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. 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** Database Connectivity with Front End Tools(Python/C/C++/JAVA)and Back End Tools(MySQL/SQLite/CASSANDRA/MONGO DB) For any problem selected, write the ER Diagram, apply ER mapping rules, normalize the relations, and follow the application development process. Make sure that the application should have five or more tables, at least one trigger and one stored procedure, using suitable frontend tool. Indicative areas include a) Inventory Control System. b) Material Requirement Processing. c) Hospital Management System. d) Railway Reservation System. e) Personal Information System. f) Web Based User Identification System. g) Timetable Management System. h) Hotel Management System 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 : _____

Ex. No. : 0 **Date:15.11.2024**

Register No.: 221701055 **Name: S Sneha**

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 2.UPDATE 3.DELETE 4.MERGE	Enter new rows, changes existing rows, removes unwanted rows from tables in the database respectively.
DDL	1.CREATE 2.ALTER 3.DROP 4.RENAME 5.TRUNCATE	Sets up, changes and removes data structures from tables.
TCL	1.COMMIT 2.ROLLBACK 3.SAVEPOINT	Manages the changes made by DML statements. Changes to the data can be grouped together into logical transactions.
DCL	1.GRANT 2.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:18.11.2024**

Register No.: 221701055 **Name:S Sneha**

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)

ANSWER:

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

ANSWER:

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.

ANSWER:

select * from my_employee

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

ANSWER:

INSERT INTO my_employee (id,first_name, last_name, salary,userid)

VALUES (3,'Ben', 'Biri', 756, CONCAT(SUBSTR('Ben', 1, 1), SUBSTR('Biri', 1, 7)));

INSERT INTO my_employee (id,first_name, last_name, salary,userid)

VALUES (4,'Chad', 'Newman', 756, CONCAT(SUBSTR('Chad', 1, 1), SUBSTR('Newman', 1, 7)));

5. Delete Betty dancs from MY _EMPLOYEE table.

ANSWER:

delete from my_employee where first_name='Betty' and last_name='Dancs';

6. Empty the fourth row of the emp table.

ANSWER:

update my_employee set first_name=null,last_name=null,salary=null,userid=null where id=4;

7. Make the data additions permanent.

ANSWER:

Commit;

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

ANSWER:

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.

ANSWER:

update my_employee set salary=1000 where salary<900;

Ex. No. : P-1 **Date: 18.11.2024**

Register No.: 221701055 **Name: S Sneha**

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

ANSWER:

select employee_id,first_name||' '||last_name as name ,salary from employees;

b) List out the employees who works under manager 100

ANSWER:

select first_name||' '||last_name as name from employees where manager_id=100;

---

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

ANSWER:

select first_name||' '||last_name as name from employees where salary>=4800;

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

ANSWER:

select first_name||' '||last_name as name from employees where last_name='Austin';

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

ANSWER:

select first_name||' '||last_name as name from employees where department_id in(60,70,80);

f) Display the unique Manager_Id.

ANSWER:

select distinct manager_id from employees;

Ex. No. : 2 **Date: 18.11.2024**

Register No.: 221701055 **Name: S sneha**

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)

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)

JOB_GRADE TABLE

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

LOCATION TABLE

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

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

ANSWER:

create table dept as select * from department;

OUTPUT:

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

ANSWER:

create table emp as select * from employees;

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

ANSWER:

ALTER TABLE employee MODIFY (first_name varchar(50));

ALTER TABLE employeesMODIFY (last_name varchar(50));

OUTPUT:

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.

ANSWER:

create table employees2 as select employee_id,first_name,last_name,salary,department_id from employees;

5. Drop the EMP table.

ANSWER:

drop table emp;

6. Rename the EMPLOYEES2 table as EMP.

ANSWER:

RENAME employees2 TO emp;

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

ANSWER:

COMMENT ON TABLE department IS 'This is a comment about the department';

COMMENT ON TABLE emp IS 'This is a comment about the emp';

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

ANSWER:

alter table emp drop column first_name ;

Ex. No. : 2 **Date:18.11.2024**

Register No.: 221701055 **Name: S Sneha**

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)

ANSWER:

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

ANSWER:

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.

ANSWER:

select * from my_employee

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

ANSWER:

INSERT INTO my_employee (id,first_name, last_name, salary,userid)

VALUES (3,'Ben', 'Biri', 756, CONCAT(SUBSTR('Ben', 1, 1), SUBSTR('Biri', 1, 7)));

INSERT INTO my_employee (id,first_name, last_name, salary,userid)

VALUES (4,'Chad', 'Newman', 756, CONCAT(SUBSTR('Chad', 1, 1), SUBSTR('Newman', 1, 7)))

5. Make the data additions permanent.

ANSWER:

Commit;

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

ANSWER:

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.

