

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY



NAME	Divya S
ROLL NO	241001056
DEPT	B TECH INFORMATION TECHNOLOG
SEC	FA

# CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

## Definition of a Relational Database

A relational database is a collection of relations or two-dimensional tables.

## Terminologies Used in a Relational Database

1. A single **ROW** or table representing all data required for a particular employee. Each row should be identified by a primary key which allows no duplicate rows.
2. A **COLUMN** or attribute containing the employee number which identifies a unique employee. Here Employee number is designated as a primary key ,must contain a value and must be unique.
3. A column may contain foreign key. Here Dept\_ID is a foreign key in employee table and it is a primary key in Department table.
4. A Field can be found at the intersection of a row and column. There can be only one value in it. Also it may have no value. This is called a null value.

EMP ID	FIRST NAME	LAST NAME	EMAIL
100	King	Steven	Sking
101	John	Smith	Jsmith
102	Neena	Bai	Neenba
103	Eex	De Haan	Ldehaan

## **Relational Database Properties**

### A relational database :

- Can be accessed and modified by executing structured query language (SQL) statements.

# CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

- Contains a collection of tables with no physical pointers.
- Uses a set of operators

## Relational Database Management Systems

RDBMS refers to a relational database plus supporting software for managing users and processing SQL queries, performing backups/restores and associated tasks.

(Relational Database Management System) Software for storing data using SQL (structured query language). A relational database uses SQL to store data in a series of tables that not only record existing relationships between data items, but which also permit the data to be joined in new relationships. SQL (pronounced 'sequel') is based on a system of algebra developed by E F Codd, an IBM scientist who first defined the relational model in 1970. Relational databases are optimized for storing transactional data, and the majority of modern business software applications therefore use an RDBMS as their data store. The leading RDBMS vendors are Oracle, IBM and Microsoft.

The first commercial RDBMS was the Multics Relational Data Store, first sold in 1978.

INGRES, Oracle, Sybase, Inc., Microsoft Access, and Microsoft SQL

Server are well-known database products and companies. Others include PostgreSQL, SQL/DS, and RDB.

A relational database management system (RDBMS) is a program that lets you create, update, and administer a relational database. Most commercial RDBMS's use the Structured Query Language (SQL) to access the database, although SQL was invented after the development of the relational model and is not necessary for its use.

The leading RDBMS products are Oracle, IBM's DB2 and Microsoft's SQL Server. Despite repeated challenges by competing technologies, as well as the claim by some experts that no current RDBMS has fully implemented relational principles, the majority of new corporate databases are still being created and managed with an RDBMS.

## SQL Statements

1. Data Retrieval(DR)
2. Data Manipulation Language(DML)
3. Data Definition Language(DDL)
4. Data Control Language(DCL)
5. Transaction Control Language(TCL)

TYPE	STATEMENT	DESCRIPTION
DR	SELECT	Retrieves the data from the database

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

DML	1.INSERT 2.UPDATE 3.DELETE 4.MERGE	Enter new rows, changes existing rows, removes unwanted rows from tables in the database respectively.
DDL	1.CREATE 2.ALTER 3.DROP 4.RENAME 5.TRUNCATE	Sets up, changes and removes data structures from tables.
TCL	1.COMMIT 2.ROLLBACK 3.SAVEPOINT	Manages the changes made by DML statements. Changes to the data can be grouped together into logical transactions.
DCL	1.GRANT 2.REVOKE	Gives or removes access rights to both the oracle database and the structures within it.

### DATA TYPES

#### 1. Character Data types:

- Char – fixed length character string that can varies between 1-2000 bytes
- Varchar / Varchar2 – variable length character string, size ranges from 1-4000 bytes.it saves the disk space(only length of the entered value will be assigned as the size of column) ▪ Long - variable length character string, maximum size is 2 GB

#### 2. Number Data types : Can store +ve,-ve,zero,fixed point, floating point with 38 precission.

- Number – {p=38,s=0}
- Number(p) - fixed point
- Number(p,s) –floating point (p=1 to 38,s= -84 to 127)

#### 3. Date Time Data type: used to store date and time in the table.

- DB uses its own format of storing in fixed length of 7 bytes for century, date, month, year, hour, minutes, and seconds.

- Default data type is “dd-mon-yy”

- New Date time data types have been introduced. They are

TIMESTAMP-Date with fractional seconds

INTERVAL YEAR TO MONTH-stored as an interval of years and months

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

INTERVAL DAY TO SECOND-stored as o interval of days to hour's minutes and seconds

4. **Raw Data type:** used to store byte oriented data like binary data and byte string.

5. **Other :**

- CLOB – stores character object with single byte character.
- BLOB – stores large binary objects such as graphics, video, sounds.
- BFILE – stores file pointers to the LOB's.

### EXERCISE-1

#### Creating and Managing Tables

#### OBJECTIVE

After the completion of this exercise, students should be able to do the following:

Create tables

Describing the data types that can be used when specifying column definition

Alter table definitions

Drop, rename, and truncate tables

#### NAMING RULES

Table names and column names:

- Must begin with a letter
- Must be 1-30 characters long

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

- Must contain only A-Z, a-z, 0-9, \_, \$, and #
- Must not duplicate the name of another object owned by the same user
- Must not be an oracle server reserve words
- 2 different tables should not have same name.
- Should specify a unique column name.
- Should specify proper data type along with width
- Can include “not null” condition when needed. By default it is ‘null’.

### The CREATE TABLE Statement

**Table:** Basic unit of storage; composed of rows and columns

**Syntax: 1** Create table table\_name (column\_name1 data\_type (size) column\_name2 data\_type (size)...);

**Syntax: 2** Create table table\_name (column\_name1 data\_type (size) constraints, column\_name2 data\_type constraints ...); **Example:**

Create table employees ( employee\_id number(6), first\_name varchar2(20), ..job\_id varchar2(10), CONSTRAINT emp\_emp\_id\_pk PRIMARY KEY (employee\_id));

### Tables Used in this course

### Creating a table by using a Sub query

#### SYNTAX

```
// CREATE TABLE table_name(column_name type(size)...);
```

```
Create table table_name as select column_name1,column_name2,.....colmn_namen  
from table_name where predicate;
```

#### AS Subquery

Subquery is the select statement that defines the set of rows to be inserted into the new table.

### Example

Create table dept80 as select employee\_id, last\_name, salary\*12 Annsal, hire\_date  
from employees where dept\_id=80;

# CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

## The ALTER TABLE Statement

The ALTER statement is used to

- Add a new column
- Modify an existing column
- Define a default value to the new column
- Drop a column
- To include or drop integrity constraint.

## SYNTAX

ALTER TABLE table\_name ADD /MODIFY(Column\_name type(size));

ALTER TABLE table\_name DROP COLUMN (Column\_name);

*ALTER TABLE ADD CONSTRAINT Constraint\_name PRIMARY KEY (Column\_Name);*

## Example:

Alter table dept80 add (job\_id varchar2(9));

Alter table dept80 modify (last\_name varchar2(30));

Alter table dept80 drop column job\_id;

**NOTE:** Once the column is dropped it cannot be recovered.

## DROPPING A TABLE

- All data and structure in the table is deleted.
- Any pending transactions are committed.
- All indexes are dropped.
- Cannot roll back the drop table statement.

## Syntax:

**Drop table *tablename*;**

## Example:

Drop table dept80;

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

### RENAMING A TABLE

To rename a table or view.

#### Syntax

RENAME old\_name to new\_name

#### Example:

Rename dept to detail\_dept;

### TRUNCATING A TABLE

Removes all rows from the table.

Releases the storage space used by that table.

#### Syntax

TRUNCATE TABLE *table\_name*;

#### Example:

TRUNCATE TABLE copy\_emp;

### Find the Solution for the following:

Create the following tables with the given structure.

#### EMPLOYEES TABLE

NAME	NULL?	TYPE
Employee_id	Not null	Number(6)
First_Name		Varchar(20)
Last_Name	Not null	Varchar(25)

CREATE TABLE EMPLOYEE (



## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

Email	Not null	Varchar(25)
Phone_Number		Varchar(20)
Hire_date	Not null	Date
Job_id	Not null	Varchar(10)
Salary		Number(8,2)
Commission_pct		Number(2,2)
Manager_id		Number(6)
Department_id		Number(4)

```
Employee_id  NUMBER(6)  NOT NULL,  
First_Name   VARCHAR2(20),  
Last_Name    VARCHAR2(25) NOT NULL,  
Email        VARCHAR2(25) NOT NULL,  
Phone_Number VARCHAR2(20),  
Hire_date    DATE      NOT NULL,  
Job_id       VARCHAR2(10) NOT NULL,  
Salary       NUMBER(8,2),  
Commission_pct NUMBER(2,2),  
Manager_id   NUMBER(6),  
Department_id NUMBER(4)  
);
```

### DEPARTMENT TABLE

NAME	NULL?	TYPE
Dept_id	Not null	Number(6)
Dept_name	Not null	Varchar(20)
Manager_id		Number(6)
Location_id		Number(4)

```
CREATE TABLE DEPARTMENT (  
  Dept_id  NUMBER(6) NOT NULL,  
  Dept_name VARCHAR2(20) NOT NULL,  
  Manager_id NUMBER(6),  
  Location_id NUMBER(4)  
);
```

### JOB\_GRADE TABLE

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

NAME	NULL?	TYPE
Grade_level		Varchar(2)
Lowest_sal		Number
Highest_sal		Number

```
CREATE TABLE SAL_GRADE (
    Grade_level VARCHAR2(2),
    Lowest_sal NUMBER,
    Highest_sal NUMBER
);
```

### LOCATION TABLE

NAME	NULL?	TYPE
Location_id	Not null	Number(4)
St_addr		Varchar(40)
Postal_code		Varchar(12)
City	Not null	Varchar(30)
State_province		Varchar(25)
Country_id		Char(2)

```
CREATE TABLE LOCATION (
    Location_id NUMBER(4) NOT NULL,
    St_addr VARCHAR2(40),
    Postal_code VARCHAR2(12),
    City VARCHAR2(30) NOT NULL,
    State_province VARCHAR2(25),
    Country_id CHAR(2)
);
```

below. Confirm that the table is created.

Column name	ID	NAME
Key Type		
Nulls/Unique		
EK table		
EK column		
Data Type	Number	Varchar2
Length	7	25

- Create the DEPT table based on the DEPARTMENT following the table instance chart  

```
CREATE TABLE DEPT (ID NUMBER(7), NAME VARCHAR2(25) );
```

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

2. Create the EMP table based on the following instance chart. Confirm that the table is created.

Column name	ID	LAST_NAME	FIRST_NAME	DEPT_ID
Key Type				
Nulls/Unique				
FK table				
FK column				
Data Type	Number	Varchar2	Varchar2	Number
Length	7	25	25	7

```
CREATE TABLE EMPLOYEE_SIMPLE (  
    ID      NUMBER(7),  
    LAST_NAME VARCHAR2(25),  
    FIRST_NAME VARCHAR2(25),  
    DEPT_ID  NUMBER(7)  
);
```

With enhancements

```
CREATE TABLE EMPLOYEE_SIMPLE (  
    ID      NUMBER(7) PRIMARY KEY,  
    LAST_NAME VARCHAR2(25) NOT NULL,  
    FIRST_NAME VARCHAR2(25),  
    DEPT_ID  NUMBER(7),  
    CONSTRAINT fk_dept  
        FOREIGN KEY (DEPT_ID) REFERENCES DEPT(ID)  
);
```

3. Modify the EMP table to allow for longer employee last names. Confirm the modification.(Hint: Increase the size to 50)

```
ALTER TABLE EMP MODIFY LAST_NAME VARCHAR2(50);
```

4. Create the EMPLOYEES2 table based on the structure of EMPLOYEES table. Include Only the Employee\_id, First\_name, Last\_name, Salary and Dept\_id coloumns. Name the columns Id, First\_name, Last\_name, salary and Dept\_id respectively.

```
CREATE TABLE EMPLOYEES2 AS  
SELECT  
    Employee_id AS Id,  
    First_name,
```

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

```
Last_name,  
Salary,  
Dept_id  
FROM EMPLOYEES  
WHERE 1=0;
```

WHERE 1=0 ensures **no data is copied**, only the structure.

5. Drop the EMP table.

```
DROP TABLE EMP;
```

6. Rename the EMPLOYEES2 table as EMP.

```
RENAME EMPLOYEES2 TO EMP;
```

7. Add a comment on DEPT and EMP tables. Confirm the modification by describing the table.

```
COMMENT ON TABLE DEPT IS 'This table stores department information';
```

```
COMMENT ON TABLE EMP IS 'This table stores employee details';
```

8. Drop the First\_name column from the EMP table and confirm it.

```
ALTER TABLE EMP DROP COLUMN First_name;
```

```
DESC EMP;
```

# **CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY**

## **EXERCISE-2**

### **MANIPULATING DATA**

#### **OBJECTIVE**

After, the completion of this exercise the students will be able to do the following

- Describe each DML statement
- Insert rows into tables
- Update rows into table
- Delete rows from table
- Control Transactions

A DML statement is executed when you:

- Add new rows to a table
  - Modify existing rows
- Removing existing rows

A transaction consists of a collection of DML statements that form a logical unit of work.

#### **To Add a New Row**

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

INSERT Statement

### **Syntax**

INSERT INTO table\_name VALUES (column1 values, column2 values, ..., columnn values);

### **Example:**

INSERT INTO department (70, 'Public relations', 100,1700);

### **Inserting rows with null values**

**Implicit Method:** (Omit the column)

INSERT INTO department VALUES (30,'purchasing');

**Explicit Method:** (Specify NULL keyword)

INSERT INTO department VALUES (100,'finance', NULL, NULL);

### **Inserting Special Values**

### **Example:**

Using SYSDATE

INSERT INTO employees VALUES (113,'louis', 'popp', 'lpopp','5151244567',**SYSDATE**, 'ac\_account', 6900, NULL, 205, 100);

### **Inserting Specific Date Values**

### **Example:**

INSERT INTO employees VALUES ( 114,'den', 'raphealy', 'drapheal', '5151274561', **TO\_DATE('feb 3,1999','mon, dd ,yyyy')**, 'ac\_account', 11000,100,30);

### **To Insert Multiple Rows**

& is the placeholder for the variable value **Example:**

INSERT INTO department VALUES (&dept\_id, &dept\_name, &location);

# CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

## Copying Rows from another table

Using Subquery **Example:**

```
INSERT INTO sales_reps(id, name, salary, commission_pct)
      SELECT employee_id, Last_name, salary, commission_pct
FROM employees
WHERE job_id LIKE '%REP');
```

## CHANGING DATA IN A TABLE

UPDATE Statement

**Syntax1:** ( to update specific rows)

```
UPDATE table_name SET column=value WHERE condition;
```

**Syntax 2:** (To update all rows)

```
UPDATE table_name SET column=value;
```

## Updating columns with a subquery

```
UPDATE employees
SET job_id= (SELECT job_id
FROM employees
WHERE employee_id=205)
WHERE employee_id=114;
```

## REMOVING A ROW FROM A TABLE

### DELETE STATEMENT

#### Syntax

```
DELETE FROM table_name WHERE conditions;
```

#### **Example:**

```
DELETE FROM department WHERE dept_name='finance';
```

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

### Find the Solution for the following:

1. Create MY\_EMPLOYEE table with the following structure

NAME	NULL?	TYPE
ID	Not null	Number(4)
Last_name		Varchar(25)
First_name		Varchar(25)
Userid		Varchar(25)
Salary		Number(9,2)

```
CREATE TABLE STAFF ( ID      NUMBER(4)   NOT NULL,
  Last_name VARCHAR2(25),
  First_name VARCHAR2(25),
  Userid   VARCHAR2(25),
  Salary   NUMBER(9,2)
);
```

2. Add the first and second rows data to MY\_EMPLOYEE table from the following sample data.

ID	Last_name	First_name	Userid	salary
1	Patel	Ralph	rpatel	895
2	Dancs	Betty	bdancs	860
3	Biri	Ben	bbiri	1100
4	Newman	Chad	Cnewman	750
5	Ropebur	Audrey	aropebur	1550

```
INSERT INTO MY_EMPLOYEE (ID, Last_name, First_name, Userid, Salary)
VALUES (1, 'Patel', 'Ralph', 'rpatel', 895);
```

```
INSERT INTO MY_EMPLOYEE (ID, Last_name, First_name, Userid, Salary)
VALUES (2, 'Dancs', 'Betty', 'bdancs', 860);
```

3. Display the table with values.

```
SELECT * FROM MY_EMPLOYEE;
```

4. Populate the next two rows of data from the sample data. Concatenate the first letter of the first\_name with the first seven characters of the last\_name to produce Userid.

```
INSERT INTO MY_EMPLOYEE (ID, Last_name, First_name, Userid, Salary)
```



## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

```
VALUES (  
    3,  
    'Biri',  
    'Ben',  
    SUBSTR('Ben', 1, 1) || SUBSTR('Biri', 1, 7),  
    1100  
);
```

```
INSERT INTO MY_EMPLOYEE (ID, Last_name, First_name, Userid, Salary)  
VALUES (  
    4,  
    'Newman',  
    'Chad',  
    SUBSTR('Chad', 1, 1) || SUBSTR('Newman', 1, 7),  
    750  
);
```

5. Make the data additions permanent.

```
COMMIT;
```

6. Change the last name of employee 3 to Drexler.

```
UPDATE MY_EMPLOYEE  
SET Last_name = 'Drexler'  
WHERE ID = 3;
```

7. Change the salary to 1000 for all the employees with a salary less than 900.

```
UPDATE MY_EMPLOYEE  
SET Salary = 1000  
WHERE Salary < 900;
```

8. Delete Betty dances from MY \_EMPLOYEE table.

```
DELETE FROM MY_EMPLOYEE  
WHERE First_name = 'Betty' AND Last_name = 'Dances';
```

9. Empty the fourth row of the emp table.

# CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

```
UPDATE MY_EMPLOYEE SET    Last_name = NULL,    First_name = NULL,  
Userid = NULL,    Salary = NULL WHERE ID = 4;
```

## EXERCISE-3

### INCLUDING CONSTRAINTS

#### OBJECTIVE

After the completion of this exercise the students should be able to do the following

- Describe the constraints
- Create and maintain the constraints

#### **What are Integrity constraints?**

- Constraints enforce rules at the table level.
- Constraints prevent the deletion of a table if there are dependencies

**The following types of integrity constraints are valid**

a) **Domain Integrity**

- ✓ NOT NULL
- ✓ CHECK

b) **Entity Integrity**

- ✓ UNIQUE
- ✓ PRIMARY KEY

c) **Referential Integrity**

- ✓ FOREIGN KEY

**Constraints can be created in either of two ways**

1. At the same time as the table is created
2. After the table has been created.

#### Defining Constraints

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

Create table tablename (column\_name1 data\_type constraints, column\_name2 data\_type constraints ...); **Example:**

Create table employees ( employee\_id number(6), first\_name varchar2(20), ..job\_id varchar2 (10), CONSTRAINT emp\_emp\_id\_pk PRIMARY KEY (employee\_id));

### **Domain Integrity**

This constraint sets a range and any violations that takes place will prevent the user from performing the manipulation that caused the breach.It includes:

### **NOT NULL Constraint**

While creating tables, by default the rows can have null value.the enforcement of not null constraint in a table ensure that the table contains values.

### **Principle of null values:**

- o Setting null value is appropriate when the actual value is unknown, or when a value would not be meaningful. o A null value is not equivalent to a value of zero. o A null value will always evaluate to null in any expression.
- o When a column name is defined as not null, that column becomes a mandatory i.e., the user has to enter data into it.
- o Not null Integrity constraint cannot be defined using the alter table command when the table contain rows.

### **Example**

CREATE TABLE employees (employee\_id number (6), last\_name varchar2(25) NOT NULL, salary number(8,2), commission\_pct number(2,2), hire\_date date constraint emp\_hire\_date\_nn NOT NULL'....);

### **CHECK**

Check constraint can be defined to allow only a particular range of values.when the manipulation violates this constraint,the record will be rejected.Check condition cannot contain sub queries.

CREATE TABLE employees (employee\_id number (6), last\_name varchar2 (25) NOT NULL, salary number(8,2), commission\_pct number(2,2), hire\_date date constraint emp\_hire\_date\_nn NOT NULL'...,CONSTRAINT emp\_salary\_mi CHECK(salary > 0));

### **Entity Integrity**

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

Maintains uniqueness in a record. An entity represents a table and each row of a table represents an instance of that entity. To identify each row in a table uniquely we need to use this constraint.

There are 2 entity constraints:

### **a) Unique key constraint**

It is used to ensure that information in the column for each record is unique, as with telephone or driver's license numbers. It prevents the duplication of value with rows of a specified column in a set of column. A column defined with the constraint can allow null value.

If unique key constraint is defined in more than one column i.e., combination of column cannot be specified. Maximum combination of columns that a composite unique key can contain is 16.

### **Example:**

```
CREATE TABLE employees (employee_id number(6), last_name varchar2(25) NOT  
NULL, email varchar2(25), salary number(8,2), commission_pct number(2,2), hire_date date constraint  
emp_hire_date_nn NOT NULL' CONSTRAINT emp_email_uk UNIQUE(email));
```

### **PRIMARY KEY CONSTRAINT**

A primary key avoids duplication of rows and does not allow null values. Can be defined on one or more columns in a table and is used to uniquely identify each row in a table. These values should never be changed and should never be null.

A table should have only one primary key. If a primary key constraint is assigned to more than one column or combination of column is said to be composite primary key, which can contain 16 columns

### **Example:**

```
CREATE TABLE employees (employee_id number(6) , last_name varchar2(25) NOT  
NULL, email varchar2(25), salary number(8,2), commission_pct number(2,2), hire_date date  
constraint emp_hire_date_nn NOT NULL, Constraint emp_id pk PRIMARY KEY  
(employee_id),CONSTRAINT emp_email_uk UNIQUE(email));
```

### **c) Referential Integrity**

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

It enforces relationship between tables. To establish parent-child relationship between 2 tables having a common column definition, we make use of this constraint. To implement this, we should define the column in the parent table as primary key and same column in the child table as foreign key referring to the corresponding parent entry.

### Foreign key

A column or combination of column included in the definition of referential integrity, which would refer to a referenced key.

### Referenced key

It is a unique or primary key upon which is defined on a column belonging to the parent table. Keywords:

**FOREIGN KEY:** Defines the column in the child table at the table level constraint.

**REFERENCES:** Identifies the table and column in the parent table.

**ON DELETE CASCADE:** Deletes the dependent rows in the child table when a row in the parent table is deleted.

**ON DELETE SET NULL:** converts dependent foreign key values to null when the parent value is removed.

```
CREATE TABLE employees (employee_id number(6) , last_name varchar2(25) NOT
NULL,email varchar2(25), salary number(8,2), commission_pct number(2,2), hire_date date
constraint emp_hire_date_nn NOT NULL, Constraint emp_id pk PRIMARY KEY
(employee_id),CONSTRAINT emp_email_uk UNIQUE(email),CONSTRAINT emp_dept_fk
FOREIGN KEY (department_id) references departments(dept_id));
```

### ADDING A CONSTRAINT

Use the ALTER to

- Add or Drop a constraint, but not modify the structure
- Enable or Disable the constraints
- Add a not null constraint by using the Modify clause

### Syntax

```
ALTER TABLE table name ADD CONSTRAINT Cons_name type(column name);
```

# CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

## Example:

```
ALTER TABLE employees ADD CONSTRAINT emp_manager_fk FOREIGN KEY  
(manager_id) REFERENCES employees (employee_id);
```

## DROPPING A CONSTRAINT

### Example:

```
ALTER TABLE employees DROP CONSTRAINT emp_manager_fk;
```

## CASCADE IN DROP

- The CASCADE option of the DROP clause causes any dependent constraints also to be dropped.

## Syntax

```
ALTER TABLE departments DROP PRIMARY KEY|UNIQUE (column)| CONSTRAINT  
constraint _name CASCADE;
```

## DISABLING CONSTRAINTS

- Execute the DISABLE clause of the ALTER TABLE statement to deactivate an integrity constraint
- Apply the CASCADE option to disable dependent integrity constraints.

### Example

```
ALTER TABLE employees DISABLE CONSTRAINT emp_emp_id_pk CASCADE;
```

## ENABLING CONSTRAINTS

- Activate an integrity constraint currently disabled in the table definition by using the ENABLE clause.

### Example

```
ALTER TABLE employees ENABLE CONSTRAINT emp_emp_id_pk CASCADE;
```

## CASCADING CONSTRAINTS

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

The CASCADE CONSTRAINTS clause is used along with the DROP column clause. It drops all referential integrity constraints that refer to the primary and unique keys defined on the dropped Columns.

This clause also drops all multicolumn constraints defined on the dropped column.

### **Example:**

**Assume table TEST1 with the following structure**

```
CREATE TABLE test1 ( pk number PRIMARY KEY, fk number, col1 number,col2 number,  
CONSTRAINT fk_constraint FOREIGN KEY(fk) references test1, CONSTRAINT ck1 CHECK (pk>0  
and col1>0), CONSTRAINT ck2 CHECK (col2>0));
```

**An error is returned for the following statements**

```
ALTER TABLE test1 DROP (pk);
```

```
ALTER TABLE test1 DROP (col1);
```

**The above statement can be written with CASCADE CONSTRAINT**

```
ALTER TABLE test 1 DROP(pk) CASCADE CONSTRAINTS;  
(OR)
```

```
ALTER TABLE test 1 DROP(pk, fk, col1) CASCADE CONSTRAINTS;
```

### **VIEWING CONSTRAINTS**

Query the USER\_CONSTRAINTS table to view all the constraints definition and names.

### **Example:**

```
SELECT constraint_name, constraint_type, search_condition FROM user_constraints  
WHERE table_name='employees';
```

### **Viewing the columns associated with constraints**

```
SELECT constraint_name, constraint_type, FROM user_cons_columns  
WHERE table_name='employees';
```

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

### Find the Solution for the following:

1. Add a table-level PRIMARY KEY constraint to the EMP table on the ID column. The constraint should be named at creation. Name the constraint my\_emp\_id\_pk.

**ALTER TABLE EMP ADD CONSTRAINT my\_emp\_id\_pk PRIMARY KEY (ID);**

To check null and duplicates

**SELECT \* FROM EMP WHERE ID IS NULL;**

**SELECT ID, COUNT(\*) AS count  
FROM EMP  
GROUP BY ID  
HAVING COUNT(\*) > 1;**

2. Create a PRIMARY KEY constraint to the DEPT table using the ID column. The constraint should be named at creation. Name the constraint my\_dept\_id\_pk.

**ALTER TABLE DEPT ADD CONSTRAINT my\_dept\_id\_pk PRIMARY KEY (ID);**

3. Add a column DEPT\_ID to the EMP table. Add a foreign key reference on the EMP table that ensures that the employee is not assigned to nonexistent department. Name the constraint my\_emp\_dept\_id\_fk.

**ALTER TABLE EMP ADD DEPT\_ID NUMBER;**

**ALTER TABLE EMP  
ADD CONSTRAINT my\_emp\_dept\_id\_fk  
FOREIGN KEY (DEPT\_ID)  
REFERENCES DEPT(ID);**

4. Modify the EMP table. Add a COMMISSION column of NUMBER data type, precision 2, scale 2. Add a constraint to the commission column that ensures that a commission value is greater than zero.



## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

```
ALTER TABLE EMP ADD COMMISSION NUMBER(2, 2);
```

```
ALTER TABLE EMP  
ADD CONSTRAINT emp_commission_check  
CHECK (COMMISSION > 0);
```

### EXERCISE-4

#### Writing Basic SQL SELECT Statements

##### OBJECTIVES

After the completion of this exercise, the students will be able to do the following:

- List the capabilities of SQL SELECT Statement
- Execute a basic SELECT statement

##### Capabilities of SQL SELECT statement

A SELECT statement retrieves information from the database. Using a select statement, we can perform

- ✓ Projection: To choose the columns in a table
- ✓ Selection: To choose the rows in a table
- ✓ Joining: To bring together the data that is stored in different tables

# CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

## Basic SELECT Statement

### Syntax

```
SELECT *|DISTINCT Column_name| alias  
FROM table_name;
```

### NOTE:

DISTINCT—Suppress the duplicates.

Alias—gives selected columns different headings.

### Example: 1

```
SELECT * FROM departments;
```

### Example: 2

```
SELECT location_id, department_id FROM departments;
```

## Writing SQL Statements

- SQL statements are not case sensitive
- SQL statements can be on one or more lines.
- Keywords cannot be abbreviated or split across lines
- Clauses are usually placed on separate lines
- Indents are used to enhance readability

## Using Arithmetic Expressions

Basic Arithmetic operators like \*, /, +, - can be used

### Example:1

```
SELECT last_name, salary, salary+300 FROM employees;
```

### Example:2

```
SELECT last_name, salary, 12*salary+100 FROM employees;
```

The statement is not same as

```
SELECT last_name, salary, 12*(salary+100) FROM employees;
```

### Example:3

```
SELECT last_name, job_id, salary, commission_pct FROM employees;
```

### Example:4

```
SELECT last_name, job_id, salary, 12*salary*commission_pct FROM employees;
```

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

### Using Column Alias

- To rename a column heading with or without AS keyword.

