Московский Государственный Технический Университет им. Н.Э. Баумана
Факультет «Информатика и системы управления»
Кафедра «Компьютерные системы и сети»

А.В. Брешенков, Е.В. Галямова, С.В. Ефремов

Методическое пособие по лабораторной работе №3

Процедуры и функции в Oracle

Методическое пособие по лабораторной работе №4

Триггеры в Oracle

Москва 2010

ОГЛАВЛЕНИЕ

1. Введение.

В данной лабораторной работе Вы познакомитесь с существующими встроенными процедурами и функциями, с алгоритмом создания, применения и удаления новых объектов и изменения встроенных процедур и функций.

Вы можете создавать, запускать, удалять и изменять процедуры и функции разными способами: на странице SQL команд, на странице просмотра объектов, на странице создания SQL скриптов или с помощью командной строки SQL команд.

SQL выражение CREATE используется для создания любых объектов в Oracle: CREATE PROCEDURE позволит создать процедуру, CREATE FUNCTION – функцию.

Процедуры и функции похожи на маленькие программы, они состоят из декларативной (описательной) части, части выполнения или перехвата исключений. Процедура – это подпрограмма, которая выполняет какое-то конкретное действие. Вы указываете ее имя, параметры, переменные, создаете блок выполнения (BEGIN – END),

который содержит основной код и перехватывает какие-либо исключения. Функция – это подпрограмма, которая считает и выдает искомое значение. Процедуры и функции очень похожи, за исключением того, что функции кроме выполнения действий возвращают значение.

2. **Создание процедур и функций с помощью страницы SQL команд.**

Вы можете создать объекты данного типа с помощью страницы SQL команд, для этого вам необходимо выполнить следующие действия:

1. Войдите на страницу базы данных с помощью пароля, полученного от преподавателя.
2. На главной странице щелкните мышкой по иконке «SQL», чтобы отобразить страницу управления SQL.
3. На странице SQL нажмите на иконку «SQL Commands», чтобы перейти на следующую страницу.
4. Введите в появившемся окне PL/SQL код, для создания PL/SQL процедуры или функции. Можете воспользоваться примером процедуры, отображенной на рисунке №1.
5. Выберите (выделите) код создания процедуры или функции и нажмите кнопку «RUN».

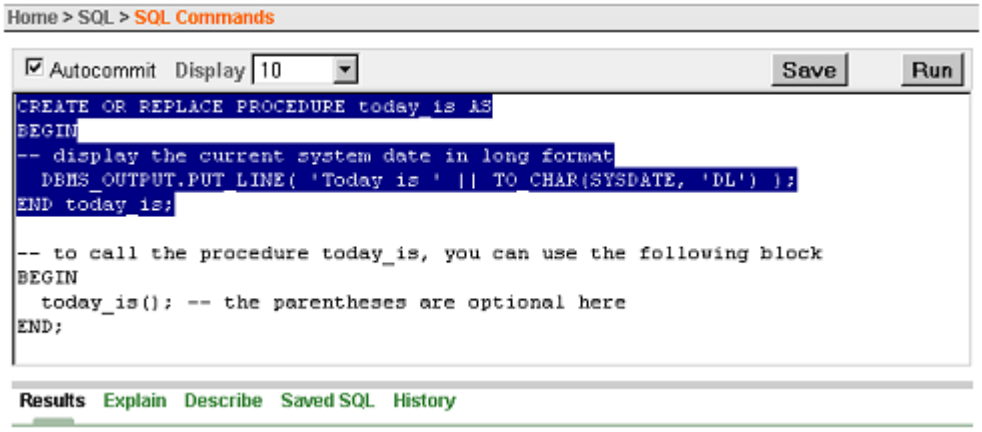

Рисунок 1. Создание процедуры today_is.

6. Выберите (выделите) код создания процедуры или функции и нажмите кнопку «RUN».

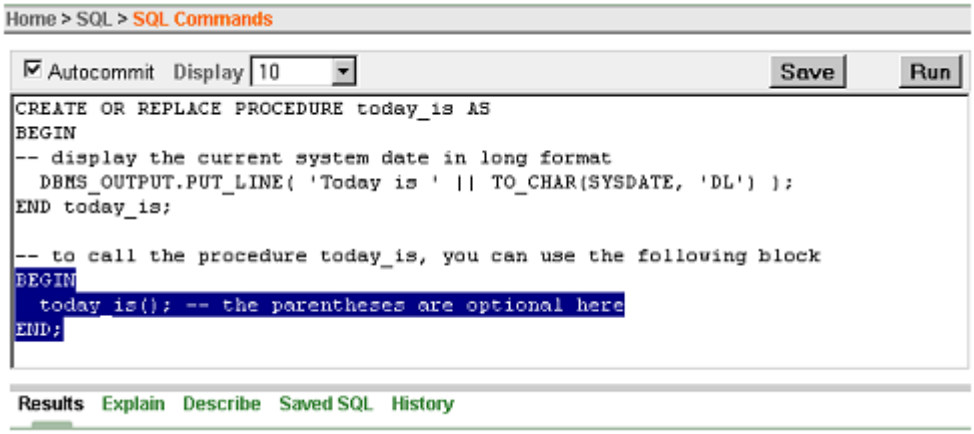

Рисунок 2. Выполнение процедуры today_is.

7. Если Вы хотите сохранить PL/SQL код для будущего использования, нажмите кнопку «SAVE».
8. В поле «NAME» введите название Вашего кода. Вы можете также ввести дополнительные опции. Нажмите кнопку «SAVE», чтобы сохранить Ваш код.
9. Для доступа к сохраненным процедурам и функциям нажмите на закладку «Saved SQL» и выберите необходимую Вам процедуру или функцию.

3. Создание процедур и функций с помощью страницы просмотра объектов.

Вы можете создать объекты данного типа с помощью страницы просмотра объектов, для этого вам необходимо выполнить следующие действия:

1. Войдите на страницу базы данных с помощью пароля, полученного от преподавателя.
2. На главной странице щелкните мышкой по треугольнику справа от кнопки «Object Browser», чтобы появился выпадающий список.
3. Выберите из списка пункт «Create» --} «Procedure».
4. Введите имя процедуры «award_bonus», проверьте стоит ли галочка напротив «Include Arguments» и нажмите кнопку «Next».
5. Введите информацию об аргументах и нажмите «Next». В качестве примера можете взять ее из рассмотренной ниже процедуры Example.

Name	In/Out	type
Emp_id	IN	Number
Bonus_rate	IN	Number

Таблица 1. Процедуры и аргументы.

6. Введите исходный код в «Procedure body» и нажмите «Next». В качестве примера можете взять ее из рассмотренной в примере №2 процедуры .

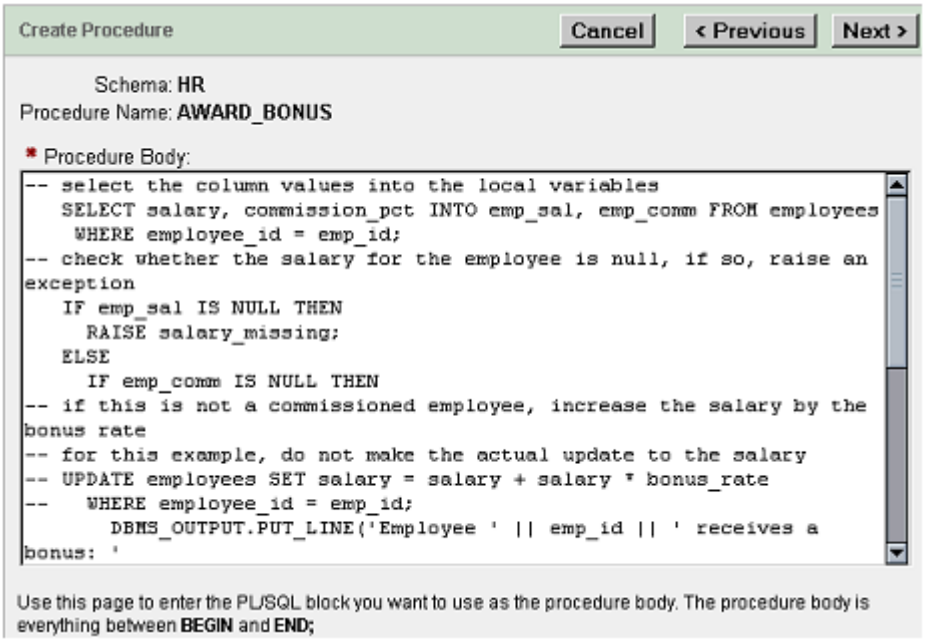

Рисунок 3. Исходный код процедуры «Award_Bonus».