ANSWER:

update my_employee set salary=1000 where salary<900;

8. Delete Betty dancs from MY _EMPLOYEE table.

ANSWER:

delete from my_employee where first_name='Betty' and last_name='Dancs';

9. Empty the fourth row of the emp table.

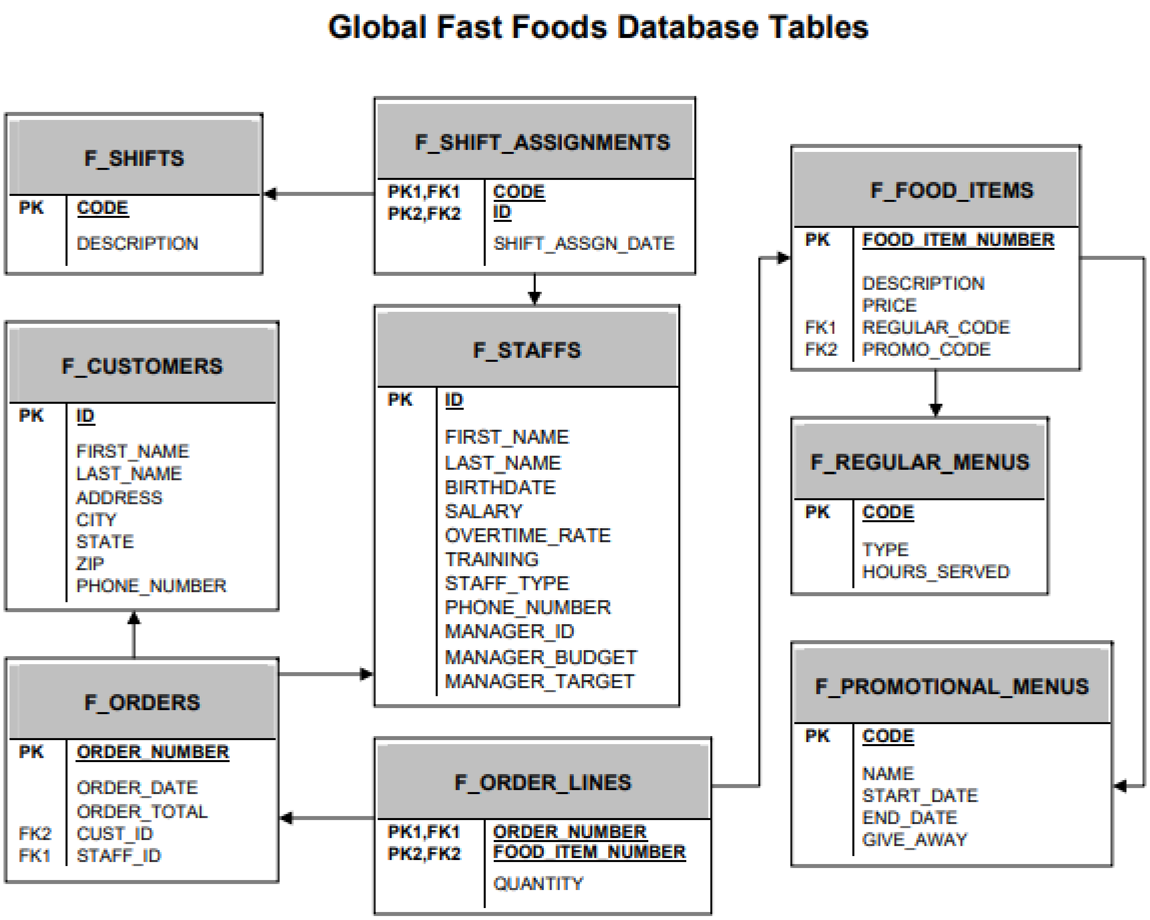
ANSWER:

update my_employee set first_name=null,last_name=null,salary=null,userid=null where id=4;

Ex. No. : 3 **Date:18.11.2024**

Register No.: 221701055 **Name: S Sneha**

Working With Column, Characters and rows



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.

ANSWER:

Select last_name,address from f_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;

ANSWER:

SELECT first_name FROM

f_staffs;

b.

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

d_clients;

ANSWER:

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

d_clients;

c.

SELECT DISCTINCT f_order_lines FROM

quantity;

ANSWER:

SELECT DISTINCT quantity FROM

f_order_lines;

d.

SELECT order number FROM

f_orders;

ANSWER:

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 ***

ANSWER:

select “***”||first_name||”***”||first_name||”***" as “Super Star” from f_staffs;

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.

ANSWER:

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

ANSWER:

Create table increment as select employee_last_name,salary*0.05 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?

ANSWER:

All the values are marked as nullable except for ID attribute because we didn’t give not null condition to other attributes also ID is act as primary key for that particular table;

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.

ANSWER:

Create table D_CDs(inventory_item number,cd_title varchar(100),music_producer varchar(100),year_purchased date);

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

ANSWER:

True.

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

ANSWER:

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;

ANSWER:

i) The “,”(comma) need to come after last_name in above query

ii) You need to “*”(Astrid) symbol to multiply 2 values

iii) You need to use “AS” keyword in order to alias a column .

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

ANSWER:

Salary * 12 , According to BODMAS rule the multiplication operation is 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

ANSWER:

b)*(Astrid) symbol use to return all the columns from the table.

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

a. selection

b. projection

c. partitioning

d. join

ANSWER:

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”

ANSWER:

a) 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

ANSWER:

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

ANSWER:

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.

ANSWER:

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;

ANSWER:

b)SELECT DEPARTMENT_ID, LAST_NAME, FIRST_NAME FROM

employees;

and

d)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;

ANSWER:

a)SELCT * FROM employees;

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.

ANSWER:

221701055@RAJALAKSHMI.EDU.IN CT_superplus.sql select “+++”||first_name||”+++”||first_name||”+++” as “Super Star” from f_staffs; 221701055@RAJALAKSHMI.EDU.IN 3 months ago

Ex. No. : 4 **Date:18.11.2024**

Register No.: 221701055 **Name:S Sneha**

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.