#### **Example:1**

```
SELECT last_name AS Name  
FROM employees;
```

#### **Example: 2**

```
SELECT last_name "Name" salary*12 "Annual Salary "  
FROM employees;
```

### Concatenation Operator

- Concatenates columns or character strings to other columns
- Represented by two vertical bars (||)
- Creates a resultant column that is a character expression **Example:**

```
SELECT last_name||job_id AS "EMPLOYEES JOB" FROM employees;
```

### Using Literal Character String

- A literal is a character, a number, or a date included in the SELECT list.
- Date and character literal values must be enclosed within single quotation marks.

#### **Example:**

```
SELECT last_name||'is a'||job_id AS "EMPLOYEES JOB" FROM employees;
```

### Eliminating Duplicate Rows

- Using DISTINCT keyword.

#### **Example:**

```
SELECT DISTINCT department_id FROM employees;
```

### Displaying Table Structure

- Using DESC keyword.

#### **Syntax**

```
DESC table_name; Example:
```

```
DESC employees;
```

### **Find the Solution for the following:**

#### **True OR False**

1. The following statement executes successfully.

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

### Identify the Errors

```
SELECT employee_id, last_name sal*12
ANNUAL SALARY
FROM employees;
```

### Queries

```
SELECT employee_id, last_name,
       salary * 12 AS annual_salary
FROM employees;
```

2. Show the structure of departments the table. Select all the data from it.

```
DESC departments;
SELECT * FROM departments;
```

3. Create a query to display the last name, job code, hire date, and employee number for each employee, with employee number appearing first.

```
SELECT employee_id, last_name, job_id, hire_date FROM employees;
```

4. Provide an alias STARTDATE for the hire date.

```
SELECT employee_id, last_name, job_id, hire_date AS STARTDATE
FROM employees;
```

5. Create a query to display unique job codes from the employee table.

```
SELECT DISTINCT job_id FROM employees;
```

6. Display the last name concatenated with the job ID , separated by a comma and space, and name the column EMPLOYEE and TITLE.

```
SELECT last_name || ', ' || job_id AS "EMPLOYEE AND TITLE"
FROM employees;
```

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

7. Create a query to display all the data from the employees table. Separate each column by a comma. Name the column THE\_OUTPUT.

**SELECT**

```
    employee_id || ', ' ||  
first_name || ', ' || last_name  
|| ', ' ||  
    email || ', ' ||  
phone_number || ', ' ||  
hire_date || ', ' || job_id  
|| ', ' ||  
    salary || ', ' ||  
commission_pct || ', ' ||  
    manager_id || ', ' ||  
    department_id AS "THE_OUTPUT"  
FROM employees;
```

# CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

## EXERCISE-5

### Restricting and Sorting data

After the completion of this exercise, the students will be able to do the following:

- Limit the rows retrieved by the queries
- Sort the rows retrieved by the queries
- 

#### Limiting the Rows selected

- Using WHERE clause
- Alias cannot be used in WHERE clause

#### Syntax

SELECT-----

FROM----- WHERE

condition; **Example:**

SELECT employee\_id,last\_name, job\_id, department\_id FROM employees WHERE department\_id=90;

#### Character strings and Dates

Character strings and date values are enclosed in single quotation marks.

Character values are case sensitive and date values are format sensitive.

#### **Example:**

SELECT employee\_id,last\_name, job\_id, department\_id FROM employees  
WHERE last\_name='WHALEN';

#### **Comparison Conditions**

All relational operators can be used. (=, >, >=, <, <=, <>, !=)

#### **Example:**

SELECT last\_name, salary  
FROM employees  
WHERE salary<=3000;

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

### Other comparison conditions

Operator	Meaning
BETWEEN ...AND...	Between two values
IN	Match any of a list of values
LIKE	Match a character pattern
IS NULL	Is a null values

#### Example:1

```
SELECT last_name, salary
FROM employees
WHERE salary BETWEEN 2500 AND 3500;
```

#### Example:2

```
SELECT employee_id, last_name, salary , manager_id
FROM employees
WHERE manager_id IN (101, 100,201);
```

#### Example:3

- Use the LIKE condition to perform wildcard searches of valid string values.
- Two symbols can be used to construct the search string
  - % denotes zero or more characters
  - \_ denotes one character

```
SELECT first_name, salary
FROM employees
WHERE first_name LIKE '%s';
```

#### Example:4

```
SELECT last_name, salary
FROM employees
WHERE last_name LIKE '_o%';
```

#### Example:5

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

**ESCAPE option**-To have an exact match for the actual % and\_ characters  
To search for the string that contain 'SA\_'

```
SELECT employee_id, first_name, salary, job_id
FROM employees
WHERE job_id LIKE '%sa\_%' ESCAPE '\';
```

### Test for NULL

- Using IS NULL operator **Example:**

```
SELECT employee_id, last_name, salary, manager_id
FROM employees
WHERE manager_id IS NULL;
```

### Logical Conditions

All logical operators can be used.( AND,OR,NOT)

### Example:1

```
SELECT employee_id, last_name, salary, job_id
FROM employees
WHERE salary >= 10000
AND job_id LIKE '%MAN%';
```

### Example:2

```
SELECT employee_id, last_name, salary, job_id
FROM employees
WHERE salary >= 10000
OR job_id LIKE '%MAN%';
```

### Example:3

```
SELECT employee_id, last_name, salary, job_id
FROM employees
WHERE job_id NOT IN ('it_prog', 'st_clerk', 'sa_rep');
```

### Rules of Precedence



## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

Order Evaluated	Operator
1	Arithmetic
2	Concatenation
3	Comparison
4	IS [NOT] NULL, LIKE, [NOT] IN
5	[NOT] BETWEEN
6	Logical NOT
7	Logical AND
8	Logical OR

### **Example:1**

```
SELECT employee_id, last_name, salary , job_id
FROM employees
WHERE job_id='sa_rep'
OR job_id='ad_pres'
AND salary>15000;
```

### **Example:2**

```
SELECT employee_id, last_name, salary , job_id
FROM employees
WHERE (job_id='sa_rep'
OR job_id='ad_pres')
AND salary>15000;
```

### **Sorting the rows**

Using ORDER BY Clause

**ASC**-Ascending Order,Default

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

**DESC**-Descending order

### **Example:1**

```
SELECT last_name, salary , job_id,department_id,hire_date
FROM employees
ORDER BY hire_date;
```

### **Example:2**

```
SELECT last_name, salary , job_id,department_id,hire_date
FROM employees
ORDER BY hire_date DESC;
```

### **Example:3**

#### **Sorting by column alias**

```
SELECT last_name, salary*12 annsal , job_id,department_id,hire_date
FROM employees
ORDER BY annsal;
```

### **Example:4**

#### **Sorting by Multiple columns**

```
SELECT last_name, salary , job_id,department_id,hire_date
FROM employees ORDER BY department_id, salary DESC;
```

#### **Find the Solution for the following:**

1. Create a query to display the last name and salary of employees earning more than 12000.

```
SELECT last_name, salary
FROM employees
WHERE salary > 12000;
```

2. Create a query to display the employee last name and department number for employee number 176.

```
SELECT last_name, department_id
FROM employees
WHERE employee_id = 176;
```

3. Create a query to display the last name and salary of employees whose salary is not in the range of 5000 and 12000. (hints: not between )

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

```
SELECT last_name, salary
FROM employees
WHERE salary NOT BETWEEN 5000 AND 12000;
```

4. Display the employee last name, job ID, and start date of employees hired between February 20,1998 and May 1,1998.order the query in ascending order by start date.(hints: between)

```
SELECT last_name, job_id, hire_date
FROM employees
WHERE hire_date BETWEEN TO_DATE('20-FEB-1998', 'DD-MON-YYYY')
AND TO_DATE('01-MAY-1998', 'DD-MON-YYYY')
ORDER BY hire_date ASC;
```

5. Display the last name and department number of all employees in departments 20 and 50 in alphabetical order by name.(hints: in, orderby)

```
SELECT lastname, department_id
FROM employees
WHERE department_id IN (20, 50)
ORDER BY lastname asc;
```

6. Display the last name and salary of all employees who earn between 5000 and 12000 and are in departments 20 and 50 in alphabetical order by name. Label the columns EMPLOYEE, MONTHLY SALARY respectively.(hints: between, in)

```
SELECT last_name AS "EMPLOYEE", salary AS "MONTHLY SALARY"
FROM employees
WHERE salary BETWEEN 5000 AND 12000
AND department_id IN (20, 50) ORDER BY last_name ASC;
```

7. Display the last name and hire date of every employee who was hired in 1994.(hints: like)

```
SELECT last_name, hire_date
FROM employees
WHERE hire_date LIKE '1994%'; or
```

```
SELECT last_name, hire_date
FROM employees
WHERE TO_CHAR(hire_date, 'YYYY') LIKE '1994';
```

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

8. Display the last name and job title of all employees who do not have a manager.(hints: is null)

```
SELECT last_name, job_title  
FROM employees  
WHERE manager_id IS NULL;
```

9. Display the last name, salary, and commission for all employees who earn commissions. Sort data in descending order of salary and commissions.(hints: is not null, orderby)

```
SELECT last_name, salary, commission_pct  
FROM employees  
WHERE commission_pct IS NOT NULL  
ORDER BY salary DESC, commission_pct DESC;
```

10. Display the last name of all employees where the third letter of the name is *a*.(hints:like)

```
SELECT last_name  
FROM employees  
WHERE last_name LIKE '__a%';
```

11. Display the last name of all employees who have an *a* and an *e* in their last name.(hints: like) **SELECT last\_name**

```
FROM employees  
WHERE last_name LIKE '%a%'  
AND last_name LIKE '%e%';
```

12. Display the last name and job and salary for all employees whose job is sales representative or stock clerk and whose salary is not equal to 2500, 3500 or 7000.(hints:in,not in)

```
SELECT last_name, job_id, salary  
FROM employees  
WHERE job_id IN ('SA_REP', 'ST_CLERK')  
AND salary NOT IN (2500, 3500, 7000);
```

13. Display the last name, salary, and commission for all employees whose commission amount is 20%.(hints:use predicate logic)

```
SELECT last_name, salary, commission_pct  
FROM employees  
WHERE commission_pct = 0.20;
```

# CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

## Single Row Functions

### Objective

After the completion of this exercise, the students will be able to do the following:

- Describe various types of functions available in SQL.
- Use character, number and date functions in SELECT statement.
- Describe the use of conversion functions.

### Single row functions:

Manipulate data items.

Accept arguments and return one value.

Act on each row returned. Return one result per row.

May modify the data type.

Can be nested.

Accept arguments which can be a column or an expression

### Syntax

Function\_name(arg1,...argn)

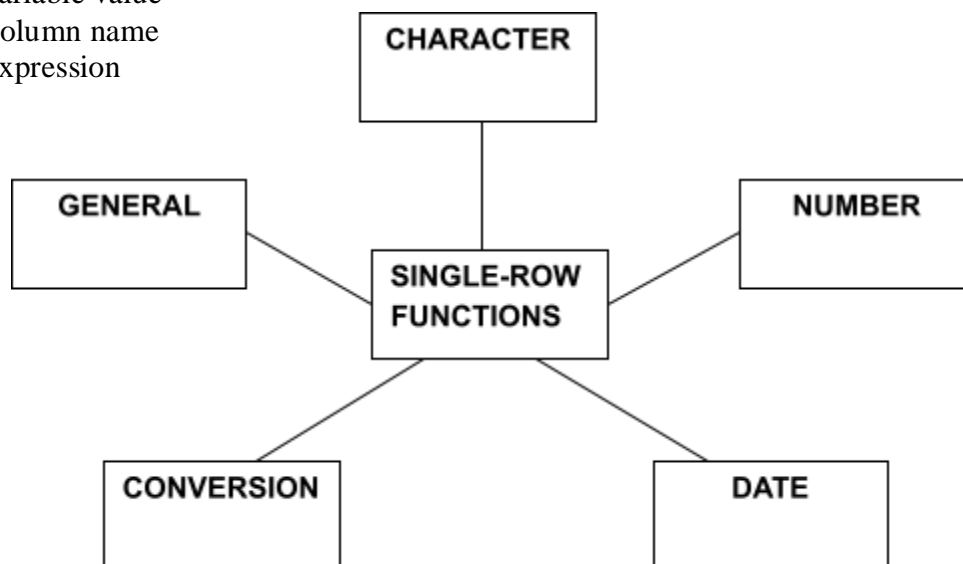
An argument can be one of the following

✓ User-supplied constant

✓ Variable value

✓ Column name

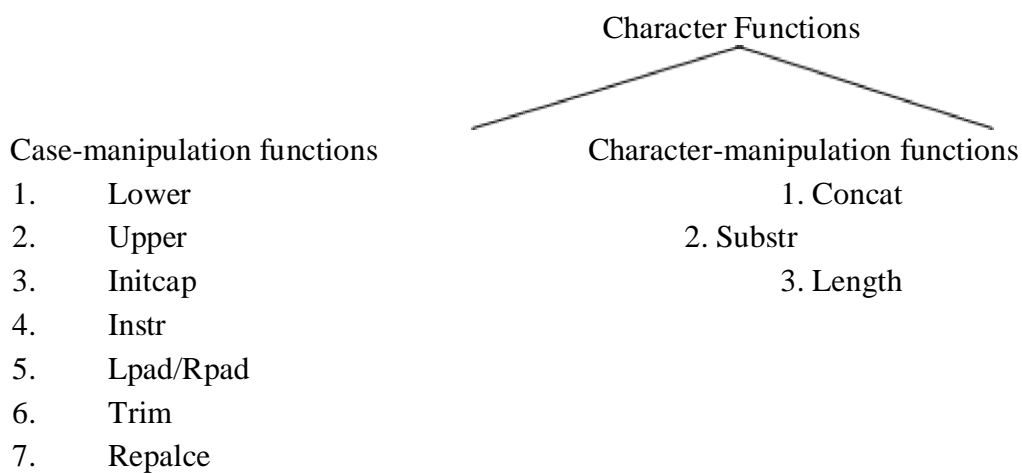
✓ Expression



## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

- Character Functions: Accept character input and can return both character and number values.
- Number functions: Accept numeric input and return numeric values.
- Date Functions: Operate on values of the DATE data type.
- Conversion Functions: Convert a value from one type to another.

### Character Functions



Function	Purpose
lower(column/expr)	Converts alpha character values to lowercase
upper(column/expr)	Converts alpha character values to uppercase
initcap(column/expr)	Converts alpha character values the to uppercase for the first letter of each word, all other letters in lowercase
concat(column1/expr1, column2/expr2)	Concatenates the first character to the second character
substr(column/expr,m,n)	Returns specified characters from character value starting at character position m, n characters long
length(column/expr)	Returns the number of characters in the expression
instr(column/expr,'string',m,n)	Returns the numeric position of a named string
lpad(column/expr, n,'string')	Pads the character value right-justified to a total width of n character positions

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

replace(text, search_string, replacement_string)	
rpad(column/expr,'string',m,n)	Pads the character value left-justified to a total width of n character positions
trim(leading/trailing/both, trim_character FROM trim_source)	Enables you to trim heading or string. trailing or both from a character

**Example:** lower('SQL Course') sql

course upper('SQL Course') SQL

COURSE initcap('SQL Course') Sql

Course

SELECT 'The job id for' || upper(last\_name || 'is') || lower(job\_id) AS "EMPLOYEE DETAILS" FROM employees;

SELECT employee\_id, last\_name, department\_id

FROM employees

WHERE LOWER(last\_name)='higgins';

Function	Result
CONCAT('hello', 'world')	helloworld
Substr('helloworld',1,5)	Hello
Length('helloworld')	10
Instr('helloworld','w')	6
Lpad(salary,10,'*')	*****24000
Rpad(salary,10,'*')	24000*****
Trim('h' FROM 'helloworld')	elloworld

Command	Query	Output
initcap(char);	<i>select initcap("hello") from dual;</i>	Hello

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

lower (char); upper (char);	<i>select lower ('HELLO') from dual;</i> <i>select upper ('hello') from dual;</i>	Hello HELLO
ltrim (char,[set]);	<i>select ltrim ('cseit', 'cse') from dual;</i>	IT
rtrim (char,[set]);	<i>select rtrim ('cseit', 'it') from dual;</i>	CSE
replace (char,search string, replace string);	<i>select replace ('jack and jue', 'j', 'bl') from dual;</i>	black and blue
substr (char,m,n);	<i>select substr ('information', 3, 4) from dual;</i>	form

### Example:

```
SELECT employee_id, CONCAT (first_name,last_name) NAME , job_id,LENGTH(last_name),  
INSTR(last_name,'a') "contains'a'?"  
FROM employees WHERE SUBSTR(job_id,4)='ERP';
```

### NUMBER FUNCTIONS

Function	Purpose
round(column/expr, n)	Rounds the value to specified decimal
trunc(column/expr,n)	Truncates value to specified decimal
mod(m,n)	Returns remainder of division

### Example

Function	Result
round(45.926,2)	45.93
trunc(45.926,2)	45.92
mod(1600,300)	100

```
SELECT ROUND(45.923,2), ROUND(45.923,0), ROUND(45.923,-1) FROM dual;
```

**NOTE:** Dual is a dummy table you can use to view results from functions and calculations.

```
SELECT TRUNC(45.923,2), TRUNC(45.923), TRUNC(45.923,-2) FROM dual;
```



## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

```
SELECT last_name,salary,MOD(salary,5000) FROM employees WHERE job_id='sa_rep';
```

### Working with Dates

The Oracle database stores dates in an internal numeric format: century, year, month, day, hours, minutes, and seconds.

- The default date display format is DD-MON-RR.
  - Enables you to store 21st-century dates in the 20th century by specifying only the last two digits of the year
  - Enables you to store 20th-century dates in the 21st century in the same way

### Example

```
SELECT last_name, hire_date FROM employees WHERE hire_date < '01-FEB-88;
```

### Working with Dates

SYSDATE is a function that returns:

- Date
- Time

### Example

**Display the current date using the DUAL table.**

```
SELECT SYSDATE FROM DUAL;
```

### Arithmetic with Dates

- Add or subtract a number to or from a date for a resultant date value.
- Subtract two dates to find the number of days between those dates.
- Add hours to a date by dividing the number of hours by 24.

### Arithmetic with Dates

Because the database stores dates as numbers, you can perform calculations using arithmetic Operators such as addition and subtraction. You can add and subtract number constants as well as dates.

You can perform the following operations:

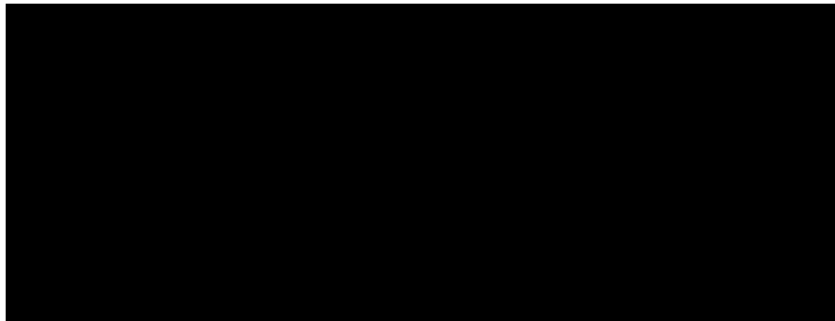
Operation	Result	Description
date + number	Date	Adds a number of days to a date date + number
date - number	Date	Subtracts a number of days from a date date - number
date - date	Number of days	Subtracts one date from another date - date
date + number/24	Date	Adds a number of hours to a date

### Example

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

```
SELECT last_name, (SYSDATE-hire_date)/7 AS WEEKS
FROM employees
WHERE department_id = 90;
```

### Date Functions



### Date Functions

Date functions operate on Oracle dates. All date functions return a value of DATE data type except MONTHS\_BETWEEN, which returns a numeric value.

- MONTHS\_BETWEEN(date1, date2)::: Finds the number of months between date1 and date2. The result can be positive or negative. If date1 is later than date2, the result is positive; if date1 is earlier than date2, the result is negative. The noninteger part of the result represents a portion of the month.
- ADD\_MONTHS(date, n)::: Adds n number of calendar months to date. The value of n must be an integer and can be negative.
- NEXT\_DAY(date, 'char')::: Finds the date of the next specified day of the week ('char') following date. The value of char may be a number representing a day or a character string.
- LAST\_DAY(date)::: Finds the date of the last day of the month that contains date
- ROUND(date[, 'fmt']::: Returns date rounded to the unit that is specified by the format model fmt. If the format model fmt is omitted, date is rounded to the nearest day.
- TRUNC(date[, 'fmt']::: Returns date with the time portion of the day truncated to the unit that is specified by the format model fmt. If the format model fmt is omitted, date is truncated to the nearest day.

### Using Date Functions

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY



### **Example**

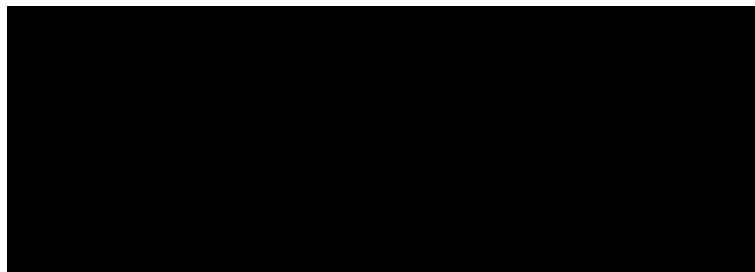
Display the employee number, hire date, number of months employed, sixmonth review date, first Friday after hire date, and last day of the hire month for all employees who have been employed for fewer than 70 months.

```
SELECT employee_id, hire_date, MONTHS_BETWEEN (SYSDATE, hire_date)
TENURE, ADD_MONTHS (hire_date, 6) REVIEW, NEXT_DAY (hire_date, 'FRIDAY'),
LAST_DAY(hire_date)
FROM employees
WHERE MONTHS_BETWEEN (SYSDATE, hire_date) < 70;
```

### **Conversion Functions**

This covers the following topics:

- Writing a query that displays the current date
- Creating queries that require the use of numeric, character, and date functions
- Performing calculations of years and months of service for an employee



### **Implicit Data Type Conversion**

For assignments, the Oracle server can automatically convert the following:

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

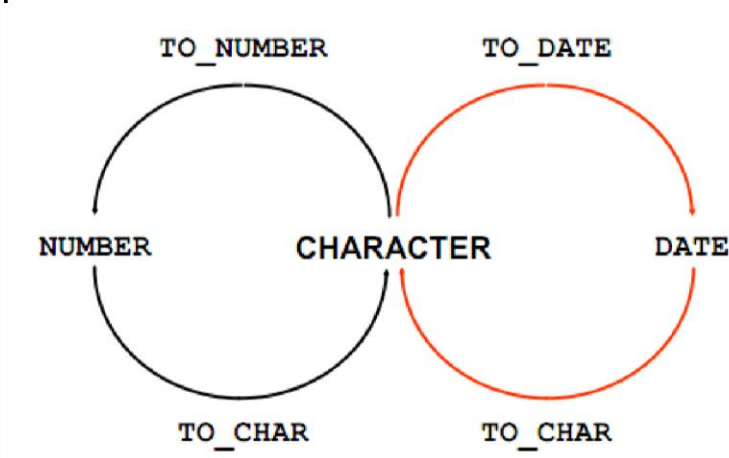


For example, the expression `hire_date > '01-JAN-90'` results in the implicit conversion from the string '01-JAN-90' to a date.

For expression evaluation, the Oracle Server can automatically convert the following:



### Explicit Data Type Conversion



SQL provides three functions to convert a value from one data type to another:

#### Example:

#### Using the TO\_CHAR Function with Dates

`TO_CHAR(date, 'format_model')` The

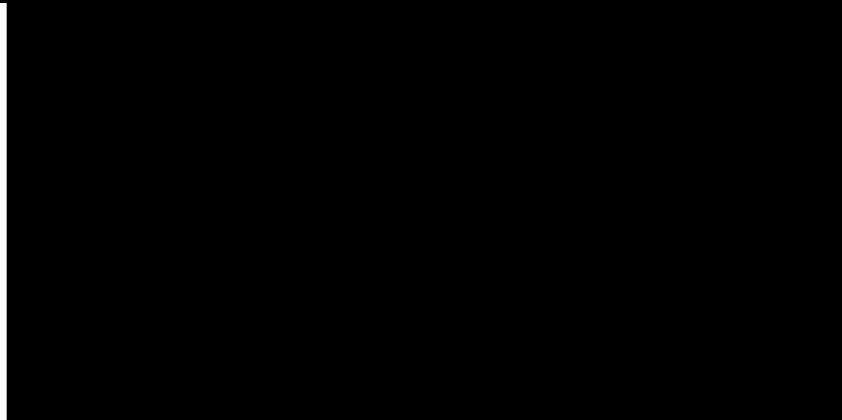
#### format model:

- Must be enclosed by single quotation marks
- Is case-sensitive
- Can include any valid date format element
- Has an fm element to remove padded blanks or suppress leading zeros
- Is separated from the date value by a comma

```
SELECT employee_id, TO_CHAR(hire_date, 'MM/YY') Month_Hired
FROM employees WHERE last_name = 'Higgins';
```

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

### Elements of the Date Format Model



### Sample Format Elements of Valid Date

Element	Description
SCC or CC	Century; server prefixes B.C. date with -
Years in dates YYYY or SYYYY	Year; server prefixes B.C. date with -
YYY or YY or Y	Last three, two, or one digits of year
Y,YYY	Year with comma in this position
IYYY, IYY, IY, I	Four-, three-, two-, or one-digit year based on the ISO standard
SYEAR or YEAR	Year spelled out; server prefixes B.C. date with -
BC or AD	Indicates B.C. or A.D. year
B.C. or A.D.	Indicates B.C. or A.D. year using periods
Q	Quarter of year
MM	Month: two-digit value
MONTH	Name of month padded with blanks to length of nine characters
MON	Name of month, three-letter abbreviation
RM	Roman numeral month
WW or W	Week of year or month
DDD or DD or D	Day of year, month, or week
DAY	Name of day padded with blanks to a length of nine characters
DY	Name of day; three-letter abbreviation
J	Julian day; the number of days since December 31, 4713 B.C.

### **Date Format Elements: Time Formats**

Use the formats that are listed in the following tables to display time information and literals and to change numerals to spelled numbers.

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

Element	Description
AM or PM	Meridian indicator
A.M. or P.M.	Meridian indicator with periods
HH or HH12 or HH24	Hour of day, or hour (1–12), or hour (0–23)
MI	Minute (0–59)
SS	Second (0–59)
SSSSS	Seconds past midnight (0–86399)

### Other Formats

Element	Description
/ . ,	Punctuation is reproduced in the result.
“of the”	Quoted string is reproduced in the result.

### Specifying Suffixes to Influence Number Display

Element	Description
TH	Ordinal number (for example, DDTH for 4TH)
SP	Spelled-out number (for example, DDSP for FOUR)
SPTH or THSP	Spelled-out ordinal numbers (for example, DDSPTH for FOURTH)

### Example

```
SELECT last_name,  
       TO_CHAR(hire_date, 'fmDD Month YYYY') AS HIREDATE  
FROM   employees;
```

Modify example to display the dates in a format that appears as “Seventeenth of June 1987 12:00:00 AM.”

```
SELECT last_name,  
       TO_CHAR (hire_date, 'fmDdspth "of" Month YYYY fmHH:MI:SS AM') HIREDATE  
FROM   employees;
```

### Using the TO\_CHAR Function with Numbers

```
TO_CHAR(number, 'format_model')
```

These are some of the format elements that you can use with the TO\_CHAR function to display a number value as a character:

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY



### Number Format Elements

If you are converting a number to the character data type, you can use the following format elements:

Element	Description	Example	Result
9	Numeric position (number of 9s determine display width)	999999	1234
0	Display leading zeros	099999	001234
\$	Floating dollar sign	\$999999	\$1234
L	Floating local currency symbol	L999999	FF1234
D	Returns in the specified position the decimal character. The default is a period (.).	99D99	99.99
.	Decimal point in position specified	999999.99	1234.00
G	Returns the group separator in the specified position. You can specify multiple group separators in a number format model.	9,999	9G999
,	Comma in position specified	999,999	1,234
MI	Minus signs to right (negative values)	999999MI	1234-
PR	Parenthesize negative numbers	999999PR	<1234>
EEEE	Scientific notation (format must specify four Es)	99.999EEEE	1.234E+03
U	Returns in the specified position the "Euro" (or other) dual currency	U9999	€1234
V	Multiply by 10 <i>n</i> times ( <i>n</i> = number of 9s after V)	9999V99	123400
S	Returns the negative or positive value	S9999	-1234 or +1234
B	Display zero values as blank, not 0	B9999.99	1234.00

```
SELECT TO_CHAR(salary, '$99,999.00') SALARY
FROM employees
WHERE last_name = 'Ernst';
```

### Using the TO\_NUMBER and TO\_DATE Functions

- Convert a character string to a number format using the TO\_NUMBER function:

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

TO\_NUMBER(char[, 'format\_model']

- Convert a character string to a date format using the TO\_DATE function:

TO\_DATE(char[, 'format\_model']

- These functions have an fx modifier. This modifier specifies the exact matching for the character argument and date format model of a TO\_DATE function.

