# RAJALAKSHMI ENGINEERING COLLEGE

## RAJALAKSHMI NAGAR, THANDALAM – 602 105

RAJALAKSHMI
ENGINEERING COLLEGE

**CS19443**

**DATABASE MANAGEMENT SYSTEMS LABORATORY**

**Laboratory Manual Note Book**

Name : . . . . . .

Year / Branch / Section : . . . . . .

Register No. : . . . . . .

Semester : . . . . . .

Academic Year : . . . . . .

## Vision

To promote highly Ethical and Innovative Computer Professionals through excellence in teaching, training and research.

## Mission

• To produce globally competent professionals, motivated to learn the emerging technologies and to be innovative in solving real world problems.

• To promote research activities amongst the students and the members of faculty that could benefit the society.

• To impart moral and ethical values in their profession.

## PROGRAMME EDUCATIONAL OBJECTIVES (PEOs)

**PEO 1**:To equip students with essential background in computer science, basic electronics and applied mathematics.

**PEO 2**:To prepare students with fundamental knowledge in programming languages, and tools and enable them to develop applications.

**PEO 3**:To develop professionally ethical individuals enhanced with analytical skills, communication skills and organizing ability to meet industry requirements.

## PROGRAMME OUTCOMES (POs)

**PO1**: Engineering knowledge: Apply the knowledge of Mathematics, Science, Engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

**PO2**: Problem analysis: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

**PO3**: Design/development of solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

**PO4**: Conduct investigations of complex problems: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

**PO5**: Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.

**PO6**: The engineer and society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

**PO7**: Environment and sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

**PO8**: Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

**PO9**: Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

**PO10**: Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

**PO11**: Project management and finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

**PO12**: Life-long learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

**PROGRAM SPECIFIC OUTCOMES (PSOs)**

A graduate of the Computer Science and Design Program will have an

**PSO 1:** Ability to understand, analyze and develop efficient software solutions using suitable algorithms, data structures, and other computing techniques.

**PSO 2:** Ability to independently investigate a problem which can be solved by a Human Computer Interaction (HCI) design process and then design an end-to-end solution to it (i.e., from user need identification to UI design to technical coding and evaluation). Ability to effectively use suitable tools and platforms, as well as enhance them, to develop applications/products using for new media design in areas like animation, gaming, virtual reality, etc.

**PSO 3:** Ability to apply knowledge in various domains to identify research gaps and to provide solution to new ideas, inculcate passion towards higher studies, creating innovative career paths to be an entrepreneur and evolve as an ethically social responsible computer science and design professional.

**CO – PO and PSO matrices of course**

| PO/PSO CO | PO 1 | PO 2 | PO 3 | PO 4 | PO 5 | PO 6 | PO 7 | PO 8 | PO 9 | PO 10 | PO 11 | PO 12 | PSO 1 | PSO 2 | PSO 3 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CS19443.1 | 2 | 2 | 2 | - | - | - | - | - | 1 | - | - | 1 | 2 | 2 | - |
| CS19443.2 | 2 | 2 | 3 | 3 | 3 | - | - | - | 2 | 1 | 2 | 1 | 2 | 1 | - |
| CS19443.3 | 2 | 2 | 2 | 2 | 2 | - | - | - | 2 | 1 | 2 | 1 | 1 | 2 | 1 |
| CS19443.4 | 2 | 2 | 2 | 2 | 2 | - | - | - | 1 | 1 | - | - | 1 | 2 | 1 |
| CS19443.5 | 2 | 2 | 2 | 4 | 2 | - | - | - | 2 | - | 2 | 2 | 1 | 2 | 3 |
| **Average** | **2.0** | **2.0** | **2.2** | **2.8** | **2.3** | **-** | **-** | **-** | **1.6** | **1.0** | **2.0** | **1.3** | **1.4** | **1.8** | **1.7** |

| List of Experiments | |
|---|---|
| **1** | Introduction to SQL : DDL,DML,DCL,TCL.<br>SQL clause :SELECT FROM WHERE GROUPBY,HAVING,ORDERBY Using SQLite/MySQL/Oracle |
| **2** | Creation of Views, Synonyms, Sequence, Indexes, Save point. |
| **3** | Creating an Employee database to set various constraints and subqueries. |
| **4** | Optimize a SQL query construct considering time complexity. |
| **5** | Write a PL/SQL block to specify constraints by accepting input from the user. |
| **6** | Implementation of PL/SQL Procedure (IN, OUT, INOUT ) with Exception Handling. |
| **7** | Implementation of PL/SQL Function. |
| **8** | Implementation of PL/SQL Cursor. |
| **9** | Implementation of PL/SQL Trigger, Packages. |
| **10** | Implementation of NoSQL basic commands using Cassandra/Mongo DB. |
| **11** | Implementation of Data Model in NoSQL. |
| **12** | Implementation of Aggregation , Indexes in NoSQL. |
| **13** | **MINI PROJECT**<br>Database Connectivity with Front End Tools(Python/C/C++/JAVA)and Back End Tools(MySQL/SQLite/CASSANDRA/MONGO DB)<br>For any problem selected, write the ER Diagram, apply ER mapping rules, normalize the relations, and follow the application development process.<br>Make sure that the application should have five or more tables, at least one trigger and one stored procedure, using suitable frontend tool.<br>Indicative areas include<br>a) Inventory Control System.<br>b) Material Requirement Processing.<br>c) Hospital Management System.<br>d) Railway Reservation System.<br>e) Personal Information System.<br>f) Web Based User Identification System.<br>g) Timetable Management System.<br>h) Hotel Management System<br>i)Library Management System. |

| | | | |
|---|---|---|---|
| | **Contact Hours** | | **: 60** |
| | **Total Contact Hours** | | **: 90** |

Safety Precautions

- Regular Backups: Ensure regular backups of all databases to prevent data loss.
- Secure Passwords: Use complex and unique passwords for database access and change them regularly.
- Antivirus Protection: Install and maintain updated antivirus software on all laboratory computers.
- Data Encryption: Encrypt sensitive data both in transit and at rest to protect against data breaches.
- Software Updates: Keep all database management software and operating systems up to date with the latest security patches.
- Environment Control: Ensure proper environmental controls, such as temperature and humidity, to protect hardware.
- Power Protection: Use Uninterruptible Power Supplies (UPS) to prevent data loss due to power outages.

Dos:

- Regular Maintenance: Perform regular maintenance and updates on the database systems to ensure optimal performance.
- Documentation: Maintain comprehensive documentation of database structures, procedures, and security policies.
- Monitoring: Continuously monitor database performance and security to detect and respond to issues promptly.
- Training: Provide regular training to staff and students on database management best practices and security measures.
- Data Integrity: Implement and enforce data integrity constraints to maintain accurate and reliable data.

Don'ts

- Sharing Passwords: Do not share passwords or leave them written down in accessible places.
- Ignoring Errors: Do not ignore system errors or warnings; investigate and resolve them promptly.
- Unauthorized Software: Do not install unauthorized software on lab computers as it may pose security risks.
- Neglecting Backups: Do not neglect regular backups; always have a backup strategy in place.
- Weak Passwords: Do not use weak or easily guessable passwords.
- Bypassing Security: Do not bypass or disable security features for convenience.
- Unverified Sources: Do not download or install software from unverified sources as they may contain malware.
- Public Wi-Fi: Avoid accessing the database from public Wi-Fi networks to prevent unauthorized interception of data.

## INDEX

Reg. No. : _____ Name : _____

Year : _____ Branch : _____ Sec : ____

| S. No. | Date | Title | Page No. | Teacher's Signature / Remarks |
|---|---|---|---|---|
| 0 | | Introduction to RDBMS – study | | |
| 1 | | Creating and Managing Tables | | |
| 2 | | Manipulating Data | | |
| 3 | | Working with Columns, Characters, and Rows | | |
| 4 | | Including Constraints | | |
| 5 | | Writing Basic SQL SELECT Statements | | |
| 6 | | Restricting and Sorting data | | |
| 7 | | Single Row Functions | | |
| 8 | | Displaying data from multiple tables | | |
| 9 | | Aggregating Data Using Group Functions | | |
| 10 | | Sub queries | | |
| 11 | | Using The Set Operators | | |
| 12 | | NULL Functions | | |
| 13 | | CREATING VIEWS | | |

| | | | | |
|---|---|---|---|---|
| 14 | | Intro to Constraints; NOT NULL and UNIQUE Constraints | | |
| 15 | | Creating Views | | |
| 16 | | OTHER DATABASE OBJECTS | | |
| 17 | | Controlling User Access | | |
| 18 | | PROCEDURES AND FUNCTIONS PROCEDURES | | |
| 19 | | TRIGGER | | |
| 20 | | NoSQL-1 | | |
| | | NoSQL-2 | | |

**Ex. No. : 0** **Date:**

**Register No.:** **Name:**

---

**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.
- 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 |
| DML | 1.INSERT<br>2.UPDATE<br>3.DELETE<br>4.MERGE | Enter new rows, changes existing rows, removes unwanted rows from tables in the database respectively. |
| DDL | 1.CREATE<br>2.ALTER<br>3.DROP<br>4.RENAME<br>5.TRUNCATE | Sets up, changes and removes data structures from tables. |
| TCL | 1.COMMIT<br>2.ROLLBACK<br>3.SAVEPOINT | Manages the changes made by DML statements. Changes to the data can be |

| | | |
|---|---|---|
| | | grouped together into logical transactions. |
| DCL | 1.GRANT<br>2.RREVOKE | 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
- 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.



**Ex. No. : 1** **Date:**

**Register No.:** **Name:**

## Creating of Base Table and Managing Tables

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 MY_EMPLOYEE (ID number(4) not null,
Last_name varchar(25), First_name varchar(25),
Userid varchar(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 |

9

```
insert into MY_EMPLOYEE values (1, 'Patel' ,' Ralph','rpatel' , 895);
insert into MY_EMPLOYEE values (2, 'Dancs', 'Betty', 'bdancs', 860);
```

3. Display the table with values.

```
select * from MY_EMPLOYEES;
```

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,salary) values
        (3, 'Biri', 'Ben', 1100);
insert into MY_EMPLOYEE(ID, Last_name,First_name,salary) values
        (4 'Newman', 'Chad', 750);
update MY_EMPLOYEE set Userid =
concat(substr(First_name,1,1),substr(Last_name,1,7)) where Userid is null;
```

5. Delete Betty dancs from MY _EMPLOYEE table.

```
delete from MY_EMPLOYEE  where First_name = 'Betty';
```

6. Empty the fourth row of the emp table.

```
delete from MY_EMPLOYEE where id = 4;
```

7. Make the data additions permanent.

```
commit;
```

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

```
update MY_EMPLOYEE set Last_name = 'Drexler' where id=3;
```

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

```
update MY_EMPLOYEE set salary = 1000 where salary<900;
```



**Ex. No. : P-1 Date:**

**Register No.: Name:**

---

## DATA MANIPULATIONS

**Create the following table 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) |
| 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 | First_Name | Last_Name | Email | Phone_Number | Hire_Date | Job_ID | Salary | Commission_Pct | Manager_ID | Department_ID |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | John | Doe | johndoe@example.com | 555-5555 | 1/1/2023 | IT_PROG | 5000 | NULL | 100 | 60 |
| 2 | Jane | Austin | janeaustin@example.com | 555-5556 | 2/1/2023 | SA_REP | 6000 | 0.1 | 101 | 70 |
| 3 | Mike | Smith | mikesmith@example.com | 555-5557 | 3/1/2023 | AD_VP | 7000 | 0.15 | 102 | 80 |
| 4 | Anna | Austin | annaaustin@example.com | 555-5558 | 4/1/2023 | FI_MGR | 4800 | 0.2 | 103 | 60 |
| 5 | Bob | Brown | bobbrown@example.com | 555-5559 | 5/1/2023 | MK_MAN | 4500 | NULL | 104 | 70 |
| 6 | Alice | Johnson | alicejohnson@example.com | 555-5560 | 6/1/2023 | HR_REP | 5500 | 0.05 | 100 | 60 |
| 7 | Steve | Wilson | stevewilson@example.com | 555-5561 | 7/1/2023 | IT_PROG | 5200 | NULL | 100 | 80 |
| 8 | Laura | White | laurawhite@example.com | 555-5562 | 8/1/2023 | AD_ASST | 4700 | NULL | 105 | 70 |
| 9 | David | Harris | davidharris@example.com | 555-5563 | 9/1/2023 | MK_REP | 5100 | 0.1 | 101 | 60 |
| 10 | Emma | Martinez | emmarmartinez@example.com | 555-5564 | 10/1/2023 | SA_MAN | 4900 | NULL | 104 | 80 |



a) Find out the employee id, names, salaries of all the employees

```
select employee_id, First_name, Last_name, salary from EMPLOYEES;
```

b) List out the employees who works under manager 100

**c)** Find the names of the employees who have a salary greater than or equal to 4800

select * from employees where salary>=4800;

d) List out the employees whose last name is _AUSTIN‘

select * from employees where Last_name = ‘AUSTIN’;

e) Find the names of the employees who works in departments 60,70 and 80

select * from employees where Department_id is in (60,70,80);

f) Display the unique Manager_Id.

**Ex. No. : 2** **Date:**

**Register No.:** **Name:**

---

## 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
- 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 employlees ( employee_id number(6), first_name varchar2(20), ..job_id varchar2(10),
CONSTRAINT emp_emp_id_pk PRIMARY KEY (employlee_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;

**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_nname);

*ALTER TABLE ADD CONSTRAINT Constraint_name PRIMARY KEY (Colum_Name);*

**Example:**

Alter table dept80 add (jod_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;

**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) |

| | | |
|---|---|---|
| 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) |

```
create table EMPLOYEES (
    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),
    primary key (employee_id)
);
```

**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),  primary key (dept_id)
);
```

**JOB_GRADE TABLE**

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

```
create table JOB_GRADE (
    grade_level varchar2(2),
    lowest_sal number,  highest_sal number,
    primary key (grade_level)
);
```

**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),
   primary key (location_id) );
```

1. Create the DEPT table based on the DEPARTMENT following the table instance chart below. 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) primary key,  name varchar2(25)
);
```

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 EMP (
    id number(7) primary key,
    last_name varchar2(25),
    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,
            last_name, salary,
            dept_id
        from EMPLOYEES;
```

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.

```
-- Table DEPT and EMP
desc DEPT;
desc EMP;
```

8. Drop the First_name column from the EMP table and confirm it.

```
alter table EMP drop column first_name;
desc EMP;
```

**Ex. No. : 2** **Date:**

**Register No.:** **Name:**

---

## 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**

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);

**Copying Rows from another table**

- Using Subquery

**Example:**

INSER INTO sales_reps(id, name, salary, commission_pct)
SELECT employee_id, Last_name, salary, commission_pct
FROM employees WHERE jod_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 updae 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'';

**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 MY_EMPLOYEE (ID number(4) not null,

Last_name varchar(25), First_name varchar(25),

Userid varchar(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 values (1, 'Patel' ,' Ralph','rpatel' , 895);
insert into MY_EMPLOYEE values (2, 'Dancs', 'Betty', 'bdancs', 860);
```

3. Display the table with values.

```
select * from MY_EMPLOYEES;
```

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,salary) values
```

(3, ‘Biri’, ‘Ben’, 1100);

insert into MY_EMPLOYEE(ID, Last_name,First_name,salary) values

(4 ‘Newman’, ‘Chad’, 750);

update MY_EMPLOYEE set Userid =

concat(substr(First_name,1,1),substr(Last_name,1,7)) where Userid is null;

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 dancs from MY _EMPLOYEE table.

```
delete from MY_EMPLOYEE  where First_name = ‘Betty’;
```

9. Empty the fourth row of the emp table.

```
delete from MY_EMPLOYEE where id = 4;
```

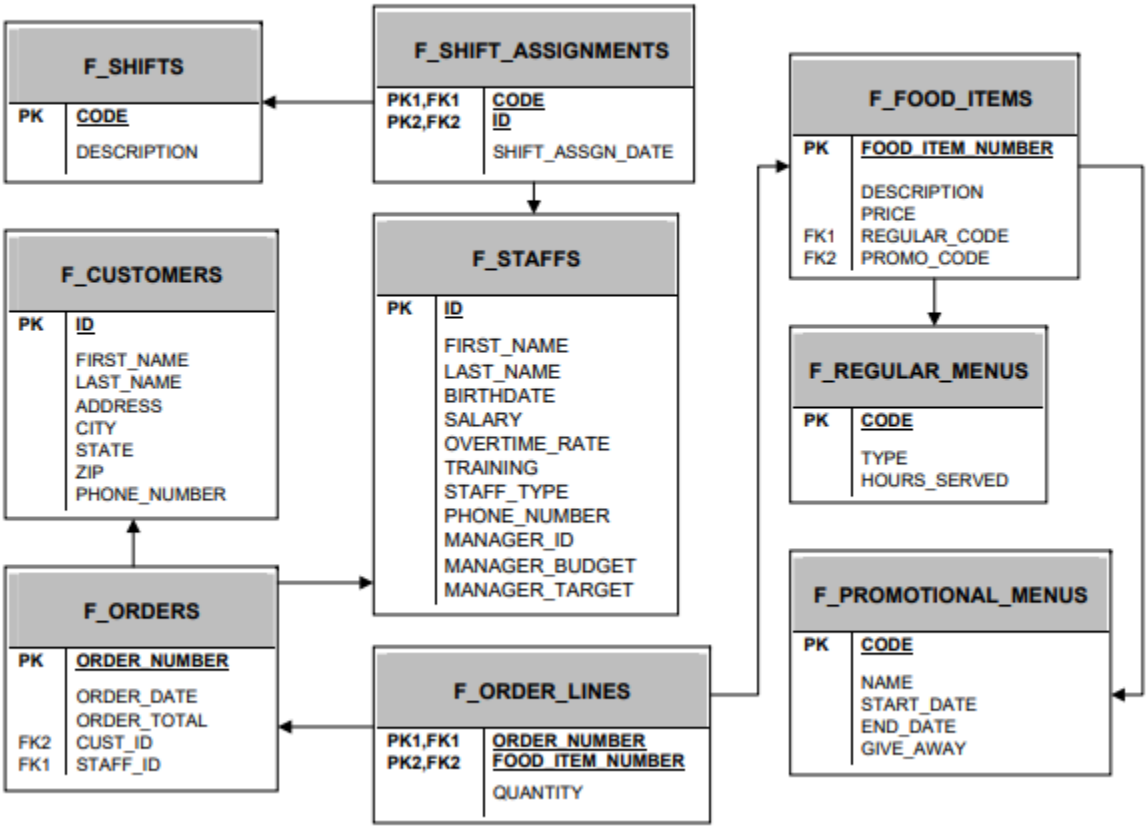
**Ex. No. : 3 Date:**

**Register No.: Name:**

---

## Working With Column, Characters and rows

### Global Fast Foods Database Tables




1. The manager of Global Fast Foods would like to send out coupons for the upcoming sale. He wants to send one coupon to each household. Create the SELECT statement that returns the customer last name and a mailing address.

```
SELECT DISTINCT HouseholdID, LastName, Address FROM Customers;
```

2. Each statement below has errors. Correct the errors and execute the query in Oracle Application Express.

a.

SELECT first name FROM f_staffs;

b.

SELECT first_name |" " | last_name AS "DJs on Demand Clients" FROM d_clients;

c.

SELECT DISCTINCT f_order_lines FROM quantity;

d.

SELECT order number FROM f_orders;

a. SELECT first_name FROM f_staffs;
b. SELECT first_name || ' ' || last_name AS "DJs on Demand Clients" FROM d_clients;
c. SELECT DISTINCT f_order_lines FROM quantity;
d. SELECT order_number FROM f_orders;

3. Sue, Bob, and Monique were the employees of the month. Using the f_staffs table, create a SELECT statement to display the results as shown in the Super Star chart.

| Super Star |
|---|
| *** Sue *** Sue *** |
| *** Bob *** Bob *** |
| *** Monique *** Monique *** |

```
select '*** ' || first_name || ' *** ' || first_name || ' ***' as "Super Star" from f_staffs where first_name in ('Sue', 'Bob', 'Monique');
```

4.Which of the following is TRUE about the following query?

SELECT first_name, DISTINCT birthdate

FROM f_staffs;

a. Only two rows will be returned.

b. Four rows will be returned.

c. Only Fred 05-Jan-1988 and Lizzie 10-Nov-1987 will be returned.

d. No rows will be returned.

| |
|---|
| d.No rows will be returned. |

5. Global Fast Foods has decided to give all staff members a 5% raise. Prepare a report that presents the output as shown in the chart.

| EMPLOYEE LAST NAME | CURRENT SALARY | SALARY WITH 5% RAISE |
|---|---|---|
| | | |

```
select last_name as "EMPLOYEE LAST NAME",   salary as "CURRENT SALARY",
salary*1.05 as "SALARY WITH 5% RAISE" from  f_staffs;
```

6. Create a query that will return the structure of the Oracle database EMPLOYEES table. Which columns are marked “nullable”? What does this mean?

```
DESC EMPLOYEES;
```

If a column is marked as "nullable," it means that the column can contain NULL values , i.e., it is not mandatory to have a value in this column for every record in the table. If it is marked as "not nullable," every record must contain a value in that column.

7. The owners of DJs on Demand would like a report of all items in their D_CDs table with the following column headings: Inventory Item, CD Title, Music Producer, and Year Purchased. Prepare this report.

