|  |
| --- |
| Introduction to Database |
| Lab Manual Finalterm |
| Aiub-1jpg.JPG  **American International University-Bangladesh** |

# Lab 7 (Sub-query)

#### Objective:

* + Describe the types of problems that subqueries can solve
  + Define subqueries
  + List the types of subqueries
  + Write single-row and multiple-row subqueries

In this lab, you will learn about more advanced features of the SELECT statement. You can write subqueries in the WHERE clause of another SQL statement to obtain values based on an unknown conditional value. This lab covers single-row subqueries and multiple-row subqueries.

### Subqueries

A subquery is a SELECT statement that is embedded in a clause of another SELECT statement. You can build powerful statements out of simple ones by using subqueries. They can be very useful when you need to select rows from a table with a condition that depends on the data in the table itself.

You can place the subquery in a number of SQL clauses:

* + WHERE clause
  + HAVING clause
  + FROM clause In the syntax:

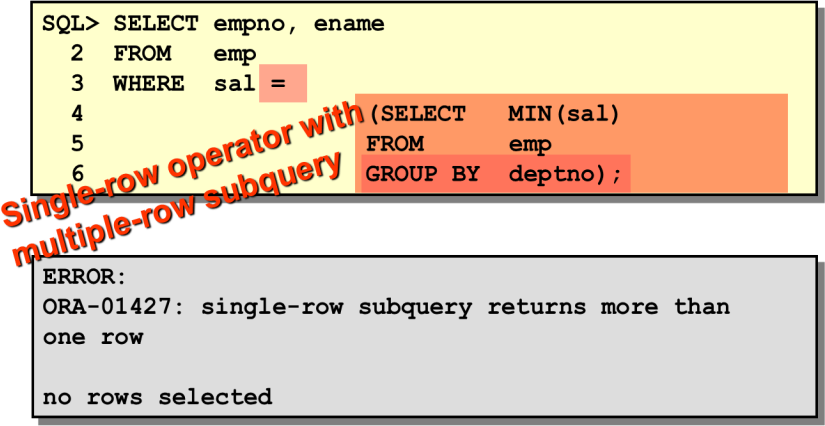
*operator* includes a comparison operator such as >, =, or IN

### Guidelines for Using Subqueries

* + A subquery must be enclosed in parentheses.
  + A subquery must appear on the right side of the comparison operator.
  + Subqueries cannot contain an ORDER BY clause. You can have only one ORDER BY clause for a SELECT statement, and if specified it must be the last clause in the main SELECT statement.
  + Two classes of comparison operators are used in subqueries: single-row operators and multiple-=row operators.

### Errors with Subqueries

One common error with subqueries is more than one row returned for a single-row subquery. To correct this error, change the = operator to IN.



***Exercise:***

1. Display all the employees who are earning more than all the managers.
2. Display all the employees who are earning more than any of the managers.
3. Select employee number, job & salaries of all the Analysts who are earning more than any of the managers.
4. Select all the employees who work in DALLAS.
5. Select department name & location of all the employees working for CLARK.
6. Select all the departmental information for all the managers
7. Display the first maximum salary.
8. Display the second maximum salary.
9. Display the third maximum salary.
10. Display all the managers & clerks who work in Accounts and Marketing departments.
11. Display all the salesmen who are not located at DALLAS.
12. Get all the employees who work in the same departments as of SCOTT.
13. Select all the employees who are earning same as SMITH.
14. Display all the employees who are getting some commission in marketing department where the employees have joined only on weekdays.
15. Display all the employees who are getting more than the average salaries of all the employees.