The fx modifier specifies exact matching for the character argument and date format model of a TO\_DATE function:

- Punctuation and quoted text in the character argument must exactly match (except for case) the corresponding parts of the format model.
- The character argument cannot have extra blanks. Without fx, Oracle ignores extra blanks.
- Numeric data in the character argument must have the same number of digits as the corresponding element in the format model. Without fx, numbers in the character argument can omit leading zeros.

```
SELECT last_name, hire_date
```

```
FROM employees
```

```
WHERE hire_date = TO_DATE('May 24, 1999', 'fxMonth DD, YYYY');
```

### **Find the Solution for the following:**

1. Write a query to display the current date. Label the column Date.

```
SELECT SYSDATE AS "Date" FROM dual;
```

2. The HR department needs a report to display the employee number, last name, salary, and increased by 15.5% (expressed as a whole number) for each employee. Label the column New Salary.

```
SELECT employee_id,  
       last_name,  
       salary,  
       ROUND(salary * 1.155) AS "New Salary"  
FROM employees;
```

3. Modify your query lab\_03\_02.sql to add a column that subtracts the old salary from the new salary. Label the column Increase.



## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

```
SELECT employee_id, last_name, salary,  
       ROUND(salary * 1.155) AS "New Salary",  
       ROUND(salary * 1.155) - salary AS Increase FROM employees;
```

4. Write a query that displays the last name (with the first letter uppercase and all other letters lowercase) and the length of the last name for all employees whose name starts with the letters J, A, or M. Give each column an appropriate label. Sort the results by the employees' last names.

```
SELECT  
       INITCAP(last_name) AS "Last Name",  
       LENGTH(last_name) AS "Name Length"  
FROM employees  
WHERE UPPER(SUBSTR(last_name, 1, 1)) IN ('J', 'A', 'M')  
ORDER BY last_name;
```

5. Rewrite the query so that the user is prompted to enter a letter that starts the last name. For example, if the user enters H when prompted for a letter, then the output should show all employees whose last name starts with the letter H.

```
SELECT last_name,  
       LENGTH(last_name) AS "Name Length"  
FROM employees  
WHERE UPPER(SUBSTR(last_name, 1, 1)) = UPPER('&start_letter')  
ORDER BY last_name;
```

6. The HR department wants to find the length of employment for each employee. For each employee, display the last name and calculate the number of months between today and the date on which the employee was hired. Label the column MONTHS\_WORKED. Order your results by the number of months employed. Round the number of months up to the closest whole number.

**Note:** Your results will differ.

```
SELECT last_name,  
       CEIL(MONTHS_BETWEEN(SYSDATE, hire_date)) AS MONTHS_WORKED  
FROM employees  
ORDER BY MONTHS_WORKED;
```

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

7. Create a report that produces the following for each employee:  
<employee last name> earns <salary> monthly but wants <3 times salary>. Label the column Dream Salaries.

```
SELECT last_name || ' earns ' || TO_CHAR(salary) || ' monthly but wants ' ||  
TO_CHAR(salary * 3) AS "Dream Salaries" FROM employees;
```

8. Create a query to display the last name and salary for all employees. Format the salary to be 15 characters long, left-padded with the \$ symbol. Label the column SALARY.

```
SELECT last_name,  
LPAD('$' || TO_CHAR(salary), 15, '$') AS SALARY  
FROM employees;
```

9. Display each employee's last name, hire date, and salary review date, which is the first Monday after six months of service. Label the column REVIEW. Format the dates to appear in the format similar to "Monday, the Thirty-First of July, 2000."

```
SELECT last_name,  
TO_CHAR(hire_date, 'fmDay, "the" fmDDth "of" Month, YYYY') AS hire_date,  
TO_CHAR(  
NEXT_DAY(ADD_MONTHS(hire_date, 6) - 1, 'MONDAY'),  
'fmDay, "the" fmDDth "of" Month, YYYY'  
) AS REVIEW FROM  
employees;
```

10. Display the last name, hire date, and day of the week on which the employee started. Label the column DAY. Order the results by the day of the week, starting with Monday.

```
SELECT last_name, hire_date,  
TO_CHAR(hire_date, 'fmDay') AS DAY  
FROM employees ORDER BY  
CASE TO_CHAR(hire_date, 'DY', 'NLS_DATE_LANGUAGE=ENGLISH')  
WHEN 'MON' THEN 1  
WHEN 'TUE' THEN 2  
WHEN 'WED' THEN 3  
WHEN 'THU' THEN 4  
WHEN 'FRI' THEN 5  
WHEN 'SAT' THEN 6
```

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

**WHEN 'SUN' THEN 7**  
**END;**

### EXERCISE-7

#### Displaying data from multiple tables

##### Objective

After the completion of this exercise, the students will be able to do the following:

- 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 by using a self join

Sometimes you need to use data from more than one table.

##### Cartesian Products

- A Cartesian product is formed when:
  - A join condition is omitted
  - A join condition is invalid
  - All rows in the first table are joined to all rows in the second table
- To avoid a Cartesian product, always include a valid join condition in a WHERE clause. A Cartesian product tends to generate a large number of rows, and the result is rarely useful. You should always include a valid join condition in a WHERE clause, unless you have a specific need to combine all rows from all tables.

Cartesian products are useful for some tests when you need to generate a large number of rows to simulate a reasonable amount of data.

##### Example:

To display employee last name and department name from the EMPLOYEES and DEPARTMENTS tables.

```
SELECT last_name, department_name dept_name
FROM employees, departments;
```

# CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

## Types of Joins

- **Equijoin**
- **Non-equijoin**
- **Outer join**
- **Self join**
- **Cross joins**
- **Natural joins**
- **Using clause**
- **Full or two sided outer joins**
- **Arbitrary join conditions for outer joins**

## Joining Tables Using Oracle Syntax

```
SELECT table1.column, table2.column  
FROM table1, table2  
WHERE table1.column1 = table2.column2;
```

Write the join condition in the WHERE clause.

- Prefix the column name with the table name when the same column name appears in more than one table.

## Guidelines

- When writing a SELECT statement that joins tables, precede the column name with the table name for clarity and to enhance database access.
- If the same column name appears in more than one table, the column name must be prefixed with the table name.
- To join n tables together, you need a minimum of n-1 join conditions. For example, to join four tables, a minimum of three joins is required. This rule may not apply if your table has a concatenated primary key, in which case more than one column is required to uniquely identify each row

## **What is an Equijoin?**

To determine an employee's department name, you compare the value in the DEPARTMENT\_ID column in the EMPLOYEES table with the DEPARTMENT\_ID values in the DEPARTMENTS table.

The relationship between the EMPLOYEES and DEPARTMENTS tables is an equijoin—that is, values in the DEPARTMENT\_ID column on both tables must be equal. Frequently, this type of join involves primary and foreign key complements.

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

Note: Equijoins are also called simple joins or inner joins

```
SELECT employees.employee_id, employees.last_name, employees.department_id,  
departments.department_id, departments.location_id  
FROM employees, departments  
WHERE employees.department_id = departments.department_id;
```

### **Additional Search Conditions**

#### **Using the AND Operator Example:**

To display employee Matos' department number and department name, you need an additional condition in the WHERE clause.

```
SELECT last_name, employees.department_id, department_name  
FROM employees, departments  
WHERE employees.department_id = departments.department_id AND last_name = 'Matos';
```

### **Qualifying Ambiguous**

#### **Column Names**

- Use table prefixes to qualify column names that are in multiple tables.
- Improve performance by using table prefixes.
- Distinguish columns that have identical names but reside in different tables by using column aliases.

### **Using Table Aliases**

- Simplify queries by using table aliases.
- Improve performance by using table prefixes **Example:**

```
SELECT e.employee_id, e.last_name, e.department_id, d.department_id,  
d.location_id  
FROM employees e, departments d  
WHERE e.department_id = d.department_id;
```

### **Joining More than Two Tables**

To join n tables together, you need a minimum of n-1 join conditions. For example, to join three tables, a minimum of two joins is required.

#### **Example:**

To display the last name, the department name, and the city for each employee, you have to join the EMPLOYEES, DEPARTMENTS, and LOCATIONS tables.

```
SELECT e.last_name, d.department_name, l.city
```

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

```
FROM employees e, departments d, locations l
WHERE e.department_id = d.department_id
AND d.location_id = l.location_id;
```

### Non-Equi Joins

A non-equi join is a join condition containing something other than an equality operator. The relationship between the EMPLOYEES table and the JOB\_GRADES table has an example of a non-equi join. A relationship between the two tables is that the SALARY column in the EMPLOYEES table must be between the values in the LOWEST\_SALARY and HIGHEST\_SALARY columns of the JOB\_GRADES table. The relationship is obtained using an operator other than equals (=).

#### Example:

```
SELECT e.last_name, e.salary, j.grade_level
FROM employees e, job_grades j
WHERE e.salary
BETWEEN j.lowest_sal AND j.highest_sal;
```

### Outer Joins

#### **Syntax**

- You use an outer join to also see rows that do not meet the join condition.
- The Outer join operator is the plus sign (+).

```
SELECT table1.column, table2.column
FROM table1, table2
WHERE table1.column(+) = table2.column;
SELECT table1.column, table2.column
FROM table1, table2
WHERE table1.column = table2.column(+);
```

The missing rows can be returned if an outer join operator is used in the join condition. The operator is a plus sign enclosed in parentheses (+), and it is placed on the “side” of the join that is deficient in information. This operator has the effect of creating one or more null rows, to which one or more rows from the nondeficient table can be joined.

#### Example:

```
SELECT e.last_name, e.department_id, d.department_name
FROM employees e, departments d
WHERE e.department_id(+) = d.department_id ;
```

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

### Outer Join Restrictions

- The outer join operator can appear on only one side of the expression—the side that has information missing. It returns those rows from one table that have no direct match in the other table.
- A condition involving an outer join cannot use the IN operator or be linked to another condition by the OR operator

### Self Join

Sometimes you need to join a table to itself.

### Example:

To find the name of each employee's manager, you need to join the EMPLOYEES table to itself, or perform a self join.

```
SELECT worker.last_name || ' works for '
|| manager.last_name
FROM employees worker, employees manager
WHERE worker.manager_id = manager.employee_id ;
```

### Use a join to query data from more than one table.

```
SELECT table1.column, table2.column
FROM table1
[CROSS JOIN table2] |
[NATURAL JOIN table2] |
[JOIN table2 USING (column_name)] |
[JOIN table2
ON(table1.column_name = table2.column_name)] |
[LEFT|RIGHT|FULL OUTER JOIN table2
ON (table1.column_name = table2.column_name)];
```

In the syntax: table1.column Denotes the table and column from which data is retrieved

CROSS JOIN Returns a Cartesian product from the two tables

NATURAL JOIN Joins two tables based on the same column name

JOIN table USING column\_name Performs an equijoin based on the column name

JOIN table ON table1.column\_name Performs an equijoin based on the condition in the ON clause  
= table2.column\_name

### LEFT/RIGHT/FULL OUTER

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

### Creating Cross Joins

- The CROSS JOIN clause produces the crossproduct of two tables.
- This is the same as a Cartesian product between the two tables.

#### Example:

```
SELECT last_name, department_name
FROM employees
CROSS JOIN departments ;
SELECT last_name, department_name
FROM employees, departments;
```

### Creating Natural Joins

- The NATURAL JOIN clause is based on all columns in the two tables that have the same name.
- It selects rows from the two tables that have equal values in all matched columns.
- If the columns having the same names have different data types, an error is returned. **Example:**

```
SELECT department_id, department_name, location_id,
city
FROM departments
NATURAL JOIN locations ;
```

LOCATIONS table is joined to the DEPARTMENT table by the LOCATION\_ID column, which is the only column of the same name in both tables. If other common columns were present, the join would have used them all.

#### Example:

```
SELECT department_id, department_name,
location_id, city
FROM departments
NATURAL JOIN locations
WHERE department_id IN (20, 50);
```

### Creating Joins with the USING Clause

- If several columns have the same names but the data types do not match, the NATURAL JOIN clause can be modified with the USING clause to specify the columns that should be used for an equijoin.
- Use the USING clause to match only one column when more than one column matches.
- Do not use a table name or alias in the referenced columns.



## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

- The NATURAL JOIN and USING clauses are mutually exclusive.

### Example:

```
SELECT l.city, d.department_name
FROM locations l JOIN departments d USING (location_id)
WHERE location_id = 1400; EXAMPLE:
```

```
SELECT e.employee_id, e.last_name, d.location_id
FROM employees e JOIN departments d
USING (department_id);
```

### Creating Joins with the ON Clause

- The join condition for the natural join is basically an equijoin of all columns with the same name.
- To specify arbitrary conditions or specify columns to join, the ON clause is used.
- The join condition is separated from other searchconditions.
- The ON clause makes code easy to understand.

### Example:

```
SELECT e.employee_id, e.last_name, e.department_id, d.department_id,
d.location_id
FROM employees e JOIN departments d ON
(e.department_id = d.department_id);
EXAMPLE:
```

```
SELECT e.last_name emp, m.last_name mgr
FROM employees e JOIN employees m
ON (e.manager_id = m.employee_id);
INNER Versus OUTER Joins
```

• A join between two tables that returns the results of the inner join as well as unmatched rows left (or right) tables is a left (or right) outer join.

- A join between two tables that returns the results of an inner join as well as the results of a left and right join is a full outer join.

### LEFT OUTER JOIN Example:

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

```
SELECT e.last_name, e.department_id, d.department_name
FROM   employees e
LEFT OUTER JOIN departments d
ON    (e.department_id = d.department_id) ;
```

Example of LEFT OUTER JOIN

This query retrieves all rows in the EMPLOYEES table, which is the left table even if there is no match in the DEPARTMENTS table.

This query was completed in earlier releases as follows:

```
SELECT e.last_name, e.department_id, d.department_name
FROM   employees e, departments d
WHERE  d.department_id (+) = e.department_id;
```

### **RIGHT OUTER JOIN Example:**

```
SELECT e.last_name, e.department_id, d.department_name
FROM   employees e
RIGHT OUTER JOIN departments d
ON    (e.department_id = d.department_id) ;
```

This query retrieves all rows in the DEPARTMENTS table, which is the right table even if there is no match in the EMPLOYEES table.

This query was completed in earlier releases as follows:

```
SELECT e.last_name, e.department_id, d.department_name
FROM   employees e, departments d
WHERE  d.department_id = e.department_id (+);
```

### **FULL OUTER JOIN Example:**

```
SELECT e.last_name, e.department_id, d.department_name
```

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

```
FROM employees e  
FULL OUTER JOIN departments d  
ON (e.department_id = d.department_id) ;
```

This query retrieves all rows in the EMPLOYEES table, even if there is no match in the DEPARTMENTS table. It also retrieves all rows in the DEPARTMENTS table, even if there is no match in the EMPLOYEES table.

### Find the Solution for the following:

1. Write a query to display the last name, department number, and department name for all employees.

```
SELECT  
  e.last_name,  
  e.department_id,  
  d.department_name FROM  
  employees e JOIN  
  departments d ON e.department_id = d.department_id;
```

2. Create a unique listing of all jobs that are in department 80. Include the location of the department in the output.

```
SELECT DISTINCT  
  E.job_id, l.location_name FROM employees e JOIN  
  departments d ON e.department_id = d.department_id JOIN  
  locations l ON d.location_id = l.location_id WHERE e.department_id = 80;
```

3. Write a query to display the employee last name, department name, location ID, and city of all employees who earn a commission

```
SELECT  
  e.last_name,  
  d.department_name,  
  l.location_id,  
  l.city  
FROM  
  employees e JOIN  
  departments d ON e.department_id = d.department_id JOIN  
  locations l ON d.location_id = l.location_id  
WHERE  
  e.commission_pct IS NOT NULL;
```

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

4. Display the employee last name and department name for all employees who have an a(lowercase) in their last names. P

```
SELECT
    e.last_name,
    d.department_name FROM
    employees e JOIN
    departments d ON e.department_id = d.department_id
WHERE
    e.last_name LIKE '%a%';
```

5. Write a query to display the last name, job, department number, and department name for all employees who work in Toronto.

```
SELECT
    e.last_name, j.job_title, d.department_id, d.department_name
FROM employees e JOIN
    jobs j ON e.job_id = j.job_id
JOIN
    departments d ON e.department_id = d.department_id JOIN
    locations l ON d.location_id = l.location_id
WHERE
    l.city = 'Toronto';
```

6. Display the employee last name and employee number along with their manager's last name and manager number. Label the columns Employee, Emp#, Manager, and Mgr#, Respectively

```
SELECT
    e.last_name AS Employee,
    e.employee_id AS "Emp#",
    m.last_name AS Manager,
    m.employee_id AS "Mgr#" FROM
    employees e
LEFT JOIN
    employees m ON e.manager_id = m.employee_id;
```

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

7. Modify lab4\_6.sql to display all employees including King, who has no manager. Order the results by the employee number.

```
SELECT  
    e.last_name AS Employee,  
    e.employee_id AS "Emp#",  
    m.last_name AS Manager,  
    m.employee_id AS "Mgr#" FROM  
    employees e  
LEFT JOIN  
    employees m ON e.manager_id = m.employee_id  
ORDER BY  
    e.employee_id;
```

8. Create a query that displays employee last names, department numbers, and all the employees who work in the same department as a given employee. Give each column an appropriate label

```
SELECT  
    e1.last_name AS "Employee",  
    e1.department_id AS "Department Number",  
    e2.last_name AS "Colleague"  
FROM  
    employees e1 JOIN  
    employees e2 ON e1.department_id = e2.department_id ORDER  
BY  
    e1.last_name, e2.last_name;
```

9. Show the structure of the JOB\_GRADES table. Create a query that displays the name, job, department name, salary, and grade for all employees

```
DESC JOB_GRADES;  
  
SELECT  
    e.last_name AS "Name",  
    j.job_title AS "Job",  
    d.department_name AS "Department Name",  
    e.salary AS "Salary",  
    g.grade AS "Grade"  
FROM
```

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

```
employees e JOIN  
jobs j ON e.job_id = j.job_id JOIN  
departments d ON e.department_id = d.department_id JOIN  
job_grades g ON e.salary BETWEEN g.low_salary AND g.high_salary;
```

10. Create a query to display the name and hire date of any employee hired after employee Davies.

```
SELECT  
last_name AS Name, hire_date AS "Hire Date" FROM employees  
WHERE  
hire_date > (  
SELECT hire_date  
FROM employees  
WHERE last_name = 'Davies'  
  
FETCH FIRST 1 ROW ONLY --it may require if multipl values for daves  
);
```

11. Display the names and hire dates for all employees who were hired before their managers, along with their manager's names and hire dates. Label the columns Employee, Emp Hired, Manager, and Mgr Hired, respectively.

```
SELECT  
e.last_name AS Employee, e.hire_date AS "Emp Hired",  
m.last_name AS Manager, m.hire_date AS "Mgr Hired"  
FROM employees e JOIN  
employees m ON e.manager_id = m.employee_id  
WHERE  
e.hire_date < m.hire_date;
```

### EXERCISE-8

#### Aggregating Data Using Group Functions

##### Objectives

After the completion of this exercise, the students be will be able to do the following:

- Identify the available group functions

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

- Describe the use of group functions
- Group data by using the GROUP BY clause
- Include or exclude grouped rows by using the HAVING clause

### What Are Group Functions?

Group functions operate on sets of rows to give one result per group

### Types of Group Functions

- AVG
- COUNT
- MAX
- MIN
- STDDEV
- SUM
- VARIANCE

Each of the functions accepts an argument. The following table identifies the options that you can use in the syntax:

Function	Description
AVG ( [DISTINCT   <u>ALL</u> ] <i>n</i> )	Average value of <i>n</i> , ignoring null values
COUNT ( { *   [DISTINCT   <u>ALL</u> ] <i>expr</i> } )	Number of rows, where <i>expr</i> evaluates to something other than null (count all selected rows using *, including duplicates and rows with nulls)
MAX ( [DISTINCT   <u>ALL</u> ] <i>expr</i> )	Maximum value of <i>expr</i> , ignoring null values
MIN ( [DISTINCT   <u>ALL</u> ] <i>expr</i> )	Minimum value of <i>expr</i> , ignoring null values
STDDEV ( [DISTINCT   <u>ALL</u> ] <i>x</i> )	Standard deviation of <i>n</i> , ignoring null values
SUM ( [DISTINCT   <u>ALL</u> ] <i>n</i> )	Sum values of <i>n</i> , ignoring null values
VARIANCE ( [DISTINCT   <u>ALL</u> ] <i>x</i> )	Variance of <i>n</i> , ignoring null values

### Group Functions: Syntax

SELECT [*column*,] *group\_function*(*column*), ... FROM  
*table*

[WHERE *condition*]

[GROUP BY *column*]

[ORDER BY *column*];

### Guidelines for Using Group Functions

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

- DISTINCT makes the function consider only nonduplicate values; ALL makes it consider every value, including duplicates. The default is ALL and therefore does not need to be specified.
- The data types for the functions with an expr argument may be CHAR, VARCHAR2, NUMBER, or DATE.
- All group functions ignore null values.

### **Using the AVG and SUM Functions**

You can use AVG and SUM for numeric data.

```
SELECT AVG(salary), MAX(salary),  
MIN(salary), SUM(salary)  
FROM employees  
WHERE job_id LIKE '%REP%';
```

### **Using the MIN and MAX Functions**

You can use MIN and MAX for numeric, character, and date data types.

```
SELECT MIN(hire_date), MAX(hire_date)  
FROM employees;
```

You can use the MAX and MIN functions for numeric, character, and date data types. example displays the most junior and most senior employees.

The following example displays the employee last name that is first and the employee last name that is last in an alphabetized list of all employees:

```
SELECT MIN(last_name), MAX(last_name)  
FROM employees;
```

**Note:** The AVG, SUM, VARIANCE, and STDDEV functions can be used only with numeric data types. MAX and MIN cannot be used with LOB or LONG data types.

### **Using the COUNT Function**

COUNT(\*) returns the number of rows in a table:  
SELECT COUNT(\*)



## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

FROM employees

WHERE department\_id = 50;

COUNT(*expr*) returns the number of rows with nonnull values for the *expr*:

SELECT COUNT(commission\_pct)

FROM employees

WHERE department\_id = 80;

### **Using the DISTINCT Keyword**

- COUNT(DISTINCT *expr*) returns the number of distinct non-null values of the *expr*.
- To display the number of distinct department values in the EMPLOYEES table:

SELECT COUNT(DISTINCT department\_id) FROM employees;

Use the DISTINCT keyword to suppress the counting of any duplicate values in a column.

### **Group Functions and Null Values**

Group functions ignore null values in the column:

SELECT AVG(commission\_pct)

FROM employees;

The NVL function forces group functions to include null values:

SELECT AVG(NVL(commission\_pct, 0))

FROM employees;

### **Creating Groups of Data**

To divide the table of information into smaller groups. This can be done by using the GROUP BY clause.

### **GROUP BY Clause Syntax**

SELECT *column*, *group\_function(column)*

FROM *table*

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

[WHERE *condition*]

[GROUP BY *group\_by\_expression*]

[ORDER BY *column*];

**In the syntax:** *group\_by\_expression* specifies columns whose values determine the basis for grouping rows

### **Guidelines**

- If you include a group function in a SELECT clause, you cannot select individual results as well, *unless* the individual column appears in the GROUP BY clause. You receive an error message if you fail to include the column list in the GROUP BY clause.
- Using a WHERE clause, you can exclude rows before dividing them into groups.
- You must include the *columns* in the GROUP BY clause.
- You cannot use a column alias in the GROUP BY clause.

### **Using the GROUP BY Clause**

All columns in the SELECT list that are not in group functions must be in the GROUP BY clause.

```
SELECT department_id, AVG(salary)
FROM employees
GROUP BY department_id ;
```

The GROUP BY column does not have to be in the SELECT list.

```
SELECT AVG(salary) FROM employees GROUP BY department_id ;
```

You can use the group function in the ORDER BY clause:

```
SELECT department_id, AVG(salary) FROM employees GROUP BY department_id ORDER
BY AVG(salary);
```

### **Grouping by More Than One Column**

```
SELECT department_id dept_id, job_id, SUM(salary) FROM employees
GROUP BY department_id, job_id ;
```

# CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

## Illegal Queries Using Group Functions

Any column or expression in the SELECT list that is not an aggregate function must be in the GROUP

### **BY clause:**

```
SELECT department_id, COUNT(last_name) FROM employees;
```

You can correct the error by adding the GROUP BY clause:

```
SELECT department_id, count(last_name) FROM employees GROUP BY department_id;
```

You cannot use the WHERE clause to restrict groups.

- You use the HAVING clause to restrict groups.
- You cannot use group functions in the WHERE clause.

```
SELECT department_id, AVG(salary) FROM employees WHERE AVG(salary) > 8000  
GROUP BY department_id;
```

You can correct the error in the example by using the HAVING clause to restrict groups:

```
SELECT department_id, AVG(salary) FROM employees  
HAVING AVG(salary) > 8000 GROUP BY department_id;
```

## Restricting Group Results

With the HAVING Clause .When you use the HAVING clause, the Oracle server restricts groups as follows:

1. Rows are grouped.
2. The group function is applied.
3. Groups matching the HAVING clause are displayed.

## Using the HAVING Clause

```
SELECT department_id, MAX(salary) FROM employees  
GROUP BY department_id HAVING MAX(salary)>10000 ;
```

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

The following example displays the department numbers and average salaries for those departments with a maximum salary that is greater than \$10,000:

```
SELECT department_id, AVG(salary) FROM employees GROUP BY department_id  
HAVING max(salary)>10000;
```

Example displays the job ID and total monthly salary for each job that has a total payroll exceeding \$13,000. The example excludes sales representatives and sorts the list by the total monthly salary.

```
SELECT job_id, SUM(salary) PAYROLL FROM employees WHERE job_id NOT LIKE  
'%REP%'  
GROUP BY job_id HAVING SUM(salary) > 13000 ORDER BY SUM(salary);
```

### Nesting Group Functions

#### **Display the maximum average salary:**

Group functions can be nested to a depth of two. The slide example displays the maximum average salary.

```
SELECT MAX(AVG(salary)) FROM employees GROUP BY department_id;
```

#### **Summary**

In this exercise, students should have learned how to:

- Use the group functions COUNT, MAX, MIN, and AVG
- Write queries that use the GROUP BY clause
- Write queries that use the HAVING clause

```
SELECT column, group_function  
FROM table  
[WHERE condition]  
[GROUP BY group_by_expression]  
[HAVING group_condition]  
[ORDER BY column];
```

#### **Find the Solution for the following:**

Determine the validity of the following three statements. Circle either True or False.

1. Group functions work across many rows to produce one result per group.

**True**/False

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

2. Group functions include nulls in calculations.

True/**False**

3. The WHERE clause restricts rows prior to inclusion in a group calculation.

**True**/False

**The HR department needs the following reports:**

4. Find the highest, lowest, sum, and average salary of all employees. Label the columns Maximum, Minimum, Sum, and Average, respectively. Round your results to the nearest whole number

**SELECT**

**ROUND(MAX(salary)) AS Maximum,**

**ROUND(MIN(salary)) AS Minimum,**

**ROUND(SUM(salary)) AS Sum,**

**ROUND(AVG(salary)) AS Average**

**FROM employees;**

5. Modify the above query to display the minimum, maximum, sum, and average salary for each job type.

**SELECT**

**job\_type,**

**ROUND(MIN(salary)) AS Minimum,**

**ROUND(MAX(salary)) AS Maximum,**

**ROUND(SUM(salary)) AS Sum,**

**ROUND(AVG(salary)) AS Average**

**FROM employees**

**GROUP BY job\_type;**

6. Write a query to display the number of people with the same job. Generalize the query so that the user in the HR department is prompted for a job title.