7. Выберите вкладку «SQL», чтобы посмотреть исходный код. Нажмите «Prevoius», если вы хотите внести в него изменения.
8. Если больше не нужно вносить никаких изменений, нажмите кнопку «Finish». Нажмите кнопку «Edit», чтобы добавить или удалить подпрограммы и переменные снаружи блока BEGIN..END.
9. Нажмите кнопку «Compile», чтобы скомпилировать текст. Если появятся ошибки, исправьте исходный код и нажмите еще раз «Compile». После каждой компиляции сохраняются все внесенные изменения.
10. Нажмите кнопку «Finish», чтобы закончить процесс создания процедуры.

4. Просмотр процедур и функций с помощью страницы просмотра объектов.

Для того, чтобы посмотреть, какие процедуры и функции существуют в Вашей базе используете страницу просмотра объектов «Object Browser».

1. Войдите на страницу базы данных с помощью пароля, полученного от преподавателя.
2. На главной странице щелкните мышкой по объекту «Object Browser», чтобы открылась страница просмотра объектов.

3. Из выпадающего списка выберите процедуру или функцию, затем щелкните по имени процедуры, которую создали в предыдущем пункте («Award_bonus»). Появится информация о выбранной процедуре или функции.

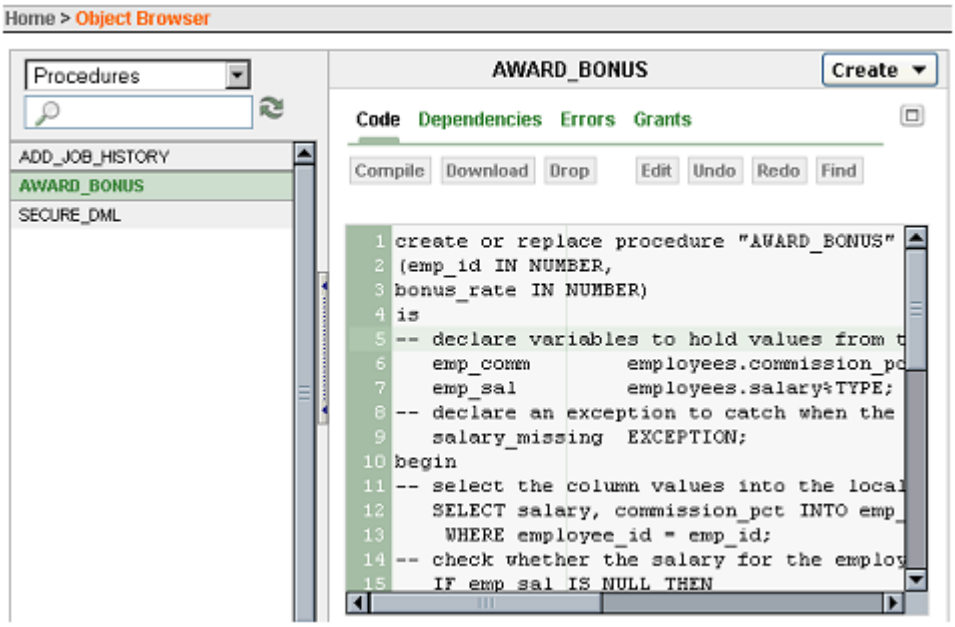

Рисунок 4. Информация о процедуре Award_bonus.

5. **Создание процедуры с помощью SQL команды CREATE PROCEDURE.**

Команда CREATE PROCEDURE позволяет создавать процедуры, которые будут храниться в базе данных. К таким хранимым (на схемном уровне) подпрограммам организован доступ из объекта «SQL». Вы можете использовать опцию OR REPLACE, чтобы изменить существующую процедуру, не удаляя ее сначала.

Пример №1. Хранимая процедура «today_is», возвращающая текущую дату.

```
CREATE OR REPLACE PROCEDURE today_is AS
BEGIN
-- display the current system date in long format
  DBMS_OUTPUT.PUT_LINE( 'Today is ' || TO_CHAR(SYSDATE, 'DL') );
END today_is;
/
-- to call the procedure today_is, you can use the following block
BEGIN
  today_is(); -- the parentheses are optional here
END;
/
```

Когда Вы создаете процедуру или функцию, то можете указать, какие параметры будут переданы ей или получены из нее. В следующем примере будет использоваться опция IN для передачи параметров в процедуры emp_id() и bonus_rate().

Пример №2. Хранимые процедуры с параметрами.

```
-- including OR REPLACE is more convenient when updating a subprogram
-- IN is the default for parameter declarations so it could be omitted
CREATE OR REPLACE
PROCEDURE
award_bonus
 (emp_id IN NUMBER, bonus_rate IN NUMBER)
  AS
-- declare variables to hold values from table columns, use %TYPE attribute
   emp_comm        employees.commission_pct%TYPE;
   emp_sal         employees.salary%TYPE;
-- declare an exception to catch when the salary is NULL
   salary_missing  EXCEPTION;
BEGIN  -- executable part starts here
-- select the column values into the local variables
   SELECT salary, commission_pct INTO emp_sal, emp_comm FROM employees
    WHERE employee_id = emp_id;
-- check whether the salary for the employee is null, if so, raise an exception
   IF emp_sal IS NULL THEN
     RAISE salary_missing;
   ELSE
     IF emp_comm IS NULL THEN
-- if this is not a commissioned employee, increase the salary by the bonus rate
-- for this example, do not make the actual update to the salary
-- UPDATE employees SET salary = salary + salary * bonus_rate
--   WHERE employee_id = emp_id;
       DBMS_OUTPUT.PUT_LINE('Employee ' || emp_id || ' receives a bonus: '
                     || TO_CHAR(emp_sal * bonus_rate) );
     ELSE
       DBMS_OUTPUT.PUT_LINE('Employee ' || emp_id
                     || ' receives a commission. No bonus allowed.');
     END IF;
   END IF;
EXCEPTION  -- exception-handling part starts here
   WHEN salary_missing THEN
     DBMS_OUTPUT.PUT_LINE('Employee ' || emp_id ||
                    ' does not have a value for salary. No update.');
   WHEN OTHERS THEN
      NULL; -- for other exceptions do nothing
END award_bonus;
/

-- the following BEGIN..END block calls, or executes, the award_bonus procedure
```

```
-- using employee IDs 123 and 179 with the bonus rate 0.05 (5%)
BEGIN

award_bonus (123, 0.05);

award_bonus (179, 0.05);
END;
/
```

После вызова этих процедур появятся сообщения:

```
Employee 123 received a bonus: 325
Employee 179 receives a commission. No bonus allowed.
```

Данная процедура использует два параметра, - идентификатор служащего и размер бонуса. Идентификатор используется для получения размера зарплаты и выяснения получает ли данный служащий комиссию из таблицы служащие (emplyees). Если зарплата у служащего нулевая, вызывается исключение. Если служащий не получает комиссию, то его зарплата автоматически изменяется с помощью бонуса, в других случаях все остается, как и было.

6. Создание функций с помощью SQL команды CREATE FUNCTION.

Команда CREATE FUNCTION позволяет создавать функции, которые будут храниться в базе данных. К таким хранимым (на схемном уровне) подпрограммам организован доступ из объекта «SQL». Вы можете использовать опцию OR REPLACE, чтобы изменить существующую функцию, не удаляя ее сначала.

Пример №3 демонстрирует работу функции, которая возвращает объект строкового типа, в котором первые буквы имени и фамилии заменяются на заглавные, а также вызов функции.

