означающий новую строку, поэтому обращение к нему идет через двоеточие, далее через точку идет обращение к нужным нам полям этой строки.

USER – системная переменная, возвращающая имя текущего пользователя в строковом формате.

SYSDATE – системная переменная, возвращающая текущую дату.

Таким образом, соответствующие поля в новой (вставляемой) строке будут заполнены, и после окончания работы триггера в таблицу добавится полностью сформированная и заполненная строка (**оставшиеся поля должны будут заполняться в операторе Insert**)!

3. После окончания редактирования скрипта триггера его необходимо скомпилировать: (Ctrl+Shift+F9) или кнопка  на панели инструментов. При успешной компиляции триггер добавится к ассоциированной с ним таблице. Его можно будет просмотреть в редакторе таблицы на закладке **Triggers** или в одноименном пункте навигатора.

4. Задание следует выполнить для каждой таблицы. Для таблиц, где есть первичный ключ, в триггер нужно добавлять строчку для заполнения ключевого поля. Необходимо создать триггеры двух типов, реагирующих на вставку записи (Insert) и на ее изменение (Update). При изменении записи в триггере должны заполняться только поля ID_CRE и DAT_CRE.

Задание 4. Организовать заполнение таблиц данными.

Порядок выполнения задания 4.

1. Создать несколько скриптов с операторами Insert. При написании оператора Insert поля, которые являются первичными ключами, и поля ID_CRE и

DAT_CRE заполнять не следует – они будут заполняться автоматически триггерами. Создать не более 5-ти операторов для добавления записей по каждой таблице. Просмотреть, как работает система триггеров и как формируются значения первичных ключей.

3 Контрольные вопросы

1. Для чего нужны последовательности?
2. Объяснить назначение параметров при создании сиквенса. Каково назначение параметра **Cache**?
3. Для чего предназначены триггеры?
4. Объяснить смысл скрипта PL/SQL в теле любого из созданных триггеров.

Лабораторная работа №3. Разработка функций и процедур для базы данных

Цель работы заключается в овладении навыками работы с языком PL/SQL. Научиться создавать хранимые процедуры (Procedure) и функции (Function).

1. Основные положения

Хороший способ познакомиться с PL/SQL – это посмотреть на образец программы. Приведенная ниже программа обрабатывает заказ на теннисные ракетки. Сначала в ней объявляется переменная типа NUMBER, куда будет считано из базы данных количество имеющихся теннисных ракеток. Затем это количество извлекается из таблицы с именем *Inventory*. Если число ракеток больше нуля, то программа обновляет таблицу и вставляет запись о покупке в другую таблицу с именем *Order_record*. В противном случае в таблицу *Order_record* вставляется запись об исчерпании запаса ракеток.

```
DECLARE
```

```
    in_inventory NUMBER(5);
```

```
BEGIN
```

```
    SELECT quantity INTO in_inventory FROM Inventory
```

```
    WHERE product = 'ТЕННИСНАЯ ПАКЕТКА';
```

```
    IF in_inventory > 0 THEN -- проверка количества
```

```
        UPDATE Inventory SET quantity = quantity - 1
```

```
        WHERE product = 'ТЕННИСНАЯ ПАКЕТКА';
```

```
        INSERT INTO Order_record
```

```
        VALUES ('Теннисная ракетка заказана', SYSDATE);
```

```
    ELSE
```

```
        INSERT INTO Order_record
```

```
        VALUES ('Теннисные ракетки закончились', SYSDATE);
```

```
    END IF;
```

```
    COMMIT;
```

END;

В PL/SQL можно использовать команды SQL для манипулирования данными ORACLE и операторы потока управления для обработки данных. Кроме того, можно объявлять константы и переменные, определять подпрограммы (процедуры и функции) и ставить ловушки для ошибок периода исполнения. Таким образом, PL/SQL объединяет силу SQL в манипулировании данными и мощь процедурных языков в их обработке.