**SELECT job\_type, COUNT(\*) AS Number\_of\_People**

**FROM employees WHERE job\_type = ?**

**GROUP BY job\_type;**

7. Determine the number of managers without listing them. Label the column Number of Managers. *Hint: Use the MANAGER\_ID column to determine the number of managers.*

**SELECT COUNT(DISTINCT MANAGER\_ID) AS "Number of Managers"**

**FROM employees**

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

**WHERE MANAGER\_ID IS NOT NULL;**

8. Find the difference between the highest and lowest salaries. Label the column DIFFERENCE.

```
SELECT  
  ROUND(MAX(salary) - MIN(salary)) AS DIFFERENCE  
FROM employees;
```

9. Create a report to display the manager number and the salary of the lowest-paid employee for that manager. Exclude anyone whose manager is not known. Exclude any groups where the minimum salary is \$6,000 or less. Sort the output in descending order of salary.

```
SELECT  
  MANAGER_ID AS Manager_Number,  
  MIN(salary) AS Lowest_Salary  
FROM employees  
WHERE MANAGER_ID IS NOT NULL  
GROUP BY MANAGER_ID  
HAVING MIN(salary) > 6000  
ORDER BY Lowest_Salary DESC;
```

10. Create a query to display the total number of employees and, of that total, the number of employees hired in 1995, 1996, 1997, and 1998. Create appropriate column headings.

```
SELECT  
  COUNT(*) AS Total_Employees,  
  COUNT(CASE WHEN EXTRACT(YEAR FROM hire_date) = 1995 THEN 1 END) AS Hired_1995,  
  COUNT(CASE WHEN EXTRACT(YEAR FROM hire_date) = 1996 THEN 1 END) AS Hired_1996,  
  COUNT(CASE WHEN EXTRACT(YEAR FROM hire_date) = 1997 THEN 1 END) AS Hired_1997,  
  COUNT(CASE WHEN EXTRACT(YEAR FROM hire_date) = 1998 THEN 1 END) AS Hired_1998  
FROM employees;
```

11. Create a matrix query to display the job, the salary for that job based on department number, and the total salary for that job, for departments 20, 50, 80, and 90, giving each column an appropriate heading.

```
SELECT  
  job,  
  SUM(CASE WHEN department_id = 20 THEN salary ELSE 0 END) AS Salary_Dept_20,  
  SUM(CASE WHEN department_id = 50 THEN salary ELSE 0 END) AS Salary_Dept_50,  
  SUM(CASE WHEN department_id = 80 THEN salary ELSE 0 END) AS Salary_Dept_80,  
  SUM(CASE WHEN department_id = 90 THEN salary ELSE 0 END) AS Salary_Dept_90,  
  SUM(salary) AS Total_Salary  
FROM employees  
WHERE department_id IN (20, 50, 80, 90)  
GROUP BY job
```

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

### **ORDER BY job;**

12. Write a query to display each department's name, location, number of employees, and the average salary for all the employees in that department. Label the column name-Location, Number of people, and salary respectively. Round the average salary to two decimal places.

### **SELECT**

```
d.department_name || '-' || d.location AS "name-Location",  
COUNT(e.employee_id) AS "Number of people",  
ROUND(AVG(e.salary), 2) AS salary  
FROM departments d  
LEFT JOIN employees e ON d.department_id = e.department_id  
GROUP BY d.department_name, d.location  
ORDER BY d.department_name;
```

## **EXERCISE-9**

### **Sub queries**

#### **Objectives**

After completing this lesson, you should be able to do the following:

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

**Using a Subquery to Solve a Problem Who**  
has a salary greater than Abel's?

**Main query:**

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

Which employees have salaries greater than Abel's salary?

### **Subquery:**

What is Abel's salary?

### **Subquery Syntax**

`SELECT select_list FROM table WHERE expr operator (SELECT select_list FROM table);`

- The subquery (inner query) executes once before the main query (outer query).
- The result of the subquery is used by the main query.

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, including the following:

- WHERE clause
- HAVING clause
- FROM clause

### **In the syntax:**

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

**Note:** Comparison conditions fall into two classes: single-row operators

(>, =, >=, <, <>, <=) and multiple-row operators (IN, ANY, ALL). statement. The subquery generally executes first, and its output is used to complete the query condition for the main (or outer) query

### **Using a Subquery**

```
SELECT last_name FROM employees WHERE salary > (SELECT salary FROM employees WHERE last_name = 'Abel');
```

The inner query determines the salary of employee Abel. The outer query takes the result of the inner query and uses this result to display all the employees who earn more than this amount.



# CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

## Guidelines for Using Subqueries

- Enclose subqueries in parentheses.
- Place subqueries on the right side of the comparison condition.
- The ORDER BY clause in the subquery is not needed unless you are performing Top-N analysis.
- Use single-row operators with single-row

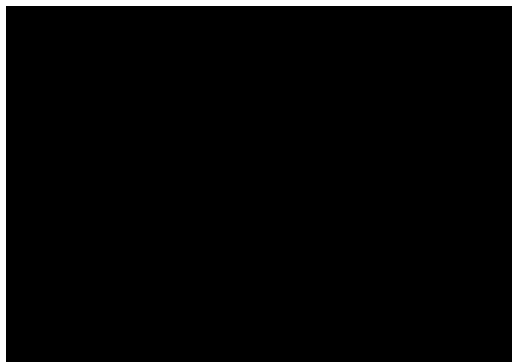
subqueries, and use multiple-row operators with multiple-row subqueries.

## Types of Subqueries

- Single-row subqueries: Queries that return only one row from the inner SELECT statement.
- Multiple-row subqueries: Queries that return more than one row from the inner SELECT statement.

## Single-Row Subqueries

- Return only one row
- Use single-row comparison operators



## Example

Display the employees whose job ID is the same as that of employee 141:

```
SELECT last_name, job_id FROM employees WHERE job_id = (SELECT job_id FROM employees  
WHERE employee_id = 141);
```

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

Displays employees whose job ID is the same as that of employee 141 and whose salary is greater than that of employee 143.

```
SELECT last_name, job_id, salary FROM employees WHERE job_id =(SELECT job_id FROM employees WHERE employee_id = 141) AND salary > (SELECT salary FROM employees WHERE employee_id = 143);
```

### Using Group Functions in a Subquery

Displays the employee last name, job ID, and salary of all employees whose salary is equal to the minimum salary. The MIN group function returns a single value (2500) to the outer query.

```
SELECT last_name, job_id, salary FROM employees WHERE salary = (SELECT MIN(salary) FROM employees);
```

### The HAVING Clause with Subqueries

- The Oracle server executes subqueries first.
- The Oracle server returns results into the HAVING clause of the main query.

Displays all the departments that have a minimum salary greater than that of department 50.

```
SELECT department_id, MIN(salary)
FROM employees
GROUP BY department_id
HAVING MIN(salary) >
(SELECT MIN(salary)
FROM employees
WHERE department_id = 50);
```

### Example

**Find the job with the lowest average salary.**

```
SELECT job_id, AVG(salary)
FROM employees
GROUP BY job_id
HAVING AVG(salary) = (SELECT MIN(AVG(salary))
```

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

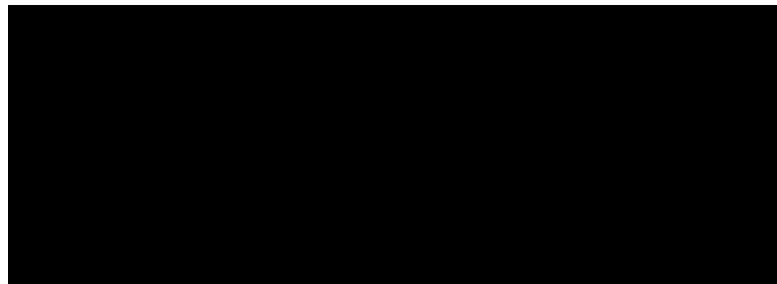
```
FROM employees  
GROUP BY job_id);
```

**What Is Wrong in this Statements?**

```
SELECT employee_id, last_name  
FROM employees  
WHERE salary =(SELECT MIN(salary) FROM employees GROUP BY department_id); Will  
This Statement Return Rows?  
SELECT last_name, job_id  
FROM employees  
WHERE job_id =(SELECT job_id FROM employees WHERE last_name = 'Haas');
```

### Multiple-Row Subqueries

- Return more than one row
- Use multiple-row comparison operators



### Example

Find the employees who earn the same salary as the minimum salary for each department.

```
SELECT last_name, salary, department_id FROM employees WHERE salary IN (SELECT  
MIN(salary)  
FROM employees GROUP BY department_id);
```

Using the ANY Operator in Multiple-Row Subqueries

```
SELECT employee_id, last_name, job_id, salary FROM employees WHERE salary < ANY  
(SELECT salary FROM employees WHERE job_id = 'IT_PROG') AND job_id <> 'IT_PROG';
```

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

Displays employees who are not IT programmers and whose salary is less than that of any IT programmer. The maximum salary that a programmer earns is \$9,000.

< ANY means less than the maximum. >ANY means more than the minimum. =ANY is equivalent to IN.

### **Using the ALL Operator in Multiple-Row Subqueries**

```
SELECT employee_id, last_name, job_id, salary
FROM employees
WHERE salary < ALL (SELECT salary FROM employees WHERE job_id = 'IT_PROG') AND
job_id <> 'IT_PROG';
```

Displays employees whose salary is less than the salary of all employees with a job ID of IT\_PROG and whose job is not IT\_PROG.

ALL means more than the maximum, and <ALL means less than the minimum.

The NOT operator can be used with IN, ANY, and ALL operators.

### **Null Values in a Subquery**

```
SELECT emp.last_name FROM employees emp
WHERE emp.employee_id NOT IN (SELECT mgr.manager_id FROM employees mgr);
```

Notice that the null value as part of the results set of a subquery is not a problem if you use the IN operator. The IN operator is equivalent to =ANY. For example, to display the employees who have subordinates, use the following SQL statement:

```
SELECT emp.last_name
FROM employees emp
WHERE emp.employee_id IN (SELECT mgr.manager_id FROM employees mgr);
```

Display all employees who do not have any subordinates:

```
SELECT last_name FROM employees
WHERE employee_id NOT IN (SELECT manager_id FROM employees WHERE manager_id IS
NOT NULL);
```

### **Find the Solution for the following:**

1. The HR department needs a query that prompts the user for an employee last name. The query then displays the last name and hire date of any employee in the same department as the

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

employee whose name they supply (excluding that employee). For example, if the user enters Zlotkey, find all employees who work with Zlotkey (excluding Zlotkey).

```
SELECT last_name, hire_date  
FROM employees  
WHERE department_id = (  
    SELECT department_id  
    FROM employees WHERE  
    last_name = ?  
)  
AND last_name <> ?  
ORDER BY last_name;
```

2. Create a report that displays the employee number, last name, and salary of all employees who earn more than the average salary. Sort the results in order of ascending salary.

```
SELECT employee_id, last_name, salary  
FROM employees  
WHERE salary > (SELECT AVG(salary) FROM employees)  
ORDER BY salary ASC;
```

3. Write a query that displays the employee number and last name of all employees who work in a department with any employee whose last name contains a *u*.

```
SELECT employee_id, last_name  
FROM employees  
WHERE department_id IN (  
    SELECT DISTINCT department_id  
    FROM employees  
    WHERE last_name LIKE '%u%'  
)  
ORDER BY employee_id;
```

4. The HR department needs a report that displays the last name, department number, and job ID of all employees whose department location ID is 1700.

```
SELECT e.last_name, e.department_id, e.job_id  
FROM employees e  
JOIN departments d ON e.department_id = d.department_id  
WHERE d.location_id = 1700  
ORDER BY e.last_name;
```

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

5. Create a report for HR that displays the last name and salary of every employee who reports to King.

```
SELECT e.last_name, e.salary  
FROM employees e  
JOIN employees m ON e.manager_id = m.employee_id  
WHERE m.last_name = 'King'  
ORDER BY e.last_name;
```

6. Create a report for HR that displays the department number, last name, and job ID for every employee in the Executive department.

```
SELECT e.department_id, e.last_name, e.job_id  
FROM employees e  
JOIN departments d ON e.department_id = d.department_id  
WHERE d.department_name = 'Executive'  
ORDER BY e.last_name;
```

7. Modify the query 3 to display the employee number, last name, and salary of all employees who earn more than the average salary and who work in a department with any employee whose last name contains a *u*.

```
SELECT employee_id, last_name, salary  
FROM employees  
WHERE salary > (SELECT AVG(salary) FROM employees)  
AND department_id IN (  
    SELECT DISTINCT department_id  
    FROM employees  
    WHERE last_name LIKE '%u%'  
)  
ORDER BY salary ASC;
```

# CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

## EXERCISE-10

### USING THE SET OPERATORS

#### Objectives

After the completion this exercise, the students should be able to do the following:

- Describe set operators
- Use a set operator to combine multiple queries into a single query
- Control the order of rows returned

The set operators combine the results of two or more component queries into one result.

Queries containing set operators are called *compound queries*.

Operator	Returns
UNION	All distinct rows selected by either query
UNION ALL	All rows selected by either query, including all duplicates
INTERSECT	All distinct rows selected by both queries
MINUS	All distinct rows that are selected by the first SELECT statement and not selected in the second SELECT statement

#### The tables used in this lesson are:

- EMPLOYEES: Provides details regarding all current employees
- JOB\_HISTORY: Records the details of the start date and end date of the former job, and the job identification number and department when an employee switches jobs

#### UNION Operator

#### Guidelines

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

- The number of columns and the data types of the columns being selected must be identical in all the SELECT statements used in the query. The names of the columns need not be identical.
- UNION operates over all of the columns being selected.
- NULL values are not ignored during duplicate checking.
- The IN operator has a higher precedence than the UNION operator.
- By default, the output is sorted in ascending order of the first column of the SELECT clause.

### **Example:**

Display the current and previous job details of all employees. Display each employee only once.

```
SELECT employee_id, job_id FROM employees UNION SELECT employee_id, job_id FROM  
job_history;
```

### **Example:**

```
SELECT employee_id, job_id, department_id  
FROM employees  
UNION  
SELECT employee_id, job_id, department_id  
FROM job_history;
```

### **UNION ALL Operator**

#### **Guidelines**

The guidelines for UNION and UNION ALL are the same, with the following two exceptions that pertain to UNION ALL:

- Unlike UNION, duplicate rows are not eliminated and the output is not sorted by default.
- The DISTINCT keyword cannot be used.

### **Example:**

Display the current and previous departments of all employees.

```
SELECT employee_id, job_id, department_id  
FROM employees  
UNION ALL
```



## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

```
SELECT employee_id, job_id, department_id  
FROM job_history  
ORDER BY employee_id;
```

### **INTERSECT Operator**

#### **Guidelines**

- The number of columns and the data types of the columns being selected by the SELECT statements in the queries must be identical in all the SELECT statements used in the query. The names of the columns need not be identical.
- Reversing the order of the intersected tables does not alter the result.
- INTERSECT does not ignore NULL values.

#### **Example:**

Display the employee IDs and job IDs of those employees who currently have a job title that is the same as their job title when they were initially hired (that is, they changed jobs but have now gone back to doing their original job).

```
SELECT employee_id, job_id FROM employees  
INTERSECT  
SELECT employee_id, job_id  
FROM job_history;
```

#### **Example**

```
SELECT employee_id, job_id, department_id  
FROM employees  
INTERSECT  
SELECT employee_id, job_id, department_id  
FROM job_history;
```

### **MINUS Operator**

#### **Guidelines**

- The number of columns and the data types of the columns being selected by the SELECT statements in the queries must be identical in all the SELECT statements used in the query. The names of the columns need not be identical.
- All of the columns in the WHERE clause must be in the SELECT clause for the MINUS operator to work.

#### **Example:**

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

Display the employee IDs of those employees who have not changed their jobs even once.

```
SELECT employee_id,job_id
FROM employees
MINUS
SELECT employee_id,job_id
FROM job_history;
```

### **Find the Solution for the following:**

1. The HR department needs a list of department IDs for departments that do not contain the job ID ST\_CLERK. Use set operators to create this report.
2. The HR department needs a list of countries that have no departments located in them. Display the country ID and the name of the countries. Use set operators to create this report.
3. Produce a list of jobs for departments 10, 50, and 20, in that order. Display job ID and department ID using set operators.
4. Create a report that lists the employee IDs and job IDs of those employees who currently have a job title that is the same as their job title when they were initially hired by the company (that is, they changed jobs but have now gone back to doing their original job).
5. The HR department needs a report with the following specifications:
  - Last name and department ID of all the employees from the EMPLOYEES table, regardless of whether or not they belong to a department.
  - Department ID and department name of all the departments from the DEPARTMENTS table, regardless of whether or not they have employees working in them Write a compound query to accomplish this.

# **CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY**

## **EXERCISE-11**

### **CREATING VIEWS**

After the completion of this exercise, students will be able to do the following:

- Describe a view
- Create, alter the definition of, and drop a view
- Retrieve data through a view
- Insert, update, and delete data through a view
- Create and use an inline view

### **View**

A view is a logical table based on a table or another view. A view contains no data but is like a window through which data from tables can be viewed or changed. The tables on which a view is based are called base tables.

### **Advantages of Views**

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

- To restrict data access
- To make complex queries easy
- To provide data independence

Feature	Simple	Complex
No. of tables	One	One or more
Contains functions	No	Yes
Contains groups of data	No	Yes
DML operations thr' view	Yes	Not always

- To present different views of the same data

### Classification of views

1. Simple view
2. Complex view

### Creating a view

#### Syntax

CREATE OR REPLACE FORCE/NOFORCE VIEW view\_name AS Subquery WITH CHECK OPTION CONSTRAINT constraint WITH READ ONLY CONSTRAINT constraint;

**FORCE** - Creates the view regardless of whether or not the base tables exist.

**NOFORCE** - Creates the view only if the base table exist.

WITH CHECK OPTION CONSTRAINT-specifies that only rows accessible to the view can be inserted or updated.

WITH READ ONLY CONSTRAINT-ensures that no DML operations can be performed on the view.

### Example: 1 (Without using Column aliases)

Create a view EMPVU80 that contains details of employees in department80.

### Example 2:

CREATE VIEW empvu80 AS SELECT employee\_id, last\_name, salary FROM employees WHERE department\_id=80;

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

### **Example:1** (Using column aliases)

```
CREATE VIEW salvu50
AS SELECT employee_id,id_number, last_name NAME, salary *12 ANN_SALARY
FROM employees
WHERE department_id=50;
```

### **Retrieving data from a view**

#### **Example:**

```
SELECT * from salvu50;
```

### **Modifying a view**

A view can be altered without dropping, re-creating.

#### **Example:** (Simple view)

Modify the EMPVU80 view by using CREATE OR REPLACE.

```
CREATE OR REPLACE VIEW empvu80 (id_number, name, sal, department_id)
AS SELECT employee_id,first_name, last_name, salary, department_id
FROM employees
WHERE department_id=80;
```

#### **Example:** (complex view)

```
CREATE VIEW dept_sum_vu (name, minsal, maxsal,avgsal)
AS SELECT d.department_name, MIN(e.salary), MAX(e.salary), AVG(e.salary)
FROM employees e, department d
WHERE e.deparment_id=d.deparment_id
GROUP BY d.department_name;
```

### **Rules for performing DML operations on view**

- Can perform operations on simple views
- Cannot remove a row if the view contains the following:
  - Group functions

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

- Group By clause
- Distinct keyword
- Cannot modify data in a view if it contains
- Group functions
- Group By clause
- Distinct keyword
- Columns contain by expressions
- 
- Cannot add data thr' a view if it contains
- Group functions
- Group By clause
- Distinct keyword
- Columns contain by expressions
- NOT NULL columns in the base table that are not selected by the view

**Example:** (Using the WITH CHECK OPTION clause)

```
CREATE OR REPLACE VIEW empvu20
AS   SELECT *
FROM employees
WHERE department_id=20
WITH CHECK OPTION CONSTRAINT empvu20_ck;
```

**Note:** Any attempt to change the department number for any row in the view fails because it violates the WITH CHECK OPTION constraint.

**Example** – (Execute this and note the error)

```
UPDATE empvu20 SET department_id=10 WHERE employee_id=201;
```

### **Denying DML operations**

Use of WITH READ ONLY option.

Any attempt to perform a DML on any row in the view results in an oracle server error.

**Try this code:**

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

```
CREATE OR REPLACE VIEW empvu10(employee_number, employee_name,job_title)
AS SELECT employee_id, last_name, job_id
FROM employees
WHERE department_id=10
WITH READ ONLY;
```

### **Find the Solution for the following:**

1. Create a view called EMPLOYEE\_VU based on the employee numbers, employee names and department numbers from the EMPLOYEES table. Change the heading for the employee name to EMPLOYEE.
2. Display the contents of the EMPLOYEES\_VU view.
3. Select the view name and text from the USER\_VIEWS data dictionary views.
4. Using your EMPLOYEES\_VU view, enter a query to display all employees names and department.
5. Create a view named DEPT50 that contains the employee number, employee last names and department numbers for all employees in department 50. Label the view columns EMPNO, EMPLOYEE and DEPTNO. Do not allow an employee to be reassigned to another department through the view.
6. Display the structure and contents of the DEPT50 view.

## **CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY**

7. Attempt to reassign Matos to department 80.
  
8. Create a view called SALARY\_VU based on the employee last names, department names, salaries, and salary grades for all employees. Use the Employees, DEPARTMENTS and JOB\_GRADE tables. Label the column Employee, Department, salary, and Grade respectively.



# CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

## PL/SQL

### Control Structures

In addition to SQL commands, PL/SQL can also process data using flow of statements. The flow of control statements are classified into the following categories.

- Conditional control -  
Branching
- Iterative control - looping
  - Sequential control

### **BRANCHING in PL/SQL:**

Sequence of statements can be executed on satisfying certain condition .

If statements are being used and different forms of if are:

1. Simple IF

2. ELSIF

3. ELSE IF **SIMPLE**

### **IF:**

### **Syntax:**

IF condition THEN

    statement1;

statement2;

END IF;

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

### **IF-THEN-ELSE STATEMENT:**

#### **Syntax:**

IF condition THEN

statement1;

ELSE

statement2;

END IF;

### **ELSIF STATEMENTS:**

#### **Syntax:** IF

condition1 THEN

statement1; ELSIF

condition2 THEN

statement2; ELSIF

condition3 THEN

statement3;

ELSE

statementn;

END IF;

### **NESTED IF :**

#### **Syntax:**

IF condition THEN

statement1;

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

ELSE

IF condition THEN

statement2; ELSE

statement3;

END IF;

END IF; ELSE

statement3;

END IF;

### **SELECTION IN PL/SQL(Sequential Controls)**

#### **SIMPLE CASE**

##### **Syntax:**

CASE SELECTOR

WHEN Expr1 THEN statement1;

WHEN Expr2 THEN statement2;

:

ELSE

Statement n;

END CASE;

#### **SEARCHED CASE:**

CASE

WHEN searchcondition1 THEN statement1;

WHEN searchcondition2 THEN statement2;

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

```
:  
:  
ELSE  
    statementn;  
END CASE;
```

### **ITERATIONS IN PL/SQL**

Sequence of statements can be executed any number of times using loop construct.

It is broadly classified into:

- Simple Loop
- For Loop
- While Loop

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

### **SIMPLE LOOP**

**Syntax:** LOOP

statement1;

EXIT [ WHEN Condition];

END LOOP;

### **WHILE LOOP**

**Syntax:**

WHILE condition LOOP

statement1;

statement2; END LOOP;

### **FOR LOOP**

**Syntax:**

FOR counter IN [REVERSE]

LowerBound..UpperBound

LOOP statement1;

statement2;END LOOP;

## **EXERCISE-12**

# CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

## PROCEDURES AND FUNCTIONS

### PROCEDURES

#### DEFINITION

A procedure or function is a logically grouped set of SQL and PL/SQL statements that perform a specific task. They are essentially sub-programs. Procedures and functions are made up of,

- Declarative part
- Executable part
- Optional exception handling part

These procedures and functions do not show the errors.

#### KEYWORDS AND THEIR PURPOSES

**REPLACE:** It recreates the procedure if it already exists.

**PROCEDURE:** It is the name of the procedure to be created.

**ARGUMENT:** It is the name of the argument to the procedure. Paranthesis can be omitted if no arguments are present.

**IN:** Specifies that a value for the argument must be specified when calling the procedure ie. used to pass values to a sub-program. This is the default parameter.

**OUT:** Specifies that the procedure passes a value for this argument back to it's calling environment after execution ie. used to return values to a caller of the sub-program.

**INOUT:** Specifies that a value for the argument must be specified when calling the procedure and that procedure passes a value for this argument back to it's calling environment after execution.

**RETURN:** It is the datatype of the function's return value because every function must return a value, this clause is required.

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

### PROCEDURES – SYNTAX

```
create or replace procedure <procedure name> (argument {in,out,inout} datatype ) {is,as}
variable declaration; constant declaration; begin
PL/SQL subprogram body;
exception exception
PL/SQL block; end;
```

### FUNCTIONS – SYNTAX

```
create or replace function <function name> (argument in datatype,.....) return datatype {is,as}
variable declaration; constant declaration; begin
PL/SQL subprogram body;
exception exception
PL/SQL block;
end;
```

### CREATING THE TABLE 'ITITEMS' AND DISPLAYING THE CONTENTS

```
SQL> create table ititems(itemid number(3), actualprice number(5), ordid number(4), prodid
number(4));
Table created.
```

```
SQL> insert into ititems values(101, 2000, 500, 201); 1
row created.
```

```
SQL> insert into ititems values(102, 3000, 1600, 202); 1
row created.
```

```
SQL> insert into ititems values(103, 4000, 600, 202); 1
row created.
```

```
SQL> select * from ititems;
```

ITEMID	ACTUALPRICE	ORDID	PRODID
-----	-----	-----	-----
101	2000	500	201
102	3000	1600	202
103	4000	600	202

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

### PROGRAM FOR GENERAL PROCEDURE – SELECTED RECORD’S PRICE IS INCREMENTED BY 500 , EXECUTING THE PROCEDURE CREATED AND DISPLAYING THE UPDATED TABLE

```
SQL> create procedure itsum(identity number, total number) is price number;
2 null_price exception;
3 begin
4 select actualprice into price from ititems where itemid=identity; 5
   if price is null then
6   raise null_price;
7   else
8   update ititems set actualprice=actualprice+total where itemid=identity;
9   end if;
10 exception
11 when null_price then
12 dbms_output.put_line('price is null');
13 end;
14 /
```

Procedure created.

```
SQL> exec itsum(101, 500);
PL/SQL procedure successfully completed.
```

```
SQL> select * from ititems;
```

ITEMID	ACTUALPRICE	ORDID	PRODID
101	2500	500	201
102	3000	1600	202
103	4000	600	202

### PROCEDURE FOR ‘IN’ PARAMETER – CREATION, EXECUTION

```
SQL> set serveroutput on;
```

```
SQL> create procedure yyy (a IN number) is price number;
```



## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

```
2 begin
3 select actualprice into price from ititems where itemid=a;
4 dbms_output.put_line('Actual price is ' || price);
5 if price is null then
6 dbms_output.put_line('price is null');
7 end if;
8 end;
9 /
```

Procedure created.

SQL> exec yyy(103);

Actual price is 4000

PL/SQL procedure successfully completed.

### **PROCEDURE FOR 'OUT' PARAMETER – CREATION, EXECUTION**

SQL> set serveroutput on;

SQL> create procedure zzz (a in number, b out number) is identity number;

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

```
2 begin
3 select ordid into identity from ititems where itemid=a;
4 if identity<1000 then
5 b:=100;
6 end if;
7 end;
8 /
```

Procedure created.

SQL> declare

```
2 a number;
3 b number;
4 begin
5 zzz(101,b);
6 dbms_output.put_line('The value of b is '|| b);
7 end;
8 /
```

The value of b is 100

PL/SQL procedure successfully completed.

### **PROCEDURE FOR 'INOUT' PARAMETER – CREATION, EXECUTION**

SQL> create procedure itit ( a in out number) is

```
2 begin
3 a:=a+1;
4 end;
5 /
```

Procedure created.

SQL> declare

```
2 a number:=7;
3 begin
4 itit(a);
```

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

```
5 dbms_output.put_line('The updated value is '||a);
6 end;
7 /
```

The updated value is 8

PL/SQL procedure successfully completed.

### **CREATE THE TABLE 'ITTRAIN' TO BE USED FOR FUNCTIONS**

```
SQL>create table ittrain ( tno number(10), tfare number(10));
Table created.
```

```
SQL>insert into ittrain values (1001, 550);
1 row created.
```

```
SQL>insert into ittrain values (1002, 600); 1
row created.
```

```
SQL>select * from ittrain;
```

TNO	TFARE
1001	550
1002	600

### **PROGRAM FOR FUNCTION AND IT'S EXECUTION**

```
SQL> create function aaa (trainnumber number) return number is
2  trainfunction ittrain.tfare % type;
3  begin
4  select tfare into trainfunction from ittrain where tno=trainnumber;
5  return(trainfunction);
6  end;
7  /
```

Function created.

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

```
SQL> set serveroutput on;
```

```
SQL> declare
```

```
2 total number;
```

```
3 begin
```

```
4 total:=aaa (1001);
```

```
5 dbms_output.put_line('Train fare is Rs. '||total);
```

```
6 end;
```

```
7 /
```

Train fare is Rs.550

PL/SQL procedure successfully completed.

