Lesson-01 : Privileges, Multi-Table Inserts, External Tables

Privileges

- Oracle Database security:
  - System security

    Accessing the Oracle Database
    Done using username and password

  - Data security

- System privileges: Gaining access to the database

- Object privileges: Manipulating the content of the database objects
  i.e. For a table do we have insert permission, update permission or delete permission.

- Schemas: Collections of objects, such as tables, views, and sequences

System Privileges

- More than 100 privileges are available.
- The database administrator (DBA) has high-level system privileges for tasks such as:
  - Creating new users
  - Removing users
  - Removing tables
  - Backing up tables

Create Users

The DBA creates users by using the CREATE USER statement.

```
NOTE: Connect as DBA using:  conn sys as sysdba
                             Enter password:
```

Syntax:

```
CREATE USER user IDENTIFIED BY password;
```

Example:

```
SQL> CREATE USER amar IDENTIFIED BY amar123;

User created.
```

NOTE: Just creating a user will not allow the user to connect to the Oracle Database.
Bare minimum, the user should have CREATE SESSION privilege

Otherwise we get an error as shown below

```
SQL> conn amar
Enter password:
ERROR:
ORA-01045: user AMAR lacks CREATE SESSION privilege; logon denied
```

User System Privileges
======================

- Once a user is created, the DBA can grant specific system privileges to a user.

Syntax:

```
GRANT privilege [, privilege...]
TO user [, user| role, PUBLIC...]
```

- An application developer, for example, may have the following system privileges:
  - CREATE SESSION
  - CREATE TABLE
  - CREATE SEQUENCE
  - CREATE VIEW
  - CREATE PROCEDURE

Granting System Privileges
==========================

The DBA can grant specific system privileges to a user.

Now, issuing CREATE SESSION privileage to user 'amar'

Example:

```
SQL> GRANT create session TO amar;

Grant succeeded.

SQL> conn amar
Enter password:
Connected.

SQL> select user from dual;

USER
------------------------------
AMAR

SQL> CREATE TABLE abcd
  2  (
  3     col1   number
  4  );
CREATE TABLE abcd
*
ERROR at line 1:
ORA-01031: insufficient privileges
```

Thus, we can connect once again as DBA and grant the required privileges

```
SQL> GRANT create table, create view, create sequence TO amar;

Grant succeeded.
```

What is a Role?

Oracle provides for easy and controlled privilege management through roles.

Roles are named groups of related privileges that you grant to users or other roles.

Roles are designed to ease the administration of end-user system and object privileges.

Creating & Granting Privileges to Role

[1] Create a Role

```
SQL> CREATE ROLE manager;

Role created.
```

[2] Granting privileges to a role

```
SQL> GRANT create table, create view TO manager;

Grant succeeded.
```

[3] Grant a role to user(s)

```
SQL> GRANT manager TO akbar, antony;

Grant succeeded.
```

Checking if the role and the users assigned with the role is working or not.

Example

```
SQL> conn antony
Enter password:
Connected.
SQL> select user from dual;

USER
------------------------------
ANTONY

SQL> CREATE TABLE abcd
  2  (
  3     col1   number
  4  );

Table created.

SQL> CREATE VIEW abcd_vu AS SELECT * FROM abcd;

View created.

SQL>
```

Changing Password
=================

The DBA creates your user account and initializes your password.

You can change your password by using the ALTER USER statement

```
ALTER USER user IDENTIFIED BY newpassword;
```

Object Privileges
=================

The different database object like Table, View, Sequence etc can have their own privileges.

Such privileges are called Object Privileges.

Example:
The Database TABLE can have ALTER, INDEX, INSERT, UPDATE, DELETE, SELECT, REFERENCE etc.

Similarly, the Sequence object can have ALTER, SELECT etc.

- Object privileges vary from object to object.
- An owner has all the privileges on the object.
- An owner can give specific privileges on that owner's object.

Syntax:

```
GRANT object_priv [(columns)] ON object
TO {user|role|PUBLIC}
[WITH GRANT OPTION];
```

Example:

```
SQL> conn hr
Enter password:
Connected.
SQL> GRANT select ON employees TO amar, akbar, antony;

Grant succeeded.
```

Now connect as user amar or akbar or antony and check to see if the 'select' privilege on the 'employees' table works or not.

```
SQL> select user from dual;

USER
------------------------------
AMAR

SQL> SELECT * FROM hr.employees;
```

We should be able to see the rows of the employees table.