ANSWER:

alter table emp add constraint my_emp_id_pk primary key (employee_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.

ANSWER:

alter table dept add constraint my_dept_id_pk primary key (dept_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.

ANSWER:

alter table emp add dept_id number;

alter table emp add constraint my_emp_dept_id_fk foreign key(dept_id) references dept(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.

ANSWER:

alter table emp add commission_pct number(2,2);

alter table emp add constraint comm_check check (commission_pct>0);

Ex. No. : 5 **Date:19.11.2024**

Register No.: 221701055 **Name: S Sneha**

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.

Identify the Errors

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

Queries

ANSWER:

```
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.

ANSWER:

```
desc employees;
select * from employees;
```

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

ANSWER:

```
select employee_id, last_name, job_id, hire_date from employees order by employee_id;
```

4. Provide an alias STARTDATE for the hire date.

ANSWER:

Select hire_date as STARTDATE from employees;

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

ANSWER:

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.

ANSWER:

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.

ANSWER:

Selectemployee_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:19.11.2024

Register No.: 2217010555 Name:S Sneha

COMPARISON OPERATORS

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

ANSWER:

SELECT * FROM d_partners WHERE auth_expense_amt = 0 OR auth_expense_amt IS NULL;

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

ANSWER:

SELECT last_name as "Possible Candidates" 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;

ANSWER:

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.

ANSWER:

```
SELECT title as "Song" FROM d_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');

ANSWER:

First uses AND, so either the sides must return true, but in second case – OR, more records may come, because either of the sides need to be true for row selection.

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

ANSWER:

SELECT last_name FROM f_staffs WHERE last_name LIKE '%e%' AND 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."

ANSWER:

SELECT first_name || ' ' || last_name as "Full Name" FROM f_staffs WHERE salary > 6.5 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.

ANSWER:

SELECT first_name || ' ' || last_name as "Full Name" FROM employees

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?

ANSWER:

SELECT DISTINCT d_venues.loc_type FROM d_events JOIN d_venues ON d_events.venue_id = d_venues.id

WHERE d_venues.loc_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

ANSWER:

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.

ANSWER:

SELECT first_name || ' ' || last_name as "Full Name"

FROM employees

WHERE hire_date > '31-May-1998' AND hire_date < '01-Jun-1999' AND salary < 8000 AND last_name like '%en%';

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

ANSWER:

SELECT LOWER(email) || '@institutedomain.com' as "Email Address"

FROM employees

WHERE salary > 9000 AND (commission_pct = 0 OR commission_pct IS NULL) AND hire_date >= '01-Jan-1996' AND hire_date <= '31-Mar-1996' ;

Ex. No. : 6 **Date:19.11.2024**

Register No.: 221701055 **Name: S Sneha**

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.

ANSWER:

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.

ANSWER:

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)

ANSWER:

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)

ANSWER:

SELECT last_name, job_id, hire_date FROM employees WHERE hire_date BETWEEN '20-Feb-1998' AND '01-May-1998' ORDER BY hire_date;

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

ANSWER:

SELECT FROM WHERE last_name, department_id employees department_id IN (20, 50) ORDER BY last_name;

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)

ANSWER:

SELECT FROM WHERE AND last_name "Employee", salary "Monthly Salary" employees salary BETWEEN 5000 AND 12000 department_id IN (20, 50) order by last_name;

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

ANSWER:

SELECT FROM WHERE last_name, hire_date employees hire_date LIKE '%94';

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

ANSWER:

SELECT FROM WHERE last_name, job_id employees manager_id IS NULL;

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

ANSWER:

SELECT FROM WHERE last_name, salary, commission_pct employees 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)

ANSWER:

SELECT FROM WHERE last_name employees 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)

ANSWER:

SELECT FROM WHERE AND last_name employees last_name LIKE '%a%' 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)

ANSWER:

SELECT FROM WHERE AND last_name, job_id, salary employees job_id IN ('SA_REP', 'ST_CLERK') 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)

ANSWER:

SELECT FROM WHERE last_name "Employee", salary "Monthly Salary", commission_pct employees commission_pct = .20;

Ex. No. : P-3 **Date:19.11.2024**

Register No.: 221701055 **Name:S Sneha**

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;

ANSWER:

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.

ANSWER:

SELECT title FROM d_cds ORDER BY year, title;

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

ANSWER:

SELECT title as "Our Collection" FROM d_cds ORDER BY title DESC;

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

ANSWER:

```
SELECT department_id , last_name, manager_id
FROM employees
WHERE employee_id<125
ORDER BY  department_id DESC , last_name DESC;
```