Пример №3. Функция last_first_name(empid NUMBER).

```
CREATE OR REPLACE FUNCTION last_first_name (empid NUMBER)
RETURN VARCHAR2 IS
lastname employees.last_name%TYPE; -- declare a variable same as last_name
firstname employees.first_name%TYPE; -- declare a variable same as first_name
BEGIN
SELECT last_name, first_name INTO lastname, firstname FROM employees
WHERE employee_id = empid;
RETURN ( 'Employee: ' || empid || ' - ' || UPPER(lastname)
|| ', ' || UPPER(firstname) );
END last_first_name;
/
-- you can use the following block to call the function
DECLARE
empid NUMBER := 163; -- pick an employee ID to test the function
BEGIN
-- display the output of the function
```

```
DBMS_OUTPUT.PUT_LINE( last_first_name(empid) );
END;
/

-- you can also call a function from a SQL SELECT statement
-- using the dummy DUAL table
    SELECT last_first_name(163) FROM DUAL;
```

Пример №4 демонстрирует работу функции, возвращающую объект числового типа. Функция возвращает вычисленный уровень зарплаты служащих определенной профессии, основанный на минимальной и максимальной зарплате и идентификаторе профессии.

Пример №4. Функция emp_sal_rankig(empid NUMBER).

```
-- function calculates the salary ranking of the employee based on the current
-- minimum and maximum salaries for employees in the same job category
CREATE OR REPLACE FUNCTION emp_sal_ranking (empid NUMBER)
RETURN NUMBER IS
minsal employees.salary%TYPE; -- declare a variable same as salary
maxsal employees.salary%TYPE; -- declare a variable same as salary
jobid employees.job_id%TYPE; -- declare a variable same as job_id
sal employees.salary%TYPE; -- declare a variable same as salary
BEGIN
-- retrieve the jobid and salary for the specific employee ID
SELECT job_id, salary INTO jobid, sal FROM employees WHERE employee_id = empid;

-- retrieve the minimum and maximum salaries for employees with the same job ID
SELECT MIN(salary), MAX(salary) INTO minsal, maxsal FROM employees
WHERE job_id = jobid;
-- return the ranking as a decimal, based on the following calculation
RETURN ((sal - minsal)/(maxsal - minsal));
END emp_sal_ranking;
/
-- create a PL/SQL block to call the function, you can also use another subprogram
-- because a function returns a value, it is called as part of a line of code
DECLARE
empid NUMBER := 163; -- pick an employee ID to test the function
BEGIN
-- display the output of the function, round to 2 decimal places
DBMS_OUTPUT.PUT_LINE('The salary ranking for employee ' || empid || ' is: '
|| ROUND(emp_sal_ranking(empid),2) );
END;
/
```

После вызова этой процедуры появится сообщение:

```
The salary ranking for employee 163 is: .63
```

7. **Вызов хранимых процедур и функций**

Вы можете запускать процедуры и функции из BEGIN..END блока или из других подпрограмм.

Вызывая процедуру или функцию необходимо принимать во внимание следующие особенности:

- Указание позиции: Вы указываете те же значения параметров и в той же позиции, на которой они были заданы при создании процедуры. Обращайте внимание именно на порядок, которым идут параметры.
- Указание наименования: Вы указываете название параметра и его значение, соединяете их знаком « => ». В этом случае порядок не важен.
- Смешанные указания: Вы можете использовать оба способа в одном месте, но при этом необходимо сохранять порядок, с которым были заданы параметры.

Пример №5. Различные техники вызова процедур и функций.

```
-- use a PL/SQL block to execute the procedure
BEGIN
award_bonus(179, 0.05);
END;
/
-- using named notation for the parameters, rather than positional
BEGIN
award_bonus(bonus_rate=>0.05, emp_id=>123);
END;
/
```

Вы также можете вызывать процедуры или функции во время создания приложений, Java ил PHP программ.

8. **Изменение процедур и функций**

Вы можете изменять процедуры и функции разными способами: на странице SQL команд, на странице просмотра объектов, на странице создания SQL скриптов или с помощью SQL команды CREATE OR REPLACE.

Если Вы хотите изменить процедуру или функцию с помощью SQL команды CREATE OR REPLACE, то Вы вместо старого исходного кода создаете новый, и тогда после компиляции он изменится.

Чтобы изменить процедуру или функцию с помощью страницы создания SQL скриптов, необходимо выполнить следующие действия:

1. Войдите на страницу базы данных с помощью пароля, полученного от преподавателя.
2. На главной странице щелкните мышкой по иконке «SQL», чтобы отобразить страницу управления SQL.

3. На странице SQL нажмите на иконку «SQL Commands», чтобы перейти на следующую страницу.
4. Перейдите на закладку «Saved SQL».
5. Выберите по имени процедуру или функцию, которую хотите изменить.
6. Измените исходный код и нажмите «RUN», если хотите запустить на исполнение ее.
7. Нажмите «SAVE», чтобы сохранить внесенные изменения.

Изменение процедур или функций с помощью страницы просмотра объектов.

1. Войдите на страницу базы данных с помощью пароля, полученного от преподавателя.
2. На главной странице щелкните мышкой по объекту «Object Browser», чтобы открылась страница просмотра объектов.
3. Из выпадающего списка выберите процедуру или функцию, затем щелкните по имени процедуры, которую хотите изменить.
4. Когда появилась информация о данном объекте, нажмите «Edit», чтобы изменить исходный код.
5. Нажмите кнопку «Compile», чтобы убедиться в том, что измененный код работает, и в случае появления ошибок их исправить.

9. **Удаление процедур и функций**

Вы можете удалять процедуры и функции разными способами: на странице SQL команд, на странице просмотра объектов или с помощью SQL команды DROP.

Чтобы удалить процедуру или функцию с помощью страницы просмотра объектов необходимо выполнить следующие действия:

1. Войдите на страницу базы данных с помощью пароля, полученного от преподавателя.
2. На главной странице щелкните мышкой по объекту «Object Browser», чтобы открылась страница просмотра объектов.
3. Из выпадающего списка выберите процедуру или функцию, затем щелкните по имени процедуры, которую хотите удалить.
4. Когда появилась информация о данном объекте, нажмите «Drop».
5. Нажмите кнопку «Finish», чтобы подтвердить данное действие.

Пример использовании команды Drop показан ниже.

Пример №6. Удаление процедур и функций с помощью команды «DROP».

```
-- drop the procedure award_bonus to remove from the database
DROP PROCEDURE award_bonus;
-- drop the function emp_sal_ranking to remove from database
DROP FUNCTION emp_sal_ranking;
```

10. **Заключение**

В данной лабораторной работе Вы познакомились с существующими встроенными процедурами и функциями, с алгоритмом создания, применения и удаления новых объектов и изменения встроенных процедур и функций.

Надеемся, что полученные знания Вы сможете применить на практике при создании собственных приложений под Oracle 10g X.E.

11. **Контрольные вопросы.**

1.«Что такое процедура?», «Что такое функция?»

2.«Какими средствами можно создать процедуру или функцию?»

3.«Какие PL\SQL команды используются для создания процедур и функций и каков их синтаксис?»

4.«Как можно посмотреть существующие процедуры или функции?»

5.«Как изменить или удалить процедуры или функции?»

12. **Рекомендуемая литература.**

1. Гарсиа – Молина, Ульман, Уидом. Системы баз данных. Полный Курс. Пер. с англ.- М.: Издательский дом Вильямс, 2003 – 1088 стр.
2. Грабер М. Введение в SQL: Пер с англ. – М.:Изд-во’ЛОРИ’,1996. – 380с.
3. 3. Джеймс Перри, Джеральд Пост. Введение в Oracle 10g “И.Д. Вильямс”, 2006. – 700 с.
4. Гринвальд Рик, Становьяк Робеот, Додж Гери, Кляйн Девид, Шапиро Бен, Челья Кристофер Дж. Программирование баз данных Oracle для профессионалов.: Пер. с англ.: – М. : ООО “И.Д. Вильямс”, 2007. – 784 с.
5. Кайт Томас. Oracle для профессионалов: архитектура, методика программирования и основные особенности версии 9i и 10j.: Пер с англ. – М.:Издательский дом ”Вильямс”. 2008. – 848 с.
6. Кевин Луни, Боб Брила. Oracle Database 10g. Настольная книга администратора баз данных. – М.:Издательство’Лори’,2008. – 732 с.
7. Райан стивенс, Рональд Плю. SQL. Пер с англ. – М.:ЗАО ’Издательство Бином’,1998.-400 с.

Методическое пособие по лабораторной работе №4
Триггеры в Oracle

1 Введение.