1.1.1 Структура блока

PL/SQL – язык с блочной структурой, т.е. базовые компоненты (процедуры, функции и неименованные блоки), из которых состоит программа на PL/SQL, представляют собой логические блоки, которые могут содержать любое число рекурсивно вложенных подблоков. Обычно каждый логический блок отвечает за решение некоторой задачи или ее подзадачи. Таким образом, PL/SQL поддерживает подход «разделяй и побеждай» к решению задач, называемый *пошаговым уточнением*.

Блок (или подблок) позволяет собрать вместе логически связанные объявления и операторы. Таким образом, вы можете размещать объявления в непосредственной близости от места их использования. Объявления локальны в блоке и перестают существовать, когда блок завершается.

Ниже показано, что блок PL/SQL состоит из трех частей: раздел объявлений, исполняемый раздел и раздел обработки исключений. (В PL/SQL состояние, вызывающее ошибку или предупреждение, называется *исключением*.) Обязательным является лишь исполняемый раздел.

```
[DECLARE  -- объявления]
```

```
BEGIN
```

```
    -- операторы
```

```
[EXCEPTION -- обработчики]
```

```
END;
```

Порядок частей обусловлен логически. Сначала идет раздел объявлений, где объекты могут быть объявлены. После объявления с объектами можно работать в исполняемом разделе. Возникшие при выполнении исключения могут быть обслужены разделом обработки исключений.

Исполняемый раздел и раздел обработки исключений блока или подпрограммы на PL/SQL могут содержать рекурсивно вложенные подблоки, а раздел объявлений – нет. Зато в разделе объявлений любого блока могут быть определены локальные подпрограммы.

Однако вызывать локальные подпрограммы можно только из блока, где они определены.

1.1.2 Переменные и константы

PL/SQL позволяет объявлять переменные и константы и затем использовать их в командах SQL и процедурных операторах в любом месте, где может быть использовано выражение. Однако ссылки вперед не разрешаются. Так что необходимо объявить переменную или константу *до* ссылки на нее в других операторах, в том числе и в других объявлениях.

Переменные могут принадлежать к любому типу данных SQL, как, например, VARCHAR2, DATE, NUMBER, или к любому типу данных PL/SQL, например BOOLEAN, BINARY и INTEGER. Предположим, что вы хотите объявить переменную с именем *part_no* для хранения 4-разрядных целых чисел и переменную с именем *in_stock* для хранения булевых значений TRUE или FALSE. Вы можете объявить эти переменные так:

```
part_no  NUMBER(4);  
in_stock BOOLEAN;
```

Можно также объявлять записи и таблицы PL/SQL, используя составные типы данных RECORD и TABLE.

Значения переменной можно присваивать двумя способами. В первом способе используется оператор присваивания (:=), двоеточие и сразу за ним знак равенства.

Переменная помещается слева от оператора, а выражение – справа. Ниже приводятся несколько примеров:

```
tax := price * tax_rate;  
bonus := current_salary * 0.10;  
raise := TO_NUMBER(SUBSTR('750 raise', 1, 3));  
valid := FALSE;
```

Второй способ присваивания значений переменной – это SELECT или FETCH значения из базы данных прямо в нее. В следующем примере ORACLE вычисляет премию в 10%, когда вы делаете выборку оклада служащего:

```
SELECT sal*0.10 INTO bonus FROM emp  
WHERE empno=emp_id;
```

Переменную *bonus* можно затем использовать в других вычислениях или поместить значение в таблицу базы данных.

Объявление константы происходит аналогично объявлению переменной, но здесь вы должны добавить ключевое слово **CONSTANT** и сразу присвоить значение константе. С этого момента никакие присваивания константе больше не допускаются. В следующем примере объявляется константа с именем *minimum_balance*:

```
miniraura_balance  CONSTANT REAL := 10.00;
```

1.1.3 Атрибуты

Переменные и константы в PL/SQL имеют *атрибуты* – свойства, которые позволяют ссылаться на тип данных и структуру объекта, не повторяя его определение. Таблицы и столбцы в базе данных имеют аналогичные атрибуты, которые можно использовать для того, чтобы облегчить сопровождение.