# Lab 8 (Joining: Displaying Data from Multiple Tables)

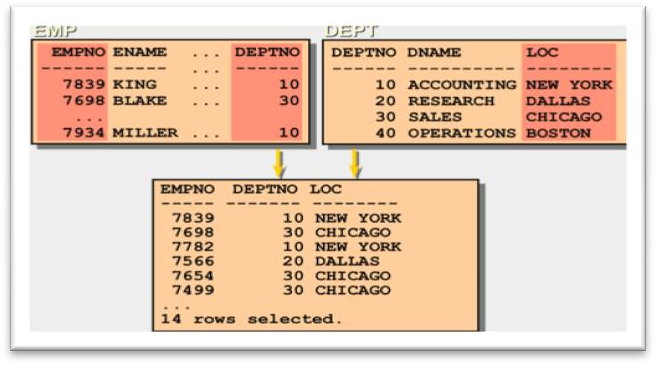
#### Objective:

* Write SELECT statements to access data from more than one table using equality and nonequality joins.
* View data that generally does not meet a join condition by using outer joins.
* Join a table to itself.

#### Data from Multiple Tables

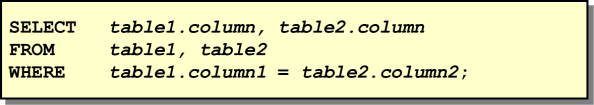
Sometimes you need to use data from more than one table. In the example stated below, the report displays data from two separate tables.

* EMPNO exists in the EMP table.
* DEPTNO exists in both the EMP and DEPT the tables.
* LOC exists in the DEPT table.



To produce the report, you need to link EMP and DEPT tables and access data from both of them.

**Defining Joins**

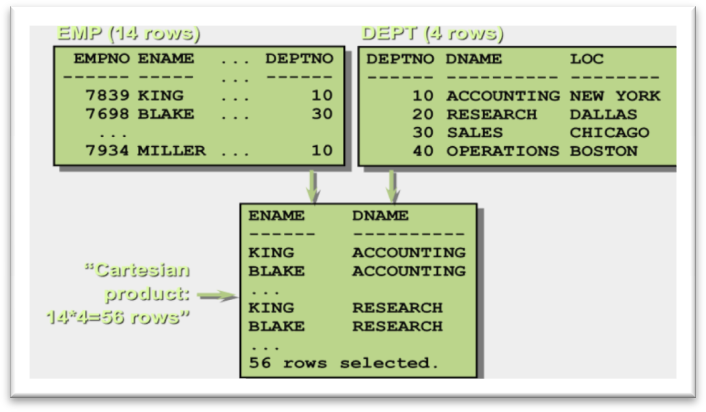


In the syntax:

*table1.column* denotes the table and column from which data is retrieved

*table1.column1* = is the condition that joins (or relates) the tables together *table2.column2*

### Cartesian Product



### Types of Joins

There are two main types of join conditions:

* Equijoins
* Non-equijoins

Additional join methods include the following:

* Outer joins
* Self joins

### Equijoins

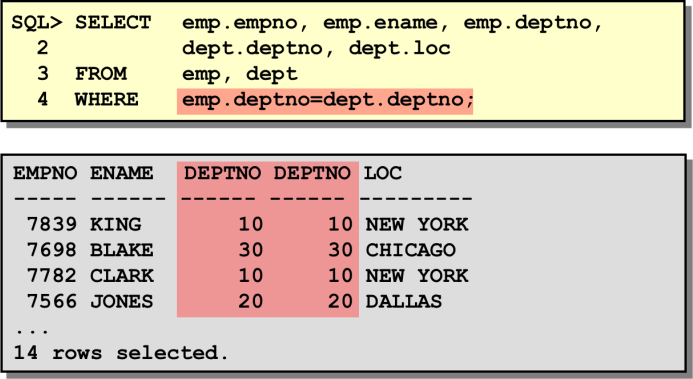
To determine the name of an employee’s department, you compare the value in the DEPTNO column in the EMP table with the DEPTNO values in the DEPT table. The relationship between the EMP and DEPT tables is an *equijoin*—that is, values in the DEPTNO column on both tables must be equal. Frequently, this type of join involves primary and foreign key complements. Equijoins are also called *simple joins* or *inner joins*.