Ex. No. : 7 **Date:19.11.2024**

Register No.: 221701055 **Name: S Sneha**

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

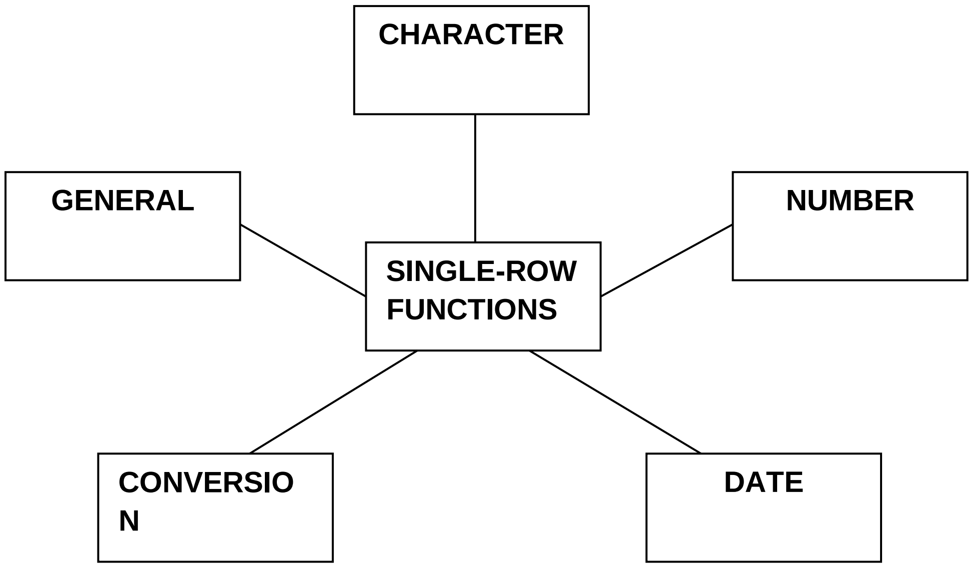
Syntax

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

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)	

<u>Example:</u>

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); upper (char);	*select lower (‘HELLO’) from dual;* *select upper (‘hello’) from dual;*	Hello 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;

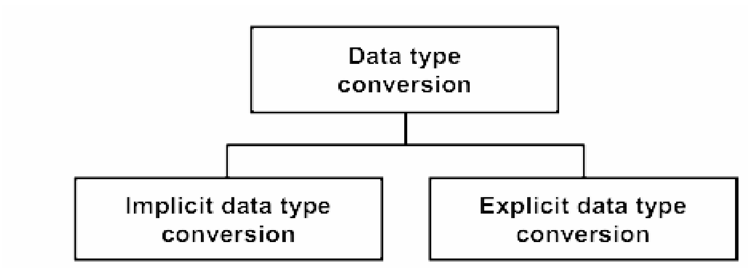
Conversion Functions

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:

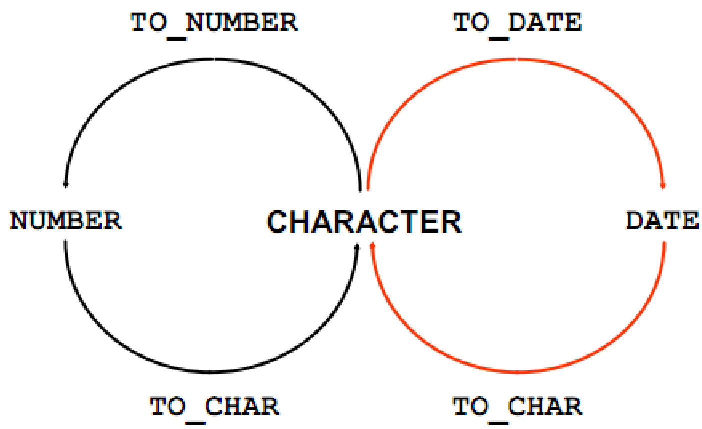
From	To
VARCHAR2 or CHAR	NUMBER
VARCHAR2 or CHAR	DATE
NUMBER	VARCHAR2
DATE	VARCHAR2

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:

<u>Example:</u>

Using the TO_CHAR Function with Dates

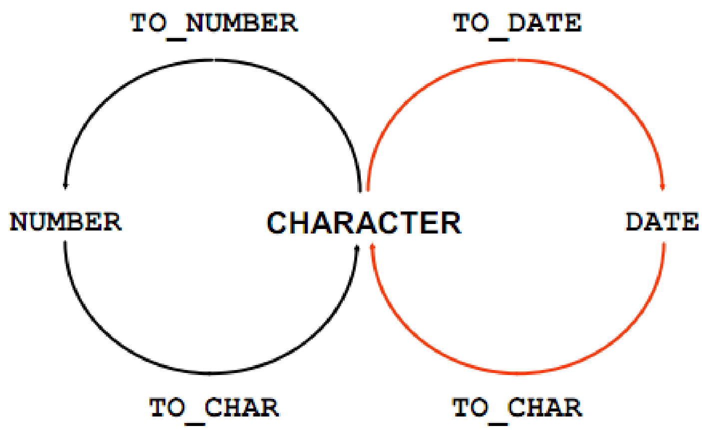
TO_CHAR(date, 'format_model')

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.

ANSWER:

Select SYSDATE as system_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.

ANSWER:

select employee_id as employee_number ,last_name,(salary *15.5)/100 as New_Salary from employees order by salary desc;

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.

ANSWER:

alter table employees add increase number;

update employees set increase=salary-(salary*15.5)/100;

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.

ANSWER:

select initcap(last_name),length (last_name) from employees where last_name like'J%' orlast_name like'M%' or last_name like'W%' 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.

ANSWER:

select initcap(last_name),length(last_name) from employees where last_name like "J%" or last_name like"M%" or last_name like"W%" or last_name like "H%" 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.

ANSWER:

select length(last_name),round(SYSDATE-hire_date) as MONTHS_WORKED 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.

ANSWER:

select(last_name||" "||'earns'||" "||salary||" "||'monthly but want'||" "||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.

ANSWER:

select last_name, lpad(salary,15,'$') as SALARY from employees;

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

ANSWER:

Select last_name ,T)_CHAR(Next_day(Add_months(hire_date,6),'Monday'),'Day fmDdspth "of" Month YYYY')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.