### **FACTORIAL OF A NUMBER USING FUNCTION — PROGRAM AND EXECUTION**

```
SQL> create function itfact (a number) return number is
```

```
2 fact number:=1;
```

```
3 b number;
```

```
4 begin
```

```
5 b:=a;
```

```
6 while b>0
```

```
7 loop
```

```
8 fact:=fact*b;
```

```
9 b:=b-1;
```

```
10 end loop;
```

```
11 return(fact);
```

```
12 end;
```

```
13 /
```

Function created.

```
SQL> set serveroutput on;
```

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

```
SQL> declare
2 a number:=7;
3 f number(10);
4 begin
5 f:=itfact(a);
6 dbms_output.put_line('The factorial of the given number is'||f);
7 end;
8 /
```

The factorial of the given number is 5040  
PL/SQL procedure successfully completed.

### **EXERCISE-13**

# CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

## TRIGGER

### DEFINITION

A trigger is a statement that is executed automatically by the system as a side effect of a modification to the database. The parts of a trigger are,

- **Trigger statement:** Specifies the DML statements and fires the trigger body. It also specifies the table to which the trigger is associated.
- **Trigger body or trigger action:** It is a PL/SQL block that is executed when the triggering statement is used.
- **Trigger restriction:** Restrictions on the trigger can be achieved

The different uses of triggers are as follows,

- *To generate data automatically*
- *To enforce complex integrity constraints*
- *To customize complex securing authorizations*
- *To maintain the replicate table*
- *To audit data modifications*

### TYPES OF TRIGGERS

The various types of triggers are as follows,

- **Before:** It fires the trigger before executing the trigger statement.
- **After:** It fires the trigger after executing the trigger statement
- .
- **For each row:** It specifies that the trigger fires once per row
- .
- **For each statement:** This is the default trigger that is invoked. It specifies that the trigger fires once per statement.

### VARIABLES USED IN TRIGGERS

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

- :new
- :old

These two variables retain the new and old values of the column updated in the database. The values in these variables can be used in the database triggers for data manipulation

### **SYNTAX**

```
create or replace trigger triggername [before/after] {DML statements}
on [tablename] [for each row/statement] begin
-----
-----
----- exception
end;
```

### **USER DEFINED ERROR MESSAGE**

The package “raise\_application\_error” is used to issue the user defined error messages

**Syntax:** raise\_application\_error(error number, ‘error message’);

The error number can lie between -20000 and -20999.

The error message should be a character string.

### **TO CREATE THE TABLE ‘ITEMPLS’**

```
SQL> create table itempls (ename varchar2(10), eid number(5), salary number(10));
Table created.
```

```
SQL> insert into itempls values('xxx',11,10000); 1
row created.
```

```
SQL> insert into itempls values('yyy',12,10500); 1
row created.
```

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

```
SQL> insert into itempls values('zzz',13,15500); 1
row created.
```

```
SQL> select * from itempls;
ENAME      EID  SALARY
-----
11  10000 yyy      12
10500
zzz      13  15500
```

### **TO CREATE A SIMPLE TRIGGER THAT DOES NOT ALLOW INSERT UPDATE AND DELETE OPERATIONS ON THE TABLE**

```
SQL> create trigger ittrigg before insert or update or delete on itempls for each row
2 begin
3  raise_application_error(-20010,'You cannot do manipulation');
4 end;
5
6 /
Trigger created.
```

```
SQL> insert into itempls values('aaa',14,34000);
insert into itempls values('aaa',14,34000)
*
ERROR at line 1:
ORA-20010: You cannot do manipulation
ORA-06512: at "STUDENT.ITTRIGG", line 2
ORA-04088: error during execution of trigger 'STUDENT.ITTRIGG'
```

```
SQL> delete from itempls where ename='xxx';
delete from itempls where ename='xxx'
*
ERROR at line 1:
ORA-20010: You cannot do manipulation
ORA-06512: at "STUDENT.ITTRIGG", line 2
```



## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

ORA-04088: error during execution of trigger 'STUDENT.ITTRIGG'

```
SQL> update itempls set eid=15 where ename='yyy';
```

```
update itempls set eid=15 where ename='yyy'
```

\*

ERROR at line 1:

ORA-20010: You cannot do manipulation

ORA-06512: at "STUDENT.ITTRIGG", line 2

ORA-04088: error during execution of trigger 'STUDENT.ITTRIGG'

### **TO DROP THE CREATED TRIGGER**

```
SQL> drop trigger ittrigg;
```

Trigger dropped.

### **TO CREATE A TRIGGER THAT RAISES AN USER DEFINED ERROR MESSAGE AND DOES NOT ALLOW UPDATION AND INSERTION**

```
SQL> create trigger ittriggs before insert or update of salary on itempls for each row
```

```
2 declare
```

```
3  triggsal itempls.salary%type;
```

```
4  begin
```

```
5  select salary into triggsal from itempls where eid=12;
```

```
6  if(:new.salary>triggsal or :new.salary<triggsal) then
```

```
7  raise_application_error(-20100,'Salary has not been changed');
```

```
8  end if;
```

```
9  end;
```

```
10 /
```

Trigger created.

```
SQL> insert into itempls values ('bbb',16,45000);
```

```
insert into itempls values ('bbb',16,45000)
```

\*

ERROR at line 1:

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

ORA-04098: trigger 'STUDENT.ITTRIGGS' is invalid and failed re-validation

```
SQL> update itempls set eid=18 where ename='zzz';
```

```
update itempls set eid=18 where ename='zzz'
```

\*

ERROR at line 1:

ORA-04298: trigger 'STUDENT.ITTRIGGS' is invalid and failed re-validation

Cursor for loop

- Explicit cursor
- Implicit cursor

### **TO CREATE THE TABLE 'SSEMPP'**

```
SQL> create table ssempp( eid number(10), ename varchar2(20), job varchar2(20), sal number(10),dnnumber(5));
```

Table created.

```
SQL> insert into ssempp values(1,'nala','lecturer',34000,11); 1  
row created.
```

```
SQL> insert into ssempp values(2,'kala',' seniorlecturer',20000,12); 1  
row created.
```

```
SQL> insert into ssempp values(5,'ajay','lecturer',30000,11); 1  
row created.
```

```
SQL> insert into ssempp values(6,'vijay','lecturer',18000,11); 1  
row created.
```

```
SQL> insert into ssempp values(3,'nila','professor',60000,12); 1  
row created.
```

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

SQL> select \* from ssemp;

EID	ENAME	JOB	SAL	DNO
1	nala	lecturer	34000	11
2	kala	seniorlecturer	20000	12
5	ajay	lecturer	30000	11
6	vijay	lecturer	18000	11
3	nila	professor	60000	12

### EXTRA PROGRAMS

#### TO WRITE A PL/SQL BLOCK TO DISPLAY THE EMPLOYEE ID AND EMPLOYEE NAME USING CURSOR FOR LOOP

SQL> set serveroutput on;

SQL> declare

2 begin

3 for emy in (select eid,ename from ssemp)

4 loop

5 dbms\_output.put\_line('Employee id and employee name are '|| emy.eid || 'and' || emy.ename);

6 end loop;

7 end;

8 /

Employee id and employee name are 1 and nala

Employee id and employee name are 2 and kala

Employee id and employee name are 5 and ajay

Employee id and employee name are 6 and vijay

Employee id and employee name are 3 and nila

PL/SQL procedure successfully completed.

#### TO WRITE A PL/SQL BLOCK TO UPDATE THE SALARY OF ALL EMPLOYEES WHERE DEPARTMENT NO IS 11 BY 5000 USING CURSOR FOR LOOP AND TO DISPLAY THE UPDATED TABLE

SQL> set serveroutput on;

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

```
SQL> declare
2  cursor cem is select eid,ename,sal,dno from ssemp where dno=11;
3  begin
4  --open cem;
5  for rem in cem
6  loop
7  update ssemp set sal=rem.sal+5000 where eid=rem.eid;

8  end loop;
9  --close cem;
10 end;
11 /
```

PL/SQL procedure successfully completed.

```
SQL> select * from ssemp;
```

	EID	ENAME	JOB	SAL	DNO
	1	nala	lecturer	39000	11
2	kala		seniorlecturer	20000	12
	5	ajay	lecturer	35000	
		11			
	6	vijay	lecturer	23000	
		11	3 nila	professor	
		60000	12		

**TO WRITE A PL/SQL BLOCK TO DISPLAY THE EMPLOYEE ID AND EMPLOYEE NAME WHERE DEPARTMENT NUMBER IS 11 USING EXPLICIT CURSORS**

```
1 declare
2  cursor cenl is select eid,sal from ssemp where dno=11;
3  ecode ssemp.eid%type;
4  esal empp.sal%type;
5  begin
```

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

```
6 open cenl;
7 loop
8 fetch cenl into ecode,esal;
9 exit when cenl%notfound;
10 dbms_output.put_line(' Employee code and employee salary are' || ecode 'and' || esal); 11 end
   loop;
12 close cenl;
13* end;
```

SQL> /

Employee code and employee salary are 1 and 39000

Employee code and employee salary are 5 and 35000

Employee code and employee salary are 6 and 23000

PL/SQL procedure successfully completed.

**TO WRITE A PL/SQL BLOCK TO UPDATE THE SALARY BY 5000 WHERE THE  
JOB IS LECTURER , TO CHECK IF UPDATES ARE MADE USING IMPLICIT  
CURSORS**

**AND TO DISPLAY THE UPDATED TABLE**

SQL> declare

```
2  county number;
3  begin
4  update ssempp set sal=sal+10000 where job='lecturer';
5  county:= sql%rowcount;
6  if county > 0 then
7  dbms_output.put_line('The number of rows are ' || county);
8  end if;
9  if sql %found then
10 dbms_output.put_line('Employee record modification successful');
11 else if sql%notfound then
12 dbms_output.put_line('Employee record is not found');
13 end if;
14 end if;
15 end;
16 /
```

The number of rows are 3

Employee record modification successful

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

PL/SQL procedure successfully completed.

SQL> select \* from ssemp;

	EID	ENAME	JOB	SAL	DNO
	1	nala	lecturer	44000	11
2	kala	seniorlecturer	20000	12	5
ajay		lecturer	40000	11	
	6	vijay	lecturer	28000	11
	3	nila	professor	60000	12

### PROGRAMS

#### TO DISPLAY HELLO MESSAGE

SQL> set serveroutput on;

SQL> declare

2 a varchar2(20);

3 begin

4 a:='Hello';

5 dbms\_output.put\_line(a);

6 end;

7 /

Hello

## **CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY**

PL/SQL procedure successfully completed.

### **TO INPUT A VALUE FROM THE USER AND DISPLAY IT**

```
SQL> set serveroutput on;
SQL> declare
2  a varchar2(20);
3  begin
4  a:=&a;
5  dbms_output.put_line(a);
6  end;
7  /
Enter value for a: 5
old  4: a:=&a; new
4: a:=5;
5
```

PL/SQL procedure successfully completed.

### **GREATEST OF TWO NUMBERS**

```
SQL> set serveroutput on;

SQL> declare
2  a number(7);
3  b number(7);
4  begin
5  a:=&a;
6  b:=&b;
7  if(a>b) then
8  dbms_output.put_line (' The grerater of the two is'|| a);
9  else
10 dbms_output.put_line (' The grerater of the two is'|| b);
```

# CS23332 DATABASE MANAGEMENT SYSTEM

## LABORATORY

```
11 end if;
12 end;
13 /
Enter value for a: 5 old 5:
a:=&a; new 5: a:=5;
Enter value for b: 9 old 6:
b:=&b; new 6: b:=9; The
grerater of the two is9
PL/SQL procedure
successfully completed.
```

### **GREATEST OF THREE NUMBERS**

```
SQL> set serveroutput on;
```

```
SQL> declare
2  a number(7);
3  b number(7);
4  c number(7);
5  begin
6  a:=&a;
7  b:=&b;
8  c:=&c;
9  if(a>b and a>c) then
10 dbms_output.put_line (' The greatest of the three is ' || a);
11 else if (b>c) then
12 dbms_output.put_line (' The greatest of the three is ' || b);
13 else
14 dbms_output.put_line (' The greatest of the three is ' || c);
15 end if;
16 end if;
17 end;
18 /
```



## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

```
Enter value for a: 5
old 6: a:=&a; new
6: a:=5; Enter value
for b: 7 old 7:
b:=&b; new 7:
b:=7; Enter value
for c: 1 old 8:
c:=&c; new 8:
c:=1;
The greatest of the three is 7
```

PL/SQL procedure successfully completed.

### **PRINT NUMBERS FROM 1 TO 5 USING SIMPLE LOOP**

```
SQL> set serveroutput on;
```

```
SQL> declare
2  a number:=1;
3  begin
4  loop
5  dbms_output.put_line (a);
6  a:=a+1;
7  exit when a>5;
8  end loop;
9  end;
10 /
1
2
3
4
5
```

PL/SQL procedure successfully completed.

# CS23332 DATABASE MANAGEMENT SYSTEM

## LABORATORY

### **PRINT NUMBERS FROM 1 TO 4 USING WHILE LOOP**

SQL> set serveroutput on;

SQL> declare

```
2  a number:=1;
3  begin
4  while(a<5)
5  loop
6  dbms_output.put_line (a);
7  a:=a+1;
8  end loop;
9  end;
10 /
1
2
3
4
```

PL/SQL procedure successfully completed.

### **PRINT NUMBERS FROM 1 TO 5 USING FOR LOOP**

SQL> set serveroutput on;

SQL> declare

```
2  a number:=1;
3  begin
4  for a in 1..5
5  loop
6  dbms_output.put_line (a);
7  end loop;
```

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

```
8 end;
```

```
9 /
```

```
1
```

```
2
```

```
3
```

```
4
```

```
5
```

PL/SQL procedure successfully completed.

### **PRINT NUMBERS FROM 1 TO 5 IN REVERSE ORDER USING FOR LOOP**

```
SQL> set serveroutput on;
```

```
SQL> declare
```

```
2 a number:=1;
```

```
3 begin
```

```
4 for a in reverse 1..5
```

```
5 loop
```

```
6 dbms_output.put_line (a);
```

```
7 end loop;
```

```
8 end;
```

```
9 /
```

```
5
```

```
4
```

```
3
```

```
2
```

```
1
```

PL/SQL procedure successfully completed.

### **TO CALCULATE AREA OF CIRCLE**

```
SQL> set serveroutput on;
```

```
SQL> declare
```

```
2 pi constant number(4,2):=3.14;
```

```
3 a number(20);
```

```
4 r number(20);
```

```
5 begin
```

```
6 r:=&r;
```

```
7 a:= pi* power(r,2);
```

```
8 dbms_output.put_line (' The area of circle is ' || a);
```

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

9 end;

10 /

Enter value for r: 2

old 6: r:=&r; new

6: r:=2;

The area of circle is 13

PL/SQL procedure successfully completed.

### **TO CREATE SACCOUNT TABLE**

SQL> create table saccount ( accno number(5), name varchar2(20), bal number(10));

Table created.

SQL> insert into saccount values ( 1,'mala',20000); 1

row created.

SQL> insert into saccount values (2,'kala',30000); 1

row created.

SQL> select \* from saccount;

ACCNO	NAME	BAL
1	mala	20000
2	kala	30000

SQL> set serveroutput on;

SQL> declare

2 a\_bal number(7);

3 a\_no varchar2(20);

4 debit number(7):=2000;

5 minamt number(7):=500;

6 begin

7 a\_no:=&a\_no;

8 select bal into a\_bal from saccount where accno= a\_no;

9 a\_bal:= a\_bal-debit;

10 if (a\_bal > minamt) then

11 update saccount set bal=bal-debit where accno=a\_no;

12 end if;

13 end;

14

15 /

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

Enter value for a\_no: 1 old

7: a\_no:=&a\_no;

new 7: a\_no:=1;

PL/SQL procedure successfully completed.

SQL> select \* from saccount;

ACCNO	NAME	BAL
-------	------	-----

-----

1	mala	18000
---	------	-------

2	kala	30000
---	------	-------

### **TO CREATE TABLE SROUTES**

SQL> create table sroutes ( rno number(5), origin varchar2(20), destination varchar2(20), fare number

r(10), distance number(10));

Table created.

SQL> insert into sroutes values ( 2, 'chennai', 'dindugal', 400,230); 1 row created.

SQL> insert into sroutes values ( 3, 'chennai', 'madurai', 250,300); 1 row created.

SQL> insert into sroutes values ( 6, 'thanjavur', 'palani', 350,370); 1 row created.

SQL> select \* from sroutes;

RNO	ORIGIN	DESTINATION	FARE	DISTANCE
2	chennai	dindugal	400	230
3	chennai	madurai	250	300
6	thanjavur	palani	350	370

SQL> set serveroutput on;

SQL> declare

2 route sroutes.rno % type;

3 fares sroutes.fare % type;

4 dist sroutes.distance % type;

5 begin

6 route:=&route;

7 select fare, distance into fares , dist from sroutes where rno=route;

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

```
8  if (dist < 250) then
9  update sroutes set fare=300 where rno=route;
10 else if dist between 250 and 370 then
11 update sroutes set fare=400 where rno=route;
12 else if (dist > 400) then
13 dbms_output.put_line('Sorry');
14 end if;
15 end if;
16 end if;
17 end;
18 /
```

Enter value for route: 3 old

6: route:=&route;

new 6: route:=3;

PL/SQL procedure successfully completed.

SQL> select \* from sroutes;

RNO	ORIGIN	DESTINATION	FARE	DISTANCE
2	chennai	dindugal	400	230
3	chennai	madurai	400	300
6	thanjavur	palani	350	370

### **TO CREATE SCALCULATE TABLE**

SQL> create table scalculate ( radius number(3), area number(5,2));

Table created.

SQL> desc scalculate;

Name	Null?	Type
RADIUS		NUMBER(3)
AREA		NUMBER(5,2)

SQL> set serveroutput on;

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

```
SQL> declare
2 pi constant
  number(4,2):=3.14;
3 area number(5,2);
4 radius number(3);
5 begin
6 radius:=3;
7 while (radius <=7)
8 loop
9 area:= pi* power(radius,2);
10 insert into scalculate values (radius,area);
11 radius:=radius+1;
12 end loop;
13 end;
14 /
```

PL/SQL procedure successfully completed.

```
SQL> select * from scalculate;
RADIUS    AREA
```

```
-----
3      28.26
4      50.24
5      78.5
6     113.04
7     153.86
```

### **TO CALCULATE FACTORIAL OF A GIVEN NUMBER**

```
SQL> set serveroutput on;
```

```
SQL> declare
```

```
2 f number(4):=1;
3 i number(4);
4 begin
5 i:=&i;
6 while(i>=1)
```

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

```
7  loop
8  f:=f*i;
9  i:=i-1;
10 end loop;
11 dbms_output.put_line('The value is ' || f);
12 end;
13 /
```

Enter value for i: 5

old 5: i:=&i; new

5: i:=5;

The value is 120

PL/SQL procedure successfully completed.



# CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

## EXERCISE-14

### OTHER DATABASE OBJECTS

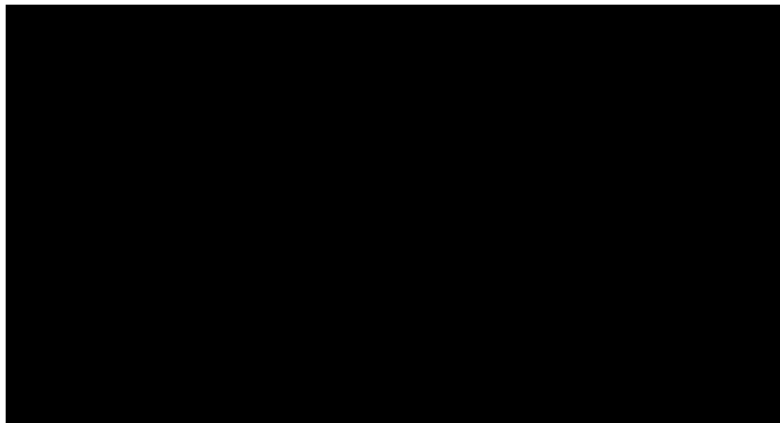
#### Objectives

After the completion of this exercise, the students will be able to do the following:

- Create, maintain, and use sequences
- Create and maintain indexes

#### Database Objects

Many applications require the use of unique numbers as primary key values. You can either build code into the application to handle this requirement or use a sequence to generate unique numbers. If you want to improve the performance of some queries, you should consider creating an index. You can also use indexes to enforce uniqueness on a column or a collection of columns. You can provide alternative names for objects by using synonyms.



#### **What Is a Sequence?**

A sequence:

- Automatically generates unique numbers
- Is a sharable object
- Is typically used to create a primary key value
- Replaces application code
- Speeds up the efficiency of accessing sequence values when cached in memory

# CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

## The CREATE SEQUENCE Statement Syntax

Define a sequence to generate sequential numbers automatically:

```
CREATE SEQUENCE sequence
[INCREMENT BY n]
[START WITH n]
[{{MAXVALUE n | NOMAXVALUE}}]
[{{MINVALUE n | NOMINVALUE}}]
[{{CYCLE | NOCYCLE}}] [{{CACHE
n | NOCACHE}}];
```

### In the syntax:

*sequence* is the name of the sequence generator

INCREMENT BY *n* specifies the interval between sequence numbers where *n* is an integer (If this clause is omitted, the sequence increments by 1.)

START WITH *n* specifies the first sequence number to be generated (If this clause is omitted, the sequence starts with 1.)

MAXVALUE *n* specifies the maximum value the sequence can generate

NOMAXVALUE specifies a maximum value of  $10^{27}$  for an ascending sequence and  $-1$  for a descending sequence (This is the default option.)

MINVALUE *n* specifies the minimum sequence value

NOMINVALUE specifies a minimum value of 1 for an ascending sequence and  $-(10^{26})$  for a descending sequence (This is the default option.)

CYCLE | NOCYCLE specifies whether the sequence continues to generate values after reaching its maximum or minimum value (NOCYCLE is the default option.)

CACHE *n* | NOCACHE specifies how many values the Oracle server preallocates and keep in memory (By default, the Oracle server caches 20 values.)

### Creating a Sequence

- Create a sequence named DEPT\_DEPTID\_SEQ to be used for the primary key of the DEPARTMENTS table.
- Do not use the CYCLE option.

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

### **EXAMPLE:**

```
CREATE SEQUENCE dept_deptid_seq  
INCREMENT BY 10  
START WITH 120  
MAXVALUE 9999  
NOCACHE  
NOCYCLE;
```

### **Confirming Sequences**

- Verify your sequence values in the USER\_SEQUENCES data dictionary table.
- The LAST\_NUMBER column displays the next available sequence number if NOCACHE is specified.

### **EXAMPLE:**

```
SELECT sequence_name, min_value, max_value, increment_by, last_number
```

### **NEXTVAL and CURRVAL Pseudocolumns**

- NEXTVAL returns the next available sequence value. It returns a unique value every time it is referenced, even for different users.
- CURRVAL obtains the current sequence value.
- NEXTVAL must be issued for that sequence before CURRVAL contains a value.

### **Rules for Using NEXTVAL and CURRVAL**

You can use NEXTVAL and CURRVAL in the following contexts:

- The SELECT list of a SELECT statement that is not part of a subquery
- The SELECT list of a subquery in an INSERT statement
- The VALUES clause of an INSERT statement
- The SET clause of an UPDATE statement

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

You cannot use NEXTVAL and CURRVAL in the following contexts:

- The SELECT list of a view
- A SELECT statement with the DISTINCT keyword
- A SELECT statement with GROUP BY, HAVING, or ORDER BY clauses
- A subquery in a SELECT, DELETE, or UPDATE statement
- The DEFAULT expression in a CREATE TABLE or ALTER TABLE statement

### Using a Sequence

- Insert a new department named “Support” in location ID 2500.
- View the current value for the DEPT\_DEPTID\_SEQ sequence.

### EXAMPLE:

```
INSERT INTO departments(department_id, department_name, location_id)
VALUES (dept_deptid_seq.NEXTVAL, 'Support', 2500);
```

```
SELECT dept_deptid_seq.CURRVAL FROM dual;
```

The example inserts a new department in the DEPARTMENTS table. It uses the DEPT\_DEPTID\_SEQ sequence for generating a new department number as follows:

You can view the current value of the sequence:

```
SELECT dept_deptid_seq.CURRVAL FROM dual;
```

### Removing a Sequence

- Remove a sequence from the data dictionary by using the DROP SEQUENCE statement.
- Once removed, the sequence can no longer be referenced.

### EXAMPLE:

```
DROP SEQUENCE dept_deptid_seq;
```

### **What is an Index?**

An index:

- Is a schema object

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

- Is used by the Oracle server to speed up the retrieval of rows by using a pointer
- Can reduce disk I/O by using a rapid path access method to locate data quickly
- Is independent of the table it indexes
- Is used and maintained automatically by the Oracle server

### How Are Indexes Created?

- Automatically: A unique index is created automatically when you define a PRIMARY KEY or UNIQUE constraint in a table definition.
- Manually: Users can create nonunique indexes on columns to speed up access to the rows.

### Types of Indexes

Two types of indexes can be created. One type is a unique index: the Oracle server automatically creates this index when you define a column in a table to have a PRIMARY KEY or a UNIQUE key constraint. The name of the index is the name given to the constraint.

The other type of index is a nonunique index, which a user can create. For example, you can create a

FOREIGN KEY column index for a join in a query to improve retrieval speed.

### Creating an Index

- Create an index on one or more columns.
- Improve the speed of query access to the LAST\_NAME column in the EMPLOYEES table.

```
CREATE INDEX index
ON table (column[, column]...);
```

#### **EXAMPLE:**

```
CREATE INDEX emp_last_name_idx
ON employees(last_name);
```

#### **syntax:**

*index* is the name of the index *table* is the name of the table  
*column* is the name of the column in the table to be indexed

### When to Create an Index

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

You should create an index if:

- A column contains a wide range of values
- A column contains a large number of null values
- One or more columns are frequently used together in a WHERE clause or a join condition
- The table is large and most queries are expected to retrieve less than 2 to 4 percent of the rows

### **When Not to Create an Index**

It is usually not worth creating an index if:

- The table is small
- The columns are not often used as a condition in the query
- Most queries are expected to retrieve more than 2 to 4 percent of the rows in the table
- The table is updated frequently
- The indexed columns are referenced as part of an Expression

### **Confirming Indexes**

- The USER\_INDEXES data dictionary view contains the name of the index and its uniqueness.
- The USER\_IND\_COLUMNS view contains the index name, the table name, and the column name.

### **EXAMPLE:**

```
SELECT ic.index_name, ic.column_name, ic.column_position col_pos, ix.uniqueness
FROM user_indexes ix, user_ind_columns ic
WHERE ic.index_name = ix.index_name
AND ic.table_name = 'EMPLOYEES';
```

### **Removing an Index**

- Remove an index from the data dictionary by using the DROP INDEX command.
- Remove the UPPER\_LAST\_NAME\_IDX index from the data dictionary.
- To drop an index, you must be the owner of the index or have the DROP ANY INDEX privilege.

```
DROP INDEX upper_last_name_idx;
```

```
DROP INDEX index;
```

### **Find the Solution for the following:**

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

1. Create a sequence to be used with the primary key column of the DEPT table. The sequence should start at 200 and have a maximum value of 1000. Have your sequence increment by ten numbers. Name the sequence DEPT\_ID\_SEQ.
2. Write a query in a script to display the following information about your sequences: sequence name, maximum value, increment size, and last number
3. Write a script to insert two rows into the DEPT table. Name your script lab12\_3.sql. Be sure to use the sequence that you created for the ID column. Add two departments named Education and Administration. Confirm your additions. Run the commands in your script.
4. Create a nonunique index on the foreign key column (DEPT\_ID) in the EMP table.
5. Display the indexes and uniqueness that exist in the data dictionary for the EMP table.

### **EXERCISE-15**

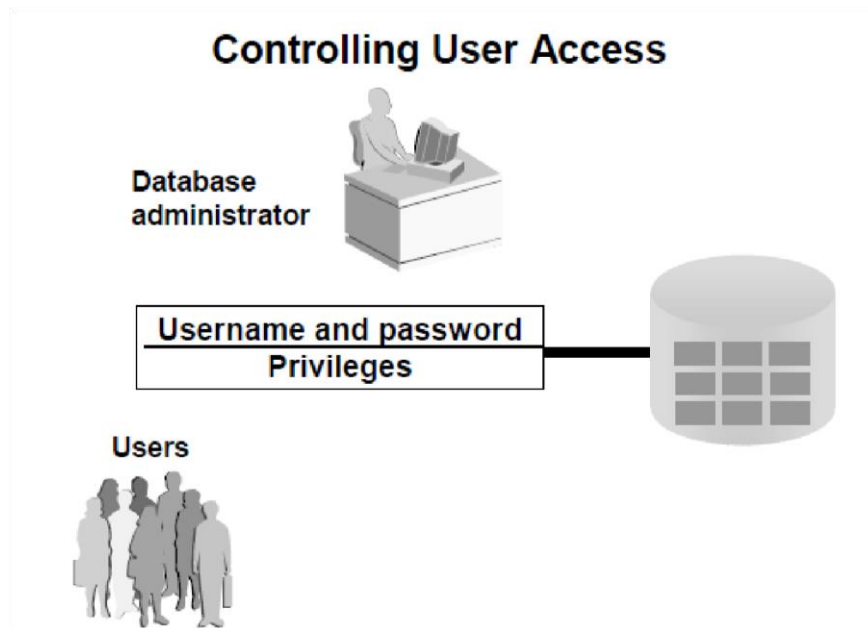
# CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

## Controlling User Access

### Objectives

After the completion of this exercise, the students will be able to do the following:

- 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



### Controlling User Access

In a multiple-user environment, you want to maintain security of the database access and use. With Oracle server database security, you can do the following:

- Control database access
- Give access to specific objects in the database
- Confirm given and received *privileges* with the Oracle data dictionary
- Create synonyms for database objects

### Privileges



# CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

- 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

### **Typical DBA Privileges**

System Privilege	Operations Authorized
CREATE USER	Grantee can create other Oracle users (a privilege required for a DBA role).
DROP USER	Grantee can drop another user.
DROP ANY TABLE	Grantee can drop a table in any schema.
BACKUP ANY TABLE	Grantee can back up any table in any schema with the export utility.
SELECT ANY TABLE	Grantee can query tables, views, or snapshots in any schema.
CREATE ANY TABLE	Grantee can create tables in any schema.