В данной лабораторной работе Вы познакомитесь с существующими встроенными триггерами, с алгоритмом создания, применения и удаления новых объектов и изменения встроенных триггеров.

Вы можете создавать, удалять и изменять триггеры разными способами: на странице SQL команд, на странице просмотра объектов, на странице создания SQL скриптов или с помощью командной строки SQL команд.

SQL выражение CREATE используется для создания любых объектов в Oracle: CREATE TRIGGER позволит создать триггер.

Триггеры баз данных это хранимые процедуры, связанные с таблицами, видами, или событиями. Триггер может быть вызван однажды, когда происходит какое-то событие, или много раз, когда происходит какое-либо изменение таблицы (вставка новой строки, удаление или изменение существующей). Триггер может быть вызван после события, чтобы записать новые данные или произвести необходимые после данного события действия. Триггер может быть вызван перед каким-либо событием, чтобы избежать ошибочных действий или изменить данные так, чтобы они удовлетворяли правилам ведения бизнеса. Исполнительная часть триггера может содержать SQL выражение или программную часть.

2 Обзор триггеров.

2.1 Типы триггеров.

Триггеры могут быть описаны в виде PL/SQL запросов или C процедур, связанных с таблицами, видами, событиями или самой базой данных. СУБД Oracle XE автоматически вызывает триггер, когда происходит специфическое событие, которым обычно является DML выражение, связанное с таблицей. Существуют следующие типы триггеров:

- DML триггеры таблиц.
- INSTEAD OF триггеры видов.
- Системные триггеры событий, связанные с базой данных (DATABASE) или схемой (SCHEMA).

Вы можете создавать триггеры, связанные со следующими выражениями:

- DML выражения (DELETE, INSERT, UPDATE).
- DDL выражения(CREATE, ALTER, DROP).
- Операции базы данных (LOGON, LOGOFF).

2.2 **Именование триггеров**

При создании триггеров необходимо иметь ввиду, что имена триггеров должны быть уникальны, то есть в рамках одной схемы (например, «HR») не должно быть триггеров с одинаковыми названиями. В СУБД Oracle допускается совпадение имен триггеров и таблиц или видов, однако не рекомендуется так делать. Одинаковые названия могут привести к путанице и усложнению управления базы данных.

2.3 **В каких случаях вызывается триггер?**

Для того, чтобы вызвался триггер необходимо в его описании указать следующие особенности:

- SQL выражение или системное событие, событие БД или DML событие, которое обратиться к «телу» триггера. Опции могут содержать выражения DELETE, INSERT или UPDATE по одному или все сразу.
- Должен быть указан объект, связанный с триггером: таблица, вид, база данных или схема.

Если триггер содержит следующее выражение:

AFTER DELETE OR INSERT OR UPDATE ON employees ...

То любое из следующих выражений запустят триггер:

```
DELETE FROM employees WHERE ...;
INSERT INTO employees VALUES ( ... );
INSERT INTO employees SELECT ... FROM ... ;
UPDATE employees SET ... ;
```

Выражение UPDATE может содержать список колонок, которые будут проверяться на наличие изменений. Если указаны названия колонок, то триггер будет срабатывать только после изменений этих колонок. Если они не заданы, то – после любого изменения таблицы. Нельзя указывать список колонок в выражениях DELETE и INSERT, потому что удаляются и добавляются сразу все колонки одной строки. В примере №1 описан триггер audit_sal, который следит только за колонкой salary из таблицы employees. Изменения других колонок этой таблицы не запустят данный триггер.

2.4 Моменты запуска триггера.

В четырех следующих разделах рассматриваются опции, с помощью которых определяется в какой момент вызывать процедуру триггера.

2.4.1 Запуск триггера с помощью опций BEFORE и AFTER.

Опции BEFORE и AFTER в выражении CREATE TRIGGER явно показывают, в какой момент запускать тело триггера в зависимости от необходимости. Опции находятся после выражения CREATE TRIGGER, до тела процедуры триггера.

Примеры №1 и №2 демонстрируют работу триггера с опциями BEFORE и AFTER.

Если в СУБД используются выражения DELETE или UPDATE, то Oracle сравнивает их с правильным вариантом написания UPDATE или DELETE. В случае некорректности сравнения СУБД использует процедуру отката (ROLLBACK) и перезапускает выражения. Так может продолжаться много раз, пока не достигнется положительный результат. Каждый раз при перезапуске запускается триггер с опцией BEFORE. Откат не производит никаких изменений в триггере, поэтому чтобы избежать избыточных действий в триггере, необходимо предусмотреть счетчик, который по достижении границы останавливал бы запуск триггера.

2.4.2 Запуск триггера с помощью опции FOR EACH ROW.

Опция FOR EACH ROW определяет, будет ли триггер строковым или будет триггером выражения. Если Вы применяете эту опцию, триггер выполнит все действия для каждой строки таблицы согласно условиям в описании триггера.

Примеры №1 и №2 демонстрируют работу триггеров с опцией FOR EACH ROW.

Суть опции FOR EACH ROW в том, чтобы показать, что триггер запускается только один раз после всех изменений, а не отдельно для каждой строки таблицы. Триггеры такого типа относятся к триггерам на уровне выражений и используются для проверок правильности занесения данных.

2.4.3 Запуск триггера с помощью задания условий (выражения WHEN)

Можно создать триггер, который будет запускаться только в случае выполнения конкретного условия. Такие опции называются опциями ограничениями и задаются с помощью SQL выражения булевского типа в условии WHEN.

Если необходимо, триггер будет проверять каждую строку таблицы на соответствия заданному условию. Если условие будет выполнено (переменная получит значение TRUE), то тело триггера будет запущено. Если условие не будет выполнено (переменная получит значение FALSE), то тело триггера запущено не будет. В примере №2 показан вариант создания триггера с условием.

Выражение в условии WHEN может быть SQL типа, но не может содержать подзапросов. Вы не можете использовать PL\SQL код в выражении WHEN. Условие WHEN не может также содержаться в описании триггера.

2.4.4 Запуск триггера с помощью опции INSTEAD OF.

Опция INSTEAD OF позволяет избежать возникновения каких-либо событий, заменяя его на другое, описанное в теле триггера.

С помощью триггеров такого типа Вы можете запускать выражения UPDATE, INSERT и DELETE применительно к сложным видам, которые не могут быть изменены другими способами. Также триггер можно использовать для проверки корректности изменений вида. Триггеры с опцией INSTEAD OF запускаются в основном «на заднем плане», чтобы выполнить необходимые действия с подчиняющимися виду таблицами. Они могут применяться только к видам и будут использоваться для каждой строки таблицы. Триггеры корректны только для выражений DML и не могут содержать выражения DDL или события базы данных.

Пример триггера с этой опцией рассматривается под третьим номером.

2.5 Доступ к колонкам таблиц в строковых триггерах.

В теле триггера строкового типа могут содержаться SQL запросы и PL\SQL процедуры, относящиеся как к старым колонкам таблицы, так и к новым, измененным триггером. Существую специальные выражения для обозначения колонок: «:OLD.column_name» и «:NEW.column_name».

В зависимости от типа выражения триггера некоторые имена могут не иметь смысла.

- Триггер, запущенный выражением INSERT, имеет доступ только к новому имени, потому что у старого значение – «NULL».
- Триггер, запущенный выражением UPDATE, имеет доступ как к старому имени, так и к новому. Это связано с тем, что изменения могут касаться как старого так и нового имени.
- Триггер, запущенный выражением DELETE, имеет доступ только к старому имени, потому что новое получит значение «NULL».

2.6 Определение DML операции, запускающей триггер.

Если необходимо, чтобы триггер запускался не по одному DML выражению, такому как ON INSERT или UPDATE, а сразу по нескольким, то тело триггера может

содержать производные от этих слов, - INSERTING, UPDATING и DELETING. Это служит для проверки, какое конкретно условие выполнилось. Пример №3 демонстрирует такой вариант.

В UPDATE триггере можно указать, какой конкретно столбец таблицы необходимо проверять. Для этого после UPDATING указывается имя колонки. Например, так:

```
CREATE OR REPLACE TRIGGER ...
... UPDATE OF salary ON employees ...
BEGIN
... IF UPDATING ('salary') THEN ... END IF;
...
```

Код после THEN будет выполнен только, если будут произведены изменения в колонке «salary». Таким образом можно уменьшить заголовок триггера, перенеся все условия в тело триггера.