ANSWER:

select last_name ,hire_date,TO_CHAR(hire_date,'Day') as hire_day from employees order by TO_CHAR(hire_date -1,'d');

Ex. No. : P-4 **Date:19.11.2024**

Register No.: 221701055 **Name:S Sneha**

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.

ANSWER:

a) Single row
b) Multiple row
c) Single row
d) Multiple row
e) Single row
f) 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.

ANSWER:

a) select job_id AVG(salary) from employees where job_id='SA_MAN' group by job_id;
b) select COUNT(employee_id) from employees group by employee_id;
c) select MAX(salary) from employees group by salary;
d) select MIN(salary) from employees group by salary;

e) select SUM(salary) from employees;

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.

ANSWER:

SELECT AVG(salary) FROM employees; --8775

SELECT COUNT(salary) FROM employees; --20

SELECT MAX(salary) FROM employees; --24000

SELECT MIN(salary) FROM employees; --2500

SELECT SUM(salary) FROM employees; --175500

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

ANSWER:

select “The Best Class ”||”Oracle”||”Internet”||”Academy” from dual;

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

The Net net

ANSWER:

select substr(“The Best
Class",1,3)||substr(“Internet”,
6,7)||’net from dual;

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

ANSWER:

select length(‘Oracle’||’Internet’||’Academy’) from dual;

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

ANSWER:

select instr(‘Oracle Internet Academy’,’I’) as matchposition from dual;

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

ANSWER:

select lpad(‘Oracle’,9,’*’)|| lpad(‘Internet’,12,’*’)||lpad(‘Academy’,11,’*’)||’****’ as LPADSTR 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.

ANSWER:

select last_name ,salary,round(salary/1.55,2) from employees where employee_id between 4 and 6;

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.

ANSWER:

select last_name ,trunc(salary *5.333/100,2) from employees where department_id =80;

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

ANSWER:

select mod(38873,2) from dual;

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

ANSWER:

select round(845.553,1) from dual;

select round(30695.348,2) from dual;

select round(30695.348,-2) from dual;

select round(2.3454,1) 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.

ANSWER:

select salary/3 from employees where mod(salary,3)=0;

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

ANSWER:

select mod(34,8) as example from dual;

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?

ANSWER:

select abs(565.784 -round(565.784)*1000 as DIFF from dual;

Ex. No. : 8 **Date:19.11.2024**

Register No.: 221701055 **Name:S Sneha**

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;
```

<u>Additional Search Conditions</u>

Using the AND Operator

<u>Example:</u>

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.

ANSWER:

select last_name,employees.department_id,department.dept_name from employees,department where employees.department_id=department.dept_id;

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

ANSWER:

select employees.job_id,department.location_id from employees,department where employees.department_id=department.dept_id and employees.department=80;

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

ANSWER:

select E.last_name,B.dept_name,L.location_id from employees E full outer join department B on(E.department_id=B.dept_id)full outer join emp_location L on(B.location_id=L.location_id);

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

ANSWER:

select E.last_name,D.Department_name from employees E,department D where E.last_name like "%a" and E.department_id=D.dept_id;

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

ANSWER:

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) JOIN locations l ON (d.location_id = l.location_id) WHERE LOWER(l.city) = 'toronto';

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

ANSWER:

```
SELECT
    e.employee_id 'Emp_Id', e.last_name 'Employee', m.employee_id 'Mgr_Id',  m.last_name 'Manager'
FROM
    employees 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.

ANSWER:

select last_name||' '||'King' from employees where commission_pct is null;

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

ANSWER:

select e1.last_name,e1.department_id,e2.last_name from employees e1 join employees e2 on e1.department_id=e2.department_id where e1.employee_id=9;

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

ANSWER:

desc salgrade;

SELECT e.ename, e.job, d.dname, e.sal, s.grade FROM emp e, dept d, salgrade s WHERE e.deptno = d.deptno;

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

ANSWER:

SELECT e.first_name, e.last_name, e.hire_date FROM employees e JOIN

employees davies

ON

(davies.last_name = 'Davies')

WHERE

davies.hire_date < e.hire_date;

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.

ANSWER:

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:19.11.2024**

Register No.: 221701055 **Name:S Sneha**

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);

<u>Nesting Group Functions</u>

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*];

<u>Find the Solution for the following:</u>

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

<u>ANSWER:</u>

True.

2. Group functions include nulls in calculations.

True/False

ANSWER:

False

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

True/False

ANSWER:

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

ANSWER:

SELECT ROUND(MAX(salary), 0) "Maximum", ROUND(MIN(salary), 0) "Minimum", ROUND(SUM(salary), 0) "Sum", ROUND(AVG(salary), 0) "Average" FROM employees;

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

ANSWER:

SELECT job_id, ROUND(MAX(salary), 0) "Maximum", ROUND(MIN(salary), 0) "Minimum", ROUND(SUM(salary), 0) "Sum", ROUND(AVG(salary), 0) "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.

ANSWER:

SELECT job_id, COUNT(*) FROM employees 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.*

ANSWER:

SELECT COUNT(DISTINCT manager_id) FROM employees;

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

ANSWER:

SELECT MAX(salary)-MIN(salary) 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.

ANSWER:

SELECT manager_id, MIN(salary) FROM employees WHERE manager_id IS NOT NULL GROUP BY manager_id HAVING MIN(salary) > 6000 ORDER BY MIN(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.