### Retrieving Records with Equijoins

In the below example:

* The SELECT clause specifies the column names to retrieve:
  + employee name, employee number, and department number, which are columns in the EMP table
  + department number, department name, and location, which are columns in the DEPT table
* The FROM clause specifies the two tables that the database must access:
  + EMP table
  + DEPT table
* The WHERE clause specifies how the tables are to be joined: EMP.DEPTNO=DEPT.DEPTNO

Because the DEPTNO column is common to both tables, it must be prefixed by the table name to avoid ambiguity.



### Qualifying Ambiguous Column Names

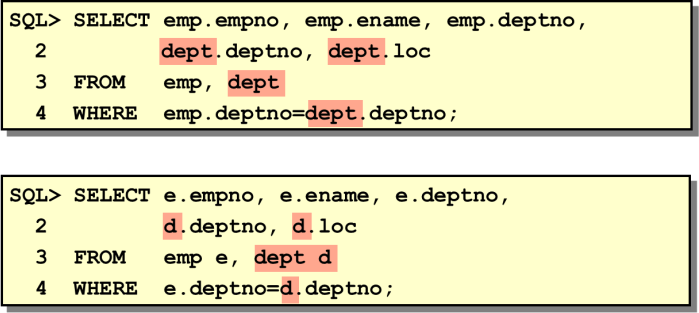
You need to qualify the names of the columns in the WHERE clause with the table name to avoid ambiguity.

### Table Aliases

Qualifying column names with table names can be very time consuming, particularly if table names are lengthy. You can use table *aliases* instead of table names.

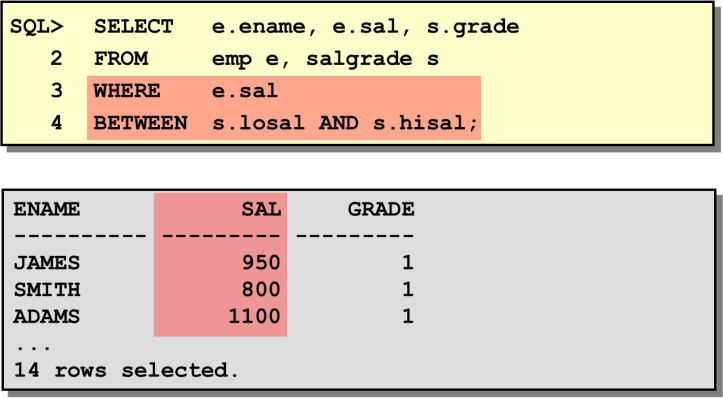
### Guidelines

* Table aliases can be up to 30 characters in length, but the shorter they are the better.
* If a table alias is used for a particular table name in the FROM clause, then that table alias must be substituted for the table name throughout the SELECT statement.
* Table aliases should be meaningful.
* The table alias is valid only for the current SELECT statement.



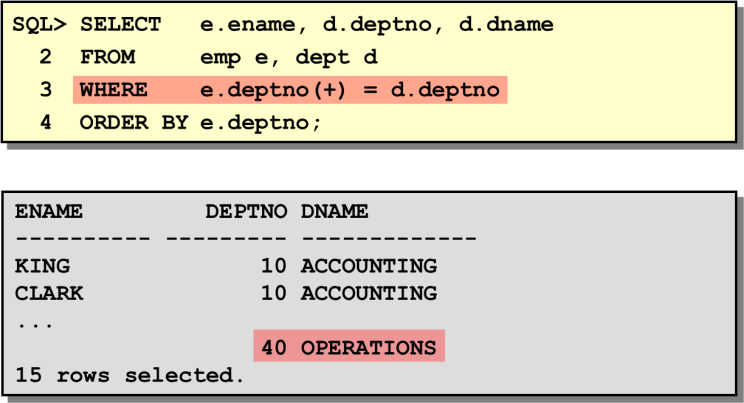
### Non-Equijoins

The relationship between the EMP table and the SALGRADE table is a non-equijoin, meaning that no column in the EMP table corresponds directly to a column in the SALGRADE table. The relationship between the two tables is that the SAL column in the EMP table is between the LOSAL and HISAL column of the SALGRADE table. The relationship is obtained using an operator other than equal (=).