2.7 **Включение и выключение триггера.**

Триггеры можно включать и выключать по желанию. Включенный триггер будет срабатывать, когда будет выполняться условие в его заголовке. Выключенный же триггер не будет срабатывать, ни при каких условиях. Отключайте триггеры в случае, когда вы проверяете работоспособность процедур и функций, связанных с таблицами, к которым привязаны триггеры.

3 Создание триггеров.

3.1 **Рекомендации по созданию триггеров.**

Используйте следующие рекомендации при создании триггеров:

- Используйте триггеры для выполнения специфических операций или действий.
- Не создавайте триггеры, которые дублируют уже существующие в Oracle встроенные триггеры. Например, триггеры, проверяющие на соответствие тип вносимых данных и тип колонки в таблице.
- Ограничивайте триггеры по размеру. Если код триггера не умещается в 60 строк кода, лучше перенесите его в хранимую процедуру, и вызывайте ее в теле триггера. Размер триггера не может быть больше 32 кБ.
- Создавайте триггеры, которые можно было бы применять глобально ко всей базе данных или схеме, не обращающие внимания на пользователя или приложение базы данных.

- Не создавайте рекурсивных триггеров. Это может привести к зацикливанию процедуры выполнения и к тому, что вся память будет отобрана для выполнения триггера.
- Используйте триггеры только там, где они действительно необходимо. Они выполняются для любого приложения, любого пользователя и любого события, описанного в условии триггера.

3.2 Ограничения по созданию триггеров.

При создании триггеров необходимо учитывать, что существуют определенные ограничения на создание триггеров.

3.2.1 SQL выражения, приемлемые при создании триггеров.

Тело триггера может содержать DML выражения. Также может содержать SELECT выражения, но они должны состоять из SELECT..INTO.. выражений или SELECT выражений, использующих указатель мыши.

В теле триггера нельзя использовать: DDL выражения, выражения по контролю транзакций, ROLLBACK, COMMIT и SAVEPOINT выражения. Для системных триггеров разрешены CREATE, ALTER, DROP TABLE и ALTER..COMPILE выражения.

Процедура, вызываемая в триггере, не может обращаться к предыдущей транзакции, потому что она выполняется в контексте выполнения триггера.

Выражения внутри триггера могут обращаться к удаленным объектам схемы. Однако следует обратить в таком случае внимание на совпадение подписей, настроек времени, чтобы не произошло сбоя.

3.2.2 Ограничение на системные триггеры.

Только утвержденные триггеры запускаются. Например, если вы создаете триггер, который должен содержать опцию CREATE, то при создании триггера он сам запускаться не будет, потому что информация о создании такого триггера еще не подтвердилась в базе данных. Чтобы было яснее, рассмотрим следующий пример.

```
CREATE OR REPLACE TRIGGER my_trigger
AFTER CREATE ON DATABASE
BEGIN
NULL;
END;
```

3.2.3 Привилегии, которые нужны для работы с триггерами.

Для того чтобы создать триггер, Вам необходимо иметь системную привилегию CREATE TRIGGER, а также одну из следующих привилегий:

- Владеть таблицей, определенной в заголовке триггера.
- Иметь ALTER привилегию над таблицей, определенной в заголовке триггера.
- Иметь системную привилегию ALTER ANY TABLE.

Системная привилегия CREATE TRIGGER включена в RESOURCE роль, которая присвоена пользователю HR.

Чтобы создать триггер базы данных, Вы должны обладать ADMINISTER DATABASE TRIGGER привилегией. Если эта привилегия позже будет снята с Вас, то вы сможете удалить созданный триггер, но не изменить его.

4 Управление триггерами.

Триггеры – это еще один вид объектов, которыми Вы можете управлять с помощью страницы просмотра объектов (Object Browser). Вы также можете создавать и изменять их с помощью страницы SQL команд или командной строки SQL.

4.1 Создание триггеров с помощью страницы SQL команд.

Вы можете создать объекты данного типа с помощью страницы SQL команд, для этого вам необходимо выполнить следующие действия:

1. Войдите на страницу базы данных с помощью пароля, полученного от преподавателя.
2. На главной странице щелкните мышкой по иконке «SQL», чтобы отобразить страницу управления SQL.
3. На странице SQL нажмите на иконку «SQL Commands», чтобы перейти на следующую страницу.
4. Для начала создайте таблицу emp_audit, к которой будет подключен триггер audit_sal. Для этого введите в появившемся окне PL/SQL код, показанный в примере №1.
5. Выберите (выделите) код создания таблицы и нажмите кнопку «RUN». После этого сотрите все строчки, чтобы повторно не создать таблицу.
6. Введите код, показанный на рисунке №1.

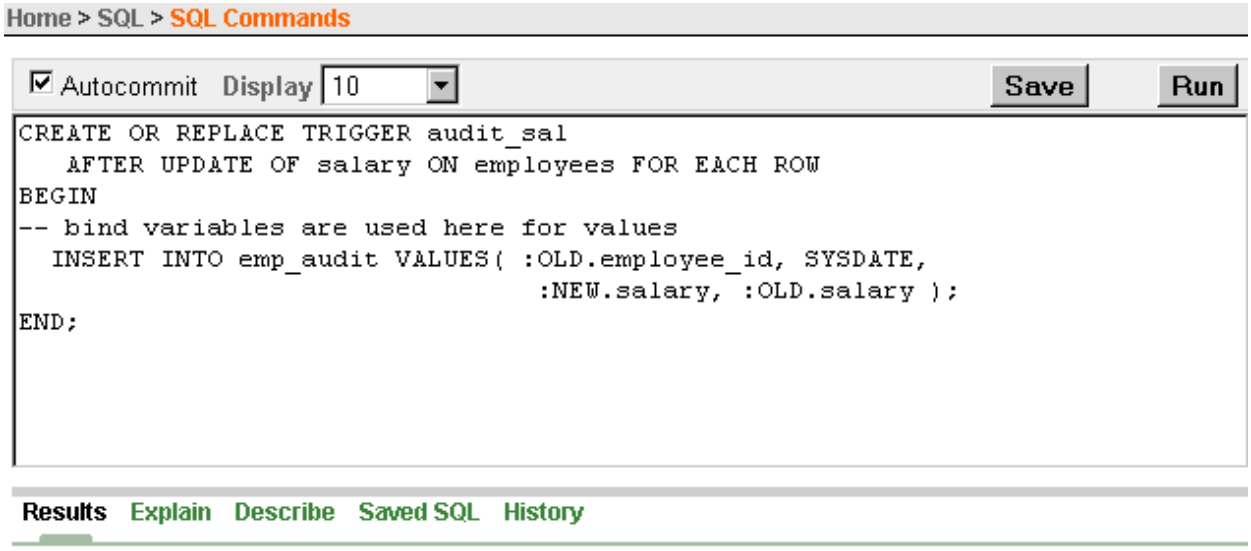

Рис. 4 Создание триггера audit_sal с помощью SQL кода.

7. Нажмите кнопку «RUN».
8. Если Вы хотите сохранить PL/SQL код для будущего использования, нажмите кнопку «SAVE».
9. В поле «NAME» введите название Вашего кода. Вы можете также ввести дополнительные опции. Нажмите кнопку «SAVE», чтобы сохранить Ваш код.
10. Для доступа к сохраненным триггерам нажмите на закладку «Saved SQL» и выберите необходимый Вам триггер.

4.2 Создание триггеров с помощью страницы просмотра объектов.

Вы можете создать объекты данного типа с помощью страницы просмотра объектов, для этого вам необходимо выполнить следующие действия:

1. Войдите на страницу базы данных с помощью пароля, полученного от преподавателя.
2. На главной странице щелкните мышкой по треугольнику справа от кнопки «Object Browser», чтобы появился выпадающий список.
3. Выберите из списка пункт «Create» --} «Trigger».
4. Введите имя таблицы «employees», к которой будет создан триггер, и нажмите кнопку «Next». Также вы можете не вводить имя таблицы, а выбрать его из списка.