Атрибут **%TYPE** представляет тип данных переменной, константы или столбца базы данных. Это, в частности, полезно при объявлении переменной, которая ссылается на столбец базы данных. Предположим, например, что в таблице с именем *books* есть столбец с именем *title*. Чтобы объявить переменную с именем *my_title*, принадлежащую к тому же типу, что и столбец *title*, вы используете точечную нотацию и атрибут **%TYPE** следующим образом:

```
my_title books.title%TYPE
```

Объявление *my_title* с помощью **%TYPE** имеет два преимущества. Первое – не нужно знать точный тип данных *title*. Второе – если в базе данных меняется определение *title* (например, увеличивается размер строки символов), то соответственно изменится и тип данных *my_title* при выполнении программы.

В PL/SQL записи используются для объединения данных в группы. Запись состоит из ряда полей, в которые могут заноситься значения данных. При помощи атрибута **%ROWTYPE** можно получить тип записи, которая будет представлять строку таблицы. Запись может содержать целиком строку данных, выбранную из таблицы или извлеченную с помощью курсора (обсуждается позднее).

Столбцы строки и соответствующие поля записи имеют одинаковые имена и типы данных. В примере, приведенном ниже, объявляется запись с именем *dept_rec*. Ее поля имеют те же имена и типы данных, что и столбцы таблицы *dept*.

```
DECLARE  
dept_rec dept%ROWTYPE;  
...
```

Для доступа к полям записи можно использовать точечную нотацию, как показывает следующий пример:

```
my_deptno := dept_rec.deptno;
```

Если объявлен курсор для выборки фамилии, оклада, даты поступления и должности служащих, то можно использовать %ROWTYPE для объявления записи, которая будет содержать ту же информацию, следующим образом:

DECLARE

```
CURSOR c1 IS SELECT ename, sal, hiredate, job FROM emp;
emp_rec c1%ROWTYPE;
```

Когда выполняется оператор `FETCH c1 INTO emp_rec;` значение столбца *ename* таблицы *emp* присваивается полю *ename* записи *emp_rec*, значение столбца *sal* присваивается полю *sal* и т.д. Ниже показан возможный результат (табл. 3.1).

Таблица 3.1 - Пример заполнения таблицы

emp_rec	
emp_rec.ename	Иван
emp_rec.sal	950.00
emp_rec.hiredate	03.12.2001
emp_rec.job	Клерк

1.1.4 Управляющие структуры

Управляющие структуры – самое важное в PL/SQL расширение по сравнению с SQL. PL/SQL позволяет не только манипулировать данными ORACLE, но и обрабатывать данные, используя операторы условного, циклического и последовательного потока управления, такие как IF-THEN-ELSE, FOR-LOOP, WHILE-LOOP, EXIT-WHEN и GOTO. В совокупности эти операторы могут обработать любую ситуацию.

Часто приходится предпринимать альтернативные действия в зависимости от обстоятельств. Оператор IF-THEN-ELSE позволяет выбирать последовательность выполнения действий в зависимости от условия. Фраза IF проверяет условие; фраза THEN определяет, что делать, если условие истинно; фраза ELSE определяет, что делать, если условие ложно или недействительно.