ANSWER:

SELECT COUNT(*) total, SUM(DECODE(TO_CHAR(hire_date, 'YYYY'), 1995,1,0)) "1995",
SUM(DECODE(TO_CHAR(hire_date, 'YYYY'), 1996,1,0)) "1996",
SUM(DECODE(TO_CHAR(hire_date, 'YYYY'), 1997,1,0)) "1997",
SUM(DECODE(TO_CHAR(hire_date, 'YYYY'), 1998,1,0)) "1998" FROM 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.

ANSWER:

SELECT job_id "Job", SUM(DECODE(department_id, 20, salary)) "Dept 20",
SUM(DECODE(department_id, 50, salary)) "Dept 50", SUM(DECODE(department_id, 80, salary)) "Dept 80", SUM(DECODE(department_id, 90, salary)) "Dept 90", from employees;

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.

ANSWER:

SELECT d.dname AS "department name" ,
d.loc AS " Location",
COUNT(*) "Number of People",
ROUND(AVG(sal),2) "Salary"
FROM dept d INNER JOIN emp e ON (d.deptno = e.deptno)
GROUP BY d.dname, d.loc;

Ex. No. : P-5 **Date:19.11.2024**

Register No.: 221701055 **Name: S Sneha**

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.

ANSWER:

SELECT name, event_date, ROUND(MONTHS_BETWEEN(SYSDATE, event_date)) as "number of months"

FROM d_events

WHERE 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."

ANSWER:

SELECT TO_DATE('20-Sep-2016', 'dd-Mon-yyyy') - TO_DATE('05-Jun-2016', 'dd-Mon-yyyy') as "Actual Days", ROUND(MONTHS_BETWEEN(TO_DATE('20-Sep-2016', 'dd-Mon-yyyy'), TO_DATE('05-Jun-2016', 'dd-Mon-yyyy'))*30.5, 0) as "Days"

FROM dual;

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

ANSWER:

SELECT TO_DATE('31-Dec-2016', 'dd-Mon-yyyy') - TO_DATE('01-Jan-2016', 'dd-Mon-yyyy') as "Actual 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.

ANSWER:

SELECT ROUND(SYSDATE, 'Month') as "nearest first day of month", ROUND(SYSDATE, 'Year') as "nearest first day of Year", TRUNC(SYSDATE, 'Month') as "current month's 1st day", TRUNC(SYSDATE, 'Year') as "current year's 1st day"
FROM dual;

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

ANSWER:

SELECT LAST_DAY(To_date('01-Jun-2005', 'dd-Mon-yyyy'))
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.

ANSWER:

SELECT first_name, last_name , ROUND(MONTHS_BETWEEN(SYSDATE, birthdate)/12) "No of Years"

FROM f_staffs

WHERE first_name || ' ' || last_name = 'Bob 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."

ANSWER:

SELECT TO_CHAR(ADD_MONTHS(SYSDATE, 6),'dd-Mon-yyyy (DY)') 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.”

ANSWER:

SELECT TO_CHAR(LAST_DAY(SYSDATE),'dd-Mon-yyyy (Day)') as "Deadline"
FROM dual;

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

ANSWER:

SELECT TO_DATE('11/05/2016','mm/dd/yyyy') "B'Day current year",
ADD_MONTHS(TO_DATE('11/05/2016','mm/dd/yyyy'),12)   "B'Day next year",
TRUNC(ADD_MONTHS(TO_DATE('11/05/2016','mm/dd/yyyy'),12), 'Year') "1st day next Year" , ROUND(MONTHS_BETWEEN(TRUNC(ADD_MONTHS(TO_DATE('11/05/2016','mm/dd/yyyy'),12), 'Year')   , TO_DATE('11/05/2016','mm/dd/yyyy'))) "Rounded Months to next 1st jan"
FROM dual;

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

ANSWER:

SELECT TO_DATE('11/05/2016','mm/dd/yyyy') "B'Day current year",
NEXT_DAY(TO_DATE('11/05/2016','mm/dd/yyyy'), 'Friday') "First Friday"
FROM dual;

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

ANSWER:

MONTHS_BETWEEN

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

ANSWER:

ADD_MONTHS

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

ANSWER:

Say, my credit card payment is due on 09^{th} every month. But if 09^{th} is on weekend or bank holiday, this due date has to be shifted to next Monday while triggering late payment workflow. This is a very common and nearest to me example of why business needs to manipulate date data.

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.

ANSWER:

SELECT last_name, TO_CHAR(birthdate, 'Month fmDD, RRRR') AS "Birthday"FROM f_staffs;

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

ANSWER:

SELECT TO_DATE('January 3, 04', 'Month dd, YY') AS "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.

ANSWER:

SELECT 'The promotion began on the '|| TO_CHAR(start_date, 'ddspth "of" Month YYYY') AS "Date"FROM f_promotional_menusWHERE code = 110;

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

ANSWER:

SELECT 'Today is the ' ||TO_CHAR(SYSDATE, 'Ddspth "of" Month, Yyyysp')FROM DUAL;

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

ANSWER:

SELECT id, first_name, TO_CHAR(salary, '$999.99')FROM f_staffs;

Ex. No. : 10 Date:19.11.2024

Register No.: 221701055 Name: S Sneha

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

<u>In the syntax:</u>

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

<u>Using a Subquery</u>

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.

<u>Guidelines for Using Subqueries</u>

• 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.

<u>Using the ALL Operator in Multiple-Row Subqueries</u>

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.