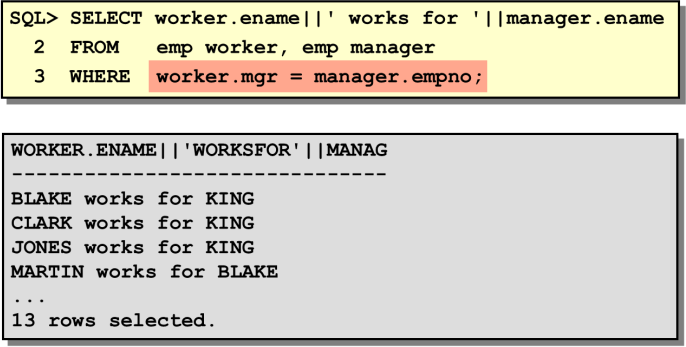
### Returning Records with No Direct Match with Outer Joins

If a row does not satisfy a join condition, the row will not appear in the query resultThe operator is a plus sign enclosed in parentheses (+), and it is *placed on the* “*side*” *of the join that is deficient in information*.



### Joining a Table to Itself by SelfJoin

Sometimes you need to join a table to itself. To find the name of each employee’s manager, you need to join the EMP table to itself, or perform a self-join.



The example stated above joins the EMP table to itself. To simulate two tables in the FROM clause, there are two aliases, namely WORKER and MANAGER, for the same table, EMP. In this example the WHERE clause contains the join condition that means “where a worker’s manager number matches the employee number for the manager.”

***Exercise:***

* 1. Display all the managers & clerks who work in Accounts and Marketing departments. 2. Display all the salesmen who are not located at DALLAS.

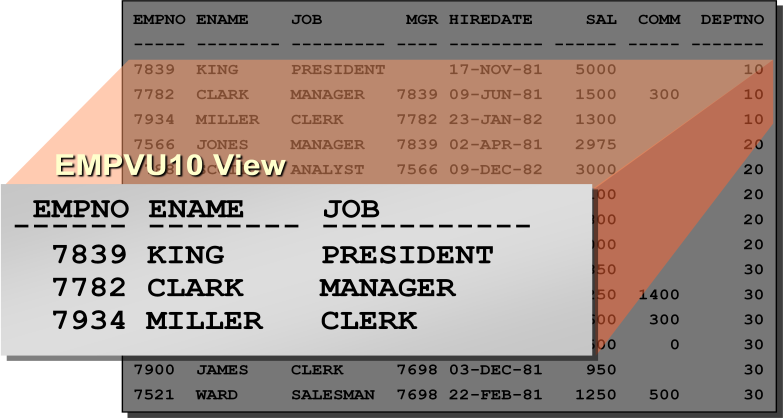
3. Select department name & location of all the employees working for CLARK. 4. Select all the departmental information for all the managers

1. Select all the employees who work in DALLAS.
2. Delete the records from the DEPT table that don’t have matching records in EMP
3. Display all the departmental information for all the existing employees and if a department has no employees display it as “No employees”.
4. Get all the matching & non-matching records from both the tables.
5. Get only the non-matching records from DEPT table (matching records shouldn’t be selected).
6. Select all the employees name along with their manager names, and if an employee does not have a manager, display him as “CEO”.
7. Get all the employees who work in the same departments as of SCOTT
8. Display all the employees who have joined before their managers.
9. List all the employees who are earning more than their managers.
10. Fetch all the employees who are earning same salaries.
11. Select all the employees who are earning same as SMITH.Display employee name , his date of joining, his manager name & his manager's date of joinin

# Lab 9 (View)

#### View:

You can present logical subsets or combinations of data by creating views of tables. A view is a logical table based on a table or another view..