Рассмотрим приведенную ниже программу, которая обрабатывает банковскую транзакцию. Прежде чем позволить снять \$500 со счета 3, она должна удостовериться, что денег на счете достаточно, чтобы покрыть расход. Если денег хватает, программа снимает сумму со счета; в противном случае вносит запись в таблицу для ревизии счетов.

```
DECLARE
```

```
    acct_balance NUMBER(11, 2) ;
```

```
    acct      CONSTANT NUMBER(4) := 3;
```

```
    debit_amt  CONSTANT NUMBER(5,2) := 500.00;
```

```
BEGIN
```

```
    SELECT bal INTO acct_balance FROM accounts
```

```
    WHERE account_id = acct FOR UPDATE OF bal;
```

```
    IF acct_balance >= debit_amt THEN
```

```
        UPDATE accounts SET bal = bal - debit_amt
```

```
        WHERE account_id = acct;
```

```
    ELSE
```

```
        INSERT INTO temp VALUES
```

```
        (acct, acct_balance, 'Insufficient funds');
```

```
        -- включить номер счета, текущий баланс и сообщение
```

```
    END IF;
```

```
    COMMIT;
```

```
END;
```

Последовательность операторов, которая использует результаты запроса, чтобы выбрать альтернативные действия, типична для приложений баз данных. Другим примером типовых действий является вставка или удаление строки при условии, что в другой таблице найдена строка, связанная по содержанию с данной. Используя условные операторы, можно собрать эти типовые цепочки действий в блок PL/SQL. Это может повысить эффективность работы и упростить проверки целостности.

1.1.5 Циклы

Оператор LOOP позволяет многократно выполнить последовательность операторов. Ключевое слово LOOP должно располагаться перед первым оператором последовательности, а ключевые слова END LOOP – за последним оператором. Следующий пример показывает простейшую форму цикла, который все время повторяет последовательность операторов:

```
LOOP
-- последовательность операторов
END LOOP;
```

Оператор FOR-LOOP позволяет указать диапазон целых чисел и выполнить последовательность операторов один раз для каждого числа из диапазона. Предположим, например, что вы – производитель заказных автомобилей и что каждый автомобиль имеет серийный номер.

Для каждого автомобиля в учетную ведомость продаж необходимо внести заказчика. Это можно сделать, используя следующий цикл FOR:

```
FOR i IN 1..order_qty LOOP
UPDATE sales SET custno = customer_id
WHERE snum = snum_seq.NEXTVAL; END LOOP;
```

Оператор WHILE-LOOP связывает с последовательностью операторов некоторое условие. Перед каждым повторением цикла условие вычисляется. Если результатом является TRUE, то выполняется последовательность операторов, и управление возвращается к началу цикла. Если же значением условия является FALSE или NULL, цикл обходится, и управление передается на следующий оператор.

В следующем примере ищется ближайший в иерархии подчиненности руководитель служащего с номером 7902, имеющий оклад не менее \$4000:

```

DECLARE
    salary      emp.sal%TYPE;
    mgr_num     emp.mgr%TYPE;
    last_name   emp.ename%TYPE;
    starting_empno CONSTANT NUMBER(4) := 7902;
BEGIN
    SELECT sal, mgr INTO salary, mgr_num FROM emp
    WHERE empno = starting_empno;
    WHILE salary < 4000 LOOP
        SELECT sal, mgr, ename INTO salary, mgr_num, last_name FROM
        emp
        WHERE empno = mgr num;
    END LOOP;
    INSERT INTO temp VALUES (NULL, salary, last_name);
    COMMIT;
END;

```

Оператор EXIT-WHEN позволяет закончить цикл, если дальнейшая обработка нежелательна или невозможна. Когда встречается оператор EXIT, то вычисляется условие во фразе WHEN. Если результатом является TRUE, цикл заканчивается и управление передается на следующий оператор. В следующем примере цикл закончится, когда значение *total* превысит 25000:

```

LOOP
    ...
    total = total + salary;
    EXIT WHEN total > 25000;
    -- ВЫЙТИ ИЗ ЦИКЛА, ЕСЛИ УСЛОВИЕ ИСТИННО
END LOOP;
-- сюда передается управление

```

Оператор GOTO позволяет безусловный переход на метку. Метка – необъявленный идентификатор, заключенный в двойные угловые скобки, – должна стоять перед выполняемым оператором или блоком PL/SQL. При исполнении оператора GOTO управление передается помеченному оператору или блоку, как показано в следующем примере:

```
IF rating > 90 THEN
  GOTO calc_raise; -- переход на метку
END IF;
...
«calc_raise»
IF job_title = 'SALESMAN' THEN -- управление передается сюда
  raise := commission * 0.25;
ELSE
  raise := salary * 0.10; END IF;
```