```
select inventory_item AS "Inventory Item",  cd_title AS "CD Title",  music_producer AS
    "Music   Producer",  year_purchased AS "Year Purchased"  from D_CDs;
```

8.True/False – The following SELECT statement executes successfully:
SELECT last_name, job_id, salary AS Sal FROM employees;

True

9.True/False – The following SELECT statement executes successfully:
SELECT * FROM job_grades;

True

10.There are four coding errors in this statement. Can you identify them?

SELECT employee_id, last_name sal x 12 ANNUAL SALARY FROM employees;

Query after Correction of 4 Error:

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

11.In the arithmetic expression salary*12 - 400, which operation will be evaluated first?

salary * 12 will be executed first.

12. Which of the following can be used in the SELECT statement to return all columns of data in the Global Fast Foods f_staffs table?

a. column names

b. *

c. DISTINCT id

d. both a and b

d. both a and b

13. Using SQL to choose the columns in a table uses which capability?

a. selection

b. projection

c. partitioning

d. join

b. projection

14. SELECT last_name AS "Employee". The column heading in the query result will appear as:

a. EMPLOYEE

b. employee

c. Employee

d. "Employee:

b. Employee

15. Which expression below will produce the largest value?

a. SELECT salary*6 + 100

b. SELECT salary* (6 + 100)

c. SELECT 6(salary+ 100)

d. SELECT salary+6*100

b. SELECT salary* (6 + 100)

16. Which statement below will return a list of employees in the following format?
Mr./Ms. Steven King is an employee of our company.

a. SELECT "Mr./Ms."||first_name||' '||last_name 'is an employee of our company.' AS "Employees" FROM employees;

b. SELECT 'Mr./Ms. 'first_name,last_name ||' '||'is an employee of our company.' FROM employees;

c. SELECT 'Mr./Ms. '||first_name||' '||last_name ||' '||'is an employee of our company.' AS "Employees" FROM employees ;

d. SELECT Mr./Ms. ||first_name||' '||last_name ||' '||"is an employee of our company." AS "Employees" FROM employees

c. SELECT 'Mr./Ms. '||first_name||' '||last_name ||' '||'is an employee of our company.' AS "Employees" FROM employees ;

17. Which is true about SQL statements?

a. SQL statements are case-sensitive

b. SQL clauses should not be written on separate lines.

c. Keywords cannot be abbreviated or split across lines.

d. SQL keywords are typically entered in lowercase; all other words in uppercase.

c. Keywords cannot be abbreviated or split across lines.

18. Which queries will return three columns each with UPPERCASE column headings?

a. SELECT "Department_id", "Last_name", "First_name" FROM employees;

b. SELECT DEPARTMENT_ID, LAST_NAME,
FIRST_NAME FROM employees;

c. SELECT department_id, last_name, first_name AS UPPER
CASE FROM employees

d. SELECT department_id, last_name, first_name
FROM employees;

b. SELECT DEPARTMENT_ID, LAST_NAME, FIRST_NAME FROM employees;

19. Which statement below will likely fail?

a. SELCT * FROM employees;

b. Select * FROM employees;

c. SELECT * FROM EMPLOYEES;

d. SelecT* FROM employees;

[redacted]

20. Click on the History link at the bottom of the SQL Commands window. Scroll or use the arrows at the bottom of the page to find the statement you wrote to solve problem 3 above. (The one with the column heading SuperStar). Click on the statement to load it back into the command window. Execute the command again, just to make sure it is the correct one that works. Once you know it works, click on the SAVE button in the top right corner of the SQL Commands window, and enter a name for your saved statement. Use your own initials and "_superstar.sql", so if your initials are CT then the filename will be CT_superstar.sql.

Log out of OAE, and log in again immediately. Navigate back to the SQL Commands window, click the Saved SQL link at the bottom of the page and load your saved SQL statement into the Edit window. This is done by clicking on the script name. Edit the statement, to make it display
+ instead of *. Run your amended statement and save it as initials_superplus.sql.

Save as: <Your Initials>_superstar.sql (e.g., CT_superstar.sql for initials CT).
Edit and Save as: <Your Initials>_superplus.sql (e.g., CT_superplus.sql for initials CT).

**Ex. No. : 4 Date:**

**Register No.: Name:**

---

## 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**

Create table tablename (column_name1 data_ type constraints, column_name2 data_ type constraints …);

**Example:**

Create table employlees ( employee_id number(6), first_name varchar2(20), ..job_id varchar2 (10), CONSTRAINT emp_emp_id_pk PRIMARY KEY (employlee_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:**

- Setting null value is appropriate when the actual value is unknown, or when a value would not be meaningful.
- A null value is not equivalent to a value of zero.
- A null value will always evaluate to null in any expression.
- When a column name is defined as not null, that column becomes a mandatory i.e., the user has to enter data into it.
- 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**

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' COSTRAINT 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**

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 deparments(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);

**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**

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, CONTRAINT 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';

**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);
```

2. Create a PRIMAY KEY constraint to the DEPT table using the ID colum. 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 deparment. 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.

```
alter table EMP add COMMISSION number(2,2);
alter table EMP add constraint chk_commission check(commission>0);
```

**Ex. No. : 5 Date:**

**Register No.: Name:**

---

## 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

### Basic SELECT Statement

**Syntax**

SELECT *|DISTINCT Column_ name| alias

` FROM table_name;

**NOTE:**

DISTINCT—Suppr

ess 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 sued 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;

**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 deparment_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.

SELECT employee_id, last_name

sal*12 ANNUAL SALARY

FROM employees;

| .False<br>"As" keyword and a comma are missing. |
|---|

**Identify the Errors**

SELECT employee_id, last_name

sal*12 ANNUAL SALARY

FROM employees;

**Queries**

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

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

describe 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;
```

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;
```

**Ex. No. : P-2 Date:**

**Register No.: Name:**

---

## COMPARISON OPERATORS

1. Who are the partners of DJs on Demand who do not get an authorized expense amount?

```
SELECT partner_name FROM Partners WHERE company = 'DJs on Demand'
  AND (authorized_expense_amount IS NULL OR authorized_expense_amount = 0);
```

2. Select all the Oracle database employees whose last names end with “s”. Change the heading of the column to read Possible Candidates.

```
select * from employees where Last_name like ‘%s’;
```

3. Which statement(s) are valid?

a. WHERE quantity <> NULL;

b. WHERE quantity = NULL;

c. WHERE quantity IS NULL;

d. WHERE quantity != NULL;

c.WHERE quantity IS NULL;

4. Write a SQL statement that lists the songs in the DJs on Demand inventory that are type code 77, 12, or 1.

```
select song_name from songs where type_code in (77,12,1);
```

## Logical Comparisons and Precedence Rules

1. Execute the two queries below. Why do these nearly identical statements produce two different results? Name the difference and explain why.

```
SELECT code, description
FROM d_themes
WHERE code >200 AND description IN('Tropical', 'Football', 'Carnival');
SELECT code, description
FROM d_themes
    WHERE code >200 OR description IN('Tropical', 'Football', 'Carnival');
```

The difference between these two queries is because of `AND` and `OR` in the `WHERE` clause:

First Query (using `AND`): Here, both conditions have to be true. So, it will only show rows where the `code` is over 200 **and** the `description` is either `''Tropical''`, `''Football''`, or `''Carnival''`. This means it's more restrictive, and fewer rows will be returned.

Second Query (using `OR`): Here, only one of the conditions needs to be true. It will show rows where either the `code` is over 200 or the

`description` is one of those values. This makes it less restrictive, so more rows will be returned because just one of the conditions needs to match.

2. Display the last names of all Global Fast Foods employees who have “e” and “i” in their last names.

```
select last_name from f_staffs where last_name like ‘%e%’ or last_name  like ‘%i%’;
```

3. “I need to know who the Global Fast Foods employees are that make more than $6.50/hour and their position is not order taker.”

```
1select First_name , Last_name, salary, staff_type form f_staffs
where salary > 6.50 and staff_type <> 'order taker';
```

4. Using the employees table, write a query to display all employees whose last names start with “D” and have “a” and “e” anywhere in their last name.

```
select lower(first_name) as first_name, lower(last_name) as last_name from f_staffs
where last_name like 'd%' and last_name like '%a%'  and last_name like '%e%';
```

5. In which venues did DJs on Demand have events that were not in private homes?

```
SELECT DISTINCT venue_name FROM Events
WHERE company = 'DJs on Demand' AND venue_type != 'private home';
```

6. Which list of operators is in the correct order from highest precedence to lowest precedence?

a. AND, NOT, OR

b. NOT, OR, AND

c. NOT, AND, OR

c.NOT, AND, OR

**For questions 7 and 8, write SQL statements that will produce the desired output.**

7. Who am I?

I was hired by Oracle after May 1998 but before June of 1999. My salary is less than $8000 per month, and I have an “en” in my last name.

8. What's my email address?

Because I have been working for Oracle since the beginning of 1996, I make more than $9000 per month. Because I make so much money, I don't get a commission

**Ex. No. : 6 Date:**

**Register No.: Name:**

---

## 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 used in WHERE clause

### Syntax

SELECT----------

FROM----------

WHERE condition;

**Example:**

SELECT employee_id,last_name, job_id, deparment_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, deparment_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;

**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**

**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**

| 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

**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 )

```
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('1998-02-20', 'YYYY-MM-DD') and to_date('1998-05-01', 'YYYY-MM-DD')
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 last_name,  department_id from EMPLOYEES where department_id in (20, 50)
order by last_name 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 to_char(hire_date, 'YYYY') like
'1994';
```

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

```
select last_name,  job_id  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 nul,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 ('Sales
Representative', 'Stock 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;
```

**Ex. No. : P-3 Date:**

**Register No.: Name:**

---

## Sorting Rows

1. In the example below, assign the employee_id column the alias of "Number." Complete the SQL statement to order the result set by the column alias.

SELECT employee_id, first_name, last_name FROM employees;

```
SELECT employee_id AS "Number", first_name, last_name
FROM EMPLOYEES
ORDER BY "Number";
```

2. Create a query that will return all the DJs on Demand CD titles ordered by year with titles in alphabetical order by year.

```
select title, year from CD  where category = 'DJs on Demand'
order by year, title;
```

3. Order the DJs on Demand songs by descending title. Use the alias "Our Collection" for the song title.

```
select title as "Our Collection", year from CD where category = 'DJs on Demand'
order by title desc;
```

4. Write a SQL statement using the ORDER BY clause that could retrieve the information needed.

```
select title as "Our Collection", year from CD
where category = 'djs on demand' order by title desc;
```

**Ex. No. : 7 Date:**

**Register No.: Name:**

---

## 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.

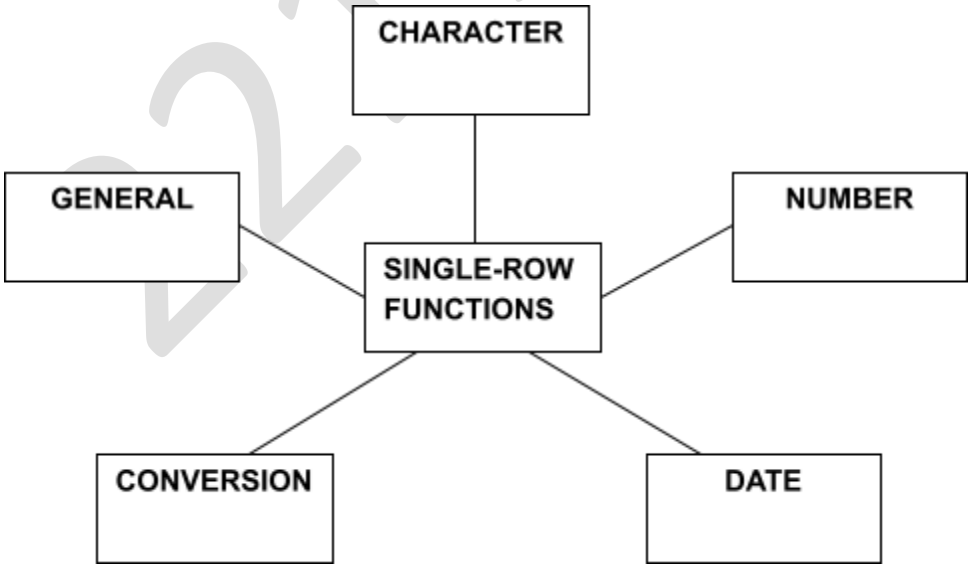
Accept arguments which can be a column or an expression

**Syntax**

Function_name(arg1,…argn)

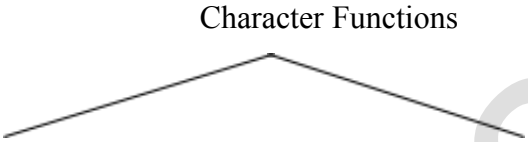
An argument can be one of the following

- ✔ User-supplied constant
- ✔ Variable value
- ✔ Column name
- ✔ Expression



- 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**




Case-manipulation functions

1. Lower
2. Upper
3. Initcap

Character-manipulation functions

1. Concat
2. Substr
3. Length
4. Instr
5. Lpad/Rpad
6. Trim
7. Repalce

| **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 |
| 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 |
| replace(text, search_string, replacement_string) | |

**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); | *select initcap("hello") from dual;* | Hello |
| lower (char);<br>upper (char); | *select lower ('HELLO') from dual;*<br>*select upper ('hello') from dual;* | Hello<br>HELLO |
| ltrim (char,[set]); | *select ltrim ('cseit', 'cse') from dual;* | IT |
| rtrim (char,[set]); | *select rtrim ('cseit', 'it') from dual;* | CSE |
| replace (char,search string, replace string); | *select replace ('jack and jue', 'j', 'bl') from dual;* | black and blue |

| | | |
|---|---|---|
| substr (char,m,n); | *select substr (‘information’, 3, 4) from dual;* | 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;

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 | Subtracts a number of days from a date |
| date – date | Number of days | Subtracts one date from another |
| date + number/24 | Date | Adds a number of hours to a date |

**Example**

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

**Date Functions**

| Function | Result |
|---|---|
| MONTHS_BETWEEN | Number of months between two dates |
| ADD_MONTHS | Add calendar months to date |
| NEXT_DAY | Next day of the date specified |
| LAST_DAY | Last day of the month |
| ROUND | Round date |
| TRUNC | Truncate date |

**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**

| Function | Result |
|---|---|
| MONTHS_BETWEEN ('01-SEP-95','11-JAN-94') | 19.6774194 |
| ADD_MONTHS ('11-JAN-94',6) | '11-JUL-94' |
| NEXT_DAY ('01-SEP-95','FRIDAY') | '08-SEP-95' |
| LAST_DAY ('01-FEB-95') | '28-FEB-95' |

**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;
```

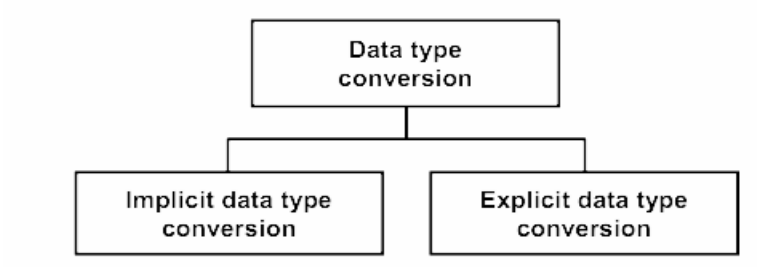
## Conversion Functions

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:

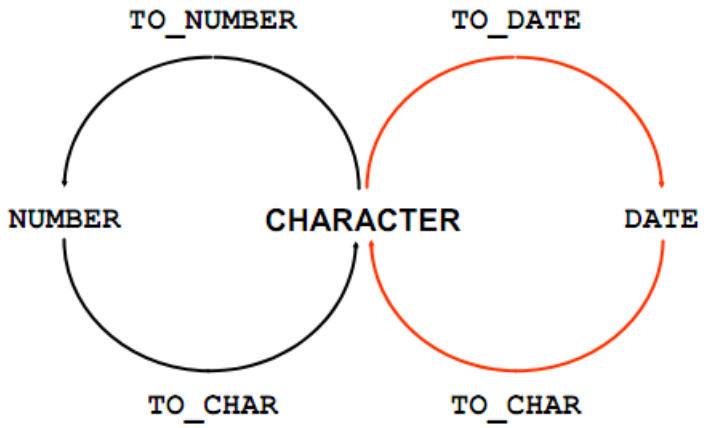
| From | To |
|---|---|
| VARCHAR2 or CHAR | NUMBER |
| VARCHAR2 or CHAR | DATE |
| NUMBER | VARCHAR2 |
| DATE | VARCHAR2 |

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:

| From | To |
|---|---|
| VARCHAR2 or CHAR | NUMBER |
| VARCHAR2 or CHAR | DATE |

## 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

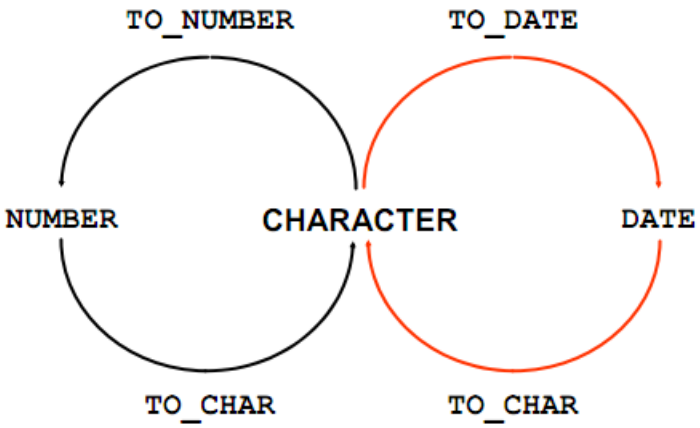
• Can include any valid date format element

• 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';
```

**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.

| 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:

| Element | Result |
|---|---|
| 9 | Represents a number |
| 0 | Forces a zero to be displayed |
| $ | Places a floating dollar sign |
| L | Uses the floating local currency symbol |
| . | Prints a decimal point |
| , | Prints a comma as thousands indicator |

**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 *n* times (*n* = 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:
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.

```
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 "Formatted Last Name",  length(last_name) as "Last Name
Length"  from EMPLOYEES  where last_name like 'J%' or last_name like 'A%' or last_name
like '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 initcap(last_name) as "Formatted Last Name",  length(last_name) as "Last Name
Length" from EMPLOYEES where last_name like upper(?) || '%' 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.

```
select last_name,  ceil(months_between(sysdate, hire_date)) as "MONTHS_WORKED"
from EMPLOYEES  order by MONTHS_WORKED;
```

**Note:** Your results will differ.

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 ' || salary || ' monthly but wants ' || (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('$' || to_char(salary, '999999999.99'), 15, '$') 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, hire_date,  to_char ( next_day(add_months(hire_date, 6), 'MONDAY'),
 'Day, the FMDDth of FMMonth, 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, 'Day') as "DAY"
from EMPLOYEES  order by case to_char(hire_date, 'D')
 when '2' then 1  -- Monday
 when '3' then 2  -- Tuesday
 when '4' then 3  -- Wednesday
 when '5' then 4  -- Thursday
 when '6' then 5  -- Friday
 when '7' then 6  -- Saturday
 when '1' then 7  -- Sunday
end;
```

**Ex. No. : P-4 Date:**

**Register No.: Name:**

---

## Introduction to Functions

1. For each task, choose whether a single-row or multiple row function would be most appropriate:

a. Showing all of the email addresses in upper case letters

b. Determining the average salary for the employees in the sales department

c. Showing hire dates with the month spelled out *(September 1, 2004)*

d. Finding out the employees in each department that had the most seniority (the earliest hire date)

e. Displaying the employees' salaries rounded to the hundreds place

f. Substituting zeros for null values when displaying employee commissions.

1. Single-row
2. Multiple-row
3. Single-row
4. Multiple-row
5. Single-row
6. Single-row

2. The most common multiple-row functions are: AVG, COUNT, MAX, MIN, and SUM. Give your own definition for each of these functions.

1. AVG: Calculates the average value of a numeric column by summing the values and dividing
by the count of non-null entries.
2. COUNT: Returns the number of rows that meet a specified condition or the number of non-null entries in a column.
3. MAX: Identifies the highest value in a specified column across all rows.
4. MIN: Determines the lowest value in a specified column across all rows.

5. SUM: Adds together all values in a specified numeric column.

3. Test your definitions by substituting each of the multiple-row functions in this query.

SELECT FUNCTION(salary)

FROM employees

Write out each query and its results.

```
SELECT SUM(salary) FROM employees;
SELECT AVG(salary) FROM employees;
SELECT MAX(salary) FROM employees;
SELECT MIN(salary) FROM employees;
SELECT COUNT(salary) FROM employees;
```

## Case and Character Manipulation

1. Using the three separate words “Oracle,” “Internet,” and “Academy,” use one command to produce the following output:

The Best Class Oracle Internet Academy

```
select 'The Best Class ' || 'Oracle ' || 'Internet ' || 'Academy' AS "Output" from dual;
```

2. Use the string “Oracle Internet Academy” to produce the following output:

```
The Net net
```

3. What is the length of the string “Oracle Internet Academy”?

```
23
```

4. What’s the position of “I” in “Oracle Internet Academy”?

```
8
```

5. Starting with the string “Oracle Internet Academy”, pad the string to create ****Oracle****Internet****Academy****

```
select  '****' || ‘Oracle’ || '****'|| ‘Internet’ || ‘Academy’  || '****'  from dual;
```

## Number Functions

1. Display Oracle database employee last_name and salary for employee_ids between 100 and 102. Include a third column that divides each salary by 1.55 and rounds the result to two decimal places.

```
SELECT last_name, salary, ROUND(salary / 1.55, 2) AS adjusted_salary FROM employees
WHERE employee_id BETWEEN 100 AND 102;
```

2. Display employee last_name and salary for those employees who work in department 80. Give each of them a raise of 5.333% and truncate the result to two decimal places.

```
SELECT last_name, salary, TRUNC(salary * 1.05333, 2) AS new_salary FROM employees
WHERE department_id = 80;
```

3. Use a MOD number function to determine whether 38873 is an even number or an odd number.

```
SELECT CASE WHEN MOD(38873, 2) = 0 THEN 'Even'
    ELSE 'Odd' END AS number_type;
```

4. Use the DUAL table to process the following numbers:

845.553 - round to one decimal place

30695.348 - round to two decimal places

30695.348 - round to -2 decimal Places 2.3454 -

truncate the 454 from the decimal place

```
SELECT ROUND(845.553, 1) AS
rounded_one_decimal,ROUND(30695.
348, 2) AS rounded_two_decimals,
   ROUND(30695.348, -2) AS
rounded_negative_two_decimals,TRU
NC(2.3454, 2) AS truncated_decimal
FROM  DUAL;
```

5. Divide each employee's salary by 3. Display only those employees' last names and salaries who earn a salary that is a multiple of 3.

```
SELECT last_name, salary FROM employees
WHERE MOD(salary, 3) = 0;
```

6. Divide 34 by 8. Show only the remainder of the division. Name the output as EXAMPLE.

```
SELECT  MOD(34, 8) AS EXAMPLE;
```

7. How would you like your paycheck – rounded or truncated? What if your paycheck was calculated to be $565.784 for the week, but you noticed that it was issued for $565.78. The loss of .004 cent would probably make very little difference to you. However, what if this was done to a thousand people, a 100,000 people, or a million people! Would it make a difference then? How much difference?

The difference of $0.004 per person may not seem much at first, but when applied to many people, it adds up quickly. For example, if 1,000 people are affected, the total loss is $4. For 100,000 people, the loss becomes $400, and for 1,000,000 people, it's $4,000. This shows that while the loss per person is small, when multiplied by a large number of people, it can result in a significant amount of money being lost, which is why rounding is usually preferred in financial systems.

**Ex. No. : 8** **Date:**

**Register No.:** **Name:**

---

## 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 displays employee last name and department name from the EMPLOYEES and DEPARTMENTS tables.