**Simple Syntax:**

CREATE VIEW <view\_name> AS

SELECT <col>,<col> FROM <table\_name> WHERE <condition> ;

**Complex Syntax:**

CREATE [OR REPLACE] [FORCE|NOFORCE] VIEW <view> [(alias[, alias]...)]

AS <subquery>

[WITH CHECK OPTION [CONSTRAINT constraint]] [WITH READ ONLY];

### Retrieving Data from a View

You can retrieve data from a view as you would from any table.

Syntax: *SELECT \* FROM <view\_name>;*

### Modifying a View

The OR REPLACE option allows a view to be created even if one exists with this name already

### Performing DML Operations on a View

You can perform DML operations on data through a view if those operations follow certain rules.

* You can remove a row from a view unless it contains any of the following:
* Group functions
* A GROUP BY clause
* The DISTINCT keyword

You can modify data in a view unless it contains any of the conditions mentioned in the previous section and any of the following:

* Columns defined by expressions—for example, SALARY \* 12
* The ROWNUM pseudocolumn

You can add data through a view unless it contains any of the above and there are NOT NULL columns, without a default value, in the base tabl that are not selected by the view.

### Removing a View

You use the DROP VIEW statement to remove a view.

**Synntax:** *DROP VIEW <view\_name>;*

### Exercise

1. Create a view called **EMP\_VU** based on the employee number, employee name, and department number from the EMP table. Change the heading for the employee name to EMPLOYEE.
2. Display the contents of the **EMP\_VU** view. EMPNO EMPLOYEE DEPTNO

|  |  |  |  |
| --- | --- | --- | --- |
| 7839 | KING |  | 10 |
| 7698 | BLAKE |  | 30 |
| 7782 | CLARK |  | 10 |
| 7566 | JONES |  | 20 |
| 7654 | MARTIN |  | 30 |
| 7499 | ALLEN |  | 30 |
| 7844 | TURNER |  | 30 |
| 7900 | JAMES |  | 30 |
| 7521 | WARD |  | 30 |
| 7902 | FORD |  | 20 |
| 7369 | SMITH |  | 20 |
| 7788 | SCOTT |  | 20 |
| 7876 | ADAMS | 20 |  |
| 7934 | MILLER |  | 10 |

1. using your view EMP\_VU, enter a query to display all employee names and department numbers.

|  |  |
| --- | --- |
| EMPLOYEE | DEPTNO |
| KING | 10 |
| BLAKE | 30 |
| CLARK | 10 |

JONES 20

MARTIN 30

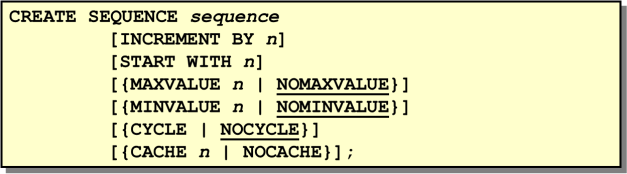
1. Create a view named **DEPT20** that contains the employee number, employee name, and department number for all employees in department 20. Label the view column EMPLOYEE\_ID, EMPLOYEE, and DEPARTMENT\_ID. Do not allow an employee to be reassigned to another department through the view.
2. Create a view called SALARY\_VU based on the employee name, department name, salary, and salary grade for all employees. Label the columns Employee, Department, Salary, and Grade, respectively.

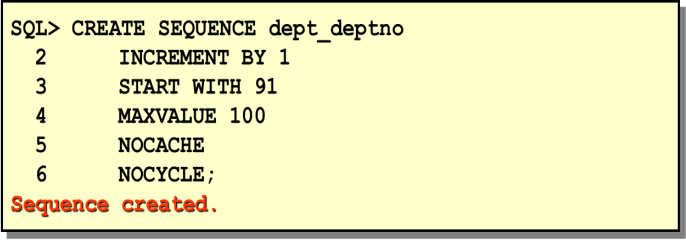
\*\* Please save the SQL commands in a text file for further use.