1.1.6 Курсоры

Для выполнения команд SQL и хранения обрабатываемой информации ORACLE использует рабочие области, называемые *приватными областями SQL*. Конструкция PL/SQL, называемая *курсором*, позволяет поименовать приватную область SQL и получить доступ к хранящейся в ней информации. Существует два вида курсоров: *явные* и *неявные*. PL/SQL неявно объявляет курсор для всех команд SQL, манипулирующих данными, включая даже запросы, возвращающие всего одну строку. Для запросов, возвращающих более одной строки, можно объявить явный курсор, чтобы обрабатывать строки каждую по отдельности. Пример:

```
DECLARE
CURSOR cl IS
  SELECT empno, ename, job FROM emp WHERE deptno = 20;
```

Множество строк, возвращаемое многострочным запросом, называется *активным набором*. Его размер определяется числом строк, удовлетворяющих критерию поиска. Как показано на рис. 3.1, явный курсор «указывает» на *текущую строку* в активном наборе. Это позволяет программе обрабатывать строки каждую в отдельности. Рассмотрим это на примере запроса:

```
SELECT empno, ename, job FROM emp WHERE deptno = 20;
```

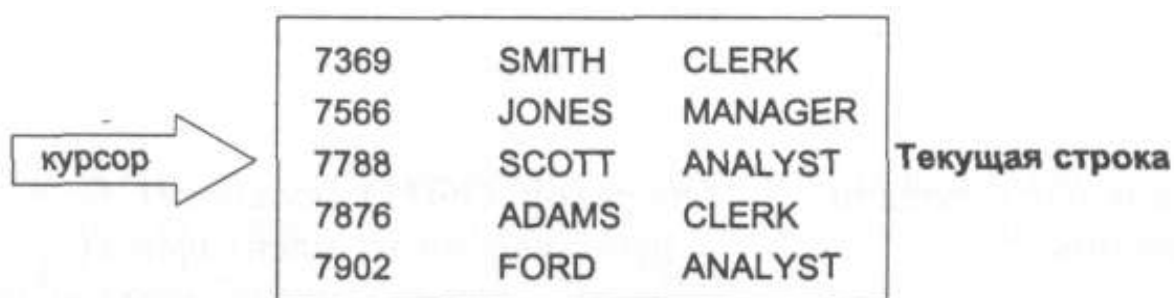


Рисунок 3.1 – Активный набор курсора

Обработка многострочного запроса в чем-то подобна обработке файла. Например, программа на COBOL открывает файл, обрабатывает записи, потом закрывает файл. Аналогично программа на PL/SQL открывает курсор, обрабатывает строки, полученные в ответ на запрос, затем закрывает курсор. Точно так же, как указатель позиции файла отмечает текущую позицию в открытом файле, курсор отмечает текущую позицию в активном наборе.