NOTE: Don't forget to prefix the schema name before the
      table name; delimited by a DOT

Example-2

```
SQL> conn antony
Enter password:
Connected.
```

```
SQL> SELECT first_name FROM hr.employees WHERE department_id IN (10,
110);

FIRST_NAME
--------------------
Jennifer
Shelley
William
```

Example-3

```
SQL> UPDATE hr.employees SET first_name = 'Willy' WHERE employee_id =
206;
UPDATE hr.employees SET first_name = 'Willy' WHERE employee_id = 206
          *
ERROR at line 1:
ORA-01031: insufficient privileges
```

Example-4
Connect as 'hr' and issuing the following GRANT

```
SQL> GRANT update(department_name, location_id) ON departments
  2  TO amar, manager;

Grant succeeded.
```

Using the WITH GRANT OPTION and PUBLIC keywords
===

The WITH GRANT OPTION clause

- Give a user authority to pass along privileges.

Example

```
GRANT select, insert ON departments
TO amar
WITH GRANT OPTION;
```

- Allow all users on the system to query data from Hr’s LOCATIONS table.

```
GRANT select ON hr.locations
TO PUBLIC;
```

Now, all the users of the Oracle DB have select privilege on the 'locations' table.

Revoking Object Privileges
==========================

To revoke object privileges, use the REVOKE statement

- You use the REVOKE statement to revoke privileges granted to other users.
- Privileges granted to others through the WITH GRANT OPTION clause are also revoked.

Syntax:

```
REVOKE {privilege [, privilege...]|ALL} ON object
FROM {user[, user...]|role|PUBLIC}
[CASCADE CONSTRAINTS];
```

Example
As user HR revoke the SELECT and INSERT privileges given to user Amar on the DEPARTMENTS table.

```
SQL> REVOKE select, insert ON departments
  2  FROM amar;

Revoke succeeded.
```

Using Subqueries to Manipulate Data

We can use subqueries in DML statements also.
Usage of subqueries:

- Retrieve data by using an inline view
- Copy data from one table to another
- Update data in one table based on the values of another table
- Delete rows from one table based on rows in another table

Example : Fetching the department name and the city which are there in Europe region

```
SQL> SELECT department_name, city
  2  FROM departments
  3  NATURAL JOIN ( SELECT l.location_id, l.city, l.country_id
  4                 FROM locations l
  5                 JOIN countries c
  6                 ON (l.country_id = c.country_id)
  7                 JOIN regions USING (region_id)
  8                 WHERE region_name = 'Europe'
  9               );

DEPARTMENT_NAME                CITY
------------------------------ ------------------------------
Human Resources                London
Sales                          Oxford
Public Relations               Munich
```

Inserting using a subquery as target

```
SQL> -- Inserting by using a subquery as target
SQL> INSERT INTO ( SELECT l.location_id, l.city, l.country_id
  2                FROM locations l
  3                JOIN countries c
  4                ON (l.country_id = c.country_id)
  5                JOIN regions USING (region_id)
  6                WHERE region_name = 'Europe'
  7              )
  8  VALUES (3300, 'Cardiff', 'UK');

1 row created.

SQL> SELECT location_id, city, country_id
  2  FROM locations;

LOCATION_ID CITY                           CO
----------- ------------------------------ --
       3300 Cardiff                        UK
```

Explicitly DEFAULT Features
===========================

Use the DEFAULT keyword as a column value where the default column value is desired.

This allows the user to control where and when the default value should be applied to data.

Explicit defaults can be used in INSERT and UPDATE statements.

Example

```
SQL> INSERT INTO customer VALUES(5, 'Adnan', 'M', 'smartguy@yahoo.com',
25, DEFAULT);

SQL> UPDATE customer
  2  SET custcity = DEFAULT
  3  WHERE custid = 1;
```

NOTE: If a DEFAULT value DOES NOT exist, NULL is taken into account.

Copying Rows From Another Table
===============================

Use the subquery with INSERT statement to copy rows from another table.

Do not use VALUES clause.

Match the number of columns in the INSERT clause with that in the subquery.

NOTE: However, the table structure should exist.