<u>Null Values in a Subquery</u>

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);
```

<u>Find the Solution for the following:</u>

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).

<u>ANSWER:</u>

```
select last_name, hire_date
from employees
where department_id =
(select department_id from employees where last_name like '&name')and last_name<> '&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.

<u>ANSWER:</u>

```
select employee_id, last_name, salary
from employees
```

where salary > (select avg(salary) from employees)

order by salary;

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*.

ANSWER:

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.

ANSWER:

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.

ANSWER:

select last_name, salary

from employees

where manager_id in (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.

ANSWER:

select department_id, last_name, job_id

from employees

where department_id in (select department_id from departments where 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*.

ANSWER:

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%');

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.

ANSWER:

SELECT first_name, last_name, TO_CHAR(salary, '$99,999.99') AS "Current Salary",TO_CHAR(salary + 2000, '$99,999.99') AS "New Salary"FROM EMPLOYEESWHERE last_name='Abel ';

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

ANSWER:

SELECT TO_CHAR(start_date, 'Day Month ddth') AS "Valentine's"FROM f_promotional_menusWHERE 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

ANSWER:

SELECT TO_CHAR(TO_DATE('25-DEC-04','dd-MON-yy'),'Month ddth,YYYY')Convert,TO_CHAR(TO_DATE('25-DEC-04','dd-MON-yy'),'MONTH DDth,YYYY')Convert,TO_CHAR(TO_DATE('25-DEC-04','dd-MON-yy'),'month ddth, YYYY') ASConvertFROM 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.

ANSWER:

SELECT TO_CHAR(low_range, '$99999.99')AS LOW, TO_CHAR(high_range,'$99999.99') AS HIGHFROM d_packages;

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

ANSWER:

SELECT TO_DATE('JUNE192004','fxMONTHDDYYYY') AS BIRTHDAYFROM DUAL; A common error when formatting dates is to omit the single quotation mark. The error message is missing a right parenthesis.

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

ANSWER:

Implicit datatype conversion occurs when the Oracle server internally converts datatypesfor example, Varchar2 to number. Explicit data conversion is when the specific function isused to convert one datatype to another, for example: TO_DATE(actual examples may vary).

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

ANSWER:

Datatype conversions offer the facility to display numbers as local currency, format datesin a variety of formats, display time to the second, and keep track of what century a daterefers to.

Ex. No. : 11 **Date: 19.11.2024**

Register No.: 221701055 **Name:S Sneha**

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.

ANSWER:

```
SELECT department_id
FROM departments
MINUS
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.

ANSWER:

```
SELECT country_id,country_name
FROM countries
MINUS
SELECT l.country_id,c.country_name
FROM locations l JOIN countries c
ON (l.country_id = c.country_id)
JOIN departments d
ON d.location_id=l.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.

ANSWER:

```
SELECT distinct job_id, department_id
FROM employees
WHERE department_id = 10
UNION ALL
SELECT DISTINCT job_id, department_id
FROM employees
WHERE department_id = 50
UNION ALL
SELECT DISTINCT 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).

ANSWER:

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

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.

ANSWER:

```
SELECT last_name,department_id,TO_CHAR(null)
FROM employees
UNION
SELECT TO_CHAR(null),department_id,department_name
FROM departments;
```

Ex. No. : 12 **Date:19.11.2024**

Register No.: 221701055 **Name:S Sneha**

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.

ANSWER:

SELECT name, start_date, end_date, NVL2(end_date, 'end in two weeks', TO_CHAR(SYSDATE, 'DD-Mon-YYYY')) as nvl2
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”.

ANSWER:

SELECT last_name, NVL(overtime_rate,0) as "Overtime Status"
FROM f_staffs;

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.

ANSWER:

SELECT last_name, TO_CHAR(NVL(overtime_rate,5), '$999.99') as "Overtime Status"

FROM f_staffs;

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.

ANSWER:

SELECT last_name, NVL(manager_id,9999) as manager_id

FROM f_staffs;

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;

ANSWER:

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;

ANSWER:

Since last_name is not nullable, it will always return last_name. If last_name would have been nullable and there had been a null last_name field, it would have fall back to manager_id converted to varchar2.

7

a. 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).

ANSWER:

SELECT NVL(first_name,'FNU') , 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.

ANSWER:

SELECT NVL(first_name,'FNU') , 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.

ANSWER:

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".

ANSWER:

```
SELECT title,
CASE
WHEN TO_NUMBER(REPLACE(NVL(duration,'0 min'), ' min', '')) = 2 THEN 'Shortest'
WHEN TO_NUMBER(REPLACE(NVL(duration,'0 min'), ' min', '')) = 10 THEN 'Longest'
ELSE NVL(duration,'0 min')
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.

ANSWER:

```
SELECT NVL(TO_CHAR(department_id), 'none')
department_id , last_name, NVL(salary,0) salary,
CASE department_id
WHEN 10 THEN 1.25*NVL(salary,0)
WHEN 90 THEN 1.5*NVL(salary,0)
```

WHEN 130 THEN 1.75*NVL(salary,0)

ELSE NVL(salary,0)

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.

ANSWER:

SELECT first_name, last_name, manager_id, commission_pct,

CASE

WHEN commission_pct IS NULL and manager_id IS NULL THEN 99999

WHEN manager_id IS NULL THEN commission_pct

ELSE manager_id

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.

ANSWER:

SELECT last_name, first_name, department_name

FROM employees CROSS JOIN departments;

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.