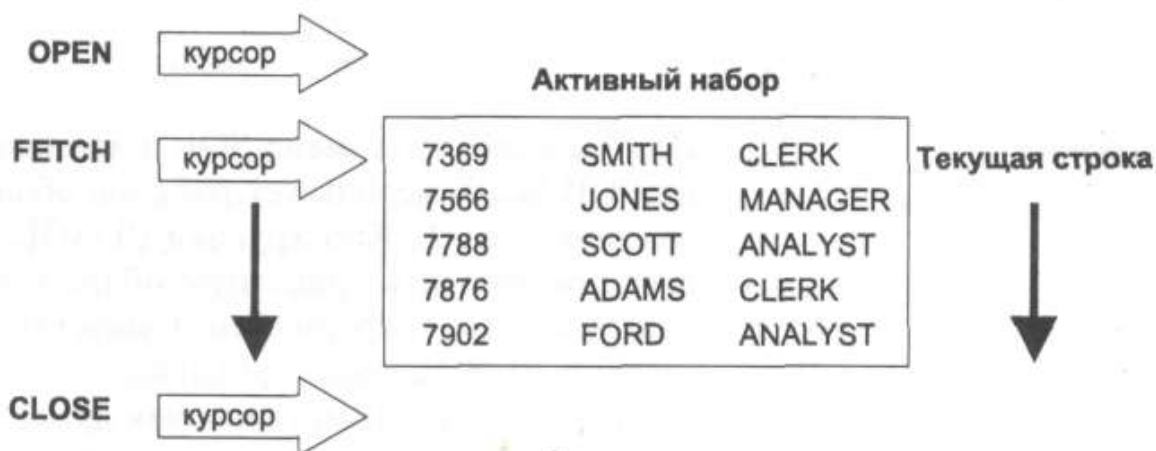


Рисунок 3.2 – Работа с курсором

Как показано на рис. 3.2, для управления курсором используются операторы OPEN, FETCH и CLOSE.

Оператор OPEN исполняет запрос, связанный с курсором, идентифицирует активный набор и устанавливает курсор перед первой строкой. Оператор FETCH выбирает текущую строку и продвигает курсор на следующую. После обработки последней строки оператор CLOSE отключает курсор.

1.1.7 Циклы FOR с курсором

В большинстве ситуаций, где требуется явный курсор, можно поступать проще, используя цикл FOR с курсором вместо операторов OPEN, FETCH и CLOSE. Цикл FOR с курсором неявно объявляет свой параметр цикла как запись типа %ROWTYPE, открывает курсор, при каждом своем повторении извлекает из активного множества строки со значениями в поля записи и закрывает курсор, когда все строки обработаны. В следующем примере цикл FOR с курсором неявно объявляет *emp_rec* как запись, принадлежащую к типу *c1%ROWTYPE*:

```
DECLARE
CURSOR c1 IS
SELECT ename, sal, hiredate, deptno FROM emp;
...
BEGIN
...
FOR emp_rec IN c1 LOOP
...
    salary_total := salary_total + emp_rec.sal;
END LOOP;
END;
```

Как показывает пример, для ссылок на отдельные поля используется точечная нотация.

1.1.8 Обработка ошибок

PL/SQL облегчает задачу обнаружения и обработки предопределенных и определяемых пользователем нештатных ситуаций, называемых *исключениями*. Когда встречается ошибка, *инициируется* исключение, т.е. нормальный ход вычислений прерывается и управление передается разделу обработки исключений вашего блока или подпрограммы на PL/SQL. Для обработки инициированных исключений пишутся отдельные подпрограммы, называемые *обработчиками исключений*.

Предопределенные исключения неявно инициируются исполнительной системой. Например, если происходит деление на ноль, то автоматически инициируется предопределенное исключение ZERO-DIVIDE. Определяемые пользователем исключения должны быть инициированы явно при помощи оператора RAISE.

DECLARE

salary NUMBER(7,2);

commission NUMBER(7,2);

comm_missing EXCEPTION; -- объявить исключение

BEGIN

SELECT sal, comm INTO salary, commission FROM emp

WHERE empno = :emp_id;

IF commission IS NULL THEN

 RAISE comm_missing; -- инициировать исключение

ELSE

 :bonus := (salary * 0.05) + (commission * 0.15);

END IF;

EXCEPTION -- начало обработчиков исключений

 WHEN comm_missing THEN

-- операторы по обработке ошибки
END;

Собственные исключения можно определять в разделе объявлений любого блока или подпрограммы на PL/SQL. В исполняемом разделе проверяется условие, требующее специального внимания. Если условие оказалось истинным, выполняется оператор RAISE. В вышеописанном примере вычисляется премия, причитающаяся коммивояжеру. Премия зависит от оклада и суммы продаж (переменная *commission*). Поэтому, если сумма продаж отсутствует, то инициируется исключение *comm_missing*.