SELECT last_name, department_name dept_name

FROM employees, departments;

**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.

Note: Equijoins are also called simple joins or inner joins

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

**Non-Equijoins**

A non-equijoin 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-equijoin. 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 ;

**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**

**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.

• 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:**

```
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
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 alslso 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  last_name,  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,   d.location_id from EMPLOYEES e join DEPARTMENTS d
on e.department_id = d.department_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, d.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;
```

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, e.job_id, e.department_id, d.department_name from
EMPLOYEES e join DEPARTMENTS d on e.department_id = d.department_id
where d.location_id = (select location_id from LOCATIONS where 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;
```

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 Last Name",  e1.department_id as "Department Number",  e2.last_name as "Colleagues" from EMPLOYEES e1 join EMPLOYEES e2 on e1.department_id = e2.department_id where e1.employee_id = :given_employee_id;
```

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 "Employee Name",  e.job_id as "Job", d.department_name as "Department Name", e.salary as "Salary",   j.grade_level as "Grade" from EMPLOYEES e join DEPARTMENTS d on e.department_id = d.department_id join JOB_GRADES j on e.salary between j.lowest_sal and j.highest_sal;
```

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

```
        select last_name as "Employee Name",   hire_date as "Hire Date" from EMPLOYEES where hire_date > (select hire_date from EMPLOYEES where last_name = 'Davies');
```

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;

**Ex. No. : 9 Date:**

**Register No.: Name:**

---

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

• 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(*)
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
[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 ;
```

**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_idHAVING MAX(salary)>10000 ;

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

True

2. Group functions include nulls in calculations.

True/False

True

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

True/False

True

**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_id,  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_id;
```

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_id, count(*) as number_of_employees from EMPLOYEES where
        job_id = :job_title group by job_id;
```

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
where manager_id is not null;
```

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

```
        select 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, 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 hire_date like '1995%' then 1 end) as employees_hired_1995,
count(case when hire_date like '1996%' then 1 end) as employees_hired_1996,
 count(case when hire_date like '1997%' then 1 end) as employees_hired_1997,
count(case when hire_date like '1998%' then 1 end) as employees_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_id as Job, department_id as Department, salary as Salary,
sum(salary) over (partition by job_id) as Total_Salary from EMPLOYEES
where department_id in (20, 50, 80, 90);
```

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 as Name,  d.location_id as 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_id;
```

**Ex. No. : P-5 Date:**

**Register No.: Name:**

## Date Functions

1. For DJs on Demand, display the number of months between the event_date of the Vigil wedding and today's date. Round to the nearest month.

```
select round(months_between(sysdate, event_date)) as months_between from events
where event_name = 'vigil wedding';
```

2. Display the days between the start of last summer's school vacation break and the day school started this year. Assume 30.5 days per month. Name the output "Days."

```
select round(months_between(start_date_this_year, start_date_last_summer) * 30.5) as days
from school_calendar;
```

3. Display the days between January 1 and December 31.

```
select round(months_between(date 'yyyy-12-31', date 'yyyy-01-01') * 30.5) as days from dual;
```

4. Using one statement, round today's date to the nearest month and nearest year and truncate it to the nearest month and nearest year. Use an alias for each column.

```
select
round(sysdate, 'MM') as rounded_month,
round(sysdate, 'YY') as rounded_year,
trunc(sysdate, 'MM') as truncated_month,
trunc(sysdate, 'YY') as truncated_year from dual;
```

5. What is the last day of the month for June 2005? Use an alias for the output.

```
select last_day(date '2005-06-01') as last_day_of_june from dual;
```

6.Display the number of years between the Global Fast Foods employee Bob Miller's birthday and today. Round to the nearest year.

```
select round(months_between(sysdate, birth_date) / 12) as years_between from employees
where first_name = 'Bob' and last_name = 'Miller';
```

7.Your next appointment with the dentist is six months from today. On what day will you go to the dentist? Name the output, "Appointment."

```
select add_months(sysdate, 6) as appointment from dual;
```

8.The teacher said you have until the last day of this month to turn in your research paper. What day will this be? Name the output, “Deadline.”

```
select last_day(sysdate) as deadline from dual;
```

9.How many months between your birthday this year and January 1 next year?

```
select months_between(date 'yyyy-01-01', date 'yyyy-mm-dd') as months_between from dual;
```

10.What’s the date of the next Friday after your birthday this year? Name the output, “First Friday.”

```
select next_day(date 'yyyy-mm-dd', 'FRIDAY') as "First Friday" from dual;
```

11. Name a date function that will return a number.

```
select months_between(date '2024-12-31', date '2024-01-01') as months_difference from dual;
```

12.Name a date function that will return a date.

```
select add_months(date '2024-01-01', 3) as new_date from dual;
```

13.Give one example of why it is important for businesses to be able to manipulate date data?

It’s important for businesses to manipulate date data for financial forecasting Analyzing historical date data helps identify seasonal trends and customer behaviors, enabling better resource allocation and inventory management to meet demand effectively.

# Conversion Functions

In each of the following exercises, feel free to use labels for the converted column to make the output more readable.

1. List the last names and birthdays of Global Fast Food Employees. Convert the birth dates to character data in the Month DD, YYYY format. Suppress any leading zeros.

```
select last_name, to_char(birth_date, 'FMMonth DD, YYYY') as formatted_birthday
from  employees where company_name = 'Global Fast Foods';
```

2. Convert January 3, 04, to the default date format 03-Jan-2004.

```
select to_char(to_date('January 3, 04', 'Month DD, YY'), 'DD-Mon-YYYY') as
formatted_date
from dual;
```

3. Format a query from the Global Fast Foods f_promotional_menus table to print out the start_date of promotional code 110 as: The promotion began on the tenth of February 2004.

```
select 'The promotion began on the ' || to_char(start_date, 'FMDay') || ' of ' || to_char(start_date,
'FMMonth YYYY') as promotion_message from f_promotional_menus
where promotional_code = 110;
```

4. Convert today's date to a format such as: "Today is the Twentieth of March, Two Thousand Four"

```
select 'Today is the ' || to_char(sysdate, 'FMDay') || ' of ' || to_char(sysdate, 'FMMonth') || ', ' ||
to_char(sysdate, 'YYYY') as formatted_date from dual;
```

5. List the ID, name and salary for all Global Fast Foods employees. Display salary with a $ sign and two decimal places.

```
select employee_id, employee_name, '$' || to_char(salary, 'FM999,999.00') as
formatted_salary
from employee where company_name = 'Global Fast Foods';
```

**Ex. No. : 10 Date:**

**Register No.: Name:**

---

## 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:**

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.

**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);
```

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 employeesWHERE 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))
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';

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 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 = '&employee_last_name'
) and last_name <> '&employee_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 department_id from employees where last_name like '%u%');
```

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 last_name, department_id, job_id from employees where department_id in (
select department_id from departments where location_id = 1700 );
```

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

```
select last_name, salary from employees where manager_id = (
select employee_id from employees where last_name = 'King');
```

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';
```

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 department_id from employees where last_name like '%u%')
order by salary asc;
```

## Practice Questions

1. Ellen Abel is an employee who has received a $2,000 raise. Display her first name and last name, her current salary, and her new salary. Display both salaries with a $ and two decimal places. Label her new salary column AS New Salary.

```
select first_name, last_name,
'$' || to_char(salary, 'FM999,999.00') as current_salary,
'$' || to_char(salary + 2000, 'FM999,999.00') as "New Salary" from employees where
first_name = 'Ellen' and last_name = 'Abel';
```

2. On what day of the week and date did Global Fast Foods' promotional code 110 Valentine's Special begin?

```
select to_char(start_date, 'Day') || ', ' || to_char(start_date, 'MM-DD-YYYY') as promotion_start
from f_promotional_menus where promotional_code = 110;
```

3. Create one query that will convert 25-Dec-2004 into each of the following (you will have to convert 25-Dec-2004 to a date and then to character data):

December 25th, 2004
DECEMBER 25TH, 2004
25th december, 2004

```
select
to_char(to_date('25-Dec-2004', 'DD-Mon-YYYY'), 'FMMonth DDTH, YYYY') as
formatted_date1,
upper(to_char(to_date('25-Dec-2004', 'DD-Mon-YYYY'), 'FMMonth DDTH, YYYY')) as
formatted_date2,
to_char(to_date('25-Dec-2004', 'DD-Mon-YYYY'), 'DDTH FMMonth, YYYY')
as formatted_date3
from dual;
```

4. Create a query that will format the DJs on Demand d_packages columns, low-range and high-range package costs, in the format $2500.00.

```
select '$' || to_char(low_range, 'FM999,999.00') as formatted_low_range,
'$' || to_char(high_range, 'FM999,999.00') as formatted_high_range from d_packages;
```

5. Convert JUNE192004 to a date using the fx format model.

```
select to_date('JUNE192004', 'fxMonthYYYY') as converted_date from dual;
```

6. What is the distinction between implicit and explicit datatype conversion? Give an example of each.

Implicit conversion occurs automatically by the database system when it encounters a data type mismatch. eg:select '100' + 200 as result from dual;
Explicit conversion occurs when the user specifies a conversion using a function, instructing the database to convert a value from one data type to another. eg:select to_number('100') + 200 as result from dual.

7. Why is it important from a business perspective to have datatype conversions?

Datatype conversions are crucial for ensuring data accuracy, integration across systems, and compliance, ultimately improving reporting and user experience in business operations.

**Ex. No. : 11** **Date:**

**Register No.:** **Name:**

---

## 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**

• 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
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:**

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.

```
select department_id from employees where department_id is not null except select
department_id from employees where job_id = 'ST_CLERK';
```

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.

```
select country_id, country_name from countries except select c.country_id, c.country_name
from countries c join locations l on c.country_id = l.country_id join departments d on
l.location_id = d.location_id;
```

3. Produce a list of jobs for departments 10, 50, and 20, in that order. Display job ID and department ID using set operators.

```
select job_id, department_id from employees where department_id = 10 union all
select job_id, department_id from employees where department_id = 50 union all
select job_id, department_id from employees where department_id = 20;
```

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).

```
select e.employee_id, e.job_id from employees e join employees e2 on e.employee_id =
e2.employee_id where e.job_id = e2.job_id and e.hire_date < e2.hire_date;
```

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.

```
select last_name, department_id from employees union all
select department_id, department_name from departments;
```

**Ex. No. : 12 Date:**

**Register No.: Name:**

---

## NULL Functions

1. Create a report that shows the Global Fast Foods promotional name, start date, and end date from the f_promotional_menus table. If there is an end date, temporarily replace it with “end in two weeks”. If there is no end date, replace it with today’s date.

```
select promotional_name, start_date, case when end_date is not null then 'end in two weeks' else
to_char(sysdate, 'MM-DD-YYYY') end as end_date from f_promotional_menus;
```

2. Not all Global Fast Foods staff members receive overtime pay. Instead of displaying a null value for these employees, replace null with zero. Include the employee’s last name and overtime rate in the output. Label the overtime rate as “Overtime Status”.

```
select last_name, nvl(overtime_rate, 0) as "Overtime Status" from employees;
```

3. The manager of Global Fast Foods has decided to give all staff who currently do not earn overtime an overtime rate of $5.00. Construct a query that displays the last names and the overtime rate for each staff member, substituting $5.00 for each null overtime value.

```
select last_name, nvl(overtime_rate, 5.00) as overtime_rate from employees;
```

4. Not all Global Fast Foods staff members have a manager. Create a query that displays the employee last name and 9999 in the manager ID column for these employees

.

```
select last_name, nvl(manager_id, 9999) as manager_id from employees;
```

5. Which statement(s) below will return null if the value of v_sal is 50?

a. SELECT nvl(v_sal, 50) FROM emp;

b. SELECT nvl2(v_sal, 50) FROM emp;

c. SELECT nullif(v_sal, 50) FROM emp;

d. SELECT coalesce (v_sal, Null, 50) FROM emp;

c.SELECT nullif(v_sal, 50) FROM emp;

6. What does this query on the Global Fast Foods table return?

```
SELECT COALESCE(last_name, to_char(manager_id)) as NAME
FROM f_staffs;
```

The query returns either the last name of the staff member or, if the last name is null, the manager ID as a string, labeling the resulting column as NAME.

7a. Create a report listing the first and last names and month of hire for all employees in the EMPLOYEES table (use TO_CHAR to convert hire_date to display the month).

```
SELECT first_name, last_name, TO_CHAR(hire_date, 'Month') AS month_of_hire
FROM employees;
```

b.Modify the report to display null if the month of hire is September. Use the NULLIF function.

SELECT first_name, last_name, NULLIF(TO_CHAR(hire_date, 'Month'), 'September') AS month_of_hire FROM employees;

8. For all null values in the specialty column in the DJs on Demand d_partners table, substitute "No Specialty." Show the first name and specialty columns only.

```
SELECT first_name, NVL(specialty, 'No Specialty') AS specialty FROM d_partners;
```

## Conditional Expressions

1. From the DJs on Demand d_songs table, create a query that replaces the 2-minute songs with "shortest" and the 10-minute songs with "longest". Label the output column "Play Times".

```
SELECT CASE
  WHEN duration = 2 THEN 'shortest'
  WHEN duration = 10 THEN 'longest'   ELSE TO_CHAR(duration)
END AS "Play Times" FROM d_songs;
```

2. Use the Oracle database employees table and CASE expression to decode the department id. Display the department id, last name, salary and a column called "New Salary" whose value is based on the following conditions:

If the department id is 10 then 1.25 * salary If the department id is 90 then 1.5 * salary

If the department id is 130 then 1.75 * salary

Otherwise, display the old salary.

```
select department_id, last_name, salary, case when department_id = 10 then salary * 1.25 when department_id = 90 then salary * 1.5 when department_id = 130 then salary * 1.75 else salary end as "new salary" from employees;
```

3. Display the first name, last name, manager ID, and commission percentage of all employees in departments 80 and 90. In a 5th column called "Review", again display the manager ID. If they don't

have a manager, display the commission percentage. If they don't have a commission, display 99999.

```
select first_name, last_name, manager_id, commission_pct, case when manager_id
is not null then manager_id  when commission_pct is not null then commission_pct
else 99999 end as "Review" from employees where department_id in (80, 90);
```

## Cross Joins and Natural Joins

Use the Oracle database for problems 1-4.

1. Create a cross-join that displays the last name and department name from the employees and departments tables.

```
select e.last_name, d.department_name from employees e cross join departments d;
```

2. Create a query that uses a natural join to join the departments table and the locations table. Display the department id, department name, location id, and city.

```
select d.department_id, d.department_name, l.location_id, l.city from departments d natural
join locations l;
```

3. Create a query that uses a natural join to join the departments table and the locations table. Restrict the output to only department IDs of 20 and 50. Display the department id, department name, location id, and city.

```
select d.department_id, d.department_name, l.location_id, l.city from departments d natural join
locations l where d.department_id in (20, 50);
```

**Ex. No. : 13 Date:**

**Register No.: Name:**

---

## 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

- To restrict data access
- To make complex queries easy
- To provide data independence
- To present different views of the same data

### Classification of views

1. Simple view
2. Complex view

| 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 |

**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 ase 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;

**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
  - 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:**

```
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.

```
create view EMPLOYEE_VU as select employee_id, first_name || ' ' || last_name as
EMPLOYEE, department_id from employees;
```

2. Display the contents of the EMPLOYEES_VU view.

```
select * from EMPLOYEES_VU;
```

3. Select the view name and text from the USER_VIEWS data dictionary views.

```
select view_name, text from user_views;
```

4. Using your EMPLOYEES_VU view, enter a query to display all employees names and department.

```
select EMPLOYEE, department_id from EMPLOYEES_VU;
```

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.

```
create view dept50 as select empno, last_name as employee, deptno
from employees where deptno = 50;
```

6. Display the structure and contents of the DEPT50 view.

```
describe dept50;
```

7. Attempt to reassign Matos to department 80.

```
update dept50 set deptno = 80 where employee = 'Matos';
```

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.

```
create view salary_vu as select e.last_name as employee, d.department_name as
department, e.salary as salary, j.grade as grade from employees e
join departments d on e.department_id = d.department_id
join job_grades j on e.salary between j.lowest_sal and j.highest_sal;
```

**Ex. No. : P-6 Date:**

**Register No.: Name:**

---

# Join Clauses

Use the Oracle database for problems 1-6.

1. Join the Oracle database locations and departments table using the location_id column. Limit the results to location 1400 only.

```
select l.location_id, l.city, d.department_id, d.department_name from locations l
join departments d on l.location_id = d.location_id where l.location_id = 1400;
```

2. Join DJs on Demand d_play_list_items, d_track_listings, and d_cds tables with the JOIN USING syntax. Include the song ID, CD number, title, and comments in the output.

```
select dpi.song_id, dt.cd_number, dt.title, dc.comments
from d_play_list_items dpi join d_track_listings dt using (song_id)
```

```
join d_cds dc using (cd_number);
```

3. Display the city, department name, location ID, and department ID for departments 10, 20, and 30 for the city of Seattle.

```
select l.city, d.department_name, l.location_id, d.department_id from locations l
join departments d on l.location_id = d.location_id where d.department_id in (10, 20, 30)
and l.city = 'Seattle';
```

4. Display country name, region ID, and region name for Americas.

```
select c.country_name, r.region_id, r.region_name
    from countries c join regions r on c.region_id = r.region_id
    where r.region_name = 'Americas';
```

5. Write a statement joining the employees and jobs tables. Display the first and last names, hire date, job id, job title, and maximum salary. Limit the query to those employees who are in jobs that can earn more than $12,000.

```
select e.first_name, e.last_name, e.hire_date, e.job_id, j.job_title, j.max_salary
from employees e join jobs j on e.job_id = j.job_id where j.max_salary > 12000;
```

## Inner versus Outer Joins

Use the Oracle database for problems 1-7.

1. Return the first name, last name, and department name for all employees including those employees not assigned to a department.

```
    select e.first_name, e.last_name, d.department_name from employees e
left join departments d on e.department_id = d.department_id;
```

2. Return the first name, last name, and department name for all employees including those departments that do not have an employee assigned to them.

```
select e.first_name, e.last_name, d.department_name from employees e
full outer join departments d on e.department_id = d.department_id;
```

3. Return the first name, last name, and department name for all employees including those departments that do not have an employee assigned to them and those employees not assigned to a department.

```
select e.first_name, e.last_name, d.department_name from employees e full outer join departments d on e.department_id = d.department_id;
```

4. Create a query of the DJs on Demand database to return the first name, last name, event date, and description of the event the client held. Include all the clients even if they have not had an event scheduled.

```
select c.first_name, c.last_name, e.event_date, e.description from clients c
left join events e on c.client_id = e.client_id;
```

5. Using the Global Fast Foods database, show the shift description and shift assignment date even if there is no date assigned for each shift description.

```
select s.shift_description, sa.shift_assignment_date from shifts s
left join shift_assignments sa on s.shift_id = sa.shift_id;
```

# Self Joins and Hierarchical Queries

For each problem, use the Oracle database.

1. Display the employee's last name and employee number along with the 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;
```

2. Modify question 1 to display all employees and their managers, even if the employee does not have a manager. Order the list alphabetically by the last name of the employee.

```
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.last_name;
```

3. Display the names and hire dates for all employees who were hired before their managers, along with their managers' 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;
```

4. Write a report that shows the hierarchy for Lex De Haans department. Include last name, salary, and department id in the report.

```
select e.last_name, e.salary, e.department_id from employees e join (
select manager_id from employees where last_name = 'De Haan' ) m on e.manager_id =
.manager_id order by e.department_id, e.last_name;
```

5. What is wrong in the following statement:

```
SELECT last_name, department_id, salary
FROM employees
START WITH last_name = 'King'
CONNECT BY PRIOR manager_id = employee_id;
```

```
select last_name, department_id, salary from employees start with last_name = 'King' connect by
prior manager_id = employee_id;
```

6. Create a report that shows the organization chart for the entire employee table. Write the report so that each level will indent each employee 2 spaces. Since Oracle Application Express cannot display the spaces in front of the column, use - (minus) instead.

```
select lpad('-', 2 * (level - 1)) || last_name as "Employee", employee_id,manager_id,department_id
from employees start with manager_id is null connect by prior employee_id = manager_id;
```

7. Re-write the report from 6 to exclude De Haan and all the people working for him.

```
select lpad('-', 2 * (level - 1)) || last_name as "Employee",
employee_id, manager_id, department_id from employeeswhere
employee_id not in ( select employee_id from employees start with
last_name = 'De Haan' connect by prior employee_id = manager_id )
start with manager_id is null connect by prior employee_id =
manager_id;
```

## Oracle Equijoin and Cartesian Product

1. Create a Cartesian product that displays the columns in the d_play_list_items and the d_track_listings in the DJs on Demand database.

```
select dpi.*, dt.*from d_play_list_items dpicross join d_track_listings dt;
```

2. Correct the Cartesian product produced in question 1 by creating an equijoin using a common column.

```
select dpi.*, dt.*from d_play_list_items dpi
join d_track_listings dt on dpi.song_id = dt.song_id;
```

3. Write a query to display the title, type, description, and artist from the DJs on Demand database.

```
select title, type, description, artist from d_track_listings;
```

4. Rewrite the query in question 3 to select only those titles with an ID of 47 or 48.

```
select title, type, description, artist from d_track_listings where song_id in (47, 48);
```

5. Write a query that extracts information from three tables in the DJs on Demand database, the d_clients table, the d_events table, and the d_job_assignments table.

```
select c.client_name, e.event_date, e.description, ja.job_id from d_clients c
join d_events e on c.client_id = e.client_id
join d_job_assignments ja on e.event_id = ja.event_id;
```

# Group Functions

1. Define and give an example of the seven group functions: AVG, COUNT, MAX, MIN, STDDEV, SUM, and VARIANCE.

```
select avg(salary) from employees;
select count(employee_id) from employees;
select max(salary) from employees;
select min(salary) from employees;
select stddev(salary) from employees;
select sum(salary) from employees;
select variance(salary) from employees;
```

2. Create a query that will show the average cost of the DJs on Demand events. Round to two decimal places.

```
select round(avg(cost), 2) as average_event_cost from d_events;
```

3. Find the average salary for Global Fast Foods staff members whose manager ID is 19.

```
select avg(salary) as average_salary from staff_members where manager_id = 19;
```

4. Find the sum of the salaries for Global Fast Foods staff members whose IDs are 12 and 9.

```
select sum(salary) as total_salary from staff_members where staff_member_id in (12, 9);
```

5. Using the Oracle database, select the lowest salary, the most recent hire date, the last name of the person who is at the top of an alphabetical list of employees, and the last name of the person who is at the bottom of an alphabetical list of employees. Select only employees who are in departments 50 or 60

```
select min(salary) as lowest_salary, max(hire_date) as most_recent_hire_date,
(select last_name from employees where department_id in (50, 60) order by last_name asc fetch first
1 rows only) as first_alphabetical_employee,  (select last_name from employees where
department_id in (50, 60) order by last_name desc fetch first 1 rows only) as
last_alphabetical_employee from employees where department_id in (50, 60);
```

6. Your new Internet business has had a good year financially. You have had 1,289 orders this year. Your customer order table has a column named total_sales. If you submit the following query, how many rows will be returned?

```
SELECT sum(total_sales)  fFROM orders;
```

it will return **one row**. This row contains the sum of all total_sales values in the orderstable.

