SQL AND ORACLE

SIT103 Lecture 6

1

CORRELATED SUB-QUERY

- This is also a nested query with one difference. The nested query refers to a field in an outer query.
- Since a reference is made to an outer query field, the query can NOT be executed once before outer query.

CORRELATED SUB-QUERY - EXAMPLE

- The inner query refers to a field from the outer query.

```
SELECT STUDENT_NO, SURNAME, GIVEN
FROM STUDENT
WHERE 3 =
      (SELECT COUNT(*)
       FROM STUD_COURSE
       WHERE STUD_COURSE.STUDENT_NO
                  = STUDENT.STUDENT_NO);
```

Same Table

Correlated Sub-Query - execution

- Execution of correlated queries:

1. For each record in the outer query:
   A) execute inner sub-query
   B) produce result
   C) use result in the execution of the outer query

EXISTS and Sub-Queries

- Check if the sub-query returns any answer.
- NOT version also possible

```
SELECT STUDENT_NO, SURNAME, GIVEN
FROM STUDENT
WHERE EXISTS
      (SELECT *
       FROM STUD_COURSE
       WHERE STUD_COURSE.STUDENT_NO
                  = STUDENT.STUDENT_NO);
```

NOTE: Can also use EXISTS, NOT EXISTS, IN, NOT IN instead of "<".

PL/SQL (106)

- PL/SQL is procedural programming language roughly based on SQL.
- It has similar constructs to any other programming language such as variables, IF statement and Loops
- It also has special constructs such as cursors to allow looping through a table one row at a time.
- Results of SQL statements such as SELECT are not displayed to the user, but instead put into variables.

Example of PL/SQL

```
DECLARE
-- variable declaration
message varchar2(20):= 'Hello, World!';
BEGIN
/* * PL/SQL executable statement(s) */
dbms_output.put_line(message);
END;
```

Stored Procedures

- A Stored procedure is a named block of procedural code which is compiled and stored on the server, in the schema of the user who created it.
- It is the same, conceptually, as a subroutine in any other programming language - a method in Java.
- It can be passed parameters.
- It can be called with the EXEC command from another procedure or another program.
- Stored procedures have no user input or output - they would normally store results into tables.

Stored Procedure Example

```
CREATE OR REPLACE
PROCEDURE addNewInvoice (givenCustomer
  NUMBER) AS
BEGIN
  INSERT INTO INVOICE VALUES(
  INVNUM.NEXTVAL, SYSDATE, givenCustomer);
END ;
/
```

- To call this procedure from the SQLPlus prompt, from another procedure, or from VB or Java:

```
EXEC addNewInvoice( 127 ) ;
```

Stored Procedure Notes

- Note the parameter in the previous example.
- You can type the code directly at the SQL-Plus prompt, or into a file, as with a SQL query.
- You signal the end of a PL/SQL block with a full stop ., or slash / on a line by itself.
- When you start the file, Oracle attempts to compile and store your code block.

User Defined Functions

- Functions are similar to Stored Procedures.
- They are named and stored on the server in the schema of the person who created them
- They can be made available to other users.
- Syntax rules are exactly the same
- Main difference a function returns a value.

User Defined Function Example

```
CREATE OR REPLACE
FUNCTION theYear( theDate DATE) RETURN
  NUMBER IS
BEGIN
  RETURN TO_NUMBER( TO_CHAR( theDate, 'YYYY'
  ) ) ;
END theYear;
/
```

Function Use

- Users with execute privilege on function theYear could then use it in a SQL statement.

```
SELECT theYear( DOB) FROM STUDENT ;
SELECT theYear(SYSDATE) + 20 FROM DUAL ;
```

- The user who created the function automatically has all privileges on it.
- S/he can grant the execute privilege to others:

```
GRANT EXECUTE ON theYear TO FRED;
```

Triggers

- Triggers are similar to Stored Procedures and Functions in that :
  - they are named blocks of PL/SQL code
  - they are compiled and stored on the database server
- What differs - they are not called explicitly by a user, procedure, function or program.
- When triggers are defined, they are attached to a particular table, for particular events such as INSERT, UPDATE or DELETE.