## **Creating Users**

The DBA creates users by using the CREATE USER statement.

### **EXAMPLE:**

```
CREATE USER scott IDENTIFIED BY tiger;
```

## **User System Privileges**

- Once a user is created, the DBA can grant specific system privileges to a user.
- An application developer, for example, may have the following system privileges:
  - CREATE SESSION

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

- CREATE TABLE
- CREATE SEQUENCE
- CREATE VIEW
- CREATE PROCEDURE

GRANT *privilege* [, *privilege*...]

TO *user* [, *user*/ *role*, *PUBLIC*...];

### Typical User Privileges

System Privilege	Operations Authorized
CREATE SESSION	Connect to the database
CREATE TABLE	Create tables in the user's schema
CREATE SEQUENCE	Create a sequence in the user's schema
CREATE VIEW	Create a view in the user's schema
CREATE PROCEDURE	Create a stored procedure, function, or package in the user's schema

### In the syntax:

*privilege* is the system privilege to be granted

*user* |*role*|*PUBLIC* is the name of the user, the name of the role, or *PUBLIC* designates that every user is granted the privilege

**Note:** Current system privileges can be found in the dictionary view *SESSION\_PRIVS*.

### Granting System Privileges

The DBA can grant a user specific system privileges.

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

### What is a Role?

A role is a named group of related privileges that can be granted to the user. This method makes it easier to revoke and maintain privileges.

A user can have access to several roles, and several users can be assigned the same role. Roles are typically created for a database application.

# CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

## Creating and Assigning a Role

First, the DBA must create the role. Then the DBA can assign privileges to the role and users to the role.

### Syntax

CREATE ROLE *role*;

In the syntax: *role* is the name of the role to be created

Now that the role is created, the DBA can use the GRANT statement to assign users to the role as well as assign privileges to the role.

## Creating and Granting Privileges to a Role

CREATE ROLE manager;

Role created.

GRANT create table, create view TO manager;

Grant succeeded.

GRANT manager TO DEHAAN, KOCHHAR;

Grant succeeded.

- Create a role
- Grant privileges to a role
- Grant a role to users

## 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;

User altered.

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

### Object Privileges

Object Privilege	Table	View	Sequence	Procedure
ALTER	√		√	
DELETE	√	√		
EXECUTE				√
INDEX	√			
INSERT	√	√		
REFERENCES	√	√		
SELECT	√	√	√	
UPDATE	√	√		

### Object Privileges

- 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.

GRANT *object\_priv* [(*columns*)]

ON *object*

TO {*user*|*role*|PUBLIC}

[WITH GRANT OPTION];

#### **In the syntax:**

*object\_priv* is an object privilege to be granted

ALL specifies all object privileges

*columns* specifies the column from a table or view on which privileges are granted

ON *object* is the object on which the privileges are granted

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

TO identifies to whom the privilege is granted

PUBLIC grants object privileges to all users

WITH GRANT OPTION allows the grantee to grant the object privileges to other users and roles

### **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 scott
WITH GRANT OPTION;
```

```
.
GRANT select
ON alice.departments
TO PUBLIC;
```

### **How to Revoke Object Privileges**

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

- 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];

### **In the syntax:**

CASCADE is required to remove any referential integrity constraints made to the CONSTRAINTS object by means of the REFERENCES privilege

### **Revoking Object Privileges**

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

REVOKE select, insert

ON departments

FROM scott;

### **Find the Solution for the following:**

1. What privilege should a user be given to log on to the Oracle Server? Is this a system or an object privilege?

---

2. What privilege should a user be given to create tables?

---

3. If you create a table, who can pass along privileges to other users on your table?

---

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

4. You are the DBA. You are creating many users who require the same system privileges. What should you use to make your job easier?

---

5. What command do you use to change your password?

---

- Grant another user access to your DEPARTMENTS table. Have the user grant you query access to his or her DEPARTMENTS table.

7. Query all the rows in your DEPARTMENTS table.

8. Add a new row to your DEPARTMENTS table. Team 1 should add Education as department number 500. Team 2 should add Human Resources department number 510. Query the other team's table.

9. Query the USER\_TABLES data dictionary to see information about the tables that you own.

10. Revoke the SELECT privilege on your table from the other team.

11. Remove the row you inserted into the DEPARTMENTS table in step 8 and save the changes.

-----END-----

**[ ANSWERS FOR ABOVE PROBLEMS – STUDENTS WILL NOT TAKE PRINTOUT OF THE BELOW PAGES ]**

## EXNO: CREATING AND MANAGING TABLES

**AIM:**

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

To create tables, to describe the data types that can be used when specifying column definition, to alter table definitions, to drop, rename, and truncate tables.

### SAMPLE TABLES:

#### EMPLOYEE TABLE:

```
SQL> create table employeetable (employee_id number(6) NOT NULL,first_name
varchar2(20),last_name varchar2(25) NOT NULL,email varchar2(25) NOT
NULL,phone_number varchar2(20),hire_date date NOT NULL,job_id varchar2(10) NOT
NULL,salary number(8,2),commision number(2,2),manager_id number(6),department_id
number(4));
```

Table created.

```
SQL> desc employeetable;
```

Name	Null?	Type
EMPLOYEE_ID	NOT NULL	NUMBER(6)
FIRST_NAME		VARCHAR2(20)
LAST_NAME	NOT NULL	VARCHAR2(25)
EMAIL	NOT NULL	VARCHAR2(25)
PHONE_NUMBER		VARCHAR2(25)
HIRE_DATE	NOT NULL	DATE
JOB_ID	NOT NULL	VARCHAR2(10)
SALARY		NUMBER(8,2)
COMMISSION_PCT		NUMBER(2,2)
MANAGER_ID		NUMBER(6)
DEPARTMENT_ID		NUMBER(4)



## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

SQL>select \* from employeetable;

EMPLOYEE\_ID FIRST\_NAME LAST\_NAME

EMAIL PHONE\_NUMBER HIRE\_DATE

JOB\_ID SALARY COMMISSION\_PCT MANAGER\_ID DEPARTMENT\_ID

100	stevan	king		
sking	515.123.4567	17-JUN-87	ad_pres	24000
90	101 neena	kochhar	nkochhar	
515.123.4568	21-SEP-89	ad_vp	17000	100
90				

EMPLOYEE\_ID FIRST\_NAME LAST\_NAME

EMAIL PHONE\_NUMBER HIRE\_DATE

JOB\_ID SALARY COMMISSION\_PCT MANAGER\_ID DEPARTMENT\_ID

102	lex	dehaan		
ldehaan	515.123.4569	13-JAN-93	ad_vp	17000
100	90			
103	alexandar	hunold	ahunold	590.423.4567 03-JAN-93

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

EMPLOYEE\_ID FIRST\_NAME LAST\_NAME

-----

EMAIL PHONE\_NUMBER HIRE\_DATE

-----

JOB\_ID SALARY COMMISSION\_PCT MANAGER\_ID DEPARTMENT\_ID

-----

it\_prog 9000 102 60 104

bruce ernst bernst

590.423.4568 21-MAY-91 it\_prog

6000 103 60

107 diana lorentz

EMPLOYEE\_ID FIRST\_NAME LAST\_NAME

-----

EMAIL PHONE\_NUMBER HIRE\_DATE

-----

JOB\_ID SALARY COMMISSION\_PCT MANAGER\_ID DEPARTMENT\_ID

-----

dlorentz 590.423.4568 07-FEB-99

it\_prog 4200 103 60 124

kevin mourgas kmourgas

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

650.123.5234          16-NOV-99          st\_man

5800                  100          50

EMPLOYEE\_ID FIRST\_NAME          LAST\_NAME

EMAIL                  PHONE\_NUMBER          HIRE\_DATE

JOB\_ID                  SALARY COMMISSION\_PCT MANAGER\_ID DEPARTMENT\_ID          -  
-----  
trajs                  650.121.8009          17-OCT-95          141 tenna          rajs  
124          50                  st\_clerk          3500

142      curtis          davies                  cdavies  
650.121.2994          29-JAN-97          st\_clerk          3100  
124          50

EMPLOYEE\_ID FIRST\_NAME          LAST\_NAME

EMAIL                  PHONE\_NUMBER          HIRE\_DATE

JOB\_ID                  SALARY COMMISSION\_PCT MANAGER\_ID DEPARTMENT\_ID

-----  
143      randall          matos                  rmatos  
650.121.2874          15-MAR-98          st\_clerk          2600  
124          50                  144 peter          vargas  
pvargas          650.121.2004          09-JUL-98

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

EMPLOYEE\_ID FIRST\_NAME LAST\_NAME

-----

EMAIL PHONE\_NUMBER HIRE\_DATE

-----

JOB\_ID SALARY COMMISSION\_PCT MANAGER\_ID DEPARTMENT\_ID

----- st\_clerk

2500 124 50

149 eleni zlotkey

ezlotkey 011.44.1344.429018 29-JAN-00 sa\_man

10500 .2 100 80

174 ellen abel

EMPLOYEE\_ID FIRST\_NAME LAST\_NAME

-----

EMAIL PHONE\_NUMBER HIRE\_DATE

-----

JOB\_ID SALARY COMMISSION\_PCT MANAGER\_ID DEPARTMENT\_ID

----- eabel

011.44.1644.429267 11-MAY-96 sa\_rep

11000 .3 149 80

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

176	jonathan	taylor			jtaylor
011.44.1644.429265		24-MAR-98			sa_rep
8600	.2	149	80		

EMPLOYEE_ID	FIRST_NAME	LAST_NAME
-------------	------------	-----------

EMAIL	PHONE_NUMBER	HIRE_DATE
-------	--------------	-----------

JOB_ID	SALARY	COMMISSION_PCT	MANAGER_ID	DEPARTMENT_ID
--------	--------	----------------	------------	---------------

178	kimberly	grant		
kgrant	011.44.1644.429263	24-MAY-99		
sa_rep	7000	.15	149	

200	jennifer]	whalen		jwhalen
515.123.4444		17-SEP-87		ad_asst
4400	101	10		

EMPLOYEE_ID	FIRST_NAME	LAST_NAME
-------------	------------	-----------

EMAIL	PHONE_NUMBER	HIRE_DATE
-------	--------------	-----------

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

JOB\_ID                      SALARY COMMISSION\_PCT MANAGER\_ID DEPARTMENT\_ID

----- -

201	micheal	hartstein		
mhartstein	515.123.5555	17-FEB-96		
mk_man	13000	100	20	

202	pat	fay		pfay
606.123.6666	17-AUG-97			

EMPLOYEE\_ID FIRST\_NAME              LAST\_NAME

-----

EMAIL                      PHONE\_NUMBER              HIRE\_DATE

-----

JOB\_ID    SALARY COMMISSION\_PCT MANAGER\_ID DEPARTMENT\_ID

----- -

			mk_rep	6000	
201	20	205 sheliey	higgins		shiggins
515.123.8080	07-JUN-94		ac_mgr	12000	101
110					

EMPLOYEE\_ID FIRST\_NAME              LAST\_NAME

-----

EMAIL                      PHONE\_NUMBER              HIRE\_DATE

-----

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

JOB\_ID                      SALARY COMMISSION\_PCT MANAGER\_ID DEPARTMENT\_ID

----- -                      wgeitz

515.123.8181              07-JUN-94                      ac\_account

8300                      205              110

20 rows selected.

### DEPARTMENT TABLE:

SQL> create table department(department\_id number(6) NOT NULL,dept\_name varchar2(20)  
NOT NULL,manager\_id number(6),location\_id number(4));

Table created.

SQL> desc department

Name	Null?	Type
DEPARTMENT_ID	NOT NULL	NUMBER(6)
DEPT_NAME	NOT NULL	VARCHAR2(20)
MANAGER_ID		NUMBER(6)
LOCATION_ID		NUMBER(4)

SQL> select \* from department;

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
---------------	-----------------	------------	-------------

10	administration	200	1700
20	marketing	201	1800
50	shipping	124	1500
60	it	103	1400
80	sales	149	2500

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

90 executive	100	1700
110 accounting	205	1700
190 contracting	0	1700

8 rows selected.

### JOB\_GRADE TABLE:

```
SQL> create table job_grade(grade_level varchar2(2),lowest_sal number,highest_sal number);
```

Table created.

```
SQL> desc job_grade
```

Name	Null?	Type
GRADE_LEVEL		VARCHAR2(2)
LOWEST_SAL		NUMBER
HIGHEST_SAL		NUMBER

```
SQL> select * from locations;
```

LOCATION_ID	STREET_ADDRESS	POSTAL_CODE
1400	2014 jabberwocky rd	26192
	southlake	
	texas	us

1500	2011 interior blvd	99236	south
	san francisco	california	us



## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

1700 2004 charade rd	98199	seattle
washington	us	

LOCATION_ID	STREET_ADDRESS	POSTAL_CODE
-------------	----------------	-------------

-----		
CITY	STATE_PROVINCE	CO
-----	-----	---

1800 460 bloor st	on m5s 1xb	torondo
ontario	ca	

2500 magdalen centre,the oxford science park	on99zb	
oxford	oxford	uk

### LOCATION TABLE:

```
SQL> create table location(location_id number(4) not null,street_address
varchar2(40),postal_code varchar2(12),city varchar2(30) not null,state_province
varchar2(25),country_id char(2));
```

Table created.

```
SQL> desc location;
```

Name	Null?	Type
-----	-----	-----

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

LOCATION_ID	NOT NULL NUMBER(4)
STREET_ADDRESS	VARCHAR2(40)
POSTAL_CODE	VARCHAR2(12)
CITY	NOT NULL VARCHAR2(30)
STATE_PROVINCE	VARCHAR2(25)
COUNTRY_ID	CHAR(2)

SQL> select \* from job\_grades;

GRA LOWEST\_SAL HIGHEST\_SAL

---		a
1000	2999	b
3000	5999	c
6000	9999	d
10000	14999	e
15000	24999	f
25000	40000	

6 rows selected.

SQL> select \* from job\_history;

EMPLOYEE\_ID START\_DAT END\_DATE JOB\_ID DEPARTMENT\_ID

-----				
103	13-JAN-93	24-JUL-98	it_prog	60
101	21-SEP-89	27-OCT-93	ac_account	110
101	28-OCT-89	15-MAR-97	ac_mgr	110
201	17-FEB-96	19-DEC-99	mk_rep	20

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

114	24-MAR-98	31-DEC-99	mk_rep	50
122	01-JAN-99	31-DEC-98	st_clerk	50
200	17-SEP-87	17-JUN-93	ad_asst	90
176	24-MAR-98	31-DEC-98	sa_rep	80
176	01-JAN-99	31-DEC-99	sa_man	80
200	01-JUL-94	31-DEC-98	ac_account	90

10 rows selected.

SQL> spool off;

### SOLUTION FOR THE EXERCISES

9. 1. Create the DEPT table based on the following table instance chart. Confirm that the table is created.

Column name	ID	NAME
Key Type		
Nulls/Unique		
FK table		
FK column		
Data Type	Number	Varchar2
Length	7	25

```
create table dept(id number(7),name varchar(25));
```

Table created.

SQL> desc dept;

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

Name	Null?	Type
-----		
ID		NUMBER(7)
NAME		VARCHAR2(25)

10. Populate the DEPT table with data from the DEPARTMENT table. Include only the columns that are needed.

```
SQL> alter table dept add(manager_id number(10),location_id number(10));
```

Table altered.

```
SQL> desc dept;
```

Name	Null?	Type
-----		
ID		NUMBER(7)
NAME		VARCHAR2(25)
MANAGER_ID		NUMBER(10)
LOCATION_ID		NUMBER(10)

11. Create the EMP table based on the following instance chart. Confirm that the table is created.

Column name	ID	LAST_NAME	FIRST_NAME	DEPT_ID
Key Type				
Nulls/Unique				
FK table				
FK column				

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

Data Type	Number	Varchar2	Varchar2	Number
Length	7	25	25	7

```
SQL> create table empdb(id number(7),firstname varchar(25),lastname varchar(25),dept_id
number(25));
```

Table created.

```
SQL> desc empdb
```

Name	Null?	Type
-----		
ID		NUMBER(7)
FIRSTNAME		VARCHAR2(25)
LASTNAME		VARCHAR2(25)
DEPT_ID		NUMBER(25)

12. Modify the EMP table to allow for longer employee last names. Confirm the modification.(Hint: Increase the size to 50)

```
SQL> alter table empdb modify(lastname varchar(50));
```

Table altered.

```
SQL> desc empdb;
```

Name	Null?	Type
-----		
ID		NUMBER(7)
FIRSTNAME		VARCHAR2(25)

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

LASTNAME                                      VARCHAR2(50)

DEPT\_ID                                        NUMBER(25)

13.      Create the EMPLOYEES2 table based on the structure of EMPLOYEES table. Include Only the Employee\_id, First\_name, Last\_name, Salary and Dept\_id coloumns. Name the columns Id, First\_name, Last\_name, salary and Dept\_id respectively.

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

```
create table employee2(id number(10),firstname varchar(10),lastname varchar(10),salary
number(8,2),deptid number(10));
```

Table created.

```
SQL> desc employee2;
```

Name	Null?	Type
-----		
ID		NUMBER(10)
FIRSTNAME		VARCHAR2(10)
LASTNAME		VARCHAR2(10)
SALARY		NUMBER(8,2)
DEPTID		NUMBER(10)

14. Drop the EMP table. SQL> drop table empdb;

Table dropped.

15. Rename the EMPLOYEES2 table as EMP.

```
SQL> rename employee2 to employee3;
```

Table renamed.

16. Add a comment on DEPT and EMP tables.

Confirm the modification by describing the table.

```
SQL> comment on table dept is'this is department database..';
```

Comment created.

```
SQL> select * from user_tab_comments;
```

TABLE_NAME	TABLE_TYPE	COMMENTS
-----		

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

SQL>

-----  
STUDENT                      TABLE

EMPLOYEE                    TABLE

EMPLOYEE\_DETAIL            TABLE

TABLE\_NAME                TABLE\_TYPE

-----    COMMENTS

-----EMPLOYEES                      TABLE

EMP                      TABLE

this is a employee database..

BIN\$JYmxGC1pS7KBH/ciI85irw==\$0 TABLE

TABLE\_NAME                TABLE\_TYPE

-----    COMMENTS

-----  
DEPT                      TABLE

this is department database..



## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

EMPLOYEE3                      TABLE

rows selected.

17.     Drop the First\_name column from the EMP table and confirm it. SQL>  
alter table employee3 drop column firstname;

Table altered.

desc employee3;

Name	Null?	Type
-----		
ID		NUMBER(10)
LASTNAME		VARCHAR2(10)
SALARY		NUMBER(8,2)
DEPTID		NUMBER(10)

18.     In the DEPT table mark the column Dept\_id as UNUSED and confirm it. SQL> alter table employee3 set unused(deptid);

Table altered.

SQL> desc employee3;

Name	Null?	Type
-----		
ID		NUMBER(10)
LASTNAME		VARCHAR2(10)
SALARY		NUMBER(8,2)

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

SQL>

11.Drop all the UNUSED columns from the EMP table and Confirm the modification

SQL> drop table employee3;

Table dropped.

### EXNO:2 MANIPULATING DATA

#### AIM:

To describe each DML statement, to insert rows into tables, to update rows into tables, to delete rows from tables, to execute control transactions,

#### SOLUTION FOR THE EXERCISES

10. Create MY\_EMPLOYEE table with the following structure

NAME	NULL?	TYPE
ID	Not null	Number(4)
Last_name		Varchar(25)
First_name		Varchar(25)
Userid		Varchar(25)
Salary		Number(9,2)

```
create table emp(id number(4) not null, firstname varchar(10),lastname varchar(10),userid
number(10), salary number(9,2));
```

Table created.

SQL> desc emp;

Name	Null?	Type
-----		
ID	NOT NULL	NUMBER(4)
FIRSTNAME		VARCHAR2(10)

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

LASTNAME                                      VARCHAR2(10)

USERID                                        NUMBER(10)

SALARY                                        NUMBER(9,2)

11.      Add the first and second rows data to MY\_EMPLOYEE table from the following sample data.

ID	Last_name	First_name	Userid	salary
1	Patel	Ralph	rpatel	895
2	Dancs	Betty	bdancs	860
3	Biri	Ben	bbiri	1100
4	Newman	Chad	Cnewman	750
5	Ropebur	Audrey	aropebur	1550

```
SQL> insert into my_employee values(1,'patel','ralph','rpatel',895);
```

1 row created.

```
insert into my_employee values(2,'dancs','betty','bdancs',860);
```

1 row created.

```
SQL> select * from my_employe ID LAST_NAME FIRST_NAME
```

```
-----
USER_ID          SALARY
-----
1 patel          ralph
rpatel          895
2 dancs          betty          bdancs          860
```

12.      Display the table with values.

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

SQL>

SQL> select \* from my\_employee;

ID	LAST_NAME	FIRST_NAME	USER_ID	SALARY
1	patel	ralph	rpatel	895
2	dancs	betty	bdancs	860

13. Populate the next two rows of data from the sample data. Concatenate the first letter of the first\_name with the first seven characters of the last\_name to produce Userid.

SQL> insert into my\_employee  
values(&id,&last\_name,&first\_name,lower(substr('&first\_name',1,1)||substr('&last\_name',1,7))  
,&salary);

Enter value for id: 3

Enter value for last\_name: biri

Enter value for first\_name: ben

Enter value for first\_name: ben

Enter value for last\_name: biri Enter

value for salary: 1100

old 1: insert into my\_employee  
values(&id,&last\_name,&first\_name,lower(substr('&first\_name',1,1)||substr('&last\_name',1,7))  
,&salary)

new 1: insert into my\_employee  
values(3,'biri','ben',lower(substr('ben',1,1)||substr('biri',1,7)),1100)

1 row created.

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

SQL> /

Enter value for id: 4

Enter value for last\_name: newman

Enter value for first\_name: chad

Enter value for first\_name: chad

Enter value for last\_name: newman Enter

value for salary: 750

old 1: insert into my\_employee  
values(&id,&last\_name,&first\_name,lower(substr('&first\_name',1,1)||substr('&last\_name',1,7))  
,&salary)

new 1: insert into my\_employee  
values(4,'newman','chad',lower(substr('chad',1,1)||substr('newman',1,7)),750)

1 row created.

SQL> select \* from my\_employee;

ID	LAST_NAME	FIRST_NAME	USER_ID	SALARY
----	-----------	------------	---------	--------

-----				
1	patel			

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

2	dancs	betty	bdancs	860	3	biri	ben	bbiri
	1100							

LAST_NAME	FIRST_NAME	USER_ID	SALARY
-----------	------------	---------	--------

-----				4
newman	chad	cnewman	750	

14. Make the data additions permanent.  
SQL> commit;

Commit complete.

Change the last name of employee 3 to Drexler

SQL> update my\_employee set last\_name='drexler'where id=3;

1 row updated.

SQL> select \* from my\_employee;

ID	LAST_NAME	FIRST_NAME	USER_ID	SALARY
----	-----------	------------	---------	--------

-----				
1	patel	ralph	rpatel	895
2	dancs	betty	bdancs	860
3	drexler	ben	bbiri	1100

LAST_NAME	FIRST_NAME	USER_ID	SALARY
-----------	------------	---------	--------

-----				
4	newman	chad	cnewman	750

7.Change the salary to 1000 for all the employees with a salary less than 900.

Update my\_employee set salary=1000 where salary<900;

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

SQL> select \* from my\_employee;

ID	LAST_NAME	FIRST_NAME	USER_ID	SALARY
1	patel	ralph	895	
3	drexler	ben	bbiri	1000
4	newman	chad	cnewman	750

8. Verify the changes to the table

SQL> select \* from my\_employee;

ID	LAST_NAME	FIRST_NAME	USER_ID	SALARY
1	patel	ralph	895	
3	drexler	ben	bbiri	1100
4	newman	chad	cnewman	750

ID	LAST_NAME	FIRST_NAME	USER_ID	SALARY
5	ropebur	audrey	aropebur	1550

9. Delete Betty dancs from MY \_EMPLOYEE table. SQL>

delete from my\_employee where last\_name='dancs';

1 row deleted.

10. Confirm the changes.

SQL> select \* from my\_employee;

ID	LAST_NAME	FIRST_NAME	USER_ID	SALARY
----	-----------	------------	---------	--------

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

-----

1	patel	ralph	rpatel	895	
3	drexler	ben	bbiri	1100	
4	newman	chad	cnewman	750	Commit the changes.

Commit;

12) Populate the last row of sample data as by Q:4

```
SQL> insert into my_employee
values(&id,&last_name,&first_name,lower(substr('&first_name',1,1)||
substr('&last_name',1,7)),&salary);
```

Enter value for id: 5

Enter value for last\_name: Ropebur

Enter value for first\_name: Audrey

Enter value for first\_name: Audrey

Enter value for last\_name: Ropebur Enter

value for salary: 1550

```
old 1: insert into my_employee
values(&id,&last_name,&first_name,lower(substr('&first_name',1,
```

```
new 1: insert into my_employee
values(5,'Ropebur','Audrey',lower(substr('Audrey',1,1)||substr('Ropebur',1,7)),
```

1 row created.

13) Confirm the table.

```
SQL> select * from my_employee;
```

ID	LAST_NAME	FIRST_NAME	USERID	SALARY
----	-----------	------------	--------	--------



## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

-----

1 Patel        Ralph     rpatel       1000

3 drexler     Ben        bbiri        1100

4 Newman     Chad       cnewman     1000

5 Ropebur     Audrey   aropebur     1550

14) Mark an intermediate point in the processing of the transaction.

SQL> savepoint q14;

Savepoint created.

15) Empty the fourth row of the emp table

SQL> truncate table my\_employee;

Table truncated.

16) Confirm the table is empty. SQL> select \* from my\_employee; no rows selected

17) Discard the most recent DELETE operation without discarding the earlier INSERT operation.

Rollback Q14;

SQL> rollback;

18) Confirm that the new row is still intact.

SQL> select \* from my\_employee;

ID	LAST_NAME	FIRST_NAME	USERID	SALARY
----	-----------	------------	--------	--------

-----

1	Patel	Ralph	rpatel	1000
---	-------	-------	--------	------

3	drexler	Ben	bbiri	1100
---	---------	-----	-------	------

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

4 Newman	Chad	cnewman	1000
5 Ropebur	Audrey	aropebur	1550

### EXNO:3 INCLUDING CONSTRAINTS

#### AIM:

To describe the constraints, and to create and maintain constraints

#### SOLUTION FOR THE EXERCISES

5. Add a table-level PRIMARY KEY constraint to the EMP table on the ID column. The constraint should be named at creation. Name the constraint my\_emp\_id\_pk. SQL> alter table dept add constraint my\_dept\_id\_pk primary key(id);

Table altered.

6. Create a PRIMARY KEY constraint to the DEPT table using the ID column. The constraint should be named at creation. Name the constraint my\_dept\_id\_pk. Alter table dept add constraint my\_dept\_id\_pk primary key(id);

SQL> alter table emp add(dept\_id number(7));

Table altered.

7. Add a column DEPT\_ID to the EMP table. Add a foreign key reference on the EMP table that ensures that the employee is not assigned to nonexistent department. Name the constraint my\_emp\_dept\_id\_fk.

SQL> alter table emp add constraint my\_emp\_dept\_id\_fk foreign key(dept\_id) references dept(id);

Table altered.

8. Modify the EMP table. Add a COMMISSION column of NUMBER data type, precision 2, scale 2. Add a constraint to the commission column that ensures that a commission value is greater than zero.

SQL> alter table emp add commission float constraints ck check(commission>=0);

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

Table altered.

5. Confirm that the constraints were added by querying the USER\_CONSTRAINTS view.

```
SQL> select constraint_name,constraint_type from user_constraints where table_name  
in('emp','dept');
```

No rows selected.

### EXNO:4 WRITING BASIC SQL SELECT STATEMENTS

#### AIM:

To list the capabilities of SQL SELECT statement and to execute a basic SELECT statement.

#### SOLUTIONS FOR THE EXERCISES:

1. Show the structure of departments the table. Select all the data from it.