5. You were asked to create a report of the average salaries for all employees in each division of the company. Some employees in your company are paid hourly instead of by salary. When you ran the report, it seemed as though the averages were not what you expected—they were much higher than you thought! What could have been the cause?

The high average salaries in the report could be due to including hourly employees, which inflates the average. Data entry errors might lead to incorrect salary figures, and outliers—employees with very high salaries—can skew the average upward. Additionally, if the calculation method doesn't filter out non-salaried employees, it can produce misleading results. To resolve this, ensure only salaried employees are included and verify the accuracy of the salary data.

6. Employees of Global Fast Foods have birth dates of July 1, 1980, March 19, 1979, and March 30, 1969. If you select MIN(birthdate), which date will be returned?

the date that will be returned is **March 30, 1969**.

7. Create a query that will return the average order total for all Global Fast Foods orders from January 1, 2002, to December 21, 2002.

```
SELECT AVG(order_total) AS average_order_total FROM orders
WHERE order_date BETWEEN '2002-01-01' AND '2002-12-21';
```

8. What was the hire date of the last Oracle employee hired?

```
select max(hire_date) as last_hire_date from employees;
```

9. Your new Internet business has had a good year financially. You have had 1,289 orders this year. Your customer order table has a column named total_sales. If you submit the following query, how many rows will be returned?

```
SELECT sum(total_sales)
FROM orders;
```

The query SELECT
sum(total_sales) FROM orders;
will return one row.

**Ex. No. : P-7 Date:**

**Register No.: Name:**

---

## COUNT, DISTINCT, NVL

1. How many songs are listed in the DJs on Demand D_SONGS table?

```
select count(*) as total_songs from d_songs;
```

2. In how many different location types has DJs on Demand had venues?

```
select count(distinct location_type) as unique_location_types from d_venues;
```

3. The d_track_listings table in the DJs on Demand database has a song_id column and a cd_number column. How many song IDs are in the table and how many different CD numbers are in the table?

```
select count(distinct song_id) as total_unique_songs, count(distinct cd_number) as total_unique_cds
from d_track_listings;
```

4. How many of the DJs on Demand customers have email addresses?

```
select count(*) as customers_with_email from d_customers where email is not null;
```

5. Some of the partners in DJs on Demand do not have authorized expense amounts (auth_expense_amt). How many partners do have this privilege?

```
select count(*) as partners_with_auth_expense from d_partners where auth_expense_amt is
not null;
```

6. What values will be returned when the statement below is issued?

| ID | type | shoe_color |
|---|---|---|
| 456 | oxford | brown |
| 463 | sandal | tan |
| 262 | heel | black |
| 433 | slipper | tan |

SELECT COUNT(shoe_color),
COUNT(DISTINCT shoe_color)

FROM shoes;

COUNT(shoe_color): 4 (total colors listed)
COUNT(DISTINCT shoe_color): 3 (unique colors: brown, tan, black)

7. Create a query that will convert any null values in the auth_expense_amt column on the DJs on Demand D_PARTNERS table to 100000 and find the average of the values in this column. Round the result to two decimal places.

```
select round(avg(coalesce(auth_expense_amt, 100000)), 2) as avg_auth_expense from d_partners;
```

8. Which of the following statements is/are TRUE about the following query?

```
SELECT AVG(NVL(selling_bonus, 0.10))
FROM bonuses;
```

_____a. The datatypes of the values in the NVL clause can be any datatype except date data.

_____b. If the selling_bonus column has a null value, 0.10 will be substituted.

_____c. There will be no null values in the selling_bonus column when the average is calculated.

_____ d. This statement will cause an error. There cannot be two functions in the SELECT statement.

a. The datatypes of the values in the NVL clause can be any datatype except date data.
b. If the selling_bonus column has a null value, 0.10 will be substituted.
c. There will be no null values in the selling_bonus column when the average is calculated.

9. Which of the following statements is/are TRUE about the following query?

```
SELECT DISTINCT colors, sizes
FROM items;
```

_____a. Each color will appear only once in the results set.

_____b. Each size will appear only once in the results set.

_____c. Unique combinations of color and size will appear only once in the results set.

_____d. Each color and size combination will appear more than once in the results set.

| |
|---|
| c. Unique combinations of color and size will appear only once in the results set. |

## Using GROUP BY and HAVING Clauses

1. In the SQL query shown below, which of the following are true about this query?

a. Kimberly Grant would not appear in the results set.

b. The GROUP BY clause has an error because the manager_id is not listed in the SELECT clause.

c. Only salaries greater than 16001 will be in the result set.

d. Names beginning with Ki will appear after names beginning with Ko.

e. Last names such as King and Kochhar will be returned even if they don't have salaries > 16000.

```
SELECT last_name, MAX(salary)
FROM employees
WHERE last_name LIKE 'K%'
GROUP BY manager_id, last_name
HAVING MAX(salary) >16000
ORDER BY last_name DESC ;
```

| |
|---|
| a. True<br>b. False<br>c. False<br>d. False<br>e. False |

2. Each of the following SQL queries has an error. Find the error and correct it. Use Oracle Application Express to verify that your corrections produce the desired results.

a. SELECT
manager_id FROM
employees
WHERE AVG(salary) <16000
GROUP BY manager_id;

b. SELECT cd_number,
COUNT(title) FROM d_cds
WHERE cd_number < 93;

c. SELECT ID, MAX(ID), artist AS Artist FROM d_songs
WHERE duration IN('3 min', '6 min', '10 min')
HAVING ID < 50
GROUP by ID;

d. SELECT loc_type, rental_fee AS
Fee FROM d_venues
WHERE id <100
GROUP BY "Fee"
ORDER BY 2;

a. SELECT manager_id FROM employees GROUP BY manager_id HAVING AVG(salary) < 16000;

b. SELECT cd_number, COUNT(title) FROM d_cds WHERE cd_number < 93 GROUP BYcd_number;

c. SELECT ID, MAX(ID), artist AS Artist FROM d_songs WHERE duration IN

('3 min', '6 min', '10 min') GROUP BY ID, artist HAVING ID < 50;

d. SELECT loc_type, rental_fee AS Fee FROM d_venues WHERE id < 100 GROUP BY loc_type, rental_fee ORDER BY 2;

3. Rewrite the following query to accomplish the same result:

```
SELECT DISTINCT MAX(song_id)
FROM d_track_listings
WHERE track IN ( 1, 2, 3);
```

```
SELECT MAX(song_id)  FROM d_track_listings WHERE track IN (1, 2, 3);
```

4. Indicate True or False

a. If you include a group function and any other individual columns in a SELECT clause, then each individual column must also appear in the GROUP BY clause.

b. You can use a column alias in the GROUP BY clause.

c. The GROUP BY clause always includes a group function.

a. True
b. False
c. False

5. Write a query that will return both the maximum and minimum average salary grouped by department from the employees table.

```
SELECT MAX(avg_salary) AS max_average_salary, MIN(avg_salary) AS
min_average_salary FROM (SELECT department, AVG(salary) AS avg_salary FROM
employees GROUP BY department) AS avg_salaries;
```

6. Write a query that will return the average of the maximum salaries in each department for the employees table.

```
SELECT AVG(max_salary) AS average_of_max_salaries FROM (SELECT MAX(salary) AS
max_salary FROM employees GROUP BY department) AS max_salaries;
```

## Using Set Operators

1. Name the different Set operators?

   **UNION**: Combines the results of two or more SELECT statements and removes duplicate rows. It returns all unique rows from both queries.

   1. **UNION ALL**: Similar to UNION, but it includes all rows from both queries, including duplicates.
   2. **INTERSECT**: Returns only the rows that are present in both SELECT statements. It finds the common rows between the two queries.
   3. **EXCEPT** (or **MINUS** in some databases): Returns the rows from the first SELECT statement that are not present in the second SELECT statement. It effectively finds the difference between two result sets.

2. Write one query to return the employee_id, job_id, hire_date, and department_id of all employees and a second query listing employee_id, job_id, start_date, and department_id from the job_history table and combine the results as one single output. Make sure you suppress duplicates in the output.

```
SELECT employee_id, job_id, hire_date AS date, department_id FROM employees UNION
SELECT employee_id, job_id, start_date AS date, department_id FROM job_history;
```

3. Amend the previous statement to not suppress duplicates and examine the output. How many extra rows did you get returned and which were they? Sort the output by employee_id to make it easier to spot.

**Total Extra Rows Returned**: **1**

**Details of the Extra Row**:

- **employee_id**: 176
- **job_id**: SA_REP
- **date**: (this will depend on whether it pulls `hire_date` or `start_date`, but will show as the corresponding date for employee_id 176)
- **department_id**: (this will reflect the department for employee_id 176)

There is one extra row on employee 176 with a job_id of SA_REP.

This extra row appears because employee 176 has entries in both the employees and job_history tables, leading to duplication in the results when using UNION ALL.

4. List all employees who have not changed jobs even once. (Such employees are not found in the job_history table)

```
SELECT e.employee_id, e.job_id, e.hire_date, e.department_id FROM employees e
LEFT JOIN job_history j ON e.employee_id = j.employee_id
WHERE j.employee_id IS NULL;
```

5. List the employees that HAVE changed their jobs at least once.

```
SELECT DISTINCT
e.employee_id, e.job_id,
e.hire_date,
e.department_id
FROM employees e
```

```
JOIN job_history j ON
e.employee_id =
j.employee_id;
```

6. Using the UNION operator, write a query that displays the employee_id, job_id, and salary of ALL present and past employees. If a salary is not found, then just display a 0 (zero) in its place.

```
SELECT employee_id, job_id, COALESCE(salary, 0) AS salary FROM employees
UNION
SELECT employee_id, job_id, COALESCE(salary, 0) AS salary FROM job_history;
```

**Ex. No. : P-8 Date:**

**Register No.: Name:**

---

# Fundamentals of Subqueries

1. What is the purpose of using a subquery?

A subquery, or nested query, is used in SQL to filter data based on criteria from another table, perform calculations like averages to use in the main query, and simplify complex queries by breaking them down into more manageable parts.

2. What is a subquery?

Subqueries can appear in various parts of a SQL statement, like the SELECT, FROM, or WHERE clauses, and are executed before the outer query. For example, you might use a subquery to find employees who earn more than the average salary in their department.

3. What DJs on Demand d_play_list_items song_id's have the same event_id as song_id 45?

```
SELECT song_id
FROM d_play_list_items
WHERE event_id =
(SELECT event_id FROM
d_play_list_items WHERE
song_id = 45);
```

4. Which events in the DJs on Demand database cost more than event_id = 100?

```
SELECT *
FROM events
WHERE cost >
(SELECT cost FROM
events WHERE
event_id = 100);
```

5. Find the track number of the song that has the same CD number as “Party Music for All Occasions.”

```
SELECT track_number FROM d_track_listings WHERE cd_number = (SELECT cd_number FROM
d_track_listings WHERE song_title = 'Party Music for All Occasions');
```

6. List the DJs on Demand events whose theme code is the same as the code for “Tropical.”

```
SELECT * FROM events WHERE theme_code = (SELECT theme_code FROM
themes WHERE theme_name = 'Tropical');
```

7. What are the names of the Global Fast Foods staff members whose salaries are greater than the staff member whose ID is 12?

```
SELECT name FROM staff WHERE salary > (SELECT salary FROM staff WHERE staff_id =12);
```

8. What are the names of the Global Fast Foods staff members whose staff types are not the same as Bob Miller’s?

```
SELECT name FROM staff WHERE staff_type <> (SELECT staff_type FROM staff WHERE name
= 'Bob Miller');
```

9. Which Oracle employees have the same department ID as the IT department?

```
SELECT * FROM employees WHERE department_id = (SELECT department_id FROM
departments WHERE department_name = 'IT');
```

10. What are the department names of the Oracle departments that have the same location ID as Seattle?

```
SELECT department_name FROM departments WHERE location_id = (SELECT
location_id FROM locations WHERE city = 'Seattle');
```

11. Which statement(s) regarding subqueries is/are true?

a. It is good programming practice to place a subquery on the right side of the comparison operator.

b. A subquery can reference a table that is not included in the outer query's FROM clause.

c. Single-row subqueries can return multiple values to the outer query.

```
b.A subquery can reference a table that is not included in the outer query's FROM clause.
```

## Single-Row Subqueries

1. Write a query to return all those employees who have a salary greater than that of Lorentz and are in the same department as Abel.

```
SELECT * FROM employees
WHERE salary > (SELECT salary FROM employees WHERE name = 'Lorentz')
AND department_id = (SELECT department_id FROM employees WHERE name =
'Abel');
```

2. Write a query to return all those employees who have the same job id as Rajs and were hired after Davies.

```
SELECT * FROM employees
WHERE job_id = (SELECT job_id FROM employees WHERE name = 'Rajs')
AND hire_date > (SELECT hire_date FROM employees WHERE name = 'Davies');
```

3. What DJs on Demand events have the same theme code as event ID = 100?

```
SELECT * FROM events WHERE theme_code = (SELECT theme_code FROM events WHERE event_id = 100);
```

4. What is the staff type for those Global Fast Foods jobs that have a salary less than those of any Cook staff-type jobs?

```
SELECT staff_type FROM jobs WHERE salary < ALL (SELECT salary FROM jobs WHERE staff_type = 'Cook');
```

5. Write a query to return a list of department id's and average salaries where the department's average salary is greater than Ernst's salary.

```
SELECT department_id, AVG(salary) AS average_salary
FROM employees  GROUP BY department_id
HAVING AVG(salary) > (SELECT salary FROM employees WHERE name = 'Ernst');
```

6. Return the department ID and minimum salary of all employees, grouped by department ID, having a minimum salary greater than the minimum salary of those employees whose department ID is not equal to 50.

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

# Multiple-Row Subqueries

1. What will be returned by a query if it has a subquery that returns a null?

When a subquery returns null, the outer query's behavior varies based on usage. If the null is in a comparison (e.g., `=`), it won't match any rows since comparisons with null are false. In an `IN` or `ANY` clause, it may also result in no matches. For aggregate functions, nulls are ignored and don't affect calculations. If included in the `SELECT` clause, the output will show null for that column. Overall, a null returned by a subquery typically leads to no matches or null values in the results.

2. Write a query that returns jazz and pop songs. Write a multi-row subquery and use the d_songs and d_types tables. Include the id, title, duration, and the artist name.

```
SELECT id, title, duration, artist_name  FROM d_songs

WHERE type_id IN (SELECT id FROM d_types WHERE type_name IN ('Jazz', 'Pop'));
```

3. Find the last names of all employees whose salaries are the same as the minimum salary for any department.

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

4. Which Global Fast Foods employee earns the lowest salary? Hint: You can use either a single-row or a multiple-row subquery.

```
SELECT name, salary FROM employees
```

WHERE salary = (SELECT MIN(salary) FROM employees);

5. Place the correct multiple-row comparison operators in the outer query WHERE clause of each of the following:

a. Which CDs in our d_cds collection were produced before “Carpe Diem” was produced?

WHERE year _____(SELECT year ...

b. Which employees have salaries lower than any one of the programmers in the IT department?

WHERE salary_____(SELECT salary ...

c. What CD titles were produced in the same year as “Party Music for All Occasions” or “Carpe Diem”?

WHERE year _____(SELECT year ...

d. What song title has a duration longer than every type code 77 title?

WHERE duration _____(SELECT duration ...

a.WHERE year < (SELECT year FROM d_cds WHERE title = 'Carpe Diem')
b.WHERE salary < ALL (SELECT salary FROM employees WHERE job_title = 'Programmer' AND department = 'IT')
c.WHERE year IN (SELECT year FROM d_cds WHERE title IN ('Party Music for All Occasions', 'Carpe Diem'))
d.WHERE duration > ALL (SELECT duration FROM songs WHERE type_code = 77)

6. If each WHERE clause is from the outer query, which of the following are true?

a. WHERE size > ANY -- If the inner query returns sizes ranging from 8 to 12, the value 9 could

be returned in the outer query.

b. WHERE book_number IN -- If the inner query returns books numbered 102, 105, 437, and 225 then 325 could be returned in the outer query.

c. WHERE score <= ALL -- If the inner query returns the scores 89, 98, 65, and 72, then 82 could be returned in the outer query.

d. WHERE color NOT IN -- If the inner query returns red, green, blue, black, and then the outer query could return white.

e. WHERE game_date = ANY -- If the inner query returns 05-Jun-1997, 10-Dec-2002, and 2-Jan-2004, then the outer query could return 10-Sep-2002.

**True**: a, d

**False**: b, c, e

7. The goal of the following query is to display the minimum salary for each department whose minimum salary is less than the lowest salary of the employees in department 50. However, the subquery does not execute because it has five errors. Find them, correct them, and run the query.

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

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

No data was returned from this query.

8. Which statements are true about the subquery below?

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

a. The inner query could be eliminated simply by changing the WHERE clause to WHERE MIN(salary).

b. The query wants the names of employees who make the same salary as the smallest salary in any department.

c. The query first selects the employee ID and last name, and then compares that to the salaries in every department.

d. This query will not execute.

**1**: False

**2**: True

**3**: False

**4**: False

9. Write a pair-wise subquery listing the last_name, first_name, department_id, and manager_id for all employees that have the same department_ id and manager_id as employee 141. Exclude employee 141 from the result set.

```
SELECT last_name, first_name, department_id, manager_id FROM employees
WHERE (department_id, manager_id) = ( SELECT department_id, manager_id
FROM employees WHERE employee_id = 141) AND employee_id <> 141;
```

10. Write a non-pair-wise subquery listing the last_name, first_name, department_id, and manager_id for all employees that have the same department_ id and manager_id as employee 141.

```
SELECT last_name, first_name, department_id, manager_id FROM employees
WHERE department_id = (SELECT department_id FROM employees WHERE employee_id = 141)
    AND manager_id = (SELECT manager_id FROM employees WHERE employee_id = 141);
```



## Correlated Subqueries

1. Explain the main difference between correlated and non-correlated subqueries?

Correlated Subqueries: These reference columns from the outer query and cannot run independently. They are executed once for each row of the outer query, re-evaluating the inner query for every row.

Non-Correlated Subqueries: These do not reference the outer query and can run independently. The

inner query is executed once, and its result is used in the outer query.

2. Write a query that lists the highest earners for each department. Include the last_name, department_id, and the salary for each employee.

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

3. Examine the following select statement and finish it so that it will return the last_name, department_id, and salary of employees who have at least one person reporting to them. So we are effectively looking for managers only. In the partially written SELECT statement, the WHERE clause will work as it is. It is simply testing for the existence of a row in the subquery.

SELECT (enter columns here)
FROM (enter table name here) outer
WHERE 'x' IN (SELECT 'x'
FROM (enter table name here) inner
WHERE inner(enter column name here) = inner(enter column name here) Finish
off the statement by sorting the rows on the department_id column.

```
SELECT last_name, department_id, salary FROM employees outer WHERE employee_id IN (
SELECT manager_id FROM employees inner WHERE inner.manager_id = outer.employee_id)
ORDER BY department_id;
```

4. Using a WITH clause, write a SELECT statement to list the job_title of those jobs whose maximum salary is more than half the maximum salary of the entire company. Name your subquery MAX_CALC_SAL. Name the columns in the result JOB_TITLE and JOB_TOTAL, and sort the result on JOB_TOTAL in descending order.

Hint: Examine the jobs table. You will need to join JOBS and EMPLOYEES to display the job_title.

```
WITH MAX_CALC_SAL AS ( SELECT MAX(salary) AS max_salaryFROM employees)
SELECT j.job_title AS JOB_TITLE, MAX(e.salary) AS JOB_TOTAL FROM jobs j
JOIN employees e ON j.job_id = e.job_id GROUP BY j.job_title
HAVING MAX(e.salary) > (SELECT max_salary / 2 FROM MAX_CALC_SAL)
ORDER BY JOB_TOTAL DESC;
```



# Summarizing Queries for practice

## INSERT Statements

**Students should execute DESC tablename before doing INSERT to view the data types for each column. VARCHAR2 data-type entries need single quotation marks in the VALUES statement.**

1. Give two examples of why it is important to be able to alter the data in a database.

Data Correction: Errors can occur during data entry, and being able to update records ensures accuracy and integrity. For example, correcting an employee's incorrect address is vital for reliable information.

Dynamic Business Needs: As businesses grow and change, their data requirements evolve. Updating records allows organizations to adapt to new products, markets, or processes, ensuring that the database remains relevant and useful for decision-making.