Example

```
SQL> CREATE TABLE sales_reps AS SELECT employee_id ID, last_name NAME,
salary, commission_pct
  2  FROM employees WHERE 1 = 5;

Table created.

SQL> INSERT INTO sales_reps ( id, name, salary, commission_pct )
  2  SELECT employee_id, last_name, salary, commission_pct
  3  FROM employees
  4  WHERE job_id LIKE '%REP%';
```

Multi-Table Insert
==================

The INSERT ALL statement has the ability to insert the rows in multiple tables.

Multi table INSERT statements are used in data warehousing systems to transfer data from one or more operational sources to a set of target tables.

They provide significant performance improvement over:

Single DML versus multiple INSERT…SELECT statements

Single DML versus a procedure to perform multiple inserts by using the IF...THEN

Types of MultiTable Insert Statements
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

[1] Unconditional Insert
[2] Conditional Insert
[3] Conditional Insert First
[4] Pivoting Insert

Syntax for Unconditional Insert:

```
INSERT ALL
      INTO target_a VALUES (.....)
      INTO target_b VALUES (.....)
      :
SELECT ....
FROM source_table
WHERE....;
```

Example

[1] Unconditional Insert All

```
INSERT ALL
   INTO sal_history VALUES (EMPID, HIREDATE, SAL)
   INTO mgr_history VALUES (EMPID, MGR, SAL)
SELECT employee_id EMPID, hire_date HIREDATE, salary SAL, manager_id MGR
FROM employees
WHERE employee_id > 200;
```

[2] Conditional Insert All

We want to insert into 'EMP_HISTORY' table those employees who joined before 1995 and insert into 'EMP_SALES' table those employees who have commission.

Example

```
SQL> -- Coniditonal Insert All
SQL>
INSERT ALL
   WHEN HIREDATE < '01-JAN-03' THEN
      INTO emp_history VALUES (EMPID, HIREDATE, SAL)
   WHEN COMM IS NOT NULL THEN
      INTO emp_sales VALUES (EMPID, COMM, SAL)
SELECT employee_id EMPID, hire_date HIREDATE, salary SAL, commission_pct
COMM
FROM employees;

43 rows created.
```

[3] Conditional Insert First

```
SQL> -- Conditional Insert First
SQL> INSERT FIRST
  2    WHEN salary < 5000 THEN
  3         INTO sal_low VALUES (employee_id, last_name, salary)
  4    WHEN salary BETWEEN 5000 AND 10000 THEN
  5         INTO sal_mid VALUES (employee_id, last_name, salary)
  6    ELSE
  7         INTO sal_high VALUES (employee_id, last_name, salary)
```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

```
  8  SELECT employee_id, last_name, salary
  9  FROM employees;
```

[4] Pivoting Insert

Convert the set of sales records from the non-relational table to a relational format.

Example:

```
SQL> desc  sales_source_data
 Name                                      Null?    Type
 ----------------------------------------- -------- ---------------------
-------
 EMPLOYEE_ID                                        NUMBER(3)
 WEEK_ID                                            NUMBER(2)
 SALE_MON                                           NUMBER(4)
 SALE_TUE                                           NUMBER(4)
 SALE_WED                                           NUMBER(4)
 SALE_THU                                           NUMBER(4)
 SALE_FRI                                           NUMBER(4)

SQL> SELECT * FROM sales_source_data;

EMPLOYEE_ID    WEEK_ID   SALE_MON   SALE_TUE   SALE_WED   SALE_THU
SALE_FRI
----------- ---------- ---------- ---------- ---------- ---------- ------
----
        176          6       2000       3000       4000       5000
6000

SQL> desc sales_info
 Name                                      Null?    Type
 ----------------------------------------- -------- ---------------------
-------
 EMPLOYEE_ID                                        NUMBER(3)
 WEEK_ID                                            NUMBER(2)
 SALES                                              NUMBER(4)

INSERT ALL
  INTO sales_info VALUES (employee_id, week_id, sale_mon)
  INTO sales_info VALUES (employee_id, week_id, sale_tue)
  INTO sales_info VALUES (employee_id, week_id, sale_wed)
  INTO sales_info VALUES (employee_id, week_id, sale_thu)
  INTO sales_info VALUES (employee_id, week_id, sale_fri)
SELECT employee_id, week_id, sale_mon, sale_tue, sale_wed, sale_thu,
sale_fri
FROM sales_source_data;

5 rows created.
```