```
SQL> desc department;
```

Name	Null?	Type
DEPARTMENT_ID	NOT NULL	NUMBER(4)
DEPARTMENT_NAME		VARCHAR2(30)
MANAGER_ID		NUMBER(6)
LOCATION_ID		NUMBER(4)

```
SQL> select * from department;
```

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
10	administration	200	1700
20	marketing	201	1800
50	shipping	124	1500

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

60 it	103	1400
80 sales	149	2500
90 executive	100	1700
110 accounting	205	1700
190 contracting	0	1700

8 rows selected.

2. Show the structure of the employees table. Create a query to display the last name, job code, hire date, and employee number for each employee, with employee number appearing first

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
10	admin	200	1700
20	Marketing	201	1800
50	shipping	124	1500
60	IT	103	1400
80	sales	149	2500
90	executive	100	1700
110	accounting	205	1700
190	contracting		1700
70	public relations	100	1700
70	public relations		

3. Provide an alias STARTDATE for the hire date

SQL> select employee\_id ,last\_name,job\_id,hire\_date start\_date from employeeetable;

EMPLOYEE_ID	LAST_NAME	JOB_ID	START_DAT
17-JUN-87	101 kochhar	ad_vp	21-SEP-89
102 dehaan	ad_vp	13-JAN-93	103 hunold
JAN-93	104 ernst	it_prog	21-MAY-91
07-FEB-99			107 lorentz

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

124 mourgas 17-OCT-95	st_man	16-NOV-99	141 rajs	st_clerk
142 davies	st_clerk	29-JAN-97		
143 matos 09-JUL-98	st_clerk	15-MAR-98	144 vargas	st_clerk

EMPLOYEE_ID	LAST_NAME	JOB_ID	START_DAT		
-----	-----		149 zlotkey	sa_man	
29-JAN-00	174 abel	sa_rep	11-MAY-96		
176 taylor	sa_rep	24-MAR-98			
178 grant SEP-87	sa_rep	24-MAY-99	200 whalen	ad_asst	17-
201 hartstein 17-AUG-97	mk_man	17-FEB-96	202 fay	mk_rep	
205 higgins	ac_mgr	07-JUN-94			
206 gietz	ac_account	07-JUN-94			

20 rows selected.

4.Create a query to display unique job codes from the employee table.

SQL> select distinct job\_id from employeetable;

JOB_ID	-----
-----	sa_rep
mk_rep	
ad_pres	ad_vp

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

it\_prog

st\_man

ac\_account

sa\_man

ad\_asst

ac\_mgr

st\_clerk

JOB\_ID

-----

-----

mk\_man

12 rows selected.

5. Display the last name concatenated with the job ID , separated by a comma and space, and name the column EMPLOYEE and TITLE.

SQL> select last\_name||','||job\_id"employee and title" from employeetable;

employee and title

-----

king,ad\_pres

kochhar,ad\_vp

dehaan,ad\_vp

hunold,it\_prog

ernst,it\_prog

lorentz,it\_prog

mourgas,st\_man

rajs,st\_clerk

davies,st\_clerk

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

matos,st\_clerk

vargas,st\_clerk

employee and title

--

-----  
zlotkey,sa\_man

abel,sa\_rep

taylor,sa\_rep

grant,sa\_rep

whalen,ad\_asst

hartstein,mk\_man

fay,mk\_rep

higgins,ac\_mgr

gietz,ac\_account

20 rows selected.

6.Create a query to display all the data from the employees table. Separate each column by a comma. Name the column THE\_OUTPUT.

SQL> select

employee\_id||','||first\_name||','||last\_name||','||email||','||phone\_number||','||job\_id||','||manager\_id||','||  
hire\_date||','||salary||','||commission\_pct||','||department\_id the\_output from employeeetable;

THE\_OUTPUT

-----100,stevan,king,sking,515.123.4567 ,ad\_pres,, 1  
7-JUN-87,24000,,90

101,neena,kochhar,nkochhar,515.123.4568,ad\_vp,100,21-SEP-89,17000,,90

102,lex,dehaan,ldehaan,515.123.4569,ad\_vp,100,13-JAN-93,17000,,90

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

103,alexandar,hunold,ahunold,590.423.4567,it\_prog,102,03-JAN-93,9000,,60  
104,bruce,ernst,bernst,590.423.4568,it\_prog,103,21-MAY-91,6000,,60  
107,diana,lorentz,dlorentz,590.423.4568,it\_prog,103,07-FEB-99,4200,,60  
124,kevin,mourgas,kmourgas,650.123.5234,st\_man,100,16-NOV-99,5800,,50  
141,trenna,rajs,trajs,650.121.8009,st\_clerk,124,17-OCT-95,3500,,50  
142,curtis,davies,cdavies,650.121.2994,st\_clerk,124,29-JAN-97,3100,,50  
143,randall,matos,rmatos,650.121.2874,st\_clerk,124,15-MAR-98,2600,,50  
144,peter,vargas,pvargas,650.121.2004,st\_clerk,124,09-JUL-98,2500,,50

### THE\_OUTPUT

-----  
149,eleni,zlotkey,ezlotkey,011.44.1344.429018,sa\_man,100,29-JAN-00,10500,,2,80  
174,ellen,abel,eabel,011.44.1644.429267,sa\_rep,149,11-MAY-96,11000,,3,80  
176,jonathan,taylor,jtaylor,011.44.1644.429265,sa\_rep,149,24-MAR-98,8600,,2,80  
178,kimberly,grant,kgrant,011.44.1644.429263,sa\_rep,149,24-MAY-99,7000,,15,  
200,jennifer],whalen,jwhalen,515.123.4444,ad\_asst,101,17-SEP-87,4400,,10  
201,micheal,hartstein,mhartstein,515.123.5555,mk\_man,100,17-FEB-96,13000,,20  
202,pat,fay,pfay,606.123.6666,mk\_rep,201,17-AUG-97,6000,,20  
205,sheliey,higgins,shiggins,515.123.8080,ac\_mgr,101,07-JUN-94,12000,,110  
206,william,gietz,wgietz,515.123.8181,ac\_account,205,07-JUN-94,8300,,110

20 rows selected.



## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

**EXNO:5**

### **RESTRICTING AND SORTING DATA**

**AIM:**

To limit the rows retrieved by the queries, and to sort the rows retrieved by the queries.

#### **SOLUTIONS FOR THE EXERCISES:**

14. Create a query to display the last name and salary of employees earning more than 12000. SQL> select last\_name,salary from employee where salary>12000;

LAST_NAME	SALARY
king	24000
kochhar	17000
dehaan	17000
hartstein	13000

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

15. Create a query to display the employee last name and department number for employee number 176.

```
SQL> select last_name,department_id from employeetable where employee_id=176;
```

LAST_NAME	DEPARTMENT_ID
-----	taylor
80	

16. Create a query to display the last name and salary of employees whose salary is not in the range of 5000 and 12000. (hints: not between )

```
SQL> select last_name,salary from employeetable where salary not between 5000 and 12000;
```

LAST_NAME	SALARY
-----	
king	24000
kochhar	17000
dehaan	17000
lorentz	4200
	rajs
3500	davies
3100	matos
2600	vargas
2500	whalen
4400	hartstein
13000	

10 rows selected.

17. Display the employee last name, job ID, and start date of employees hired between February 20,1998 and May 1,1998.order the query in ascending order by start date.

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

SQL> select last\_name,job\_id,hire\_date from employeetable where hire\_date between '20-feb-98' and '1-may-98';

LAST_NAME	JOB_ID	HIRE_DATE
-----	-----	matos
st_clerk	15-MAR-98	taylor
sa_rep	24-MAR-98	

18. Display the last name and department number of all employees in departments 20 and 50 in alphabetical order by name.

SQL> select last\_name,department\_id from employeetable where department\_id in(20,50) order by last\_name;

LAST_NAME	DEPARTMENT_ID
-----	-----
davies	50
20	hartstein
matos	50
mourgas	50
rajs	50
vargas	50

7 rows selected.

19. Display the last name and salary of all employees who earn between 5000 and 12000

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

and are in departments 20 and 50 in alphabetical order by name. Label the columns EMPLOYEE, MONTHLY SALARY respectively.

```
SQL> select last_name,department_id from employeetable where department_id in(20,50) order by last_name;
```

```
SQL> select last_name employee,salary monthly_salary from employeetable where salary between 5000 and 12000 and department_id in(20,50) order by last_name;
```

EMPLOYEE	MONTHLY_SALARY
-----	fay
6000	mourgas
5800	

20. Display the last name and hire date of every employee who was hired in 1994.

```
SQL> select last_name,hire_date from employeetable where hire_date between '1-jan-94' and '31-dec-94';
```

LAST_NAME	HIRE_DATE
-----	higgins
07-JUN-94	gietz 07-
JUN-94	

8.Display the last name and job title of all employees who do not have a manager

```
SQL> select last_name,job_id from employeetable where manager_id is NULL;
```

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

LAST\_NAME                      JOB\_ID

----- king

ad\_pres

9.Display the last name, salary, and commission for all employees who earn commissions. Sort data in descending order of salary and commissions.

SQL> select last\_name,salary,commission\_pct from employeetable where commission\_pct is not null;

LAST\_NAME                      SALARY COMMISSION\_PCT

-----

zlotkey                      10500                      .2

abel                      11000                      .3

taylor                      8600                      .2                      grant

7000                      .15

10.Display the last name of all employees where the third letter of the name is *a*.

SQL> select last\_name from employeetable where last\_name like '\_\_a%';

LAST\_NAME                      -

----- grant

whalen

11.Display the last name of all employees who have an *a* and an *e* in their last name

SQL> select last\_name from employeetable where last\_name like '%a%e%';

LAST\_NAME

-----

abel

hartstein

```
SQL> select last_name,job_id,salary from employeetable where job_id in('sa_rep','st_clerk') and salary not in(2500,3500,7000);
```

-----

davies	st_clerk	3100	
matos	st_clerk	2600	abel
sa_rep	11000	taylor	
sa_rep	8600		

```
SQL> select last_name ,salary, commission_pct from employeetable where commission_pct=.2;
```

-----

zlotkey	10500	.2	taylor
8600	.2		

## SINGLE ROW FUNCTIONS

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

### AIM:

To describe the various types of functions available in SQL, and to use character, number and date functions in SELECT statement and to describe the use of conversion functions.

### SOLUTION FOR THE EXERCISES:

1) Write a query to display the current date. Label the column Date.

```
SQL> select sysdate "date" from dual; date
```

-----

13-FEB-12

1 row selected.

2) The HR department needs a report to display the employee number, last name, salary, and increased by 15.5% (expressed as a whole number) for each employee. Label the column New Salary.

```
SQL> select employee_id,last_name,salary,round(salary*1.15,0)"new salary" from employees;
```

EMPLOYEE_ID	LAST_NAME	SALARY	new salary
-------------	-----------	--------	------------

-----

100	King	24000	27600
101	Kochar	17000	19550
102	Dehaan	17000	19550
103	Hunold	9000	10350
104	Ernst	6000	6900
107	Lorentz	4200	4830
124	Mourgos	5800	6670
141	Rajs	3500	4025

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

142 Davies	3100	3565
143 Matos	2600	2990
144 Vargas	2500	2875
149 Zlotkey	10500	12075
174 Abel	11000	12650
176 taylor	8600	9890
178 grant	7000	8050
200 whalen	4400	5060
201 harstein	13000	14950
202 fay	6000	6900
205 higgins	12000	13800
206 Gietz	8300	9545
113 popp	6900	7935

21 rows selected.

3) Modify your query lab\_03\_02.sql to add a column that subtracts the old salary from the new salary.

Label the column Increase.

```
SQL> select employee_id,last_name,salary,round(salary*1.15,0)"new
salary",round(salary*1.15,0)-salary "Increase" from employees;
```

EMPLOYEE_ID	LAST_NAME	SALARY	new salary	Increase
-------------	-----------	--------	------------	----------

-----

100 King	24000	27600	3600
101 Kocher	17000	19550	2550



## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

102 Dehaan	17000	19550	2550
103 Hunold	9000	10350	1350
104 Ernst	6000	6900	900
107 Lorentz	4200	4830	630
124 Mourgos	5800	6670	870
141 Rajs	3500	4025	525
142 Davies	3100	3565	465
143 Matos	2600	2990	390
144 Vargas	2500	2875	375
149 Zlotkey	10500	12075	1575
174 Abel	11000	12650	1650
176 taylor	8600	9890	1290
178 grant	7000	8050	1050
200 whalen	4400	5060	660
201 harstein	13000	14950	1950
202 fay	6000	6900	900
205 higgins	12000	13800	1800
206 Gietz	8300	9545	1245
113 popp	6900	7935	1035

21 rows selected.

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

4) Write a query that displays the last name (with the first letter uppercase and all other letters lowercase) and the length of the last name for all employees whose name starts with the letters J, A, or M. Give each column an appropriate label. Sort the results by the employees' last names

```
SQL> select initcap(last_name)"Name",length(last_name)"Length" from employees where  
last_name like 'j%' or last_name like 'M%' or last_name like 'A%' order by last_name;
```

Name	Length
Abel	4
Matos	5
Mourgos	7

3 rows selected.

5)The HR department wants to find the length of employment for each employee. For each employee,display the last name and calculate the number of months between today and the date on which the employee was hired. Label the column MONTHS\_WORKED. Order your results by the number of months employed. Round the number of months up to the closest whole number.

```
SQL> select last_name,round(MONTHS_BETWEEN(SYSDATE,hire_date))  
MONTHS_WORKED from employees order by MONTHS_BETWEEN(SYSDATE,hire_date);
```

LAST_NAME	MONTHS_WORKED
popp	0
Zlotkey	145
Mourgos	147
grant	153

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

Lorentz	156
Vargas	163
taylor	167
Matos	167
fay	174
Davies	181
Abel	189
harstein	192
Rajs	196
Gietz	212
higgins	212
Dehaan	229
Ernst	249
Hunold	265
Kochar	269 whalen
	293
King	296

21 rows selected.

7) Create a query to display the last name and salary for all employees. Format the salary to be 15 characters long, left-padded with the \$ symbol. Label the column SALARY.

```
SQL> select last_name,lpad(salary,15,'$')salary from employees;
```

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

LAST_NAME	SALARY
-----	-----
King	\$\$\$\$\$\$\$\$\$24000
Kochar	\$\$\$\$\$\$\$\$\$17000
Dehaan	\$\$\$\$\$\$\$\$\$17000
Hunold	\$\$\$\$\$\$\$\$\$9000
Ernst	\$\$\$\$\$\$\$\$\$6000
Lorentz	\$\$\$\$\$\$\$\$\$4200
Mourgos	\$\$\$\$\$\$\$\$\$5800
Rajs	\$\$\$\$\$\$\$\$\$3500
Davies	\$\$\$\$\$\$\$\$\$3100
Matos	\$\$\$\$\$\$\$\$\$2600
Vargas	\$\$\$\$\$\$\$\$\$2500
Zlotkey	\$\$\$\$\$\$\$\$\$10500
Abel	\$\$\$\$\$\$\$\$\$11000
taylor	\$\$\$\$\$\$\$\$\$8600
grant	\$\$\$\$\$\$\$\$\$7000
whalen	\$\$\$\$\$\$\$\$\$4400
harstein	\$\$\$\$\$\$\$\$\$13000
fay	\$\$\$\$\$\$\$\$\$6000
higgins	\$\$\$\$\$\$\$\$\$12000
Gietz	\$\$\$\$\$\$\$\$\$8300
popp	\$\$\$\$\$\$\$\$\$6900

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

21 rows selected.

8) Display each employee's last name, hire date, and salary review date, which is the first Monday after six months of service. Label the column REVIEW. Format the dates to appear in the format similar to

```
SQL> select last_name, hire_date, to_char(next_day(Add_months(hire_date, 6),
'monday'), 'fmday"the"ddspth"of" month, YYYY') review from employees;
```

LAST_NAME	HIRE_DATE	REVIEW
-----------	-----------	--------

King	17-JUN-87	monday the twenty-first of december, 1987
Kocher	21-SEP-89	monday the twenty-sixth of march, 1990
Dehaan	13-JAN-93	monday the nineteenth of july, 1993
Hunold	03-JAN-90	monday the ninth of july, 1990
Ernst	21-MAY-91	monday the twenty-fifth of november, 1991
Lorentz	07-FEB-99	monday the ninth of august, 1999
Mourgos	16-NOV-99	monday the twenty-second of may, 2000
Rajs	17-OCT-95	monday the twenty-second of april, 1996
Davies	29-JAN-97	monday the fourth of august, 1997
Matos	15-MAR-98	monday the twenty-first of september, 1998
Vargas	09-JUL-98	monday the eleventh of january, 1999
Zlotkey	29-JAN-00	monday the thirty-first of july, 2000

21 rows selected.

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

9) Display the last name, hire date, and day of the week on which the employee started. Label the column DAY. Order the results by the day of the week, starting with Monday.

```
SQL> select last_name,hire_date,to_char(hire_date,'DAY')DAY from employees order by  
to_char(hire_date-1,'d');
```

LAST_NAME	HIRE_DATE DAY
-----------	---------------

----- grant
-------------

24-MAY-99 MONDAY popp
-----------------------

30-JAN-12 MONDAY taylor
-------------------------

24-MAR-98 TUESDAY Gietz
-------------------------

07-JUN-94 TUESDAY higgins
---------------------------

07-JUN-94 TUESDAY Rajs
------------------------

17-OCT-95 TUESDAY
-------------------

Mourgos	16-NOV-99 TUESDAY
---------	-------------------

Ernst	21-MAY-91 TUESDAY
-------	-------------------

Davies	29-JAN-97 WEDNESDAY
--------	---------------------

Dehaan	13-JAN-93 WEDNESDAY
--------	---------------------

King	17-JUN-87 WEDNESDAY
------	---------------------

Hunold	03-JAN-90 WEDNESDAY
--------	---------------------

Vargas	09-JUL-98 THURSDAY
--------	--------------------

Kochar	21-SEP-89 THURSDAY
--------	--------------------

whalen	17-SEP-87 THURSDAY
--------	--------------------

harstein	17-FEB-96 SATURDAY
----------	--------------------

## **CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY**

Zlotkey                29-JAN-00 SATURDAY

Abel                    11-MAY-96 SATURDAY

Lorentz                07-FEB-99 SUNDAY fay

17-AUG-97 SUNDAY

Matos                  15-MAR-98 SUNDAY

21 rows selected.

### **EXNO:7 DISPLAYING DATA FROM MULTIPLE TABLE**

#### **AIM:**

To write SELECT statement to access data from more than one table using equality and nonequality joins, and to view data that generally does not meet a join condition by using outer joins and to join a table to itself by using a self join.

#### **SOLUTIONS FOR THE EXERCISES:**

1. Write a query to display the last name, department number, and department name for all employees.

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

```
SQL> select e.last_name,e.department_id,d.department_name from employee e,department
d where e.department_id=d.department_id;
```

LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
-----------	---------------	-----------------

king		
------	--	--

90 executive	kochhar	
--------------	---------	--

90 executive	dehaan	
--------------	--------	--

90 executive	hunold	
--------------	--------	--

60 it	ernst	60 it
-------	-------	-------

lorentz	60 it	
---------	-------	--

mourgas	50 shipping	
---------	-------------	--

rajs	50 shipping	
------	-------------	--

davies	50 shipping	
--------	-------------	--

matos	50 shipping	
-------	-------------	--

vargas	50 shipping	
--------	-------------	--

LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
-----------	---------------	-----------------

zlotkey	80 sales	abel
---------	----------	------

80 sales	taylor	80
----------	--------	----

sales	whalen	10
-------	--------	----

administration	hartstein	20
----------------	-----------	----

marketing	fay	20
-----------	-----	----

marketing	higgins	110
-----------	---------	-----



accounting	gietz	110
accounting		

19 rows selected.

2. Create a unique listing of all jobs that are in department 80. Include the location of the department in the output

```
SQL> select distinct job_id,location_id from employeetable,department where
employeetable.department_id=department.department_id and employeetable.department_id=80;
```

JOB\_ID                      LOCATION\_ID

-----

sa_man	2500	sa_rep
2500		

3. Write a query to display the employee last name, department name, location ID, and city of all employees who earn a commission

```
SQL> select e.last_name,d.department_name ,d.location_id ,l.city from employeetable
e,department d,locations l where e.department_id=d.department_id and
d.location_id=l.location_id and e.commission_pct is not null;
```

LAST_NAME	DEPARTMENT_NAME	LOCATION_ID	CITY
-----------	-----------------	-------------	------

-----

lotkey	sales	2500	oxford		abel	sale
2500	oxford			tay	sales	2500
oxford						

4. Display the employee last name and department name for all employees who have an a(lowercase) in their last names

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

```
SQL> select last_name,department_name from employeetable,department where
employeetable.department_id=department.department_id and last_name like '%a%';
```

LAST_NAME	DEPARTMENT_NAME
whalen	administration
marketing	hartstein
vargas	shipping
shipping	davies
rajs	shipping
shipping	taylor
abel	sales
executive	kochhar
	executive

12 rows selected.

5.rite a query to display the last name, job, department number, and department name for all employees who work in Toronto no rows selected

6. Display the employee last name and employee number along with their manager's last name and manager number. Label the columns Employee, Emp#, Manager, and Mgr#, Respectively

```
SQL> select w.last_name "employee",w.employee_id "emp#",m.last_name
"manager",m.employee_id from employeetable w join employeetable m on
(w.manager_id=m.employee_id);
```

emplo	emp#	manager	EMPLOYEE_ID
hartstein	201	king	100
zlotkey	149	king	100

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

mourgas	124 king	100
dehaan	102 king	100
kochhar	101 king	100
higgins	205 kochhar	101
whalen	200 kochhar	101
hunold	103 dehaan	102
lorentz	107 hunold	103
ernst	104 hunold	103
vargas	144 mourgas	124
matos	143 mourgas	124
davies	142 mourgas	124
rajs	141 mourgas	124
grant	178 zlotkey	149

employee	emp#	manager	EMPLOYEE_ID
taylor	176	zlotkey	149
abel	174	zlotkey	149
202	hartstein	201	gietz
206	higgins	205	

19 rows selected.

7. Modify lab4\_6.sql to display all employees including King, who has no manager. Order the results by the employee number.

```
SQL> select w.last_name"employee",w.employee_id"emp#",m.last_name
"manager",m.employee_id "mgr#" from employeeetable w left outer join employeeetable m
on(w.manager_id=m.employee_id);
```

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

employee	emp#	manager	mgr#
hartstein	201	king	100
zlotkey			
149	king	100	mourgas
124			king
100	dehaan	102	king
100			
kochhar	101	king	100
higgins			
205	kochhar	101	whalen
200			
kochhar	101	hunold	103
dehaan			
102	lorentz	107	hunold
103			ernst
104	hunold	103	vargas
144			
mourgas	124		

employee	emp#	manager	mgr#
matos	143	mourgas	124
davies	142	mourgas	24
rajs			
141	mourgas	124	grant
178			
zlotkey	149	taylor	176
zlotkey	149	abel	174
zlotkey	149	fay	202
hartstein	201	gietz	206
higgins	205	king	100

20 rows selected.

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

8. Create a query that displays employee last names, department numbers, and all the employees who work in the same department as a given employee. Give each column an appropriate label

```
SQL> select e.department_id department,e.last_name employee,c.last_name colleague from
employee e join employee c on(e.department_id=c.department_id)where
e.employee_id<>c.employee_id order by e.department_id ,e.last_name,c.last_name;
```

DEPARTMENT	EMPLOYEE	COLLEAGUE
------------	----------	-----------

-----

20	fay	hartstein
----	-----	-----------

20	hartstein	fay
----	-----------	-----

50	davies	matos
----	--------	-------

DEPARTMENT	EMPLOYEE	COLLEAGUE
------------	----------	-----------

-----

50	davies	mourgas
----	--------	---------

50	davies	rajs
----	--------	------

50	davies	vargas
----	--------	--------

50	matos	davies
----	-------	--------

50	matos	mourgas
----	-------	---------

50	matos	rajs
----	-------	------

50	matos	vargas
----	-------	--------

50	mourgas	davies
----	---------	--------

50	mourgas	matos
----	---------	-------

50	mourgas	rajs
----	---------	------

50	mourgas	vargas
----	---------	--------

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

50 rajs	davies
50 rajs	matos
50 rajs	mourgas
50 rajs	vargas
50 vargas	davies
50 vargas	matos
50 vargas	mourgas
50 vargas	rajs
60 ernst	hunold
60 ernst	lorentz
60 hunold	ernst
60 hunold	lorentz

DEPARTMENT EMPLOYEE

COLLEAGUE

-----

60 lorentz	ernst
60 lorentz	hunold
80 abel	taylor
80 abel	zlotkey
80 taylor	abel

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

80 taylor	zlotkey
80 zlotkey	abel
80 zlotkey	taylor
90 dehaan	king
90 dehaan	kochhar
90 king	dehaan
90 king	kochhar
90 kochhar	dehaan
90 kochhar	king
110 gietz	higgins
110 higgins	gietz

42 rows selected.

9. Show the structure of the JOB\_GRADES table. Create a query that displays the name, job, department name, salary, and grade for all employees

```
SQL> desc job_grades;
```

Name	Null?	Type
-----		
GRADE_LEVEL		VARCHAR2(3)
LOWEST_SAL		NUMBER(10)
HIGHEST_SAL		NUMBER

```
SQL> select * from job_grades;
```

```
GRA LOWEST_SAL HIGHEST_SAL
```

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

-----		a
1000	2999	b
3000	5999	c
6000	9999	d
10000	14999	e
15000	24999	f
25000	40000	

6 rows selected.

9. Create a query to display the name and hire date of any employee hired after employee Davies.

```
SQL> select e.last_name,e.hire_date from employeetable e,employeetable davies where
davies.last_name='davies' and davies.hire_date<e.hire_date;
```

LAST_NAME	HIRE_DATE	
-----		lorentz
07-FEB-99	mourgas	16-
NOV-99	matos	15-
MAR-98	vargas	09-
JUL-98	zlotkey	29-JAN-
00	taylor	24-MAR-98
grant	24-MAY-99	fay
17-AUG-97		

8 rows selected.

11. Display the names and hire dates for all employees who were hired before their managers, along with their manager's names and hire dates. Label the columns Employee, Emp Hired, Manager, and Mgr Hired, respectively.



## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

```
SQL> select w.last_name,w.hire_date,m.last_name,m.hire_date from employeetable
w,employeetable m where w.manager_id=m.employee_id and w.hire_date <m.hire_date;
```

LAST_NAME	HIRE_DATE	LAST_NAME	HIRE_DATE
-----	-----	-----	whalen
17-SEP-87	kochhar	21-SEP-89	hunold
03-JAN-93	dehaan	13-JAN-93	ernst 21-
MAY-91	hunold	03-JAN-93	vargas 09-
JUL-98	mourgas	16-NOV-99	matos 15-
MAR-98	mourgas	16-NOV-99	davies
29-JAN-97	mourgas	16-NOV-99	rajs
17-OCT-95	mourgas	16-NOV-99	grant
24-MAY-99	zlotkey	29-JAN-00	taylor
24-MAR-98	zlotkey	29-JAN-00	abel
11-MAY-96	zlotkey	29-JAN-00	

10 rows selected.

### EXNO:8 AGGREGATING DATA USING GROUP FUNCTIONS

**AIM:** To identify the available group functions, to describe the use of group functions, to group data by using the GROUP BY clause, to include or exclude grouped rows by using the HAVING clause.

### SOLUTION FOR THE EXERCISE:

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

1. Find the highest, lowest, sum, and average salary of all employees. Label the columns

Maximum, Minimum, Sum, and Average, respectively. Round your results to the nearest whole number

```
SQL> select  
round(max(salary),0)"maximum",round(min(salary),0)"minimum",round(sum(salary),0)"sum",ro  
und(avg(salary),0)"average" from employeetable;
```

maximum	minimum	sum	average
---------	---------	-----	---------

-----

24000	2500	175500	8775
-------	------	--------	------

2. Modify the above query to display the minimum, maximum, sum, and average salary for each job type.

```
SQL> Selectround(max(salary),0)"maximum",round(min(salary),0)  
"minimum",round(sum(salary),0)"sum",round(avg(salary),0)"average"    from    employeetable  
group by job_id;
```

maximum	minimum	sum	average
---------	---------	-----	---------

-----

11000	7000	26600	8867
-------	------	-------	------

6000	6000	6000	6000
------	------	------	------

24000	24000	24000	24000
-------	-------	-------	-------

17000	17000	34000	17000
-------	-------	-------	-------

9000	4200	19200	6400
------	------	-------	------

5800	5800	5800	5800
------	------	------	------

8300	8300	8300	8300
------	------	------	------

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

10500	10500	10500	10500	4400
4400	4400	4400		maximum
minimum	sum	average		

-----

12000	12000	12000	12000
3500	2500	11700	2925
13000	13000	13000	13000

12 rows selected.

3. Write a query to display the number of people with the same job. Generalize the query so that the user in the HR department is prompted for a job title.

SQL> select job\_id,count(\*) from employeetable group by job\_id;

JOB_ID	COUNT(*)	
sa_rep	3	mk_rep
1		ad_pres
ad_vp	2	it_prog
3		st_man
ac_account	1	
sa_man	1	
ad_asst	1	ac_mgr
1		st_clerk
mk_man	1	4

12 rows selected.

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

4. Determine the number of managers without listing them. Label the column Number of Managers.