2. DJs on Demand just purchased four new CDs. Use an explicit INSERT statement to add each CD to the copy_d_cds table. After completing the entries, execute a SELECT * statement to verify your work.

| CD_NUMBER | TITLE | PRODUCER | YEAR |
|---|---|---|---|
| 97 | Celebrate the Day | R&B Inc. | 2003 |
| 98 | Holiday Tunes for All Ages | Tunes are Us | 2004 |
| 99 | Party Music | Old Town Records | 2004 |
| 100 | Best of Rock and Roll | Old Town Records | 2004 |

```
INSERT INTO copy_d_cds (CD_NUMBER, TITLE, PRODUCER, YEAR) VALUES (97, 'Celebrate the Day', 'R&B Inc.', 2003);
INSERT INTO copy_d_cds (CD_NUMBER, TITLE, PRODUCER, YEAR) VALUES (98, 'Holiday Tunes for All Ages', 'Tunes are Us', 2004);
INSERT INTO copy_d_cds (CD_NUMBER, TITLE, PRODUCER, YEAR) VALUES (99, 'Party
```

```
Music', 'Old Town Records', 2004);
INSERT INTO copy_d_cds (CD_NUMBER, TITLE, PRODUCER, YEAR) VALUES (100, 'Best of Rock and Roll', 'Old Town Records', 2004);

SELECT * FROM copy_d_cds;
```

3. DJs on Demand has two new events coming up. One event is a fall football party and the other event is a sixties theme party. The DJs on Demand clients requested the songs shown in the table for their events. Add these songs to the copy_d_songs table using an implicit INSERT statement.

```
INSERT INTO copy_d_songs VALUES
(52, 'Surfing Summer', 'Not known', 12),
(53, 'Victory Victory', '5 min', 12)
```

| ID | TITLE | DURATION | TYPE_CODE |
|---|---|---|---|
| 52 | Surfing Summer | Not known | 12 |
| 53 | Victory Victory | 5 min | 12 |

4. Add the two new clients to the copy_d_clients table. Use either an implicit or an explicit INSERT.

| CLIENT_NUMBER | FIRST_NAME | LAST_NAME | PHONE | EMAIL |
|---|---|---|---|---|
| 6655 | Ayako | Dahish | 3608859030 | dahisha@harbor.net |
| 6689 | Nick | Neuville | 9048953049 | nnicky@charter.net |

```
INSERT INTO copy_d_clients
VALUES (6655, 'Ayako', 'Dahish', '3608859030', 'dahisha@harbor.net');

INSERT INTO copy_d_clients
VALUES (6689, 'Nick', 'Neuville', '9048953049', 'nnicky@charter.net');
```

5. Add the new client's events to the copy_d_events table. The cost of each event has not been determined at this date.

| ID | NAME | EVENT_DATE | DESCRIPTION | COST | VENUE_ID | PACKAGE_CODE | THEME_CODE | CLIENT_NUMBER |
|---|---|---|---|---|---|---|---|---|
| 110 | Ayako Anniversary | 07-Jul-2004 | Party for 50, sixties dress, decorations | | 245 | 79 | 240 | 6655 |
| 115 | Neuville Sports Banquet | 09-Sep-2004 | Barbecue at residence, college alumni, 100 people | | 315 | 87 | 340 | 6689 |

INSERT INTO copy_d_events (ID, NAME, EVENT_DATE, DESCRIPTION, COST, VENUE_ID, PACKAGE_CODE, THEME_CODE, CLIENT_NUMBER)
VALUES (110, 'Ayako Anniversary', '2004-07-07', 'Party for 50, sixties dress, decorations', NULL, 245, 79, 240, 6655);

INSERT INTO copy_d_events (ID, NAME, EVENT_DATE, DESCRIPTION, COST, VENUE_ID, PACKAGE_CODE, THEME_CODE, CLIENT_NUMBER)
VALUES (115, 'Neuville Sports Banquet', '2004-09-09', 'Barbecue at residence, college alumni, 100 people', NULL, 315, 87, 40, 6689);

6. Create a table called rep_email using the following statement:

CREATE TABLE rep_email ( id NUMBER(3) CONSTRAINT rel_id_pk PRIMARY KEY, first_name VARCHAR2(10), last_name VARCHAR2(10), email_address VARCHAR2(10))

Populate this table by running a query on the employees table that includes only those employees who are REP's.

```
CREATE TABLE rep_email ( id NUMBER(3) CONSTRAINT rel_id_pk PRIMARY KEY,
   first_name VARCHAR2(10),  last_name VARCHAR2(10),
   email_address VARCHAR2(10)
);

INSERT INTO rep_email (id, first_name, last_name, email_address)
SELECT id, first_name, last_name, email_address FROM employees WHERE job_title = 'REP';
```

# Updating Column Values and Deleting Rows

**NOTE: Copy tables in this section do not yet exist; students must create them.**

If any change is not possible, give an explanation as to why it is not possible.

1. Monique Tuttle, the manager of Global Fast Foods, sent a memo requesting an immediate change in prices. The price for a strawberry shake will be raised from $3.59 to $3.75, and the price for fries will increase to $1.20. Make these changes to the copy_f_food_items table.

Strawberry shake:

```
UPDATE copy_f_food_items SET price = 3.75 WHERE item_name = 'Strawberry Shake';
```

Fries:

```
UPDATE copy_f_food_items SET price = 1.20 WHERE item_name = 'Fries';
```

2. Bob Miller and Sue Doe have been outstanding employees at Global Fast Foods. Management has decided to reward them by increasing their overtime pay. Bob Miller will receive an additional $0.75 per hour and Sue Doe will receive an additional $0.85 per hour. Update the copy_f_staffs table to show these new values. (Note: Bob Miller currently doesn't get overtime pay. What function do you need to use to convert a null value to 0?)

Overtime pay for Bob Miller, converting NULL to 0 if necessary:

```
UPDATE copy_f_staffs SET overtime_pay = NVL(overtime_pay, 0) + 0.75
WHERE first_name = 'Bob' AND last_name = 'Miller';
```

Overtime pay for Sue Doe:

```
UPDATE copy_f_staffs SET overtime_pay = overtime_pay + 0.85
WHERE first_name = 'Sue' AND last_name = 'Doe';
```

3. Add the orders shown to the Global Fast Foods copy_f_orders table:

| ORDER_NUMBER | ORDER_DATE | ORDER_TOTAL | CUST_ID | STAFF_ID |
|---|---|---|---|---|
| 5680 | June 12, 2004 | 159.78 | 145 | 9 |
| 5691 | 09-23-2004 | 145.98 | 225 | 12 |
| 5701 | July 4, 2004 | 229.31 | 230 | 12 |

```
INSERT INTO copy_f_orders (ORDER_NUMBER, ORDER_DATE, ORDER_TOTAL, CUST_ID,
STAFF_ID)
VALUES (5680, TO_DATE('June 12, 2004', 'Month DD, YYYY'), 159.78, 145, 9);

INSERT INTO copy_f_orders (ORDER_NUMBER, ORDER_DATE, ORDER_TOTAL, CUST_ID,
STAFF_ID)
VALUES (5691, TO_DATE('09-23-2004', 'MM-DD-YYYY'), 145.98, 225, 12);

INSERT INTO copy_f_orders (ORDER_NUMBER, ORDER_DATE, ORDER_TOTAL, CUST_ID,
STAFF_ID)
VALUES (5701, TO_DATE('July 4, 2004', 'Month DD, YYYY'), 229.31, 230, 12);
```

4. Add the new customers shown below to the copy_f_customers table. You may already have added Katie Hernandez. Will you be able to add all these records successfully?

| ID | FIRST_ | LAST_NAME | ADDRESS | CITY | STATE | ZIP | PHONE_NUMBER |
|---|---|---|---|---|---|---|---|

| | NAME | | | | | | |
|---|---|---|---|---|---|---|---|
| 145 | Katie | Hernandez | 92 Chico Way | Los Angeles | CA | 98008 | 8586667641 |
| 225 | Daniel | Spode | 1923 Silverado | Denver | CO | 80219 | 7193343523 |
| 230 | Adam | Zurn | 5 Admiral Way | Seattle | WA | | 4258879009 |

If Katie Hernandez (with ID = 145) already exists in the copy_f_customers table, attempting to insert her again will result in a duplicate primary key error.

If the ZIP column in the copy_f_customers table is defined with a NOT NULL constraint, leaving this field blank (or setting it to NULL) will result in a NULL constraint violation.

5. Sue Doe has been an outstanding Global Foods staff member and has been given a salary raise. She will now be paid the same as Bob Miller. Update her record in copy_f_staffs.

```
UPDATE copy_f_staffs SET salary = (
SELECT salary FROM copy_f_staffs WHERE first_name = 'Bob' AND last_name = 'Miller'
)WHERE first_name = 'Sue' AND last_name = 'Doe';
```

update copy_f_staffs

```
set salary = ( select salary from copy_f_staffs
 where first_name = 'Bob' and last_name = 'Miller')
where first_name = 'Sue' and last_name = 'Doe';
```

6. Global Fast Foods is expanding their staff. The manager, Monique Tuttle, has hired Kai Kim. Not all information is available at this time, but add the information shown at right.

| ID | FIRST_NAME | LAST_NAME | BIRTHDATE | SALARY | STAFF TYPE |
|---|---|---|---|---|---|
| 25 | Kai | Kim | 3-Nov-1988 | 6.75 | Order Taker |

```
INSERT INTO copy_f_staffs (ID, FIRST_NAME, LAST_NAME, BIRTHDATE,
SALARY, STAFF_TYPE)
VALUES (25, 'Kai', 'Kim', TO_DATE('03-Nov-1988', 'DD-Mon-YYYY'), 6.75, 'Order
Taker');
```

7. Now that all the information is available for Kai Kim, update his Global Fast Foods record to include the following: Kai will have the same manager as Sue Doe. He does not qualify for overtime. Leave the values for training, manager budget, and manager target as null.

```
UPDATE employees SET  manager_id = (SELECT
manager_id FROM employees WHERE
employee_name = 'Sue Doe'), qualifies_for_overtime
'No', training = NULL, manager_budget = NULL,
manager_target = NULL WHERE employee_name =
'Kai Kim';
```

8. Execute the following SQL statement. Record your results.

DELETE from departments

WHERE department_id = 60;

| department_id | department_name | location_id |
|---|---|---|
| 10 | HR | 100 |
| 20 | IT | 200 |
| 30 | Marketing | 300 |
| 90 | Finance | 500 |

9. Kim Kai has decided to go back to college and does not have the time to work and go to school. Delete him from the Global Fast Foods staff. Verify that the change was made.

```
DELETE FROM employees WHERE employee_name = 'Kim Kai';
```

10. Create a copy of the employees table and call it lesson7_emp;

Once this table exists, write a correlated delete statement that will delete any employees from the lesson7_employees table that also exist in the job_history table.

```
CREATE TABLE lesson7_emp AS SELECT * FROM employees;

DELETE FROM lesson7_emp
WHERE employee_id IN (SELECT employee_id FROM job_history WHERE
lesson7_emp.employee_id = job_history.employee_id);
```

# DEFAULT Values, MERGE, and Multi-Table Inserts

1. When would you want a DEFAULT value?

Ensures that all records have a valid value, which is crucial for data integrity. Represents unknown states, such as defaulting a phone number to NULL when not provided.Minimizes redundancy for frequently used values, like defaulting a country to 'USA' in a domestic application.

2. Currently, the Global Foods F_PROMOTIONAL_MENUS table START_DATE column does not have SYSDATE set as DEFAULT. Your manager has decided she would like to be able to set the starting date of promotions to the current day for some entries. This will require three steps:

a. In your schema, Make a copy of the Global Foods F_PROMOTIONAL_MENUS table using the following SQL statement:

b. Alter the current START_DATE column attributes using:

c. INSERT the new information and check to verify the results.

INSERT a new row into the copy_f_promotional_menus table for the manager's new promotion. The promotion code is 120. The name of the promotion is 'New Customer.' Enter DEFAULT for the start date and '01-Jun-2005' for the ending date. The giveaway is a 10% discount coupon. What was the correct syntax used?

```
CREATE TABLE copy_f_promotional_menus AS SELECT * FROM
F_PROMOTIONAL_MENUS;

ALTER TABLE copy_f_promotional_menus MODIFY START_DATE DEFAULT SYSDATE;
```

INSERT INTO copy_f_promotional_menus (promotion_code, promotion_name, start_date, end_date, giveaway)
VALUES (120, 'New Customer', DEFAULT, TO_DATE('01-Jun-2005', 'DD-Mon-YYYY'), '10% discount coupon');

SELECT * FROM copy_f_promotional_menus WHERE promotion_code = 120;

3. Allison Plumb, the event planning manager for DJs on Demand, has just given you the following list of CDs she acquired from a company going out of business. She wants a new updated list of CDs in inventory in an hour, but she doesn't want the original D_CDS table changed. Prepare an updated inventory list just for her.

a. Assign new cd_numbers to each new CD acquired.

b. Create a copy of the D_CDS table called manager_copy_d_cds. What was the correct syntax used?

c. INSERT into the manager_copy_d_cds table each new CD title using an INSERT statement. Make up one example or use this data:

20, 'Hello World Here I Am', 'Middle Earth Records', '1998' What
was the correct syntax used?

d. Use a merge statement to add to the manager_copy_d_cds table, the CDs from the original table. If there is a match, update the title and year. If not, insert the data from the original table. What was the correct syntax used?

```
CREATE TABLE manager_copy_d_cds AS SELECT * FROM D_CDS;

INSERT INTO manager_copy_d_cds (cd_number, title, label, year)
VALUES (NEW_CD_NUMBER, 'Hello World Here I Am', 'Middle Earth Records', '1998');

MERGE INTO manager_copy_d_cds AS target USING D_CDS AS source
ON target.cd_number = source.cd_number WHEN MATCHED THEN
   UPDATE SET target.title = source.title, target.year = source.year
WHEN NOT MATCHED THEN INSERT (cd_number, title, label, year)
   VALUES (source.cd_number, source.title, source.label, source.year);
```

4. Run the following 3 statements to create 3 new tables for use in a Multi-table insert statement. All 3 tables should be empty on creation, hence the WHERE 1=2 condition in the WHERE clause.

```
CREATE TABLE sal_history (employee_id, hire_date, salary) AS
SELECT employee_id, hire_date, salary
FROM employees
WHERE 1=2;
CREATE TABLE mgr_history (employee_id, manager_id, salary) AS
SELECT employee_id, manager_id, salary
FROM employees
WHERE 1=2;
CREATE TABLE special_sal (employee_id, salary) AS
SELECT employee_id, salary
FROM employees
WHERE 1=2;
```

Once the tables exist in your account, write a Multi-Table insert statement to first select the employee_id, hire_date, salary, and manager_id of all employees. If the salary is more than 20000 insert the employee_id and salary into the special_sal table. Insert the details of employee_id, hire_date, and

salary into the sal_history table. Insert the employee_id, manager_id, and salary into the mgr_history table.

You should get a message back saying 39 rows were inserted. Verify you get this message and verify you have the following number of rows in each table:
Sal_history: 19 rows
Mgr_history: 19 rows
Special_sal: 1

## Expected Result:

- **Sal_history**: 19 rows
- **Mgr_history**: 19 rows
- **Special_sal**: 1 row

# Creating Tables

1. Complete the GRADUATE CANDIDATE table instance chart. Credits is a foreign-key column referencing the requirements table.

| candidate_id | first_name | last_name | email | credits | degree_program |
|---|---|---|---|---|---|
| 1 | John | Doe | john.doe@example.com | 30 | Master of Science |
| 2 | Jane | Smith | jane.smith@example.com | 24 | Master of Arts |
| 3 | Emily | Johnson | emily.j@example.com | 36 | Doctor of Philosophy |
| 4 | Michael | Brown | michael.b@example.com | 12 | Master of Engineering |
| 5 | Sarah | Davis | sarah.d@example.com | 18 | Master of Business |

2. Write the syntax to create the grad_candidates table.

```
CREATE TABLE grad_candidates (candidate_id INT PRIMARY KEY, first_name VARCHAR(50) NOT NULL, last_name VARCHAR(50) NOT NULL, email VARCHAR(100) UNIQUE NOT NULL, credits INT, degree_program VARCHAR(50), FOREIGN KEY (credits) REFERENCES requirements(credits));
```

3. Confirm creation of the table using DESCRIBE.

```
DESCRIBE grad_candidates;
```

4. Create a new table using a subquery. Name the new table your last name – e.g., smith_table. Using a subquery, copy grad_candidates into smith_table.

```
CREATE TABLE smith_table AS SELECT * FROM grad_candidates;
```

5. Insert your personal data into the table created in question 4.

```
INSERT INTO smith_table (candidate_id, first_name, last_name, email, credits,
degree_program)
VALUES (1, 'John', 'Smith', 'john.smith@example.com', 30, 'Master of Science');
```

6. Query the data dictionary for each of the following:

- USER_TABLES
- USER_OBJECTS
- USER_CATALOG or USER_CAT

In separate sentences, summarize what each query will return.

1.Returns a list of all tables owned by the current user in the database, including details such as the table name, the number of rows, and the tablespace in which the table is stored.
2.returns information about all objects owned by the current user, including tables, views, indexes, and other schema objects. It provides details such as the object name, type, status, and creation date.
3.returns a list of all the catalog objects accessible to the current user. The USER_CATALOG view provides details about various object types and their definitions,

including tables, views, and synonyms, along with their respective object names and types.

## Modifying a Table

Before beginning the practice exercises, execute a DESCRIBE for each of the following tables: o_employees and o_jobs. These tables will be used in the exercises. You will need to know which columns do not allow null values.

**NOTE: If students have not already created the o_employees, o_departments, and o_jobs tables they should create them using the four steps outlined in the practice.**

1. Create the three o_tables – jobs, employees, and departments – using the syntax:

```
CREATE TABLE o_jobs (  job_id INT PRIMARY KEY, job_title VARCHAR(100)
NOT
NULL, min_salary DECIMAL(10, 2), max_salary DECIMAL(10, 2));

CREATE TABLE o_employees (employee_id INT PRIMARY KEY, first_name
VARCHAR(50) NOT NULL,  last_name VARCHAR(50) NOT NULL, email
VARCHAR(100) UNIQUE NOT NULL,  hire_date DATE NOT NULL, job_id
INT, salary DECIMAL(10, 2), FOREIGN KEY (job_id) REFERENCES
```

o_jobs(job_id));

CREATE TABLE o_departments ( department_id INT PRIMARY KEY, department_name VARCHAR(100) NOT NULL, manager_id INT, FOREIGN KEY (manager_id) REFERENCES o_employees(employee_id);

2. Add the Human Resources job to the jobs table:

```
INSERT INTO o_jobs (job_id, job_title, min_salary, max_salary)
VALUES (1, 'Human Resources', 50000, 100000);
```

3. Add the three new employees to the employees table:

```
INSERT INTO o_employees (employee_id, first_name, last_name, email, hire_date, job_id, salary)
VALUES
(1, 'John', 'Doe', 'john.doe@example.com', '2024-11-01', 1, 60000),
(2, 'Jane', 'Smith', 'jane.smith@example.com', '2024-11-02', 1, 65000),
(3, 'Alice', 'Johnson', 'alice.johnson@example.com', '2024-11-03', 1, 70000);
```

4. Add Human Resources to the departments table:

```
INSERT INTO o_departments (department_id, department_name, manager_id)
VALUES (1, 'Human Resources', NULL);
```

5. Why is it important to be able to modify a table?

Being able to modify a table in a database is important for several reasons. First, it ensures data accuracy since information can change over time, and updates may be necessary. Second, businesses often evolve, and modifying tables allows them to adapt to new needs,

such as adding new columns or adjusting existing data.

1. CREATE a table called Artists.

a. Add the following to the table:

- artist ID
- first name
- last name
- band name
- email
- hourly rate
- song ID from d_songs table

b. INSERT one artist from the d_songs table.

c. INSERT one artist of your own choosing; leave song_id blank.

d. Give an example how each of the following may be used on the table that you have created:

1) ALTER TABLE

2) DROP TABLE

3) RENAME TABLE

4) TRUNCATE

5) COMMENT ON TABLE



CREATE TABLE Artists ( artist_id INT PRIMARY KEY,
first_name VARCHAR(50) NOT NULL, last_name VARCHAR(50) NOT NULL,
band_name VARCHAR(100), email VARCHAR(100) UNIQUE NOT NULL,
hourly_rate DECIMAL(10, 2), song_id INT,
FOREIGN KEY (song_id) REFERENCES d_songs(song_id);

```
INSERT INTO Artists (artist_id, first_name, last_name, band_name, email, hourly_rate, song_id)
VALUES (1, 'John', 'Doe', 'The Rockers', 'john.doe@example.com', 50.00, 1);

INSERT INTO Artists (artist_id, first_name, last_name, band_name, email, hourly_rate)
VALUES (2, 'Jane', 'Smith', 'The Pop Stars', 'jane.smith@example.com', 60.00);
```

ALTER TABLE Artists ADD genre VARCHAR(50);

DROP TABLE Artists;

RENAME TABLE Artists TO MusicArtists;
TRUNCATE TABLE Artists;
COMMENT ON TABLE Artists IS 'Table storing artist details including their bands and song references.';

a. Explain to students how you want the DJs on Demand artist's table assignment to be completed. Students should be able to list the term followed by the SQL statement they used. For example:

2. In your o_employees table, enter a new column called "Termination." The datatype for the new column should be VARCHAR2. Set the DEFAULT for this column as SYSDATE to appear as character data in the format: February 20th, 2003.

```
        ALTER TABLE o_employees
ADD Termination VARCHAR2(50);

CREATE OR REPLACE TRIGGER set_default_termination
BEFORE INSERT ON o_employees
FOR EACH ROW
BEGIN
  :NEW.Termination := TO_CHAR(SYSDATE, 'FMMonth DDth, YYYY');
END;
/
```

3. Create a new column in the o_employees table called start_date. Use the TIMESTAMP WITH LOCAL TIME ZONE as the datatype.

```
ALTER TABLE o_employees ADD start_date TIMESTAMP WITH LOCAL TIME ZONE;
```

4. Truncate the o_jobs table. Then do a SELECT * statement. Are the columns still there? Is the data still there?

```
TRUNCATE TABLE o_jobs;
```

Yes, the columns in the o_jobs table will still be there after the TRUNCATE operation. The structure of the table (i.e., its columns and their datatypes) remains unchanged.

No, the data will not be there. The TRUNCATE command removes all rows from the table, effectively resetting it to an empty state.

5. What is the distinction between TRUNCATE, DELETE, and DROP for tables?

- **DELETE** removes rows from a table based on a condition, can be rolled back, and maintains the table structure. It is slower for large datasets since each row is logged individually.
- **TRUNCATE** removes all rows from a table, is faster than DELETE, and deallocates the data pages. It doesn't allow selective row removal and is not easily rolled back.
- **DROP** removes both the table and its data completely from the database. It cannot be rolled back, and the table structure is also deleted.

6. List the changes that can and cannot be made to a column.

You can make several changes to a column, such as renaming it, changing its data type (if compatible), adding or modifying default values, setting or removing constraints (like `NOT NULL` or `UNIQUE`), and allowing or disallowing NULL values. However, some changes are not possible, such as changing a column's data type to an incompatible type, renaming a column in

certain databases, or dropping a column that is part of a constraint or view without first removing those dependencies. In some cases, you may need to recreate the table to perform certain column modifications.

7. Add the following comment to the o_jobs table: "New job description added"
View the data dictionary to view your comments.

```
COMMENT ON TABLE o_jobs IS 'New job description added';
SELECT * FROM user_tab_comments WHERE table_name = 'O_JOBS';
```

8. Rename the o_jobs table to o_job_description.

```
ALTER TABLE o_jobs RENAME TO o_job_description;
```

9.F_staffs table exercises:

A. Create a copy of the f_staffs table called copy_f_staffs and use this copy table for the remaining labs in this lesson.