5. Введите имя триггера «emp_salary_trigger», проверьте, не стоит ли галочка напротив «Preserve Case»(не учитывать регистр).
6. Выберите как точку запуска триггера «After».
7. Выберите в опциях «update of».
8. Выберите из списка колонок «Salary».
9. Проверьте, стоит ли галочка напротив «For Each Row». Не вводите ничего в поле «WHEN».
10. В поле тела триггера введите код из примера №1.

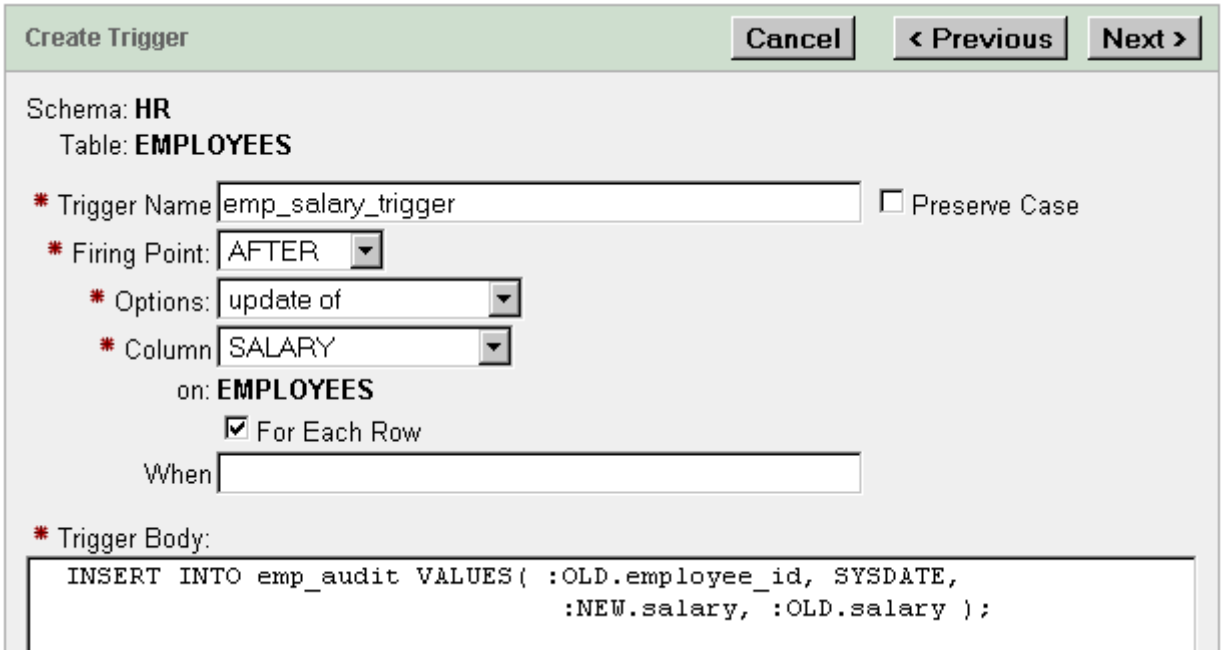

Рис. 5 Создание триггера emp_salary_trigger с помощью Object Browser.

11. Нажмите кнопку «NEXT».
12. Нажмите кнопку «SQL», чтобы посмотреть, как выглядят все сделанные действия в виде SQL кода.
13. Нажмите кнопку «FINISH».

4.3 Просмотр триггеров с помощью страницы просмотра объектов.

Для того, чтобы посмотреть, какие триггеры существуют в Вашей базе используете страницу просмотра объектов «Object Browser».

1 Войдите на страницу базы данных с помощью пароля, полученного от преподавателя.

2 На главной странице щелкните мышкой по объекту «Object Browser», чтобы открылась страница просмотра объектов.

3 Из выпадающего списка выберите триггер, затем щелкните по имени триггера, который создали в предыдущем пункте («emp_salary_trigger»).

4 Нажмите кнопку «Object Details», чтобы посмотреть информацию о триггере.

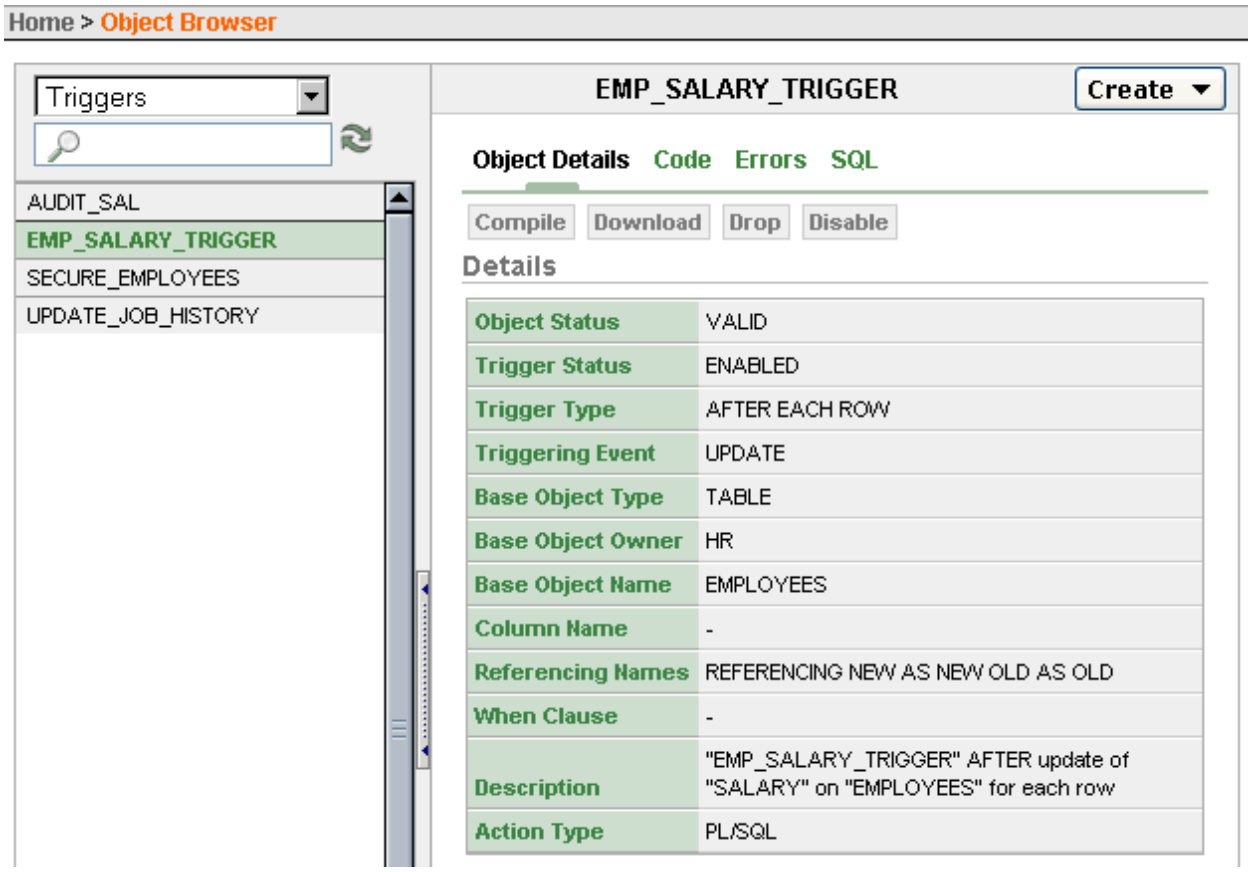

Рис. 6. Информация о триггере «emp_salary_trigger».

5. Посмотрите добавочную информацию с помощью кнопок «Code», «Errors», «SQL».

4.4 Создание триггеров с опциями «AFTER» и «FOR EACH ROW».

В примере №1 рассматривается код создания триггера «audit_sal», связанного с таблицей «employees», он запускается после того, как были произведены изменения с колонкой «salary» и записывает данные в таблицу аудита.

С помощью опции «FOR EAH ROW» триггер вносит новую информацию в таблицу «emp_audit» после каждого изменения зарплаты в таблице «emplyees». Запись содержит поля ID служащего, дату изменения, старую и новую зарплаты. Обратите внимания на то, как идет обращение к старой и новой колонке зарплаты.

С помощью ключевого слова «AFTER» извлекать данные и изменять одну и ту же таблицу. Однако он может делать это только после того, как изменения были приняты.