```
SQL> select count(distinct manager_id)"number of managers" from employeetable;
```

```
number of managers
```

```
-----
```

```
8
```

5. Find the difference between the highest and lowest salaries. Label the column DIFFERENCE.

```
SQL> select max(salary)-min(salary)"difference" from employeetable;
```

```
difference
```

```
-----
```

```
21500
```

6. Create a report to display the manager number and the salary of the lowest-paid employee for that manager. Exclude anyone whose manager is not known. Exclude any groups where the minimum salary is \$6,000 or less. Sort the output in descending order of salary.

```
SQL> select manager_id,min(salary)from employeetable where manager_id is not null group by manager_id having min(salary)>6000 order by min(salary)desc;
```

```
MANAGER_ID MIN(SALARY)
```

```
-----
```

```
102      9000
```

```
205      8300
```

```
149      7000
```

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

7. Create a query to display the total number of employees and, of that total, the number of employees hired in 1995, 1996, 1997, and 1998. Create appropriate column headings

```
SQL> select
count(*)"total",sum(decode(to_char(hire_date,'YYYY'),1995,1,0))"1995",sum(decode(to_char(hire_date,'YYYY'),1996,1,0))"1996",sum(decode(to_char(hire_date,'YYYY'),1997,1,0))"1997",sum(decode(to_char(hire_date,'YYYY'),1998,1,0))"1998" from employeetable;
```

total	1995	1996	1997	1998
20	1	2	2	3

8. Create a matrix query to display the job, the salary for that job based on department number, and the total salary for that job, for departments 20, 50, 80, and 90, giving each column an appropriate heading.

```
SQL>
```

```
Select job_id"job",sum(decode(department_id,20,salary))"department20",sum(decode(department_id,50,salary))"department50",sum(decode(department_id,80,salary))"department80",sum(decode(department_id,90,salary))"department90",sum(salary)"total" from employeetable group by job_id; job department20 department50 department80 department90 total
```

sa_rep	19600	26600		
mk_rep	6000	6000		
ad_pres	24000	24000		
ad_vp	34000	4000		
it_prog	19200	st_man	5800	5800
ac_account	8300			
sa_man	10500	10500		ad_asst
	4400			

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

```

ac_mgr      12000                                st_clerk
11700      11700                                mk_man 13000      13000

```

12 rows selected.

9. Write a query to display each department's name, location, number of employees, and the average salary for all the employees in that department. Label the column name-Location, Number of people, and salary respectively. Round the average salary to two decimal places.

```

SQL> select d.department_name "name",d.location_id "location",count(*)"no of
people",round(avg(salary),2)"salary" from employeetable e,department d where e.department_id
=d.department_id group by d.department_name,d.location_id;

```

name	location	no of people	salary
-----	-----	-----	----
1400	3	6400	shipping
5	3500	administration	1700
4400	marketing	1800	2
9500	executive	1700	3
19333.33	sales	2500	3
10033.33	accounting	1700	2
10150			

7 rows selected.

**EXNO:9**

**SUB QUERIES**

# CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

## AIM:

To define subqueries, to describe the types of problems that sub queries can solve, to list the types of subqueries, and to write single row and multiple row subqueries.

## SOLUTIONS FOR THE EXERCISES

1. The HR department needs a query that prompts the user for an employee last name.

The query then displays the last name and hire date of any employee in the same department as the employee whose name they supply (excluding that employee). For example, if the user enters Zlotkey, find all employees who work with Zlotkey (excluding Zlotkey).

```
SQL> select last_name,hire_date from employeetable where department_id=(select
department_id from employeetable where last_name='zlotkey')and last_name<>'zlotskey';
```

LAST_NAME	HIRE_DATE	
zlotkey	29-JAN-00	abel
11-MAY-96	taylor	24-
MAR-98		

2. Create a report that displays the employee number, last name, and salary of all employees who earn more than the average salary. Sort the results in order of ascending salary

```
SQL> select employee_id,last_name,salary from employeetable where salary>(select avg(salary)
from employeetable)order by salary;
```

EMPLOYEE_ID	LAST_NAME	SALARY
-------------	-----------	--------

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

103	hunold	9000
149	zlotkey	10500
174	abel	11000
205	higgins	12000
201	hartstein	13000
101	kochhar	17000
102	dehaan	17000
100	king	24000

8 rows selected.

3. Write a query that displays the employee number and last name of all employees who work in a department with any employee whose last name contains a *u*.

```
SQL> select last_name,hire_date from employeetable where department_id in(select
department_id from employeetable where last_name like '%u%');
```

LAST_NAME	HIRE_DATE	
lorentz	07-FEB-99	
ernst	21-MAY-91	
hunold	03-JAN-93	
vargas	09-JUL-98	
matos	15-MAR-98	
davies	29-JAN-97	rajs
17-OCT-95		mourgas
16-NOV-99		



## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

8 rows selected.

4. The HR department needs a report that displays the last name, department number, and job ID of all employees whose department location ID is 1700.

```
SQL> select last_name ,department_id,job_id from employeetable where department_id in(select department_id from department where location_id=1700);
```

LAST_NAME	DEPARTMENT_ID	JOB_ID
whalen	10	ad_asst
dehaan	90	ad_vp
kochhar	90	ad_vp
king	90	ad_vp
gietz	110	ad_pres
higgins	110	ac_account
ac_mgr		

6 rows selected.

5. Create a report for HR that displays the last name and salary of every employee who reports to King.

```
SQL> select last_name,salary from employeetable where manager_id=(select employee_id from employeetable where last_name='king');
```

LAST_NAME	SALARY
kochhar	17000
dehaan	17000
mourgas	5800
zlotkey	10500
hartstein	13000

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

6. Create a report for HR that displays the department number, last name, and job ID for every employee in the Executive department.

```
SQL> select last_name,department_id,job_id from employeetable where department_id in(select department_id from department where department_name='executive');
```

LAST_NAME	DEPARTMENT_ID	JOB_ID
-----		dehaan
90 ad_vp	kochhar	90 ad_vp
king	90 ad_pres	

7. Modify the query 3 to display the employee number, last name, and salary of all employees who earn

more than the average salary and who work in a department with any employee whose last name contains a *u*.

```
SQL> select employee_id,last_name,salary from employeetable where department_id in(select department_id from employeetable where last_name like '%u%') and salary>(select avg(salary) from employeetable);
```

EMPLOYEE_ID	LAST_NAME	SALARY
-----		
103	hunold	9000

# CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

**EXNO:10**

## **USING THE SET OPERATORS**

**AIM:**

To describe the set operators, to use a set operator to combine multiple queries into a single query, and to control the order of rows returned.

**FIND THE SOLUTIONS:**

1. The HR department needs a list of department IDs for departments that do not contain the job ID ST\_CLERK. Use set operators to create this report.

```
select department_id from department minus select department_id from employeetable where  
job_id='ST_CLERK';
```

DEPARTMENT\_ID

-----

10

20

50

60

80

90

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

110

190

8 rows selected.

2. The HR department needs a list of countries that have no departments located in them.

Display the country ID and the name of the countries. Use set operators to create this report.

```
SQL> select country_id,country_name from countries minus select l.country_id,c.country_name
from locations l,countries c where l.country_id=c.country_id;
```

CO COUNTRY\_NAME

-- -----

de germany

3. Produce a list of jobs for departments 10, 50, and 20, in that order. Display job ID and department ID using set operators.

```
SQL> select job_id,department_id from employeetable where department_id=10 union select
job_id,department_id from employeetable where department_id=50 union select
job_id,department_id from employeetable where department_id=20;
```

JOB\_ID            DEPARTMENT\_ID

-----

ad\_asst            10

mk\_man            20

mk\_rep            20

st\_clerk           50                    st\_man

50

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

4. Create a report that lists the employee IDs and job IDs of those employees who currently have a job title that is the same as their job title when they were initially hired by the company (that is, they changed jobs but have now gone back to doing their original job).

```
SQL> select employee_id,job_id from employeetable intersect select employee_id,job_id from job_history;
```

EMPLOYEE\_ID JOB\_ID

-----

103 it\_prog

176 sa\_rep

200 ad\_asst

5. The HR department needs a report with the following specifications:

- Last name and department ID of all the employees from the EMPLOYEES table, regardless of whether or not they belong to a department.
- Department ID and department name of all the departments from the DEPARTMENTS table, regardless of whether or not they have employees working in them Write a compound query to accomplish this.

```
SQL> select last_name,department_id,to_char(null) from employeetable union select to_char(null),department_id,department_name from department;
```

LAST\_NAME                      DEPARTMENT\_ID TO\_CHAR(NULL)

-----

abel	80	davies
50	dehaan	90
ernst	60	fay
20	gietz	110
grant		hartstein
20	higgins	110

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

hunold	60	king
--------	----	------

90 kochhar	90	lorentz
------------	----	---------

60	matos	50
----	-------	----

LAST_NAME	DEPARTMENT_ID TO_CHAR(NULL)
-----------	-----------------------------

-----	mourgas	50
-------	---------	----

rajs	50	taylor	80
------	----	--------	----

vargas	50	whalen
--------	----	--------

10	zlotkey	80
----	---------	----

administration	10
----------------	----

8 rows selected.

# CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

## EXNO:11

## CREATING VIEWS

### AIM:

To describe a view, to create, alter the definition of , and drop a view, to retrieve the data through a view, to insert, update, and delete data through a view, to create and use an inline view.

### SOLUTIONS:

- 1) Create a view called EMPLOYEE\_VU based on the employee numbers, employee names and department numbers from the EMPLOYEES table. Change the heading for the employee name to EMPLOYEE.

```
SQL> create view employee_VU as select employee_id,last_name "employee",department_id
from employees;
```

View created.

- 2) Display the contents of the EMPLOYEES\_VU view.

```
SQL> select *from employee_VU;
```

EMPLOYEE_ID	EMPLOYEE	DEPARTMENT_ID
-------------	----------	---------------

-----

1	king	90
101	kocchar	90

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

102	dehaan	90
103	hunold	60
104	ernst	60
107	lorentz	4200
124	mourgos	50
141	rajs	50
143	matos	50

9 rows selected.

3) Select the view name and text from the USER\_VIEWS data dictionary views.

SQL> select \*from tab;

TNAME	TABTYPE	CLUSTERID
-----	-----	
STUDENT	TABLE	
EMP	TABLE	
DEPT	TABLE	
EMPLOYEES	TABLE	
JOB	TABLE	
DEPARTMENT	TABLE	
MY_EMPLOYEE	TABLE	
EMPLOYEES1	TABLE	
DEPARTMENT1	TABLE	



## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

JOB_GRADE	TABLE
LOCATION	TABLE
MY_EMPLOYEES	TABLE
JOBS	TABLE
COUNTRIES	TABLE
LOCATIONS	TABLE
EMPLOYEE_VU	VIEW

16 rows selected.

SQL> select view\_name,text from user\_views;

VIEW_NAME	TEXT	EMPLOYEE_VU
-----------	------	-------------

-----

4) using your EMPLOYEES\_VU view, enter a query to display all employees names and department.

SQL> select employee\_id,department\_id from employee\_VU;

EMPLOYEE_ID	DEPARTMENT_ID
-------------	---------------

-----

1	90
101	90
102	90
103	60
104	60
107	4200
124	50

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

141	50
143	50

9 rows selected.

- 5) Create a view named DEPT50 that contains the employee number, employee last names and department numbers for all employees in department 50. Label the view columns EMPNO, EMPLOYEE and DEPTNO. Do not allow an employee to be reassigned to another department through the view.

```
SQL> create view dept50 as select
employee_id"empno",last_name"employees",department_id"deptno" from employees where
department_id=50 with check option constraints emp_department_50;
```

View created.

- 6) Display the structure and contents of the DEPT50 view.

```
SQL> desc dept50;
```

Name	Null?	Type
emp no	NOT NULL	NUMBER(6)
employees	NOT NULL	VARCHAR2(25)

dept no	NUMBER(4)
---------	-----------

```
SQL> select *from dept50; emp no
```

```
employees    dept no
```

-----	-----	
124	mourgos	50
141	rajs	50
143	matos	50

- 7) Attempt to reassign Matos to department 80.

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

```
SQL> update dept50 set deptno=80 where employee='matos';
```

1 row updated.

```
SQL> select *from dept50;
```

emp no	employees	dept no
124	mourgos	50
141	rajs	50
143	matos	80

8) Create a view called SALARY\_VU based on the employee last names, department names, salaries, and salary grades for all employees. Use the Employees, DEPARTMENTS and JOB\_GRADE tables. Label the column Employee, Department, salary, and Grade respectively.

```
SQL> create view salary_VU as select
e.last_name"employee",d.dept_name"dept",e.salary"salary",j.grade_level"grades" from
employees e,department d,job j where e.department_id=d.dept_id and e.salarybetween
j.lowest_sal and j.highest_sal;
```

View created.

**EXNO:12**

### PROCEDURES AND FUNCTIONS

**AIM:**

To implement the concept of procedures and functions which is a logically grouped set of SQL and PL/SQL statements that perform a specific task.

**PROGRAM FOR GENERAL PROCEDURE – SELECTED RECORD’S PRICE IS INCREMENTED BY 500 , EXECUTING THE PROCEDURE CREATED AND DISPLAYING THE UPDATED TABLE**

```
SQL> create procedure itsum(identity number, total number) is price number;
```

```
2  null_price exception;
```

```
3  begin
```

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

```
4  select actualprice into price from ititems where itemid=identity;
5  if price is null then
6  raise null_price;
7  else
8  update ititems set actualprice=actualprice+total where itemid=identity;
9  end if;
10 exception
11 when null_price then
12 dbms_output.put_line('price is null');
13 end;
14 /
```

Procedure created.

```
SQL> exec itsum(101, 500);
```

PL/SQL procedure successfully completed.

```
SQL> select * from ititems;
```

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

ITEMID	ACTUALPRICE	ORDID	PRODID
-----	-----	-----	-----
101	2500	500	201
102	3000	1600	202
103	4000	600	202

### CREATE THE TABLE 'ITTRAIN' TO BE USED FOR FUNCTIONS

SQL>create table ittrain (tno number(10), tfare number(10));

Table created.

SQL>insert into ittrain values (1001, 550); 1

row created.

SQL>insert into ittrain values (1002, 600);

1 row created.

SQL>select \* from ittrain;

TNO	TFARE
-----	-----
1001	550
1002	600

### PROGRAM FOR FUNCTION AND IT'S EXECUTION

SQL> create function aaa (trainnumber number) return number is

5

6

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

```
2 trainfunction ittrain.tfare % type;
3 begin
4 select tfare into trainfunction from ittrain where tno=trainnumber; return(trainfunction);
   end;
   /
```

Function created.

```
SQL> set serveroutput on;
```

```
SQL> declare
```

```
2 total number;
3 begin
4 total:=aaa (1001);
5 dbms_output.put_line('Train fare is Rs. '||total);
6 end;
7 /
```

Train fare is Rs.550

PL/SQL procedure successfully completed.

### **FACTORIAL OF A NUMBER USING FUNCTION — PROGRAM AND EXECUTION**

5

6

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

7

SQL> create function itfact (a number) return number is

2 fact number:=1;

3 b number;

4 begin

5 b:=a;

6 while b>0

7 loop

8 fact:=fact\*b;

9 b:=b-1;

10 end loop;

11 return(fact);

12 end;

13 /

Function created.

SQL> set serveroutput on;

SQL> declare

2 a number:=7;

3 f number(10);

4 begin

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

```
5 f:=itfact(a);  
6 dbms_output.put_line('The factorial of the given number is'||f);  
7 end;  
8 /
```

The factorial of the given number is 5040

PL/SQL procedure successfully completed.

### PROGRAMS

#### TO DISPLAY HELLO MESSAGE

```
SQL> set serveroutput on;  
  
SQL> declare  
  
2 a varchar2(20);  
  
3 begin  
  
4 a:='Hello'; dbms_output.put_line(a); end;  
  
/
```

Hello

PL/SQL procedure successfully completed.

#### TO INPUT A VALUE FROM THE USER AND DISPLAY IT

```
SQL> set serveroutput on;  
  
SQL> declare  
5  
  
6
```



## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

7

2 a varchar2(20);

3 begin

4 a:=&a;

5 dbms\_output.put\_line(a);

6 end;

7 /

Enter value for a: 5

old 4: a:=&a; new

4: a:=5;

5

PL/SQL procedure successfully completed.

### **GREATEST OF TWO NUMBERS**

SQL> set serveroutput on;

SQL> declare

2 a number(7);

3 b number(7);

4 begin

5 a:=&a;

6 b:=&b;

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

```
7  if(a>b) then
8  dbms_output.put_line (' The grerater of the two is'|| a);
9  else
10 dbms_output.put_line (' The grerater of the two is'|| b);
11 end if;
12 end;
13 /
```

Enter value for a: 5

old 5: a:=&a; new

5: a:=5; Enter value

for b: 9 old 6:

b:=&b; new 6:

b:=9;

The grerater of the two is9

PL/SQL procedure successfully completed.

### GREATEST OF THREE NUMBERS

SQL> set serveroutput on;

SQL> declare

2 a number(7);

5

6

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

7

3 b number(7);

4 c number(7); begin a:=&a;

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

7

b:=&b;

8 c:=&c;

9 if(a>b and a>c) then

10 dbms\_output.put\_line (' The greatest of the three is ' || a);

11 else if (b>c) then

12 dbms\_output.put\_line (' The greatest of the three is ' || b);

13 else

14 dbms\_output.put\_line (' The greatest of the three is ' || c);

15 end if;

16 end if;

17 end;

18 /

Enter value for a: 5

old 6: a:=&a; new

6: a:=5; Enter value

for b: 7 old 7:

b:=&b; new 7:

b:=7; Enter value

for c: 1 old 8:

c:=&c; new 8:

c:=1;

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

The greatest of the three is 7

PL/SQL procedure successfully completed.

**PRINT NUMBERS FROM 1 TO 5 USING SIMPLE LOOP** SQL> set  
serveroutput on;

SQL> declare

2 a number:=1;

3 begin

4 loop

5 dbms\_output.put\_line (a);

6 a:=a+1;

7 exit when a>5;

8 end loop;

9 end;

10 /

1

2

3

4

5

PL/SQL procedure successfully completed.

## **CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY**

### **PRINT NUMBERS FROM 1 TO 4 USING WHILE LOOP**

SQL> set serveroutput on;

SQL> declare

2 a number:=1;

3 begin

4 while(a<5)

5 loop

6 dbms\_output.put\_line (a);

7 a:=a+1;

8 end loop;

9 end;

10 /

1

2

3

4

PL/SQL procedure successfully completed.

### **PRINT NUMBERS FROM 1 TO 5 USING FOR LOOP**

SQL> set serveroutput on;

SQL> declare

2 a number:=1;

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

```
3 begin
4 for a in 1..5
5 loop
6 dbms_output.put_line (a);
7 end loop;
8 end;
9 /
1
2
3
4
5
```

PL/SQL procedure successfully completed.

### **PRINT NUMBERS FROM 1 TO 5 IN REVERSE ORDER USING FOR LOOP**

```
SQL> set serveroutput on;
```

```
SQL> declare
```

```
2 a number:=1;
```

```
3 begin
```

```
4 for a in reverse 1..5
```

```
5 loop
```

```
6 dbms_output.put_line (a);
```

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

```
7 end loop;
```

```
8 end;
```

```
9 /
```

```
5
```

```
4
```

```
3
```

```
2
```

```
1
```

PL/SQL procedure successfully completed.

### TO CALCULATE AREA OF CIRCLE

```
SQL> set serveroutput on;
```

```
SQL> declare
```

```
2 pi constant number(4,2):=3.14;
```

```
3 a number(20);
```

```
4 r number(20);
```

```
5 begin
```

```
6 r:=&r;
```

```
7 a:= pi* power(r,2);
```

```
8 dbms_output.put_line (' The area of circle is ' || a);
```

```
9 end;
```

```
10 /
```



## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

Enter value for r: 2

old 6: r:=&r; new

6: r:=2;

The area of circle is 13

PL/SQL procedure successfully completed.

### TO CREATE SACCOUNT TABLE

SQL> create table saccount ( accno number(5), name varchar2(20), bal number(10));

Table created.

SQL> insert into saccount values ( 1,'mala',20000); 1

row created.

SQL> insert into saccount values (2,'kala',30000);

1 row created.

SQL> select \* from saccount;

ACCNO	NAME	BAL
1	mala	20000
2	kala	30000

SQL> set serveroutput on;

SQL> declare

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

```
2  a_bal number(7);
3  a_no varchar2(20);
4  debit number(7):=2000;
5  minamt number(7):=500;
6  begin
7  a_no:=&a_no;
8  select bal into a_bal from saccount where accno= a_no;
9  a_bal:= a_bal-debit;
10 if (a_bal > minamt) then
11 update saccount set bal=bal-debit where accno=a_no;
12 end if;
13 end;
14
15 /
```

Enter value for a\_no: 1

old 7: a\_no:=&a\_no;

new 7: a\_no:=1;

PL/SQL procedure successfully completed.

SQL> select \* from saccount;

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

ACCNO	NAME	BAL
1	mala	18000
2	kala	30000

### TO CREATE TABLE SROUTES

```
SQL> create table sroutes ( rno number(5), origin varchar2(20), destination varchar2(20), fare number(10), distance number(10));
```

Table created.

```
SQL> insert into sroutes values ( 2, 'chennai', 'dindugal', 400,230); 1
row created.
```

```
SQL> insert into sroutes values ( 3, 'chennai', 'madurai', 250,300); 1
row created.
```

```
SQL> insert into sroutes values ( 6, 'thanjavur', 'palani', 350,370);
1 row created.
```

```
SQL> select * from sroutes;
```

RNO	ORIGIN	DESTINATION	FARE	DISTANCE
2	chennai	dindugal	400	230
3	chennai	madurai	250	300
6	thanjavur	palani	350	370

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

```
SQL> set serveroutput on;
```

```
SQL> declare
```

```
2 route sroutes.rno % type;
```

```
3 fares sroutes.fare % type;
```

```
4 dist sroutes.distance % type;
```

```
5 begin
```

```
6 route:=&route;
```

```
7 select fare, distance into fares , dist from sroutes where rno=route;
```

```
8 if (dist < 250) then
```

```
9 update sroutes set fare=300 where rno=route;
```

```
10 else if dist between 250 and 370 then
```

```
11 update sroutes set fare=400 where rno=route;
```

```
12 else if (dist > 400) then
```

```
13 dbms_output.put_line('Sorry');
```

```
14 end if;
```

```
15 end if;
```

```
16 end if;
```

```
17 end;
```

```
18 /
```

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

Enter value for route: 3

old 6: route:=&route;

new 6: route:=3;

PL/SQL procedure successfully completed.

SQL> select \* from sroutes;

RNO	ORIGIN	DESTINATION	FARE	DISTANCE
2	chennai	dindugal	400	230
3	chennai	madurai	400	300
6	thanjavur	palani	350	370

### TO CREATE SCALCULATE TABLE

SQL> create table scalculate ( radius number(3), area number(5,2));

Table created.

SQL> desc scalculate;

Name	Type
------	------

RADIUS	NUMBER(3)
--------	-----------

AREA	NUMBER(5,2)
------	-------------

SQL> set serveroutput on;

SQL> declare

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

```
2  pi constant number(4,2):=3.14;
3  area number(5,2);
4  radius number(3);
5  begin
6  radius:=3;
7  while (radius <=7)
8  loop
9  area:= pi* power(radius,2);
10 insert into scalculate values (radius,area);
11 radius:=radius+1;
12 end loop;
```

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

```
13 end;
```

```
14 /
```

PL/SQL procedure successfully completed.

```
SQL> select * from scalculate;
```

RADIUS	AREA
--------	------

3	28.26
4	50.24
5	78.5
6	113.04
7	153.86

### TO CALCULATE FACTORIAL OF A GIVEN NUMBER

```
SQL> set serveroutput on;
```

```
SQL> declare
```

```
2  f number(4):=1;
```

```
3  i number(4);
```

```
4  begin
```

```
5  i:=&i;
```

```
6  while(i>=1)
```

```
7  loop
```

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

```
8  f:=f*i;
9  i:=i-1;
10 end loop;
11 dbms_output.put_line('The value is ' || f);
12 end;
13 /
```

Enter value for i: 5

old 5: i:=&i; new

5: i:=5;

The value is 120

PL/SQL procedure successfully completed.

### Ex No:13

#### TRIGGERS

##### TO CREATE THE TABLE 'ITEMPLS'

```
SQL> create table itempls (ename varchar2(10), eid number(5), salary number(10));
```

Table created.

```
SQL> insert into itempls values('xxx',11,10000); 1
```

row created.



## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

```
SQL> insert into itempls values('yyy',12,10500); 1
```

row created.

```
SQL> insert into itempls values('zzz',13,15500);
```

1 row created.

```
SQL> select * from itempls;
```

ENAME	EID	SALARY
-------	-----	--------

----- xxx

11	10000 yyy	12
----	-----------	----

10500 zzz	13	
-----------	----	--

15500

### **TO CREATE A SIMPLE TRIGGER THAT DOES NOT ALLOW INSERT UPDATE AND DELETE OPERATIONS ON THE TABLE**

```
SQL> create trigger ittrigg before insert or update or delete on itempls for each row
```

```
2 begin
```

```
3 raise_application_error(-20010,'You cannot do manipulation');
```

```
4 end;
```

```
5
```

```
6 /
```

Trigger created.

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

```
SQL> insert into itempls values('aaa',14,34000); insert  
into itempls values('aaa',14,34000)
```

\*

ERROR at line 1:

ORA-20010: You cannot do manipulation

ORA-06512: at "STUDENT.ITTRIGG", line 2

ORA-04088: error during execution of trigger 'STUDENT.ITTRIGG'

```
SQL> delete from itempls where ename='xxx'; delete from itempls  
where ename='xxx'
```

\*

ERROR at line 1:

ORA-20010: You cannot do manipulation

ORA-06512: at "STUDENT.ITTRIGG", line 2

ORA-04088: error during execution of trigger 'STUDENT.ITTRIGG'

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

```
SQL> update itempls set eid=15 where ename='yyy'; update  
itempls set eid=15 where ename='yyy'
```

\*

ERROR at line 1:

ORA-20010: You cannot do manipulation

ORA-06512: at "STUDENT.ITTRIGG", line 2

ORA-04088: error during execution of trigger 'STUDENT.ITTRIGG'**TO DROP THE  
CREATED TRIGGER**

---

```
SQL> drop trigger ittrigg;
```

Trigger dropped.

**TO CREATE A TRIGGER THAT RAISES AN USER DEFINED ERROR MESSAGE  
AND DOES NOT ALLOW UPDATION AND INSERTION**

```
SQL> create trigger ittriggs before insert or update of salary on itempls for each row
```

```
2 declare
```

```
3   triggsal itempls.salary%type;
```

```
4   begin
```

```
5   select salary into triggsal from itempls where eid=12;
```

```
6   if(:new.salary>triggsal or :new.salary<triggsal) then
```

```
7   raise_application_error(-20100,'Salary has not been changed');
```

```
8   end if;
```

```
9   end;
```

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

10 /

Trigger created.

```
SQL> insert into itempls values ('bbb',16,45000); insert  
into itempls values ('bbb',16,45000)
```

\*

ERROR at line 1:

ORA-04098: trigger 'STUDENT.ITTRIGGS' is invalid and failed re-validation

```
SQL> update itempls set eid=18 where ename='zzz'; update itempls set eid=18  
where ename='zzz'
```

\*

ERROR at line 1:

ORA-04298: trigger 'STUDENT.ITTRIGGS' is invalid and failed re-validation

### 13/8/2025 Function Example:

**DECLARE**

**a INT; b FLOAT;**

**myexp EXCEPTION;**

**FUNCTION sqroot(x INT)**

## CS23332 DATABASE MANAGEMENT SYSTEM LABORATORY

**RETURN FLOAT**

**AS answer  
FLOAT; BEGIN**

**IF  $x < 0$  THEN**

**RAISE myexp;  
ELSE answer :=  
SQRT(x); END IF;**

**RETURN answer;**

**EXCEPTION**

**WHEN myexp THEN**

**DBMS\_OUTPUT.PUT\_LINE('Square root of a negative number is not allowed, so  
returning the same number');**

**RETURN x;**

**END;**

**BEGIN**

**b := sqrt(4);**

**DBMS\_OUTPUT.PUT\_LINE('The value is ' || b);**

**END;**