B. Describe the new table to make sure it exists.

C. Drop the table.

D. Try to select from the table.

E. Investigate your recyclebin to see where the table went.

```
CREATE TABLE copy_f_staffs AS SELECT * FROM f_staffs;
DESCRIBE copy_f_staffs;
DROP TABLE copy_f_staffs;
SELECT * FROM copy_f_staffs;
ERROR: table or view does not exist
SHOW RECYCLEBIN;
```

```
FLASHBACK TABLE copy_f_staffs TO BEFORE DROP;
PURGE TABLE copy_f_staffs;
```

a. Try to select from the dropped table by using the value stored in the OBJECT_NAME column. You will need to copy and paste the name as it is exactly, and enclose the new name in " " (double quotes). So if the dropped name returned to you is BIN$Q+x1nJdcUnngQESYELVIdQ==$0, you need to write a query that refers to "BIN$Q+x1nJdcUnngQESYELVIdQ==$0".

```
SELECT OBJECT_NAME FROM USER_RECYCLEBIN;
SELECT * FROM "BIN$Q+x1nJdcUnngQESYELVIdQ==$0";
FLASHBACK TABLE "BIN$Q+x1nJdcUnngQESYELVIdQ==$0" TO BEFORE DROP;
```

b. Undrop the table.

```
SELECT OBJECT_NAME FROM USER_RECYCLEBIN;
FLASHBACK TABLE "BIN$Q+x1nJdcUnngQESYELVIdQ==$0" TO BEFORE DROP;
```

c. Describe the table.

```
DESCRIBE copy_f_staffs;
```

11. Still working with the copy_f_staffs table, perform an update on the table.

a. Issue a select statement to see all rows and all columns from the copy_f_staffs table;

```
UPDATE copy_f_staffs SET staff_name = 'John Doe' WHERE staff_id = 101;
```

b. Change the salary for Sue Doe to 12 and commit the change.

```
UPDATE copy_f_staffs SET salary = 12 WHERE staff_name = 'Sue Doe';
```

c. Issue a select statement to see all rows and all columns from the copy_f_staffs table;

```
SELECT * FROM copy_f_staffs;
```

d. For Sue Doe, update the salary to 2 and commit the change.

```
UPDATE copy_f_staffs SET salary = 2 WHERE staff_name = 'Sue Doe';
```

e. Issue a select statement to see all rows and all columns from the copy_f_staffs table;

```
SELECT * FROM copy_f_staffs;
```

f. Now, issue a FLASHBACK QUERY statement against the copy_f_staffs table, so you can see all the changes made.

```
SELECT * FROM copy_f_staffs AS OF TIMESTAMP (SYSTIMESTAMP - INTERVAL '5' MINUTE);
```

g. Investigate the result of f), and find the original salary and update the copy_f_staffs table salary column for Sue Doe back to her original salary.

```
UPDATE copy_f_staffs SET salary = 15 WHERE staff_name = 'Sue Doe';
```

**Ex. No. : 14 Date:**

**Register No.: Name:**

---

## Intro to Constraints; NOT NULL and UNIQUE Constraints

Global Fast Foods has been very successful this past year and has opened several new stores. They need to add a table to their database to store information about each of their store's locations. The owners want to make sure that all entries have an identification number, date opened, address, and city and that no other entry in the table can have the same email address. Based on this information, answer the following questions about the global_locations table. Use the table for your answers.

| Global Fast Foods global_locations Table | | | | | | |
|---|---|---|---|---|---|---|
| **NAME** | **TYPE** | **LENGTH** | **PRECISION** | **SCALE** | **NULLABLE** | **DEFAULT** |
| Id | | | | | | |
| name | | | | | | |
| date_opened | | | | | | |
| address | | | | | | |
| city | | | | | | |
| zip/postal code | | | | | | |
| phone | | | | | | |

| | | | | | | |
|---|---|---|---|---|---|---|
| email | | | | | | |
| manager_id | | | | | | |
| Emergency contact | | | | | | |

1. What is a “constraint” as it relates to data integrity?

**Constraint in Data Integrity**: A constraint is a rule that enforces data accuracy and consistency in a database. Common types include PRIMARY KEY (unique identifier), FOREIGN KEY (ensures referential integrity), NOT NULL (prevents null values), and CHECK (validates data against a condition).

2. What are the limitations of constraints that may be applied at the column level and at the table level?

**Limitations of Column-Level vs. Table-Level Constraints**: Column-level constraints apply only to individual columns, while table-level constraints can involve multiple columns. However, column-level constraints are simpler, while table-level constraints may make table design complex if overused.

3. Why is it important to give meaningful names to constraints?

**Importance of Meaningful Constraint Names**: Meaningful names make constraints easier to understand, maintain, and troubleshoot. For instance, a constraint named `fk_location_id` clearly indicates it’s a foreign key related to locations, making maintenance more intuitive.

4. Based on the information provided by the owners, choose a datatype for each column. Indicate the length, precision, and scale for each NUMBER datatype.

- location_id: NUMBER(5,0), not nullable, primary key
- location_name: VARCHAR2(50), not nullable
- address: VARCHAR2(100), not nullable
- city: VARCHAR2(30), not nullable

- state: VARCHAR2(20), nullable
- postal_code: VARCHAR2(10), nullable
- phone_number: VARCHAR2(15), nullable
- manager_id: NUMBER(5,0), nullable, foreign key
- opening_date: DATE, nullable
- annual_revenue: NUMBER(10,2), nullable

5. Use "(nullable)" to indicate those columns that can have null values.

```
CREATE TABLE global_fast_foods_locations (
   location_id      NUMBER(5, 0) NOT NULL,
   location_name    VARCHAR2(50) NOT NULL,
   address          VARCHAR2(100) NOT NULL,
   city             VARCHAR2(30) NOT NULL,
   state            VARCHAR2(20),
   postal_code      VARCHAR2(10),
   phone_number     VARCHAR2(15),
   manager_id       NUMBER(5, 0),
   opening_date     DATE,
   annual_revenue   NUMBER(10, 2),
   CONSTRAINT pk_location_id PRIMARY KEY (location_id),
   CONSTRAINT fk_manager_id FOREIGN KEY (manager_id) REFERENCES
staff(staff_id)
);
```

6. Write the CREATE TABLE statement for the Global Fast Foods locations table to define the constraints at the column level.

```
CREATE TABLE global_fast_foods_locations (
id          NUMBER(4)       NOT NULL,
loc_name    VARCHAR2(20)    NOT NULL UNIQUE,
```

```
    date        DATE           NOT NULL,
    address     VARCHAR2(30)   NOT NULL,
    city        VARCHAR2(20)   NOT NULL,
    zip_postal  VARCHAR2(20)   NOT NULL,
    phone       VARCHAR2(15)   NOT NULL,
    email       VARCHAR2(80)   NOT NULL UNIQUE,
    manager_id  NUMBER(4)      NOT NULL,
    contact     VARCHAR2(40)   NULL
);
```

7. Execute the CREATE TABLE statement in Oracle Application Express.

Copy the `CREATE TABLE` statement into the SQL command interface of Oracle Application Express and execute it.

8. Execute a DESCRIBE command to view the Table Summary information.

```
DESCRIBE global_fast_foods_locations;
```

9. Rewrite the CREATE TABLE statement for the Global Fast Foods locations table to define the UNIQUE constraints at the table level. Do not execute this statement.

| NAME | TYPE | LENGTH | PRECISION | SCALE | NULLABLE | DEFAULT |
|---|---|---|---|---|---|---|
| id | number | 4 | | | | |
| loc_name | varchar2 | 20 | | | X | |
| | date | | | | | |
| address | varchar2 | 30 | | | | |
| city | varchar2 | 20 | | | | |
| zip_postal | varchar2 | 20 | | | X | |

```
CREATE TABLE global_fast_foods_locations (
   id          NUMBER(4)       NOT NULL,
   loc_name    VARCHAR2(20)    NOT NULL,
   date        DATE            NOT NULL,
   address     VARCHAR2(30)    NOT NULL,
   city        VARCHAR2(20)    NOT NULL,
   zip_postal  VARCHAR2(20)    NOT NULL,
   phone       VARCHAR2(15)    NOT NULL,
   email       VARCHAR2(80)    NOT NULL,
   manager_id  NUMBER(4)       NOT NULL,
   contact     VARCHAR2(40)    NULL,
   CONSTRAINT unique_loc_name UNIQUE (loc_name),
   CONSTRAINT unique_email UNIQUE (email)
);
```

# PRIMARY KEY, FOREIGN KEY, and CHECK Constraints

1. What is the purpose of a

- PRIMARY KEY
- FOREIGN KEY
- CHECK CONSTRAINT

A **PRIMARY KEY** uniquely identifies each record in a table, ensuring that values in the key column(s) are both unique and non-null. It maintains the integrity of each record by preventing duplicates.

A **FOREIGN KEY** establishes a link between tables by referencing the primary key in another table, enforcing referential integrity and ensuring that relationships between records are valid.A **CHECK CONSTRAINT** restricts the values allowed in a column by specifying a condition that must be met, helping to enforce specific business rules and maintain data accuracy.

2. Using the column information for the animals table below, name constraints where applicable at the table level, otherwise name them at the column level. Define the primary key (animal_id). The license_tag_number must be unique. The admit_date and vaccination_date columns cannot contain null values.

animal_id NUMBER(6)
name VARCHAR2(25)
license_tag_number NUMBER(10)
admit_date DATE

adoption_id NUMBER(5),

vaccination_date DATE

```
create table animals (
  animal_id number(6),
  name varchar2(25),
  license_tag_number number(10),
  admit_date date not null,
  adoption_id number(5),
  vaccination_date date not null,
  constraint pk_animal_id primary key (animal_id),
  constraint unq_license_tag_number unique (license_tag_number) );
```

3. Create the animals table. Write the syntax you will use to create the table.

```
create table animals (
  animal_id number(6) constraint pk_animal_id primary key,
  name varchar2(25),
  license_tag_number number(10) constraint unq_license_tag_number unique,
  admit_date date not null,
  adoption_id number(5),
  vaccination_date date not null
);
```

4. Enter one row into the table. Execute a SELECT * statement to verify your input. Refer to the graphic below for input.

| ANIMAL_ID | NAME | LICENSE_TAG_NUMBER | ADMIT_DATE | ADOPTION_ID | VACCINATION_DATE |
|---|---|---|---|---|---|
| 101 | Spot | 35540 | 10-Oct-2004 | 205 | 12-Oct-2004 |

```
insert into animals (animal_id, name, license_tag_number, admit_date, adoption_id,
vaccination_date)
values (101, 'Spot', 35540, to_date('10-Oct-2004', 'DD-Mon-YYYY'), 205, to_date('12-Oct-2004',
'DD-Mon-YYYY'));

select * from animals;
```

5. Write the syntax to create a foreign key (adoption_id) in the animals table that has a corresponding primary-key reference in the adoptions table. Show both the column-level and table-level syntax. Note that because you have not actually created an adoptions table, no adoption_id primary key exists, so the foreign key cannot be added to the animals table.

```
create table animals (
   animal_id number(6) constraint pk_animal_id primary key,
   name varchar2(25),
   license_tag_number number(10) constraint unq_license_tag_number unique,
   admit_date date not null,
   adoption_id number(5) constraint fk_adoption_id references adoptions(adoption_id),
   vaccination_date date not null );
```

6. What is the effect of setting the foreign key in the ANIMAL table as:

a. ON DELETE CASCADE

b. ON DELETE SET NULL

**ON DELETE CASCADE**: When a row in the adoptions table is deleted, any related rows in the animals_table with the same adoption_id are automatically deleted as well. This is useful for ensuring no orphaned records remain when a parent record is removed.

**ON DELETE SET NULL**: When a row in the adoptions table is deleted, the adoption_id in any related rows in the animals table is set to NULL. This allows the child records to remain while indicating the absence of a parent record.

7. What are the restrictions on defining a CHECK constraint?

A CHECK constraint can only refer to columns within the same table and cannot reference other tables.

1. The condition in a CHECKconstraint must evaluate to a Boolean value, and if it evaluates to false, the insert or update operation is rejected.
2. CHECK constraints cannot enforce data types; they can only validate that values meet specific criteria.
3. CHECK constraint cannot contain subqueries or aggregate functions, limiting it to simple expressions comparing column values.

## PRACTICE PROBLEM

### Managing Constraints

Using Oracle Application Express, click the SQL Workshop tab in the menu bar. Click the Object Browser and verify that you have a table named copy_d_clients and a table named copy_d_events. If you don't have these tables in your schema, create them before completing the exercises below. Here is how the original tables are related. The d_clients table has a primary key client_number. This has a primary-key constraint and it is referenced in the foreign-key constraint on the d_events table.

NOTE: The practice exercises use the d_clients and d_events tables in the DJs on Demand database. Students will work with copies of these two tables named copy_d_clients and copy_d_events. Make sure they have new copies of the tables (without changes made from previous exercises). Remember, tables copied using a subquery do not have the integrity constraints as established in the original tables. When using the SELECT statement to view the constraint name, the tablenames must be all capital letters.

1. What are four functions that an ALTER statement can perform on constraints?

Add a Constraint

Drop a Constraint

Modify a Constraint

Enable or Disable a Constraint

2. Since the tables are copies of the original tables, the integrity rules are not passed onto the new tables; only the column datatype definitions remain. You will need to add a PRIMARY KEY constraint to the copy_d_clients table. Name the primary key copy_d_clients_pk . What is the syntax you used to create the PRIMARY KEY constraint to the copy_d_clients.table?

```
alter table copy_d_clients add constraint copy_d_clients_pk primary key (column_name);
```

3. Create a FOREIGN KEY constraint in the copy_d_events table. Name the foreign key copy_d_events_fk. This key references the copy_d_clients table client_number column. What is the syntax you used to create the FOREIGN KEY constraint in the copy_d_events table?

```
alter table copy_d_events add constraint copy_d_events_fk foreign key
(client_number) references copy_d_clients(client_number);
```

4. Use a SELECT statement to verify the constraint names for each of the tables. Note that the tablenames must be capitalized.

a. The constraint name for the primary key in the copy_d_clients table is `copy_d_clients_pk`

5. Drop the PRIMARY KEY constraint on the copy_d_clients table. Explain your results.

The command removes the primary key constraint copy_d_clients_pk from copy_d_clients, so column_name no longer enforces unique or NOT NULL values. This may allow duplicates or NULL entries in column_name, depending on other constraints. If successful, it shows Table altered.; otherwise, an error appears if the constraint name is invalid.

6. Add the following event to the copy_d_events table. Explain your results.

| ID | NAME | EVENT_DATE | DESCRIPTION | COST | VENUE_ID | PACKAGE_CODE | THEME_CODE | CLIENT_NUMBER |
|---|---|---|---|---|---|---|---|---|
| 140 | Cline Bas Mitzvah | 15-Jul-2004 | Church and Private Home formal | 4500 | 105 | 87 | 77 | 7125 |

```
INSERT INTO copy_d_events (  id, name, event_date, description, cost, venue_id, package_code, theme_code, client_number
)VALUES (140, 'Cline Bas Mitzvah', '15-Jul-2004', 'Church and Private Home', 4500, 105, 87, 77, 7125);
```

7. Create an ALTER TABLE query to disable the primary key in the copy_d_clients table. Then add the values from #6 to the copy_d_events table. Explain your results.

```
ALTER TABLE copy_d_clients DROP CONSTRAINT copy_d_clients_pk;
INSERT INTO copy_d_events (
   id, name, event_date, description, cost, venue_id, package_code, theme_code, client_number)
VALUES (    140, 'Cline Bas Mitzvah', '15-Jul-2004', 'Church and Private Home', 4500, 105, 87, 77, 7125);
```

8. Repeat question 6: Insert the new values in the copy_d_events table. Explain your results.

```
insert into copy_d_events (  id, name, event_date, description, cost, venue_id, package_code, theme_code, client_number) values ( 140, 'Cline Bas Mitzvah', '15-Jul-2004', 'Church and Private Home', 4500, 105, 87, 77, 7125 );
```

9. Enable the primary-key constraint in the copy_d_clients table. Explain your results.

```
alter table copy_d_clients add constraint copy_d_clients_pk primary key (column_name);
```

10. If you wanted to enable the foreign-key column and reestablish the referential integrity between

these two tables, what must be done?

```
alter table copy_d_events add constraint fk_client_number foreign key (client_number) references copy_d_clients (id);
```

11. Why might you want to disable and then re-enable a constraint?

Disabling and re-enabling a constraint can facilitate bulk data operations or schema changes without causing conflicts. It can also improve performance during high-volume data processing by temporarily suspending integrity checks. Once the necessary changes are made, re-enabling the constraint ensures ongoing data integrity.

12. Query the data dictionary for some of the constraints that you have created. How does the data dictionary identify each constraint type?

```
select name as constraint_name, type_desc as constraint_type, OBJECT_NAME(parent_object_id) as table_name from sys.objects where type_desc in ('primary_key_constraint', 'foreign_key_constraint') and OBJECT_NAME(parent_object_id) in ('copy_d_clients', 'copy_d_events');
```

**Ex. No. : 15 Date:**

**Register No.: Name:**

---

## Creating Views

1. What are three uses for a view from a DBA's perspective?

Data Security
Simplified Querying
Data Consistency

2. Create a simple view called view_d_songs that contains the ID, title and artist from the DJs on Demand table for each "New Age" type code. In the subquery, use the alias "Song Title" for the title column

```
create view view_d_songs as select ID,  title as "Song Title",  artist from
DJs_on_Demand where  type_code = 'New Age';
```

3. SELECT * FROM view_d_songs. What was returned?

The query `SELECT * FROM view_d_songs` returns all ID, Song Title, and artist columns for songs with a type code of "New Age" from the `DJs on Demand` table.

4. REPLACE view_d_songs. Add type_code to the column list. Use aliases for all columns.

Or use alias after the CREATE statement as show

```
create or replace view view_d_songs as
select
ID as "Song ID",
 title as "Song Title",
artist as "Artist",
type_code as "Type Code" from   DJs_on_Demand where  type_code = 'New Age';
```

OUTPUT:

5. Jason Tsang, the disk jockey for DJs on Demand, needs a list of the past events and those planned for the coming months so he can make arrangements for each event's equipment setup. As the company manager, you do not want him to have access to the price that clients paid for their events. Create a view for Jason to use that displays the name of the event, the event date, and the theme description. Use aliases for each column name.

```
        create view view_jason_events as
select
   event_name as "Event Name",
   event_date as "Event Date",
   theme_description as "Theme Description"
from
   events_table;
```

4. It is company policy that only upper-level management be allowed access to individual employee salaries. The department managers, however, need to know the minimum, maximum, and average salaries, grouped by department. Use the Oracle database to prepare a view that displays the needed information for department managers.

```
create view view_department_salaries as
select
   department_id as "Department ID",
   min(salary) as "Minimum Salary",
   max(salary) as "Maximum Salary",
   avg(salary) as "Average Salary"
from
   employees_table  -- Replace with the actual name of the employees table
group by
   department_id;
```

## DML Operations and Views

Use the DESCRIBE statement to verify that you have tables named copy_d_songs, copy_d_events, copy_d_cds, and copy_d_clients in your schema. If you don't, write a query to create a copy of each.

1. Query the data dictionary USER_UPDATABLE_COLUMNS to make sure the columns in the base tables will allow UPDATE, INSERT, or DELETE. All table names in the data dictionary are stored in uppercase.

```
select
   table_name,
   column_name,
   updatable
from  user_updatable_columns where
    table_name in ('YOUR_TABLE_NAME_1', 'YOUR_TABLE_NAME_2');  -- Replace with
actual table names
```

Use the same syntax but change table_name of the other tables.

2. Use the CREATE or REPLACE option to create a view of *all* the columns in the copy_d_songs table called view_copy_d_songs.

```
create or replace view view_copy_d_songs as select *  from copy_d_songs;
```

3. Use view_copy_d_songs to INSERT the following data into the underlying copy_d_songs table. Execute a SELECT * from copy_d_songs to verify your DML command. See the graphic.

| ID | TITLE | DURATION | ARTIST | TYPE_CODE |
|---|---|---|---|---|
| 88 | Mello Jello | 2 | The What | 4 |

INSERT INTO view_copy_d_songs (ID, TITLE, DURATION, ARTIST, TYPE_CODE)
VALUES (88, 'Mello Jello', 2, 'The What', 4);

4. Create a view based on the DJs on Demand COPY_D_CDS table. Name the view read_copy_d_cds. Select all columns to be included in the view. Add a WHERE clause to restrict the year to 2000. Add the WITH READ ONLY option.

```
CREATE VIEW read_copy_d_cds AS SELECT * FROM copy_d_cds
WHERE year = 2000 WITH READ ONLY;
```

5. Using the read_copy_d_cds view, execute a DELETE FROM read_copy_d_cds WHERE cd_number = 90;

```
DELETE FROM read_copy_d_cds WHERE cd_number = 90;
```

6. Use REPLACE to modify read_copy_d_cds. Replace the READ ONLY option with WITH CHECK OPTION CONSTRAINT ck_read_copy_d_cds. Execute a SELECT * statement to verify that the view exists.

```
CREATE OR REPLACE VIEW read_copy_d_cds AS SELECT *
FROM copy_d_cds WHERE year = 2000 WITH CHECK OPTION CONSTRAINT
ck_read_copy_d_cds;
```

7. Use the read_copy_d_cds view to delete any CD of year 2000 from the underlying copy_d_cds.

```
SELECT * FROM read_copy_d_cds;
```

8. Use the read_copy_d_cds view to delete cd_number 90 from the underlying copy_d_cds table.

```
DELETE FROM read_copy_d_cds WHERE cd_number = 90;
```

9. Use the read_copy_d_cds view to delete year 2001 records.

```
DELETE FROM read_copy_d_cds WHERE year = 2001;
```

10. Execute a SELECT * statement for the base table copy_d_cds. What rows were deleted?

```
SELECT * FROM copy_d_cds;
```

11. What are the restrictions on modifying data through a view?

Modifying data through a view is restricted if the view is created with `WITH READ ONLY`, includes complex queries (e.g., joins, aggregates), lacks all columns needed for `INSERT`, or if updates violate the `WITH CHECK OPTION`. Constraints on the underlying tables must also be met.

12. What is Moore's Law? Do you consider that it will continue to apply indefinitely? Support your opinion with research from the internet.

Moore's Law states that the number of transistors on a chip doubles about every two years, boosting computing power. While it held true for decades, experts note it is no longer sustainable due to physical and economic limits.

13. What is the "singularity" in terms of computing?

The "singularity" in computing is a theoretical point where AI surpasses human intelligence, leading to rapid, self-sustaining technological growth that may drastically change society and human life.