Здесь подразумевается, что переменные *emp_id* и *bonus* объявлены и инициализированы в программе на базовом языке, куда должен быть встроен этот фрагмент на PL/SQL.

1.1.9 Модульность

Модульность позволяет разбить прикладную программу на удобные для сопровождения и логически простые части, или модули. Используя пошаговое уточнение, можно свести сложную задачу к набору простых, каждая из которых уже имеет легко реализуемое решение. Для этого в PL/SQL существует понятие *программного сегмента*. Кроме блоков и подпрограмм, PL/SQL предоставляет такое средство, как пакеты, которые позволяют собрать в одно целое связанные по смыслу программные объекты.

PL/SQL имеет два типа подпрограмм, называемых *процедурами* и *функциями*, которые могут принимать параметры и к которым можно обращаться (или, иначе, которые можно вызывать). Как показывает следующий пример, подпрограмма похожа на миниатюрную программу, начинающуюся с заголовка, за которым следуют необязательный раздел объявлений, исполняемый раздел и необязательный раздел обработки исключений.

При вызове эта процедура принимает номер служащего. Она использует номер для выборки суммы продаж служащего из таблицы базы данных и заодно

начисляет ему 25% премии. Затем проверяется величина премии. Если премия отсутствует, то инициируется исключение; в противном случае обновляется запись о служащем в платежной ведомости.

```
PROCEDURE award_bonus (emp_id NUMBER) IS
```

```
    bonus      REAL;
```

```
    comm_missing EXCEPTION;
```

```
BEGIN
```

```
    SELECT comm * 0.25 INTO bonus FROM emp
```

```
        WHERE empno = emp_id;
```

```
    IF bonus IS NULL THEN
```

```
        RAISE comm_missing;
```

```
    ELSE
```

```
        UPDATE payroll SET pay = pay + bonus
```

```
            WHERE empno = emp_id;
```

```
    END IF;
```

```
EXCEPTION
```

```
    WHEN comm_missing THEN
```

```
        . . .
```

```
END award_bonus;
```

2 Содержание работы

Задание 1. Разработать процедуры и функции для своей базы данных.

Порядок выполнения задания 1.

1. Функции и процедуры должны выполнять действия вычисления, склеивания строк, изменения значений полей и т.д. Требуется разработать как минимум три функции и три процедуры. Наличие параметров в функциях и процедурах обязательно.

3 Контрольные вопросы

1. Для чего предназначены хранимые функции? Описать их характеристики.
2. Для чего предназначены хранимые процедуры? Описать их характеристики.
3. Дать подробное объяснение по каждому оператору в разработанных функциях и процедурах.

Лабораторная работа №4 . Создание пакетов

Цель работы. Продолжить осваивать работу с языком PL/SQL; научиться создавать пакеты (Package).

1. Основные положения

PL/SQL позволяет объединить логически связанные типы, программные объекты и подпрограммы в **пакет**. В любом пакете можно легко разобраться, а интерфейсы между пакетами являются простыми, ясными и четко определенными, что очень полезно при разработке приложений.

Пакеты обычно состоят из двух частей: **спецификации** и **тела**. **Спецификация** – это интерфейс для приложений: здесь объявляются типы, константы, переменные, исключения, курсоры и подпрограммы, которыми можно пользоваться. **Тело** определяет курсоры и подпрограммы и таким образом реализует спецификацию. В следующем примере «пакетируются» две процедуры по найму служащих:

```
PACKAGE emp_actions IS -- спецификация пакета
PROCEDURE hire_employee (empno NUMBER, ename CHAR, ...);
PROCEDURE fire_employee (emp_id NUMBER);
END emp_actions;
```

```
PACKAGE BODY emp_actions IS -- тело пакета
PROCEDURE hire_employee (empno NUMBER, ename CHAR, ...) IS
BEGIN
    INSERT INTO emp VALUES (empno, ename, . . . ) ;
END hire_employee;
```

```
PROCEDURE fire_employee (emp_id NUMBER) IS
BEGIN
    DELETE FROM emp WHERE empno = emp_id;
```

```
END fire_employee;  
END emp_actions;
```

Прикладным программам видимы и доступны только объявления в спецификации пакета. Детали реализации в теле пакета скрыты и недоступны.