**Lab 10 (Sequence)**

### What Is a Sequence?

A sequence generator can be used to automatically generate sequence numbers for rows in tables. A sequence is a database object created by a user and can be shared by multiple users. A typical usage for sequences is to create a primary key value

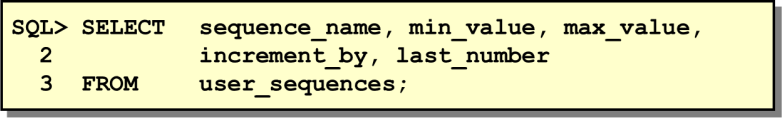




The above example creates a sequence named DEPT\_DEPTNO to be used for the DEPTNO column of the DEPT table. The sequence starts at 91, does not allow caching, and does not allow the sequence to cycle. Do not use the CYCLE option if the sequence is used to generate primary key values.

### Confirming Sequences

You can also confirm the settings of the sequence by selecting from the data dictionary USER\_SEQUENCES table.



### Using a Sequence

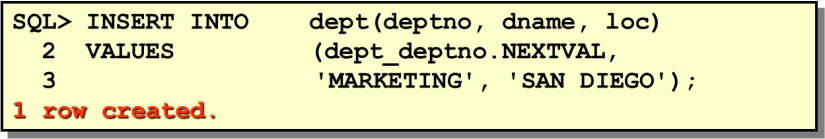
Once you create your sequence, you can use the sequence to generate sequential numbers for use in your tables. Reference the sequence values by using the NEXTVAL and CURRVAL pseudocolumns.

### NEXTVAL and CURRVAL Pseudocolumns

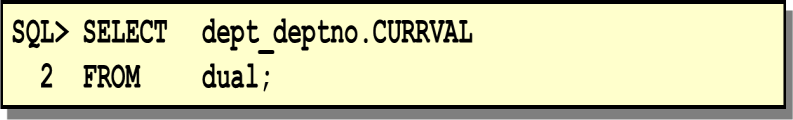
The NEXTVAL pseudocolumn is used to extract successive sequence numbers from a specified sequence. You must qualify NEXTVAL with the sequence name. When you reference *sequence*.NEXTVAL, a new sequence number is generated and the current sequence number is placed in CURRVAL. The CURRVAL pseudocolumn is used to refer to a sequence number that the current user has just generated.

### Using a Sequence

The example inserts a new department in the DEPT table. It uses the DEPT\_DEPTNO sequence for generating a new department number.



You can view the current value of the sequence:



### Viewing the Next Available Sequence Value without Incrementing It

If the sequence was created with NOCACHE, it is possible to view the next available sequence value without incrementing it by querying the USER\_SEQUENCES table.

### Altering a Sequence

### Syntax

ALTER SEQUENCE ***sequence***

[INCREMENT BY *n*]

[{MAXVALUE *n* | NOMAXVALUE}] [{MINVALUE *n* | NOMINVALUE}] [{CYCLE | NOCYCLE}]

[{CACHE *n* | NOCACHE}];

**Where: *sequence*** is the name of the sequence generator

### Removing a Sequence

To remove a sequence from the data dictionary, use the DROP SEQUENCE statement.

### Syntax

DROP SEQUENCE ***sequence***;

**Where: *sequence*** is the name of the sequence generator

***Exercise:***

1. Create a sequence to be used with the primary key column of the DEPARTMENT table. The sequence should start at 60 and have a maximum value of 200. Have your sequence increment by ten numbers. Name the sequence DEPT\_ID\_SEQ.
2. Write a script to display the following information about your sequences: sequence name, maximum value, increment size, and last number.
3. Write an interactive script to insert a row into the DEPARTMENT table. Be sure to use the sequence that you created for the ID column. Create a customized prompt to enter the department name. Execute your script. Add two departments named Education and Administration. Confirm your additions.