**Managing Views**

1. Create a view from the copy_d_songs table called view_copy_d_songs that includes only the title and artist. Execute a SELECT * statement to verify that the view exists.

```
CREATE VIEW view_copy_d_songs AS SELECT title, artist FROM copy_d_songs;
```

2. Issue a DROP view_copy_d_songs. Execute a SELECT * statement to verify that the view has been deleted.

```
SELECT * FROM view_copy_d_songs;
```

3. Create a query that selects the last name and salary from the Oracle database. Rank the salaries from highest to lowest for the top three employees.

```
SELECT last_name, salary, RANK() OVER (ORDER BY salary DESC) AS salary_rank
FROM employees WHERE ROWNUM <= 3;
```

4. Construct an inline view from the Oracle database that lists the last name, salary, department ID, and maximum salary for each department. Hint: One query will need to calculate maximum salary by department ID.

```
SELECT last_name, salary, department_id, max_salary FROM (
SELECT last_name, salary, department_id,
MAX(salary) OVER (PARTITION BY department_id) AS max_salary  FROM employees);
```

5. Create a query that will return the staff members of Global Fast Foods ranked by salary from lowest to highest.

```
SELECT staff_member_name, salary FROM staff
WHERE company = 'Global Fast Foods' ORDER BY salary ASC;
```

## Indexes and Synonyms

1. What is an index and what is it used for?

An **index** is a database structure that speeds up data retrieval by allowing quick access to rows, similar to an index in a book. It improves performance for searches and sorts.

2. What is a ROWID, and how is it used?

A **ROWID** is a unique identifier for each row in a table, indicating its physical location, which facilitates fast access to data without scanning the entire table.

3. When will an index be created automatically?

Indexes are automatically created when primary keys or unique constraints are defined on a column, ensuring data integrity and uniqueness .

4. Create a nonunique index (foreign key) for the DJs on Demand column (cd_number) in the D_TRACK_LISTINGS table. Use the Oracle Application Express SQL Workshop Data Browser to confirm that the index was created.

```
CREATE INDEX idx_cd_number ON D_TRACK_LISTINGS (cd_number);
```

5. Use the join statement to display the indexes and uniqueness that exist in the data dictionary for the DJs on Demand D_SONGS table.

```
SELECT i.index_name, i.uniqueness, i.column_name FROM all_indexes JOIN all_ind_columns c ON i.index_name = c.index_name WHERE i.table_name = 'D_SONGS';
```

6. Use a SELECT statement to display the index_name, table_name, and uniqueness from the data dictionary USER_INDEXES for the DJs on Demand D_EVENTS table.

```
SELECT index_name, table_name, uniqueness
FROM user_indexes
WHERE table_name = 'D_EVENTS';
```

7. Write a query to create a synonym called dj_tracks for the DJs on Demand d_track_listings table.

```
CREATE SYNONYM dj_tracks FOR d_track_listings
```

8. Create a function-based index for the last_name column in DJs on Demand D_PARTNERS table that makes it possible not to have to capitalize the table name for searches. Write a SELECT statement that would use this index.

```
SELECT *FROM D_PARTNERS
WHERE LOWER(last_name) = 'smith';
```

9. Create a synonym for the D_TRACK_LISTINGS table. Confirm that it has been created by querying the data dictionary.

```
CREATE SYNONYM d_track_listings_syn FOR D_TRACK_LISTINGS;
```

10.Drop the synonym that you created in question

```
DROP SYNONYM d_track_listings_syn;
```

**Ex. No. : 16 Date:**

**Register No.: Name:**

---

## 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

**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.

**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

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
• 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);

**In the 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**

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:**

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.

```
CREATE SEQUENCE DEPT_ID_SEQ START WITH 200
INCREMENT BY 10 MAXVALUE 1000;
```

2. Write a query in a script to display the following information about your sequences: sequence name, maximum value, increment size, and last number

```
SELECT sequence_name, max_value, increment_by, last_number FROM user_sequences
WHERE sequence_name = 'DEPT_ID_SEQ';
```

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.

```
INSERT INTO DEPT (DEPT_ID, DEPT_NAME) VALUES (DEPT_ID_SEQ.NEXTVAL, 'Education');
INSERT INTO DEPT (DEPT_ID, DEPT_NAME) VALUES (DEPT_ID_SEQ.NEXTVAL, 'Administration');
SELECT * FROM DEPT;
```

4. Create a nonunique index on the foreign key column (DEPT_ID) in the EMP table.

```
CREATE INDEX idx_dept_id ON EMP (DEPT_ID);
```

5. Display the indexes and uniqueness that exist in the data dictionary for the EMP table.

```
SELECT index_name, uniqueness FROM user_indexes
WHERE table_name = 'EMP';
```

**Ex. No. : 17 Date:**

**Register No.: Name:**

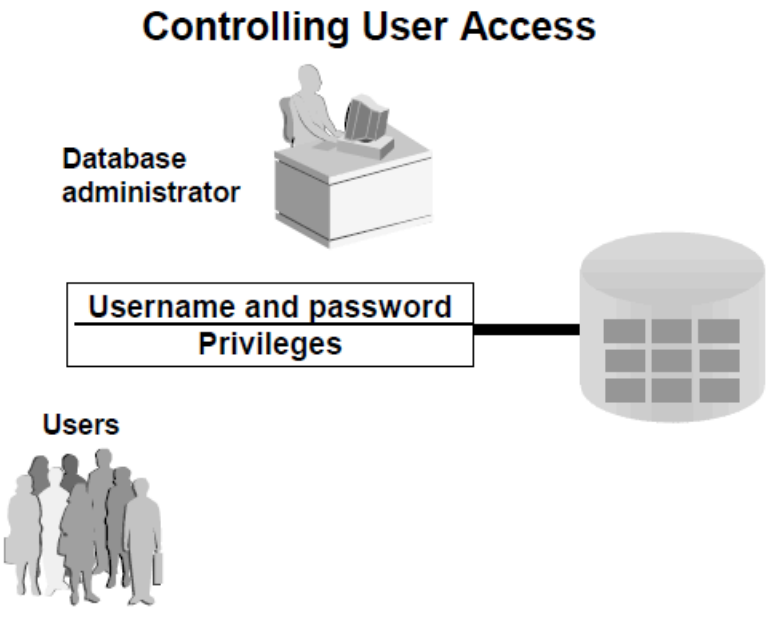
---

## 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

• 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

– 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.

**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.

**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

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**

• 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?

> To allow a user to log on to the Oracle Server, they should be granted the **CREATE SESSION** privilege. This is a **system privilege**, which enables the user to establish a session with the database.

2. What privilege should a user be given to create tables?

> A user should be granted the **CREATE TABLE** privilege to create tables in the Oracle database. This is also considered a **system privilege**, allowing users to define their own tables.

3. If you create a table, who can pass along privileges to other users on your table?

The owner of the table (the user who created it) can grant privileges to other users on that table. Specifically, the owner can use the **GRANT** statement to assign privileges such as SELECT, INSERT, UPDATE, and DELETE to other users or roles.

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?

As a DBA, you should use **roles** to manage system privileges efficiently. A role is a named group of related privileges that can be granted to users or other roles. By creating a role with the required system privileges and then assigning that role to multiple users, you can streamline privilege management and simplify user administration.

5. What command do you use to change your password?

```
ALTER USER your_username IDENTIFIED BY new_password;
```

6. Grant another user access to your DEPARTMENTS table. Have the user grant you query access to his or her DEPARTMENTS table.

```
GRANT SELECT ON DEPARTMENTS TO other_user;
```

7. Query all the rows in your DEPARTMENTS table

.

```
SELECT * FROM DEPARTMENTS;
```

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.

```
INSERT INTO DEPARTMENTS (DEPARTMENT_ID, DEPARTMENT_NAME) VALUES (500, 'Education');
INSERT INTO DEPARTMENTS (DEPARTMENT_ID, DEPARTMENT_NAME) VALUES (510, 'Human Resources');
```

9. Query the USER_TABLES data dictionary to see information about the tables that you own.

```
SELECT * FROM USER_TABLES;
```

10. Revoke the SELECT privilege on your table from the other team.

```
REVOKE SELECT ON DEPARTMENTS FROM other_user;
```

11. Remove the row you inserted into the DEPARTMENTS table in step 8 and save the changes.

```
COMMIT;
```

# PL/Sql

**Ex. No. : 18 Date:**

**Register No.: Name:**

---

## PL/SQL

### Control Structures

In addition to SQL commands,PL/SQL can also process data usin 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;

***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;
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;

:

:

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

***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;

PROGRAM 1

Write a PL/SQL block to calculate the incentive of an employee whose ID is 110.

```
declare
    emp_id number := 110;
    emp_salary number;
    emp_incentive number;
begin
     select salary
    into emp_salary
    from employees
    where employee_id = emp_id;

        emp_incentive := emp_salary * 0.05;
    dbms_output.put_line('Incentive for Employee ID ' || emp_id || ' is: ' || emp_incentive);
exception
    when no_data_found then
        dbms_output.put_line('Employee with ID ' || emp_id || ' not found.');
    when others then
        dbms_output.put_line('An error occurred: ' || sqlerrm);
end;
    /
```

PROGRAM 2

Write a PL/SQL block to show an invalid case-insensitive reference to a quoted and without quoted user-defined identifier.

```
declare
   "myvariable" number := 10;
   myVariable number := 20;
   result number;
begin
   result := "myvariable" + myVariable;

   dbms_output.put_line('Result: ' || result);
exception
   when others then
      dbms_output.put_line('An error occurred: ');
end;
/
```

PROGRAM 3

Write a PL/SQL block to adjust the salary of the employee whose ID 122.

Sample table: employees

```
declare
    emp_id number := 122;
    emp_salary number;
    new_salary number;
    salary_increase_percentage number := 0.10;  -- 10% increase
begin
    select salary into emp_salary from employees
    where employee_id = emp_id;
    new_salary := emp_salary + (emp_salary * salary_increase_percentage);
    update employees
    set salary = new_salary
    where employee_id = emp_id;
    dbms_output.put_line('Salary for Employee ID ' || emp_id || ' adjusted to: ' || new_salary);
exception
    when no_data_found then
        dbms_output.put_line('Employee with ID ' || emp_id || ' not found.');
    when others then
        dbms_output.put_line('An error occurred: ');
end;
/
```

PROGRAM 4

Write a PL/SQL block to create a procedure using the "IS [NOT] NULL Operator" and show AND operator returns TRUE if and only if both operands are TRUE.

```
create or replace procedure check_params
(
   p1 in varchar2,
   p2 in varchar2)
is
begin
   if p1 is not null and p2 is not null then
      dbms_output.put_line('Both parameters are NOT NULL.');
   else
      dbms_output.put_line('At least one parameter is NULL.');
   end if;
end check_params;
/
declare
   a varchar2(50) := 'Test';
   b varchar2(50) := 'Value';
begin
   check_params(a, b);
   a := null;
   check_params(a, b);
   a := 'Another Test';
   b := null;
```

```
    check_params(a, b);
    a := null;
    b := null;
    check_params(a, b);
end;
/
```

PROGRAM 5

Write a PL/SQL block to describe the usage of LIKE operator including wildcard characters and escape character.

```
declare
   v_name employees.first_name%type;
   cursor c1 is
      select first_name
      from employees
      where first_name like 'A%';  -- Names starting with 'A'

   cursor c2 is
      select first_name
      from employees
      where first_name like '%n\_e%' escape '\';  -- Names containing 'n_e'
begin
   dbms_output.put_line('Employees with names starting with A:');
   for emp in c1 loop
      dbms_output.put_line(emp.first_name);
   end loop;
```

```
    dbms_output.put_line('Employees with names containing n_e:');
    for emp in c2 loop
        dbms_output.put_line(emp.first_name);
    end loop;
exception
    when no_data_found then
        dbms_output.put_line('No employees found.');
    when others then
        dbms_output.put_line('An error occurred: ' || sqlerrm);
end;
/
```

PROGRAM 6

Write a PL/SQL program to arrange the number of two variable in such a way that the small number will store in num_small variable and large number will store in num_large variable.

```
declare
    num1 number := 10;  -- First number
    num2 number := 20;  -- Second number
    num_small number;    -- Variable to store the smaller number
    num_large number;    -- Variable to store the larger number
begin
```

```
    if num1 < num2 then
        num_small := num1;
        num_large := num2;
    else
        num_small := num2;
        num_large := num1;
    end if;

    dbms_output.put_line('Small number: ' || num_small);
    dbms_output.put_line('Large number: ' || num_large);
end;
/
```

PROGRAM 7

Write a PL/SQL procedure to calculate the incentive on a target achieved and display the message either the record updated or not.

```
create or replace procedure calculate_incentive (
    emp_id in number,
    target_achieved in number,
    base_incentive in number
) is
    incentive_amount number;
    rows_updated number;
begin
    incentive_amount := target_achieved * (base_incentive / 100);
```

```
        update employees  set incentive = incentive_amount
        where employee_id = emp_id;
        rows_updated := sql%rowcount;
        if rows_updated > 0 then
            dbms_output.put_line('record updated successfully. incentive: ' ||
incentive_amount);
        else
            dbms_output.put_line('no record found to update for employee id: ' || emp_id);
        end if;
    exception
        when others then
            dbms_output.put_line('an error occurred: ');
    end calculate_incentive;
    /
```

PROGRAM 8

Write a PL/SQL procedure to calculate incentive achieved according to the specific sale limit.

```
create or replace procedure calculate_incentive (
    emp_id in number,
    total_sales in number
) is
    incentive_amount number := 0;
begin
    if total_sales < 10000 then
```

```
    incentive_amount := total_sales * 0.05;  -- 5% incentive for sales less than
10,000
  elsif total_sales >= 10000 and total_sales < 20000 then
    incentive_amount := total_sales * 0.10;  -- 10% incentive for sales between
10,000 and 19,999
  elsif total_sales >= 20000 then
    incentive_amount := total_sales * 0.15;  -- 15% incentive for sales of 20,000 or
more
  end if;
  update employees
  set incentive = incentive_amount
  where employee_id = emp_id;
```

```
  if sql%rowcount > 0 then
    dbms_output.put_line('incentive calculated and updated successfully: ' ||
incentive_amount);
  else
    dbms_output.put_line('no record found to update for employee id: ' || emp_id);

  end if;
exception
  when others then
    dbms_output.put_line('an error occurred: ');
end calculate_incentive;
/
```

PROGRAM 9

Write a PL/SQL program to count number of employees in department 50 and check whether this department have any vacancies or not. There are 45 vacancies in this department.

```
declare
    v_employee_count number;
    v_vacancies number := 45;  -- Number of vacancies in department 50
begin
    select count(*)
    into v_employee_count
    from employees
    where department_id = 50;

    dbms_output.put_line('Number of employees in department 50: ' ||
v_employee_count);

    if v_employee_count < v_vacancies then
        dbms_output.put_line('Department 50 has vacancies available.');
    else
        dbms_output.put_line('Department 50 is fully occupied with no vacancies.');
    end if;
exception
```

```
    when others then
        dbms_output.put_line('An error occurred: ');
end;
/
```

PROGRAM 10

Write a PL/SQL program to count number of employees in a specific department and check whether this department have any vacancies or not. If any vacancies, how many vacancies are in that department.

```
declare
    v_department_id number := 50;  -- Specify the department ID
    v_employee_count number;
    v_vacancies number := 45;       -- Number of vacancies in the specified department
begin
    select count(*)
    into v_employee_count from employees
    where department_id = v_department_id;
    dbms_output.put_line('Number of employees in department ' || v_department_id || ': ' || v_employee_count);
    if v_employee_count < v_vacancies then
        dbms_output.put_line('Department ' || v_department_id || ' has vacancies available: ' || (v_vacancies - v_employee_count));
    else
        dbms_output.put_line('Department ' || v_department_id || ' is fully occupied with no vacancies.');
```

```
    end if;
  exception
    when others then
      dbms_output.put_line('An error occurred: ');
  end;
  /
```

PROGRAM 11

Write a PL/SQL program to display the employee IDs, names, job titles, hire dates, and salaries of all employees.

```
declare
   cursor emp_cursor is
      select employee_id, first_name || ' ' || last_name as employee_name, job_title,
hire_date, salary
      from employees
      join jobs on employees.job_id = jobs.job_id;  -- Assuming a join with jobs table to
get job titles

   v_employee_id employees.employee_id%type;
   v_employee_name varchar2(100);
   v_job_title jobs.job_title%type;
   v_hire_date employees.hire_date%type;
   v_salary employees.salary%type;
begin
   open emp_cursor;
```

```
    loop
        fetch emp_cursor into v_employee_id, v_employee_name, v_job_title,
v_hire_date, v_salary;
        exit when emp_cursor%notfound;


        dbms_output.put_line('Employee ID: ' || v_employee_id ||
                        ', Name: ' || v_employee_name ||
                        ', Job Title: ' || v_job_title ||
                        ', Hire Date: ' || to_char(v_hire_date, 'DD-MON-YYYY') ||
                        ', Salary: ' || v_salary);
    end loop;


    close emp_cursor;
exception
    when others then
        dbms_output.put_line('An error occurred:');
end;
```

PROGRAM 12

Write a PL/SQL program to display the employee IDs, names, and department names of all employees.

```
declare
    cursor emp_cursor is
```

```
        select e.employee_id, e.first_name || ' ' || e.last_name as emp_name,
d.department_name
        from employees e join departments d on e.department_id = d.department_id;
    v_emp_id employees.employee_id%type;
    v_emp_name varchar2(100);
    v_dept_name departments.department_name%type;
begin
    open emp_cursor;
    loop
        fetch emp_cursor into v_emp_id, v_emp_name, v_dept_name;
        exit when emp_cursor%notfound;
        dbms_output.put_line('Employee ID: ' || v_emp_id ||
                             ', Name: ' || v_emp_name ||
                             ', Department: ' || v_dept_name);
    end loop;
    close emp_cursor;
exception
    when others then
        dbms_output.put_line('An error occurred: ');
end;
/
```

PROGRAM 13

Write a PL/SQL program to display the job IDs, titles, and minimum salaries of all jobs.

```
declare
   cursor job_cursor is
      select job_id, job_title, min_salary from jobs;
   v_job_id jobs.job_id%type;
   v_job_title jobs.job_title%type;
   v_min_salary jobs.min_salary%type;
begin
   open job_cursor;
   loop
      fetch job_cursor into v_job_id, v_job_title, v_min_salary;
      exit when job_cursor%notfound;

      dbms_output.put_line('Job ID: ' || v_job_id ||
                           ', Title: ' || v_job_title ||
                           ', Minimum Salary: ' || v_min_salary);
   end loop;

   close job_cursor;
exception
   when others then
      dbms_output.put_line('An error occurred');
end;
/
```

PROGRAM 14

Write a PL/SQL program to display the employee IDs, names, and job history start dates of all employees.

```
declare
    cursor emp_cursor is
        select e.employee_id, e.first_name || ' ' || e.last_name as emp_name, j.start_date
        from employees e
        join job_history j on e.employee_id = j.employee_id;

    v_emp_id employees.employee_id%type;
    v_emp_name varchar2(100);
    v_start_date job_history.start_date%type;
begin
    open emp_cursor;
    loop
        fetch emp_cursor into v_emp_id, v_emp_name, v_start_date;
        exit when emp_cursor%notfound;

        dbms_output.put_line('Employee ID: ' || v_emp_id ||
                             ', Name: ' || v_emp_name ||
                             ', Job History Start Date: ' || to_char(v_start_date,
'DD-MON-YYYY'));
    end loop;
```

```
    close emp_cursor;
exception
    when others then
        dbms_output.put_line('An error occurred: ');
end;
/
```

PROGRAM 15

Write a PL/SQL program to display the employee IDs, names, and job history end dates of all employees.

```
declare
   cursor emp_cursor is
      select e.employee_id, e.first_name || ' ' || e.last_name as emp_name, j.end_date
      from employees e join job_history j on e.employee_id = j.employee_id;

   v_emp_id employees.employee_id%type;
   v_emp_name varchar2(100);
   v_end_date job_history.end_date%type;
begin
```

```
    open emp_cursor;

    loop
        fetch emp_cursor into v_emp_id, v_emp_name, v_end_date;
        exit when emp_cursor%notfound;

        dbms_output.put_line('Employee ID: ' || v_emp_id ||
                             ', Name: ' || v_emp_name ||
                             ', Job History End Date: ' || to_char(v_end_date, 'DD-MON-YYYY'));
    end loop;
    close emp_cursor;
exception
    when others then
        dbms_output.put_line('An error occurred: ');
end;
```

**Ex. No. : p-8 Date:**

**Register No.: Name:**

---

## 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.

**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
```

**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;
```

**PROCEDURE FOR 'IN' PARAMETER – CREATION, EXECUTION**

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

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

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

```
SQL> set serveroutput on;

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

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

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

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

```
SQL>insert into ittrain values (1001, 550);

SQL>insert into ittrain values (1002, 600);

SQL>select * from ittrain;
```

**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  /

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  /
```

## Program 1

## FACTORIAL OF A NUMBER USING FUNCTION

```
create or replace function calculate_factorial (
    p_number in number  -- Input number
) return number is
    fact number := 1;
begin
    -- Check if the number is negative
    if p_number < 0 then
        return null;
    elsif p_number = 0 then
        return 1;  -- Factorial of 0 is 1
    else
        -- Calculate factorial for positive numbers
        for i in 1..p_number loop
            fact := fact * i;
        end loop;
        return fact;
    end if;
end calculate_factorial;
/
```

Program 2

**Write a PL/SQL program using Procedures IN,INOUT,OUT parameters to retrieve the corresponding book information in library**

```
create or replace procedure book_info (
    p_book_id      IN number,
    p_available     INOUT number,
    p_title         OUT varchar2,
    p_author        OUT varchar2,
    p_publication   OUT number
) is
begin

    select title, author, publication_year, available_copies
    into p_title, p_author, p_publication, p_available
    from books
    where book_id = p_book_id;

    when no_data_found then
        p_title := 'Not Found';
        p_author := 'Not Found';
        p_publication := null;
        p_available := 0;
    when others then
        raise;
end book_info;
/
```

**Ex. No. : 19 Date:**

**Register No.: Name:**

---

# 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

- :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));

SQL> insert into itempls values('xxx',11,10000);

SQL> insert into itempls values('yyy',12,10500);

SQL> insert into itempls values('zzz',13,15500);

SQL> select * from itempls;

**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  /
```

```
SQL> insert into itempls values('aaa',14,34000);
```

```
SQL> delete from itempls where ename='xxx';
```

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

**TO DROP THE CREATED TRIGGER**

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

**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  /
```

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

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

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),dnonumber(5));

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

SQL> insert into ssempp values(2,'kala',' seniorlecturer',20000,12);

SQL> insert into ssempp values(5,'ajay','lecturer',30000,11);

SQL> insert into ssempp values(6,'vijay','lecturer',18000,11);

SQL> insert into ssempp values(3,'nila','professor',60000,12);

SQL> select * from ssempp;

**EXTRA PROGRAMS**

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

```
SQL> set serveroutput on;
SQL> declare
  2  begin
  3  for emy in (select eid,ename from ssempp)
  4  loop
  5  dbms_output.put_line('Employee id and employee name  are '|| emy.eid ‘and’|| emy.ename);
  6  end loop;
  7  end;
  8  /
```