Now, look into the SALES_INFO table;

```
SQL> SELECT * FROM sales_info;

EMPLOYEE_ID    WEEK_ID      SALES
----------- ---------- ----------
        176          6       2000
        176          6       3000
        176          6       4000
```

Tracking Changes in Data

======================

The Version Query will help us understand the older value(s)/data of table(s)

A example flashback version query is as follows:

#1 - Find the existing salary of an employee

```
SQL> SELECT salary FROM employees
  2  WHERE employee_id = 107;

    SALARY
----------
      4200
```

#2 - Update the salary. Increment by 30%

```
SQL> UPDATE employees
  2  SET salary = salary * 1.30
  3  WHERE employee_id = 107;

1 row updated.  --The salary of employee_id 107 will now be 5460
```

#3 - Using the flashback version query to know the old value

```
SQL> SELECT salary FROM employees
  2  VERSIONS BETWEEN SCN MINVALUE AND MAXVALUE
  3  WHERE employee_id = 107;

    SALARY
----------
      5460       -- New value
      4200       -- Old value
```

Example

```
SQL> SELECT versions_starttime "Start Date",
            versions_endtime "End Date",
            salary
     FROM employees
     VERSIONS BETWEEN SCN MINVALUE AND MAXVALUE
     WHERE employee_id = 107;
```

External Tables

===============

#1

Create a directory for the external table

We need to create a DIRECTORY object that corresponds to the directory on the file system where the external data source resides.

```
SQL> CREATE OR REPLACE DIRECTORY emp_dir AS 'C:\Temp';

Directory created.
```

i.e. Now within the Oracle Database emp_dir will actually be refering to C:\temp directory in our File System.

In case of Linux/UNIX

```
SQL> CREATE OR REPLACE DIRECTORY emp_dir AS '/home/harshan';
```

To create an External Table, the syntax is as follows:

```
CREATE TABLE <table_name>
( <col_name> <datatype>, … )            # Spefiying column names and
datatypes
ORGANIZATION EXTERNAL              # Observe ORGANIZATION EXTERNAL
(TYPE <access_driver_type>
DEFAULT DIRECTORY <directory_name># Place we use the directory object
ACCESS PARAMETERS
(… ) )
LOCATION ('<location_specifier>')
REJECT LIMIT [0 | <number> | UNLIMITED];
```

Now, creating an external table using Oracle Loader as follows:

```
SQL> CREATE TABLE oldemp (
  2  fname CHAR(20), lname CHAR(20))
  3  ORGANIZATION EXTERNAL
  4  (TYPE ORACLE_LOADER
  5  DEFAULT DIRECTORY emp_dir
  6  ACCESS PARAMETERS
  7  (RECORDS DELIMITED BY NEWLINE
  8  NOBADFILE
  9  NOLOGFILE
 10  FIELDS TERMINATED BY ','
 11  (fname POSITION (1:20) CHAR,
 12   lname POSITION (22:41) CHAR ))
 13  LOCATION ('emp1.txt'))
 14  PARALLEL 5
 15  REJECT LIMIT 200;

Table created.
```

Querying the External Table

```
SQL> SELECT * FROM oldemp;

FNAME                LNAME
-------------------- --------------------
Apoorva              Shukla
Zahwa                Haque
Mohammed             Mukthar
Aarthi               Murgan
Pathmesh             Dhaikar
```

The above query has actually fetched the data from 'emp1.txt' file existing in 'C:\Temp' directory which is having two fields, namely (fname and lname) as specified at the time creating an external table.

Creating an External Table Using Oracle DataPump

```
SQL> CREATE TABLE emp_ext
  2  (employee_id, first_name, last_name)
  3  ORGANIZATION EXTERNAL
  4  (
  5    TYPE ORACLE_DATAPUMP
  6    DEFAULT DIRECTORY emp_dir
  7    LOCATION
  8    ('emp1.dat')
  9  )
 10  PARALLEL
 11  AS
 12  SELECT employee_id, first_name, last_name
 13  FROM hr.employees;

Table created.
```

Querying the external table

```
SQL> SELECT * FROM emp_ext;

EMPLOYEE_ID FIRST_NAME           LAST_NAME
----------- -------------------- -------------------------
        100 Steven               King
        101 Neena                Kochhar
        102 Lex                  De Haan
        103 Alexander            Hunold
        104 Bruce                Ernst
        105 David                Austin
```