Если имеется процедурное расширение базы данных, пакеты могут быть откомпилированы и занесены в базу данных ORACLE, откуда их содержимым могут совместно пользоваться многие приложения. Когда в первый раз вызывается одна из подпрограмм пакета, в память загружается весь пакет. Поэтому любые последующие вызовы подпрограмм пакета не потребуют обмена с диском. Таким образом, использование пакетов может повысить производительность труда и увеличить эффективность программы.

Скрытие информации означает, что видны детали, относящиеся только к данному уровню разработки алгоритма и структуры данных. Скрытие информации обеспечивает независимость решений, принимаемых на верхнем уровне разработки, от деталей разработки нижнего уровня, которые, скорее всего, изменятся.

Скрытие информации для алгоритмов реализуется в методе ***разработки сверху вниз***. Когда определяется назначение процедуры нижнего уровня и спецификации ее интерфейса, можно игнорировать детали реализации. Они являются скрытыми для верхних уровней. Например, реализация гипотетической процедуры с именем *raise-salary* является скрытой. Все, что нужно знать, – это то, что процедура увеличит оклад данного служащего на указанную величину. Любые изменения в определении *raise-salary* прозрачны для вызывающих программ.

Скрытие информации для структур данных реализуется посредством ***инкапсуляции данных***. При разработке набора подпрограмм, обслуживающих структуру данных, они изолируются от пользователей и других разработчиков. Таким образом, другие разработчики знают, как пользоваться подпрограммами, работающими со структурой данных, не представляя точно самой структуры.

В пакетах PL/SQL можно указать, являются ли типы, программные объекты и подпрограммы приватными или общедоступными. Таким образом, пакеты навязывают инкапсуляцию данных, предлагая помещать объявления типов в черный ящик. Приватное определение типа является скрытым и недоступным. Если определение изменится, это затронет только пакет (но не вашу прикладную программу), что упрощает сопровождение и совершенствование программ.

2 Содержание работы

Задание 1. Разработать пакеты (спецификацию и тело) для своей базы данных.

Порядок выполнения задания 1.

1. Для каждой таблицы создается пакет с именем:

A_Имя таблицы.

Методы (процедуры) пакета должны выполнять следующие действия с одноименными таблицами:

Add_Имя таблицы(...) – добавление записи

Remove_Имя таблицы(...) – удаление записи

Change_ Имя поля(...) – изменение значения для поля таблицы (кроме ключевых и системных – ID_CRE и DAT_CRE). Должны быть созданы для каждого неключевого и несистемного поля).

2. В окне навигатора на пункте Packages правой клавишей мыши вызывается контекстное меню. В нем выбирается пункт New Package...

3. После описания пакета и его успешной компиляции создается тело пакета.

4. Для создания тела пакета в окне навигатора выбирается созданный ранее пакет и в контекстном меню для него выбирается пункт Create Body.

5. Заполняется и компилируется тело пакета.

3 Контрольные вопросы

1. Для чего предназначены пакеты?
2. Описать характеристики заголовка и тела пакета.
3. Подробно пояснить действия процедур пакета.

Библиографический список

1. Oracle Academy. Режим доступа: <https://academy.oracle.com/ru/solutions-database.html>
2. Култыгин, О. П. Администрирование баз данных. СУБД MS SQL Server [Электронный ресурс] : учеб. пособие / О. П. Култыгин. - М.: МФПА, 2012. - 232 с.
3. Сэм Р. Алапати. Oracle Database 11g руководство администратора баз данных. М: Издательский дом «Вильямс», 2010.- 1440 с.