Triggers

- Triggers are fired when the corresponding triggering event happens on the table.
  - Eg: a user issues an INSERT command, or UPDATE or DELETE
- They are also often used to implement complex auditing and updating:
  - Eg: to update a stock on hand value when a sale occurs

TRIGGER SYNTAX

- The general syntax is :

```
CREATE OR REPLACE TRIGGER trigger_name
{BEFORE | AFTER} triggering_event  ON
table_reference
[FOR EACH ROW [WHEN trigger_condition] ]
trigger_body ;
```

- You can query the data dictionary view user_triggers to see trigger details.

```
    SELECT trigger_type, table_name,
triggering_event
    FROM USER_TRIGGERS ;
```

TRIGGER EXAMPLE 1 (NEW)

```
CREATE OR REPLACE TRIGGER atleast15
AFTER INSERT OR UPDATE OF DOB ON
  STUDENT
FOR EACH ROW
DECLARE   thedob DATE ;
BEGIN
  thedob :=  :new.DOB;
  IF (( SYSDATE - thedob ) < ( 15 * 365.25)) THEN
      RAISE_APPLICATION_ERROR( -20000,
            'Students must be at least 15 years old');
  END IF;
END ;
/
```

:NEW AND :OLD (NEW)

- **:new** and **:old** are pseudo tables with exactly the same structure as the trigger table.
- These pseudo tables hold the replacing (after) and replaced (before) values.
- You can reference them only in a trigger.
- Each RDBMS which supports triggers provides similar pseudo tables. MS SQL Server calls them inserted and deleted.

TRIGGER EXAMPLE 2

- Suppose we want to record each time a student changes programmes. We log these changes into a log table created thus:

```
CREATE TABLE PGM_CHANGE
(  STUDENT_NO          CHAR(8),
   FROM_PGM            VARCHAR2(6),
   TO_PGM              VARCHAR2(6),
   CHANGE_DATE              DATE );
```

TRIGGER EXAMPLE 2

```
CREATE OR REPLACE TRIGGER logPgmChange
BEFORE UPDATE ON STUDENT
FOR EACH ROW
WHEN ( new.PGMCODE != old.PGMCODE )
   -- don't use colon in WHEN clause, in trigger body
   only
BEGIN
   INSERT INTO PGM_CHANGE VALUES (
   :old.STUDENT_NO, :old.PGMCODE,
   :new.PGMCODE, SYSDATE);
END logPgmChange;
```

INDEXES

- An index speeds up searching, sorting and joining operations
- Indexes slow down updates, however.
- Index is simply created.
- System handles all updates to the index.
- The system decides if the index will be used.
- A command cannot specify the use of an index.

INDEXES

- CREATE INDEX STUDSURN
  ON STUDENT(SURNAME);

- CREATE UNIQUE INDEX STUCRSPRIMARY
  ON
  STUD_COURSE(STUDENT_NO,COURSE_CODE);

Note: above creates a primary key type index – sometimes used for candidate keys

INDEXES

- Create Indexes on columns used for:
  - Primary keys (unique)
  - Foreign keys
  - search fields
  - ordering fields
- Do NOT create indexes on:
  - Fields with few different values
  - Small tables
  - NULL values

ORACLE SEQUENCES

- Sequences are used to generate unique numbers for primary keys. Oracle does not have a type to do this.

  CREATE SEQUENCE TRANSIDSEQ
  START WITH 1000 INCREMENT BY 1;

- The sequence is used explicitly within an INSERT statement to get the next value by referencing the NEXTVAL variable.

Oracle Sequences

INSERT INTO TRANSACTION
VALUES(TRANSIDSEQ.NEXTVAL,……)

- Each reference to TRANSIDSEQ.NEXTVAL increments it.
- To see the current value without incrementing it, you can reference TRANSIDSEQ.CURRVAL.

System Views (new)

- To find out about various objects in the database, there hundreds of views defined that display system information.
- Many system views are only usable by the DBA but there are many that a normal user can query to find out about their objects such as tables, views, indexes etc.
- It is useful to know some of these.

System Views - Some Examples (n)

- USER_TABLES
- USER_VIEWS
- USER_CONSTRAINTS
- USER_INDEXES
- USER_SYNONYMS
- USER_TRIGGERS

DUAL System Table (new)

- **DUAL** is a table with one column, DUMMY VARCHAR2(1), and one row with the value 'X'.
- It is useful for computing a single constant expression with the SELECT command.
- Because DUAL has only one row, the constant is only returned once. Eg:

```
SELECT SYSDATE FROM DUAL ;
```

- You could select SYSDATE from any table, but Oracle would return SYSDATE once for each row.