Из-за того, что триггер использует выражение «FOR EACH ROW», он может вызываться несколько раз подряд. Так будет происходить после удаления или изменения

нескольких строк. Если Вы только хотите зафиксировать факт внесения изменений, и Вам не нужна информация о них, вы можете не использовать эту опцию.

Пример 1. Создание триггера с опцией «AFTER».

```
-- create a table to use for with the trigger in this example if
-- it has not already been created previously
-- if the table does not exist, the trigger will be invalid
CREATE TABLE emp_audit ( emp_audit_id NUMBER(6), up_date DATE,
new_sal NUMBER(8,2), old_sal NUMBER(8,2) );
-- create or replace the trigger
CREATE OR REPLACE TRIGGER audit_sal
AFTER UPDATE OF salary ON employees FOR EACH ROW
BEGIN
-- bind variables are used here for values
INSERT INTO emp_audit VALUES( :OLD.employee_id, SYSDATE,
:NEW.salary, :OLD.salary );
END;
/
-- fire the trigger with an update of salary
UPDATE employees SET salary = salary * 1.01 WHERE manager_id = 122;
-- check the audit table to see if trigger was fired
SELECT * FROM emp_audit;
```

4.5. Создание триггеров с опциями «BEFORE» и условием «WHEN».

В примере №2 рассматривается триггер с опцией «BEFORE», который запускается после изменения каждой строки. Обратите внимание на использование условия «WHEN».

Пример 2. Создание триггера с опцией «BEFORE».

```
-- create a temporary table
CREATE TABLE emp_sal_log (emp_id NUMBER, log_date DATE,
new_salary NUMBER, action VARCHAR2(50));
CREATE OR REPLACE TRIGGER log_salary_increase -- create a trigger
BEFORE UPDATE of salary ON employees FOR EACH ROW
WHEN (OLD.salary < 8000)
BEGIN
INSERT INTO emp_sal_log (emp_id, log_date, new_salary, action)
VALUES (:NEW.employee_id, SYSDATE, :NEW.salary, 'New Salary');
END;
/
-- update the salary with the following UPDATE statement
-- trigger fires for each row that is udpated
UPDATE employees SET salary = salary * 1.01 WHERE department_id = 60;
-- view the log table
SELECT * FROM emp_sal_log;
```

4.6. Создание триггеров с опцией «INSTEAD OF».

В примере №3 создается вид с несколькими починенными таблицами. Обратите внимание на использование ключевого слова «JOIN».

Пример №3. Создание вида, который изменяется триггером с «INSTEAD OF».

```
CREATE OR REPLACE VIEW my_mgr_view AS
SELECT ( d.department_id || ' ' || d.department_name) "Department",
d.manager_id, e.first_name, e.last_name, e.email, e.hire_date "Hired On",
e.phone_number, e.salary, e.commission_pct,
(e.job_id || ' ' || j.job_title) "Job Class"
FROM departments d
JOIN employees e ON d.manager_id = e.employee_id
JOIN jobs j ON e.job_id = j.job_id
    ORDER BY d.department_id;
```

В примере №4 рассматривается триггер, который будет переносить изменения в виде в таблицы.

Пример№4. Создание триггера с опцией «INSTEAD OF».

```
CREATE OR REPLACE TRIGGER update_my_mgr_view
INSTEAD OF UPDATE ON my_mgr_view
FOR EACH ROW
BEGIN
-- allow the following updates to the underlying employees table
UPDATE employees SET
last_name = :NEW.last_name,
first_name = :NEW.first_name,
email = :NEW.email,
phone_number = :NEW.phone_number,
salary = :NEW.salary,
commission_pct = :NEW.commission_pct
WHERE employee_id = :OLD.manager_id;
END;
/
```

После того как был создан данный триггер, можно внести изменения в вид, которые перенесутся в таблицу.

```
UPDATE my_mgr_view SET first_name = 'Denis' WHERE manager_id = 114;
```

4.7.Создание триггера с перехватчиком ошибок.

В примере №5 показан код, создающий триггер с перехватчиком ошибок. В примере возникает исключение, если какой-либо процесс попытается поменять ID менеджера служащего.

Пример №5. Триггер с перехватчиком ошибок.

```
-- create a temporary table
CREATE TABLE emp_except_log (emp_id NUMBER, mgr_id_new NUMBER,
mgr_id_old NUMBER, log_date DATE, action VARCHAR2(50));
CREATE OR REPLACE TRIGGER emp_log_update -- create a trigger
BEFORE UPDATE ON employees FOR EACH ROW
```

```
DECLARE
mgrid_exception EXCEPTION;
BEGIN
IF (:NEW.manager_id <> :OLD.manager_id) THEN
RAISE mgrid_exception;
END IF;
INSERT INTO emp_except_log (emp_id, mgr_id_new, mgr_id_old, log_date, action)
VALUES (:NEW.employee_id, :NEW.manager_id, :OLD.manager_id,
        SYSDATE, 'Employee updated');

EXCEPTION
WHEN mgrid_exception THEN
INSERT INTO emp_except_log (emp_id, mgr_id_new, mgr_id_old, log_date, action)
VALUES (:NEW.employee_id, :NEW.manager_id, :OLD.manager_id,
SYSDATE, 'Employee manager ID updated!');
END;
/
-- update employees with the following UPDATE statements, firing trigger
UPDATE employees SET salary = salary * 1.01 WHERE employee_id = 105;
-- the trigger raises an exception with this UPDATE
UPDATE employees SET manager_id = 102 WHERE employee_id = 105;
-- view the log table
        SELECT * FROM emp_except_log;
```

4.8. Создание триггера, запускающегося один раз при каждом изменении.

В примере №6 рассматривается триггер, запускающийся один раз при каждом изменении. Для этого выражение «FOR EACH ROW» было опущено. Для того, чтобы триггер срабатывал при нескольких условиях, были использованы ключевые слова «IF..THEN» и производные от ключевых слов, такие как «INSERTING».

Пример №6. Триггер, запускающийся один раз при каждом изменении.

```
-- create a log table
CREATE TABLE emp_update_log (log_date DATE, action VARCHAR2(50));
-- create a trigger
CREATE OR REPLACE TRIGGER log_emp_update
AFTER UPDATE OR INSERT ON employees
DECLARE
v_action VARCHAR2(50);
BEGIN
IF UPDATING THEN
v_action := 'A row has been updated in the employees table';
END IF;
IF INSERTING THEN
v_action := 'A row has been inserted in the employees table';
END IF;
INSERT INTO emp_update_log (log_date, action)
VALUES (SYSDATE, v_action);
END;
/
-- fire the trigger with an update
```

```
UPDATE employees SET salary = salary * 1.01 WHERE department_id = 60;
INSERT INTO employees VALUES(14, 'Belden', 'Enrique', 'EBELDEN','555.111.2222',
'31-AUG-05', 'AC_MGR', 9000, .1, 101, 110);
-- view the log table
SELECT * FROM emp_update_log;
-- clean up: remove the inserted record
DELETE FROM employees WHERE employee_id = 14;
```

4.9.Создание «LOGON» и «LOGOFF» триггера.

Вы можете создавать триггеры, связанные с подключением или отключением определенного пользователя от базы данных.

В примере №7 рассматривается триггер, добавляющий запись в таблицу логов, когда пользователь подключается к схеме HR. В этом примере в таблицу логов записывается имя пользователя (USER), тип активности (LOGON или LOGOFF), дата (SYSDATE), количество служащих в таблице «employees». Как дата, так и имя пользователя являются псевдоколонками.

Пример №7. Создание LOGON триггера.

```
-- create a table to hold the data on user logons and logoffs
CREATE TABLE hr_log_table ( user_name VARCHAR2(30), activity VARCHAR2(20),
logon_date DATE, employee_count NUMBER );
-- create a trigger that inserts a record in hr_log_table
-- every time a user logs on to the HR schema
CREATE OR REPLACE TRIGGER on_hr_logon
AFTER LOGON
ON HR.schema
DECLARE
emp_count NUMBER;
BEGIN
SELECT COUNT(*) INTO emp_count FROM employees; -- count the number of employees
INSERT INTO hr_log_table VALUES(USER, 'Log on', SYSDATE, emp_count);
END;
/
```

В примере №8 рассматривается триггер, добавляющий запись в таблицу логов, когда пользователь отключается от схемы HR.