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

**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 ssempp where dno=11;
 3  ecode ssempp.eid%type;
 4  esal empp.sal%type;
 5  begin
 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> /
```

**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  /
```

```
SQL> select * from ssempp;
```

**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  /
```

**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  /
```

**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);
 11  end if;
 12  end;
 13  /
```

**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  /
```

**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  /
```

**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;
```

**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;
  8  end;
  9  /
```

**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  /
```

**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  /
```

**TO CREATE SACCOUNT TABLE**

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

SQL> insert into saccount values ( 1,'mala',20000);
SQL> insert into saccount values (2,'kala',30000);

SQL> select * from saccount;
```

```
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  /
SQL> select * from saccount;
```

**TO CREATE TABLE SROUTES**

```
SQL> create table sroutes ( rno number(5), origin varchar2(20), destination varchar2(20), fare numbe
r(10), distance number(10));

SQL> insert into sroutes values ( 2, 'chennai', 'dindugal', 400,230);

SQL> insert into sroutes values ( 3, 'chennai', 'madurai', 250,300);

SQL> insert into sroutes values ( 6, 'thanjavur', 'palani', 350,370);

SQL> select * from sroutes;
```

```
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  /

SQL> select * from sroutes;
```

**TO CREATE SCA LCULATE 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;
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  /
```

```
SQL> select * from scalculate;
```

**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
8  f:=f*i;
9  i:=i-1;
10  end loop;
11  dbms_output.put_line('The value is ' || f);
12  end;
13  /
```

Program 1

Write a code in PL/SQL to develop a trigger that enforces referential integrity by preventing the deletion of a parent record if child records exist.

```
create or replace trigger del
before delete on parent_table
for each row
declare
   child_count integer;
begin
    select count(*) into child_count from child_table where parent_id = :old.parent_id;
   if child_count > 0 then
      raise_application_error(-20001, 'ERROR.');
   end if;
end del;
/
```

Program 2

Write a code in PL/SQL to create a trigger that checks for duplicate values in a specific column and raises an exception if found

```
create or replace trigger check_duplicate
before insert or update on table
for each row
declare
   duplicate_count integer;
begin
   select count(*) into duplicate_count from example_table  where
         unique_column =:new.unique_column;
   if duplicate_count > 0 then
      raise_application_error('ERROR');
   end if;
end check_duplicate;
/
```

Program 3

Write a code in PL/SQL to create a trigger that restricts the insertion of new rows if the total of a column's values exceeds a certain threshold.

```
create or replace trigger restrict_total_amount
before insert on transactions
for each row
declare
   total_amount number;
   threshold constant number := 1000000;  -- Set the threshold limit
begin
   -- Calculate the total sum of the amount column
   select sum(amount) into total_amount from transactions;

   if total_amount + :new.amount > threshold then
      raise_application_error(-20003, 'Total amount exceeds Threshold.');
   end if;
end restrict_total_amount;
/
```

Program 4

Write a code in PL/SQL to design a trigger that captures changes made to specific columns and logs them in an audit table.

```
create or replace trigger audit_employee_changes
after update on employees
for each row
begin
   if :old.salary != :new.salary or :old.position != :new.position then
      insert into employees_audit (audit_id, employee_id, column_name, old_value, new_value, changed_by, changed_at)
      values (employees_audit_seq.nextval, :old.employee_id,
            case
               when :old.salary != :new.salary then 'salary'
               else 'position'
            end,
            case
               when :old.salary != :new.salary then to_char(:old.salary)
               else :old.position
            end,
            case
               when :old.salary != :new.salary then to_char(:new.salary)
               else :new.position
            end,
            user, sysdate);
   end if;
end audit_employee_changes;
/
```

Program 5

Write a code in PL/SQL to implement a trigger that records user activity (inserts, updates, deletes) in an audit log for a given set of tables.

```
create table activity_audit (
    audit_id number primary key,
    table_name varchar2(50),
    operation varchar2(10),
    record_id number,
    old_values varchar2(4000),
    new_values varchar2(4000),
    changed_by varchar2(30),
    changed_at date
);

create or replace trigger log_employee_activity
after insert or update or delete on employees
for each row
declare
    v_old_values varchar2(4000);
    v_new_values varchar2(4000);
begin
    -- Capture old values for updates and deletes
    if deleting or updating then
        v_old_values := 'salary=' || :old.salary || ', position=' || :old.position;
    end if;
```

```
if inserting or updating then
    v_new_values := 'salary=' || :new.salary || ', position=' || :new.position;
  end if;

  -- Insert a log record into activity_audit
  insert into activity_audit (
    audit_id, table_name, operation, record_id, old_values, new_values, changed_by,
changed_at
  )
  values (
    activity_audit_seq.nextval,  -- Assuming a sequence exists for audit_id
    'employees',
    case
      when inserting then 'INSERT'
      when updating then 'UPDATE'
      when deleting then 'DELETE'
    end,
    :old.employee_id,        -- Use :old for delete, :new for insert; both work in update
    v_old_values,
    v_new_values,
    user,                    -- Captures the current database user
    sysdate                  -- Captures the timestamp of the operation
  );
end log_employee_activity;
```



Program 6

Write a code in PL/SQL to implement a trigger that automatically calculates and updates a running total column for a table whenever new rows are inserted.

```
create or replace trigger calculate_running_total
before insert on sales
for each row
declare
    v_total number;
begin
    -- Calculate the current total of sale_amount in the table
    select nvl(sum(sale_amount), 0)
    into v_total
    from sales;

    -- Set the running total for the new row
    :new.running_total := v_total + :new.sale_amount;
end calculate_running_total;
/
```

Program 7

Write a code in PL/SQL to create a trigger that validates the availability of items before allowing an order to be placed, considering stock levels and pending orders.

```
create or replace trigger validate_item_availability
before insert on orders
for each row
declare
    v_available_stock number;
    v_pending_orders number;
begin
    -- Check current stock level of the item
    select stock_level
    into v_available_stock
    from items
    where item_id = :new.item_id;
    select nvl(sum(order_quantity), 0)
    into v_pending_orders
    from orders
    where item_id = :new.item_id;
    if :new.order_quantity > v_available_stock - v_pending_orders then
        raise_application_error(-20004, 'Insufficient stock available to fulfill this order.');
    end if;
end validate_item_availability;
/
```

# MONGO DB

**Ex. No. : 20 Date:**

**Register No.: Name:**

---

## MONGO DB

MongoDB is a free and open-source cross-platform document-oriented database. Classified as a NoSQL database, MongoDB avoids the traditional table-based relational database structure in favor of JSON-like documents with dynamic schemas, making the integration of data in certain types of applications easier and faster.

### Create Database using mongosh

After connecting to your database using mongosh, you can see which database you are using by typing db in your terminal.

If you have used the connection string provided from the MongoDB Atlas dashboard, you should be connected to the myFirstDatabase database.

### Show all databases

To see all available databases, in your terminal type show dbs.

Notice that myFirstDatabase is not listed. This is because the database is empty. An empty database is essentially non-existant.

### Change or Create a Database

You can change or create a new database by typing use then the name of the database.

### Create Collection using mongosh

You can create a collection using the createCollection() database method.

## Insert Documents

### insertOne()

```
db.posts.insertOne({
  title: "Post Title 1",
  body: "Body of post.",
  category: "News",
  likes: 1,
  tags: ["news", "events"],
  date: Date()
})
```

**Ex. No. : 21 Date:**

**Register No.: Name:**

---

Structure of 'restaurants' collection:


```
{
 "address": {
    "building": "1007",
    "coord": [ -73.856077, 40.848447 ],
    "street": "Morris Park Ave",
    "zipcode": "10462"
 },
 "borough": "Bronx",
 "cuisine": "Bakery",
 "grades": [
    { "date": { "$date": 1393804800000 }, "grade": "A", "score": 2 },
    { "date": { "$date": 1378857600000 }, "grade": "A", "score": 6 },
    { "date": { "$date": 1358985600000 }, "grade": "A", "score": 10 },
    { "date": { "$date": 1322006400000 }, "grade": "A", "score": 9 },
    { "date": { "$date": 1299715200000 }, "grade": "B", "score": 14 }
 ],
 "name": "Morris Park Bake Shop",
 "restaurant_id": "30075445"
}
```

**1. Write a MongoDB query to find the restaurant Id, name, borough and cuisine for those restaurants which prepared dish except 'American' and 'Chinees' or restaurant's name begins with letter 'Wil'.**

```
db.restaurants.find(
 {
  $or: [
   { cuisine: { $nin: ['American', 'Chinese'] } },
   { name: { $regex: /^Wil/, $options: 'i' } }
  ]
 },
 {
  restaurant_id: 1,
  name: 1,
  borough: 1,
  cuisine: 1,
  _id: })
```

**2. Write a MongoDB query to find the restaurant Id, name, and grades for those restaurants which achieved a grade of "A" and scored 11 on an ISODate "2014-08-11T00:00:00Z" among many of survey dates..**

```
db.restaurants.find(
{
 grades: {  $elemMatch: {
    grade: "A",
    score: 11,
    date: { $eq: ISODate("2014-08-11T00:00:00Z") }
   }
 }
},
```

```
 {
  restaurant_id: 1,
  name: 1,
  grades: 1,
  _id: 0
 }
)
```

**3. Write a MongoDB query to find the restaurant Id, name and grades for those restaurants where the 2nd element of grades array contains a grade of "A" and score 9 on an ISODate "2014-08-11T00:00:00Z".**

```
        db.restaurants.find(
 {
  "grades.1": {
   $exists: true,
   $elemMatch: {
    grade: "A",
    score: 9,
    date: { $eq: ISODate("2014-08-11T00:00:00Z") }
   }
  }
 },
 {
  restaurant_id: 1,
  name: 1,
  grades: 1,
```

```
  _id: 0
 }
)
```

**4. Write a MongoDB query to find the restaurant Id, name, address and geographical location for those restaurants where 2nd element of coord array contains a value which is more than 42 and upto 52..**

```
db.restaurants.find(
 {
  "address.coord.1": {
   $gt: 42,
   $lte: 52
  }
 },
 {
  restaurant_id: 1,
  name: 1,
  address: 1,
  "address.coord": 1,
  _id: 0
 }
)
```

**5. Write a MongoDB query to arrange the name of the restaurants in ascending order along with all the columns.**

```
db.restaurants.find().sort({ name: 1 })
```

**6. Write a MongoDB query to arrange the name of the restaurants in descending along with all the columns.**

```
db.restaurants.find().sort({ name: -1 })
```

**7. Write a MongoDB query to arranged the name of the cuisine in ascending order and for that same cuisine borough should be in descending order.**

```
db.restaurants.find().sort({ cuisine: 1, borough: -1 })
```

**8. Write a MongoDB query to know whether all the addresses contains the street or not.**

```
db.restaurants.find(
{ "address.street": { $exists: true } },
{ _id: 0, "address.street": 1 }
)
```

**9. Write a MongoDB query which will select all documents in the restaurants collection where the coord field value is Double.**

```
db.restaurants.find({
 "address.coord": {
  $type: "double"
 }
})
```

**10. Write a MongoDB query which will select the restaurant Id, name and grades for those restaurants which returns 0 as a remainder after dividing the score by 7.**

```
db.restaurants.find(
 {
  grades: {
```

```
        $elemMatch: {
          score: { $mod: [7, 0] }
        }
      }
    },
    {
      restaurant_id: 1,
      name: 1,
      grades: 1,
      _id: 0
    }
  )
```

**11. Write a MongoDB query to find the restaurant name, borough, longitude and attitude and cuisine for those restaurants which contains 'mon' as three letters somewhere in its name.**

```
    db.restaurants.find(
{ name: { $regex: /mon/i }  // case-insensitive search for 'mon'
},
{
  name: 1,
  borough: 1,
  "address.coord": 1,  // This will include the coord array which has longitude and latitude
  cuisine: 1,
  _id: 0
}
```

)

**12. Write a MongoDB query to find the restaurant name, borough, longitude and latitude and cuisine for those restaurants which contain 'Mad' as first three letters of its name.**

```
        db.restaurants.find(
 {
  name: { $regex: /^Mad/i }  // Matches names starting with 'Mad', case-insensitive
 },
 {
  name: 1,
  borough: 1,
  "address.coord": 1,  // This will include the coord array which has longitude and latitude
  cuisine: 1,
  _id: 0
 }
)
```

**13. Write a MongoDB query to find the restaurants that have at least one grade with a score of less than 5.**

```
db.restaurants.find(
 {
  grades: {
   $elemMatch: {
    score: { $lt: 5 }  // Score less than 5
```

```
    }
  }
 }
)
```

**14. Write a MongoDB query to find the restaurants that have at least one grade with a score of less than 5 and that are located in the borough of Manhattan.**

```
db.restaurants.find(
 {
  borough: "Manhattan",
  grades: {
   $elemMatch: {
    score: { $lt: 5 }  // Score less than 5
   }
  }
 }
)
```

**15. Write a MongoDB query to find the restaurants that have at least one grade with a score of less than 5 and that are located in the borough of Manhattan or Brooklyn.**

```
db.restaurants.find(
  {
   borough: { $in: ["Manhattan", "Brooklyn"] },
   grades: {
    $elemMatch: {
     score: { $lt: 5 }  // Score less than 5
    }
```

```
    }
   }
  )
```

**16. Write a MongoDB query to find the restaurants that have at least one grade with a score of less than 5 and that are located in the borough of Manhattan or Brooklyn, and their cuisine is not American.**

```
    db.restaurants.find(
 {
  borough: { $in: ["Manhattan", "Brooklyn"] },
  cuisine: { $ne: "American" },  // Cuisine is not American
  grades: {
   $elemMatch: {
    score: { $lt: 5 }  // Score less than 5
   }
  }
 }
)
```

**17. Write a MongoDB query to find the restaurants that have at least one grade with a score of less than 5 and that are located in the borough of Manhattan or Brooklyn, and their cuisine is not American or Chinese.**

```
    db.restaurants.find(
 {
  borough: { $in: ["Manhattan", "Brooklyn"] },
  cuisine: { $nin: ["American", "Chinese"] },  // Cuisine is not American or Chinese
  grades: {
```

```
    $elemMatch: {
      score: { $lt: 5 }  // Score less than 5
    }
  }
 }
)
```

**18. Write a MongoDB query to find the restaurants that have a grade with a score of 2 and a grade with a score of 6.**

```
        db.restaurants.find(
 {
  grades: {
   $all: [
     { $elemMatch: { score: 2 } },
     { $elemMatch: { score: 6 } }
   ]
  }
 }
)
```

**19. Write a MongoDB query to find the restaurants that have a grade with a score of 2 and a grade with a score of 6 and are located in the borough of Manhattan.**

```
db.restaurants.find(
 {
  borough: "Manhattan",
  grades: {
```

```
    $all: [
      { $elemMatch: { score: 2 } },
      { $elemMatch: { score: 6 } }
    ]
  }
 }
)
```

**20. Write a MongoDB query to find the restaurants that have a grade with a score of 2 and a grade with a score of 6 and are located in the borough of Manhattan or Brooklyn.**

```
db.restaurants.find(
 {
  borough: { $in: ["Manhattan", "Brooklyn"] },
  grades: {
    $all: [
      { $elemMatch: { score: 2 } },
      { $elemMatch: { score: 6 } }
    ]
  }
 }
)
```

**21. Write a MongoDB query to find the restaurants that have a grade with a score of 2 and a grade with a score of 6 and are located in the borough of Manhattan or Brooklyn, and their cuisine is not American.**

```
        db.restaurants.find(
 {
```

```
  borough: { $in: ["Manhattan", "Brooklyn"] },
  cuisine: { $ne: "American" },  // Cuisine is not American
  grades: {
   $all: [
    { $elemMatch: { score: 2 } },
    { $elemMatch: { score: 6 } }
   ]
  }
 }
)
```

**22. Write a MongoDB query to find the restaurants that have a grade with a score of 2 and a grade with a score of 6 and are located in the borough of Manhattan or Brooklyn, and their cuisine is not American or Chinese.**

```
db.restaurants.find(
 {
  borough: { $in: ["Manhattan", "Brooklyn"] },
  cuisine: { $nin: ["American", "Chinese"] },  // Cuisine is not American or Chinese
  grades: {
   $all: [
    { $elemMatch: { score: 2 } },
    { $elemMatch: { score: 6 } }
   ]
  }
 }
)
```

**23. Write a MongoDB query to find the restaurants that have a grade with a score of 2 or a grade with a score of 6.**

```
db.restaurants.find(
 {
  $or: [
   { grades: { $elemMatch: { score: 2 } } },
   { grades: { $elemMatch: { score: 6 } } }
  ]
 }
)
```

## Sample document of 'movies' collection

```
 {
  _id: ObjectId("573a1390f29313caabcd42e8"),
plot: 'A group of bandits stage a brazen train hold-up, only to find a determined posse hot on their heels.',
genres: [ 'Short', 'Western' ],
runtime: 11,
cast: [
    'A.C. Abadie',
    "Gilbert M. 'Broncho Billy' Anderson",
    'George Barnes',
    'Justus D. Barnes'
  ],
poster:
'https://m.media-amazon.com/images/M/MV5BMTU3NjE5NzYtYTYyNS00MDVmLWIwYjgtMmYwYWIxZDYyNzU2XkEyXkFqcGdeQXVyNzQzNzQxNzI@._V1_SY1000_SX677_AL_.jpg',
```

title: 'The Great Train Robbery',

fullplot: "Among the earliest existing films in American cinema - notable as the first film that presented a narrative story to tell - it depicts a group of cowboy outlaws who hold up a train and rob the passengers. They are then pursued by a Sheriff's posse. Several scenes have color included - all hand tinted.",

languages: [ 'English' ],

released: ISODate("1903-12-01T00:00:00.000Z"),

directors: [ 'Edwin S. Porter' ],

rated: 'TV-G',

awards: { wins: 1, nominations: 0, text: '1 win.' },

lastupdated: '2015-08-13 00:27:59.177000000',

year: 1903,

imdb: { rating: 7.4, votes: 9847, id: 439 },

countries: [ 'USA' ],

type: 'movie',

tomatoes: {

viewer: { rating: 3.7, numReviews: 2559, meter: 75 },

fresh: 6,

critic: { rating: 7.6, numReviews: 6, meter: 100 },

rotten: 0,

lastUpdated: ISODate("2015-08-08T19:16:10.000Z")

}

1.Find all movies with full information from the 'movies' collection that released in the year 1893.

```
db.movies.find(
{
 year: 1893
}
)
```

2. Find all movies with full information from the 'movies' collection that have a runtime greater than 120 minutes.

```
db.movies.find(
  {
    runtime: { $gt: 120 }  // Filter for movies with runtime greater than 120 minutes
  }
)
```

3. Find all movies with full information from the 'movies' collection that have "Short" genre.

```
db.movies.find(
  {
    genres: "Short"  // Filter for movies that include "Short" in the genres array
  }
)
```

4. Retrieve all movies from the 'movies' collection that were directed by "William K.L. Dickson" and include complete information for each movie.

```
db.movies.find(
  {
    directors: "William K.L. Dickson"  // Filter for movies directed by William K.L. Dickson
  }
)
```

5. Retrieve all movies from the 'movies' collection that were released in the USA and include complete information for each movie.

```
db.movies.find(
  {
    countries: "USA"  // Filter for movies released in the USA
  }
)
```

6. Retrieve all movies from the 'movies' collection that have complete information and are rated as "UNRATED".

```
db.movies.find(
  {
    rated: "UNRATED"  // Filter for movies that are rated "UNRATED"
  }
)
```

7. Retrieve all movies from the 'movies' collection that have complete information and have received more than 1000 votes on IMDb.

```
db.movies.find(
  {
    "imdb.votes": { $gt: 1000 }  // Filter for movies with more than 1000 votes
  }
)
```

8. Retrieve all movies from the 'movies' collection that have complete information and have an IMDb rating higher than 7.

```
db.movies.find(
  {
```

```
    "imdb.rating": { $gt: 7 }  // Filter for movies with an IMDb rating higher than
7

  }

)
```

9. Retrieve all movies from the 'movies' collection that have complete information and have a viewer rating higher than 4 on Tomatoes

```
db.movies.find(

 {

  "tomatoes.viewer.rating": { $gt: 4 }  // Filter for movies with a viewer rating higher
than 4

 }

)
```

10. Retrieve all movies from the 'movies' collection that have received an award.

```
db.movies.find(

  {

    "awards.wins": { $gt: 0 }  // Filter for movies with more than 0 wins

  }

)
```

11. Find all movies with title, languages, released, directors, writers, awards, year, genres, runtime, cast, countries from the 'movies' collection in MongoDB that have at least one nomination.

```
db.movies.find(
  {
    "awards.nominations": { $gt: 0 }  // Filter for movies with more than 0 nominations
  },
  {
    title: 1,
    languages: 1,
    released: 1,
    directors: 1,
    writers: 1,
    awards: 1,
    year: 1,
    genres: 1,
    runtime: 1,
    cast: 1,
    countries: 1,
    _id: 0  // Exclude the _id field from the results
  })
```

12. Find all movies with title, languages, released, directors, writers, awards, year, genres, runtime, cast, countries from the 'movies' collection in MongoDB with cast including "Charles Kayser".

```
        db.movies.find(
  {
   cast: "Charles Kayser"  // Filter for movies where the cast includes "Charles Kayser"
  },
  {
   title: 1,
   languages: 1,
   released: 1,
   directors: 1,
   writers: 1,
   awards: 1,
   year: 1,
   genres: 1,
   runtime: 1,
   cast: 1,
   countries: 1,
   _id: 0  // Exclude the _id field from the results
  }
)
```

13. Retrieve all movies with title, languages, released, directors, writers, countries from the 'movies' collection in MongoDB that released on May 9, 1893.

```
db.movies.find(
```

```
  {
    released: ISODate("1893-05-09T00:00:00.000Z")  // Filter for movies released on May 9, 1893
  },
  {
    title: 1,
    languages: 1,
    released: 1,
    directors: 1,
    writers: 1,
    countries: 1,
    _id: 0  // Exclude the _id field from the results
  }
)
```

14. Retrieve all movies with title, languages, released, directors, writers, countries from the 'movies' collection in MongoDB that have a word "scene" in the title.

```
db.movies.find(
  {
    title: { $regex: /scene/i }  // Filter for movies with "scene" in the title (case-insensitive)
  },
  {
```

```
    title: 1,
    languages: 1,
    released: 1,
    directors: 1,
    writers: 1,
    countries: 1,
    _id: 0  // Exclude the _id field from the results
  }
)
```