# Lab 11 (User Control Access)

#### Objective:

* + Create users
  + Create roles to ease setup and maintenance of the security model
  + Use the GRANT and REVOKE statements to grant and revoke object privileges
  + Create and access database links

#### User Privileges

* + Database security:
    - System security
    - Data security
  + System privileges: Gaining access to the database
  + Object privileges: Manipulating the content of the database objects
  + Schemas: Collections of objects, such as tables, views, and sequences

#### System Privileges

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

#### Creating Users

The DBA creates users by using the CREATE USER statement. General form: *CREATE USER userIDENTIFIED BY password;* Example:*CREATE USER scottIDENTIFIED BY tiger;*

#### User System Privileges

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

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 a user specific system privileges.

*GRANT create session, create table, create sequence, create view TO scott;*

.

#### Creating and granting privillages to ROLE

*CREATE ROLE manager;*

*GRANT create table, create view TO manager; GRANT manager TO DEHAAN, KOCHHAR;*

#### Changing Your Password

* The DBA creates your user account and initializes your password.
* You can change your password by using the ALTER USER statement.

*ALTER USER scott IDENTIFIED BY lion;*

**Object Privileges**

*GRANT object\_priv[(columns)]ONobjectTO {user|role|PUBLIC} [WITH GRANT OPTION];*

### Granting Object Privileges

* Grant query privileges on the EMPLOYEES table.
* Grant privileges to update specific columns to users and roles.

*GRANT select ON employees TO sue, rich;*

*GRANT update (department\_name, location\_id) ON departments TO scott, manager;*

### Using the WITH GRANT OPTION and PUBLIC Keywords

* Give a user authority to pass along privileges.
* Allow all users on the system to query data from Alice’s DEPARTMENTS table.

*GRANT select, insert ON departments TO scottWITH GRANT OPTION; GRANT select ON alice.departmentsTO PUBLIC;*

**How to Revoke Object Privileges**

* 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.

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

As user Alice, revoke the SELECT and INSERT privileges given to user Scott on the DEPARTMENTS table.

*REVOKE select, insert ON departments FROM scott;*

#### Exercise:

### Suppose you are the DBA for the following schemas. Complete the following task with appropriate sql command.

### Employee

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Eid | | EName | | Job | | Supervisor | Sal | Did |
| E001 | | Asif | | Manager | | E009 | 20000.00 | 10 |
| E002 | | Arif | | Manager | | E009 | 30000.00 | 10 |
| E004 | | Abul | | Salesman | | E001 | 15000.00 | 20 |
| E005 | | Kuddus | | Salesman | | E001 | 15000.00 | 20 |
| E006 | | Maruf | | Salesman | | E003 | 15000.00 | 20 |
| E009 | | Hasan | | President | |  | 40000.00 | 10 |
| Did | | Name | | Manager | | **Departments** | | | |
| 10 | | Admin | | E009 | |
| 20 | | Sales | | E002 | |

**Products OrderDetais**

|  |  |  |
| --- | --- | --- |
| ProductID | PName | Price |
| P001 | Machinery | 50000.00 |
| P002 | Hardware | 55000.00 |
| P003 | Software | 65000.00 |

|  |  |  |
| --- | --- | --- |
| OrderID | ProductID | Quantity |
| O001 | P001 | 10 |
| O002 | P001 | 10 |
| O002 | P003 | 10 |
| O003 | P002 | 10 |

1. Create a user **Rahul** with the password

### ret23erz.

1. Create a new role **Accounts**.
2. Grant system privileges create table, view and sequence to role Accounts.
3. Assign role Accounts to Rahul.
4. Change password of **Rahul** with the new password **rec34tg**
5. Grant query privilege to Asif and Arif on Departments table.
6. Grant privilege update to column Price on OrderDetails table to role Manager and user Hasan.
7. Give Asif the authority to pass along update and insert privilege on Departments table.
8. Revoke the update and delete privileges given to user kuddus on Product table.

# Lab 12 (Project Presentation and Discussion)