Пример№8. Создание LOGOFF триггера.

```
-- create a trigger that inserts a record in hr_log_table
-- every time a user logs off the HR schema
CREATE OR REPLACE TRIGGER on_hr_logoff
BEFORE LOGOFF
ON HR.schema
DECLARE
emp_count NUMBER;
BEGIN
SELECT COUNT(*) INTO emp_count FROM employees; -- count the number of employees
INSERT INTO hr_log_table VALUES(USER, 'Log off', SYSDATE, emp_count);
```

```
END;
/
```

Вы можете проверить работоспособность таких триггеров, выполнив следующие операции и посмотрев появившиеся записи в таблице «hr_log_table».

```
DISCONNECT
CONNECT hr/hr
SELECT * FROM hr_log_table;
```

4.10. Изменение триггеров.

Вы можете изменять триггеры разными способами: на странице SQL команд, на странице просмотра объектов, на странице создания SQL скриптов или с помощью SQL команды CREATE OR REPLACE.

Если Вы хотите изменить триггер с помощью SQL команды CREATE OR REPLACE, то Вы вместо старого исходного кода создаете новый, и тогда после компиляции он изменится.

Изменить триггер можно также командой ALTER. Однако необходимо помнить, что команда ALTER TRIGGER используется только для перекомпиляции, включения или выключения триггера.

4.11. Удаление триггеров.

Вы можете удалить триггеры разными способами: на странице SQL команд, на странице просмотра объектов или с помощью SQL команды DROP. Также Вы можете удалить все зависимые от триггера объекты.

В некоторых случаях можно не удалять триггер, а просто отключить его. Это позволит в будущем воспользоваться им, как примером.

Пример №9. Удаление триггеров.

```
-- first, drop the audit_sal trigger
DROP TRIGGER audit_sal;
-- then drop the table used by the trigger
DROP TABLE emp_audit;
-- drop the log_salary_increase trigger, then the table used by the trigger
DROP TRIGGER log_salary_increase;
DROP TABLE emp_sal_log;
-- drop the emp_log_update trigger, then the table used by the trigger
DROP TRIGGER emp_log_update;
DROP TABLE emp_except_log;
-- drop on_hr_logoff and on_hr_logon triggers, then drop hr_log_table
DROP TRIGGER on_hr_logon;
DROP TRIGGER on_hr_logoff;
DROP TABLE hr_log_table;
```

4.12. Включение и отключение триггеров.

Отключение триггеров может Вам понадобиться в следующих случаях:

- Объект, к которому привязан триггер, не доступен.
- Вы хотите внести большой объем данных и не хотите, чтобы триггер запускался при каждом изменении.
- Вы перезагружаете данные.

Отключить триггер можно двумя способами:

1. С помощь команды ALTER TRIGGER и опции DISABLE.

```
ALTER TRIGGER log_emp_update DISABLE;
```

2. Чтобы отключить все триггеры, относящиеся к одной таблице нужно ввести команду ALTER TABLE с опцией DISABLE ALL TRIGGERS.

```
ALTER TABLE departments DISABLE ALL TRIGGERS;
```

Аналогичная ситуация и с включением триггеров. Также существует два варианта:

1. С помощь команды ALTER TRIGGER и опции ENABLE.

```
ALTER TRIGGER log_emp_update ENABLE;
```

2. Чтобы включить все триггеры, относящиеся к одной таблице нужно ввести команду ALTER TABLE с опцией ENABLE ALL TRIGGERS.

```
ALTER TABLE departments ENABLE ALL TRIGGERS;
```

Стоит добавить, что, по умолчанию, все триггеры после создания получают статус ENABLE.

5. Компиляция триггеров.

Триггеры похожи на анонимные блоки PL\SQL с добавлением свойств :NEW и :OLD, но их компиляция все же несколько отличается. Анонимный блок PL\SQL компилируется каждый раз при загрузки в память. Триггеры же компилируются один раз при выполнении команды CREATE TRIGGER, а их исходный код хранится в памяти. Это означает, что обращение к ним идет напрямую в память.

5.6.Ошибки при создании триггеров.

Если триггер скомпилировался с ошибками, то он все равно попадет в память и будет выполняться. В таких случаях DML выражение, запустившее триггер, будет выполнено с ошибками. Чтобы посмотреть ошибки при компиляции триггера можно ввести команду SHOW ERRORS в командной строке SQL или вы можете обратиться к служебному виду USER_ERRORS с помощью служебного слова SELECT, например, так:

```
SELECT * FROM USER_ERRORS WHERE TYPE = 'TRIGGER';
```

5.7.Зависимости триггеров.

Скомпилированные триггеры зависят от объектов базы данных и перестают работать, если такие объекты, как хранимые процедуры или таблицы, связанные с ними, изменяются. Все триггеры, перестающие работать в связи с изменением зависимых объектов, перекомпилируются при следующем обращении.

Вы можете посмотреть вид ALL_DEPENDENCIES, который содержит все зависимости всех объектов базы данных.

Пример №10. Зависимости триггера.

```
SELECT NAME, REFERENCED_OWNER, REFERENCED_NAME, REFERENCED_TYPE
FROM ALL_DEPENDENCIES
WHERE OWNER = 'HR' and TYPE = 'TRIGGER' AND NAME = 'LOG_EMP_UPDATE';
```

После того, как удаляется зависимая процедура или функция триггер получает свойство «VALID WITH ERRORS», однако к зависимым событиям не относятся события открытия (STARTUP) и закрытия (SHUTDOWN) базы данных.

5.8.Перекомпиляция триггеров.

Для того, чтобы перекомпилировать триггер вручную, Вы можете использовать команду ALTER TRIGGER.

Пример №11. Перекомпиляция триггера вручную.

```
ALTER TRIGGER log_emp_update COMPILE;
-- cleanup: drop the log_emp_update trigger and emp_update_log table
DROP TRIGGER log_emp_update;
DROP TABLE emp_update_log;
```

Чтобы выполнить эти действия, Вы должны обладать правами «ALTER ANY TRIGGER» или владеть этим триггером.

6. Заключение.

В данной лабораторной работе Вы познакомились с существующими встроенными триггерами, с алгоритмом создания, применения и удаления новых объектов и изменения встроенных триггеров.

Надеемся, что полученные знания Вы сможете применить на практике при создании собственных приложений под Oracle 10g X.E.

7. Контрольные вопросы.

6. «Что такое триггер?»
7. «Какие типы триггеров существуют?»
8. «Какие имена допустимы при создании триггеров?»

9. «В каких случаях вызывается триггер?»
10. «Какие существуют точки запуска триггера?»
11. «Какие существуют ограничения при создании триггеров?»
12. «Какими привилегиями необходимо обладать для создания триггера?»
13. «Какими способами можно создать триггер?»
14. «Как можно посмотреть существующие триггеры?»
15. «Как изменить или удалить триггер?»
16. «К чему могут привести ошибки при создании триггеров?»

8. Рекомендуемая литература.

1. Гарсиа – Молина, Ульман, Уидом. Системы баз данных. Полный Курс. Пер. с англ.- М.: Издательский дом Вильямс, 2003 – 1088 стр.
2. Грабер М. Введение в SQL: Пер с англ. – М.:Изд-во'ЛОРИ',1996. – 380с.
3. Джеймс Перри, Джеральд Пост. Введение в Oracle 10g "И.Д. Вильямс", 2006. – 700 с.
4. Гринвальд Рик, Становьяк Робеот, Додж Гери, Кляйн Девид, Шапиро Бен, Челья Кристофер Дж. Программирование баз данных Oracle для профессионалов.: Пер. с англ.: – М. : ООО "И.Д. Вильямс", 2007. – 784 с.
5. Кайт Томас. Oracle для профессионалов: архитектура, методика программирования и основные особенности версии 9i и 10j.: Пер с англ. – М.:Издательский дом "Вильямс". 2008. – 848 с.
6. Кевин Луни, Боб Брила. Oracle Database 10g. Настольная книга администратора баз данных. – М.:Издательство'Лори',2008. – 732 с.
7. Райан стивенс, Рональд Плю. SQL. Пер с англ. – М.:ЗАО 'Издательство Бином',1998.-400 с.