ANSWER:

SELECT department_id,department_name, location_id, city

FROM departments NATURAL JOIN locations;

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.

ANSWER:

SELECT department_id,department_name, location_id, city

FROM departments NATURAL JOIN locations

WHERE department_id in (20, 50);

Ex. No. : 13 **Date:19.11.2024**

Register No.: 221701055 **Name: S Sneha**

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;

<u>Example:1</u> (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;

<u>Retrieving data from a view</u>

<u>Example:</u>

SELECT * from salvu50;

<u>Modifying a view</u>

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

<u>Example</u>: (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;

<u>Example</u>: (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.

ANSWER:

CREATE OR REPLACE VIEW employees_vu AS SELECT employee_id, last_name employee, department_id FROM employees;

2. Display the contents of the EMPLOYEES_VU view.

ANSWER:

SELECT * FROM employees_vu;

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

ANSWER:

SET LONG 600 SELECT view_name, text FROM user_views;

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

ANSWER:

SELECT FROM employee, department_id 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.

ANSWER:

CREATE VIEW dept50 AS SELECT employee_id empno, last_name employee, department_id deptno FROM WHERE employees department_id = 50 WITH CHECK OPTION CONSTRAINT emp_dept_50;

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

ANSWER:

DESCRIBE dept50 SELECT * FROM dept50;

7. Attempt to reassign Matos to department 80.

ANSWER:

UPDATE SET WHERE dept50 deptno = 80 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.

ANSWER:

CREATE OR REPLACE VIEW salary_vu AS SELECT e.last_name "Employee", d.department_name "Department", e.salary "Salary", j.grade_level "Grades" FROM employees e, departments d, job_grades j WHERE e.department_id = d.department_id AND e.salary BETWEEN j.lowest_sal and j.highest_sal;

Ex. No. : P-6 **Date:19.11.2024**

Register No.: 221701055 **Name: S Sneha**

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.

ANSWER:

SELECT department_id,department_name, location_id, city
FROM departments JOIN locations USING (location_id)
WHERE 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.

ANSWER:

SELECT song_id, cd_number, title, comments
FROM d_cds JOIN d_track_listings USING (cd_number) JOIN d_play_list_items USING (song_id);

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

ANSWER:

SELECT city, department_name, location_id, department_id

FROM departments JOIN locations USING (location_id)

WHERE department_id in (10, 20 , 30) AND city = 'Seattle';

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

ANSWER:

SELECT country_name, region_id, region_name

FROM countries JOIN regions USING(region_id)

WHERE 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.

ANSWER:

SELECT first_name, last_name, hire_date, job_id, job_title, max_salary

FROM employees JOIN jobs USING (job_id)

WHERE 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.

ANSWER:

SELECT emp.first_name "First Name", emp.last_name "Last Name" , dpt.department_name "Department Name"

FROM employees emp LEFT OUTER JOIN departments dpt ON emp.department_id = dpt.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.

ANSWER:

SELECT emp.first_name "First Name", emp.last_name "Last Name" , dpt.department_name "Department Name"

FROM employees emp RIGHT OUTER JOIN departments dpt ON emp.department_id = dpt.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.

ANSWER:

```sql
SELECT  emp.first_name "First Name", emp.last_name "Last Name" ,
dpt.department_name  "Department Name"
FROM  employees emp FULL OUTER JOIN departments dpt ON emp.department_id =
dpt.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.

ANSWER:

```sql
SELECT ct.first_name, ct.last_name, ev.event_date, ev.description
FROM  d_clients ct LEFT OUTER JOIN d_events ev ON ct.client_number = ev.client_number;
```

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.

ANSWER:

```sql
SELECT f_shifts.description "shift description", f_shift_assignments.shift_assign_date AS "shift
assignment date"

FROM  f_shifts LEFT OUTER JOIN f_shift_assignments ON f_shifts.code = f_shift_assignments.code;
```

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.

ANSWER:

SELECT emp.last_name "Employee", emp.employee_id "Emp#", mgr.last_name "Manager", mgr.employee_Id "Mgr#"

FROM employees emp LEFT OUTER JOIN employees mgr ON emp.manager_id = mgr.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.

ANSWER:

SELECT emp.last_name "Employee", emp.employee_id "Emp#", mgr.last_name "Manager", mgr.employee_Id "Mgr#"

FROM employees emp LEFT OUTER JOIN employees mgr ON emp.manager_id = mgr.employee_Id

ORDER BY "Employee";

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.

ANSWER:

```
SELECT emp.last_name "Employee",  emp.hire_date "Emp Hired",   mgr.last_name "Manager",
mgr.hire_date "Mgr Hired"
FROM  employees emp LEFT OUTER JOIN  employees  mgr  ON emp.manager_id  =
mgr.employee_Id
WHERE emp.hire_date < NVL( mgr.hire_date, TO_DATE('31.12.9999 23:59:59', 'dd.mm.yyyy
hh24:mi:ss'))
ORDER BY  "Employee";
```

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

ANSWER:

```
SELECT last_name, salary, department_id
FROM employees
START WITH first_name = 'Lex'  AND last_name = 'De Haan'
CONNECT BY PRIOR employee_id = manager_id;
```

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;
```

ANSWER:

Query in problem says go from tree leave to tree base, but King is itself the base, so the result of query in problem is only one row, the king itself. So there, is nothing wrong as per SQL in the query in problem, but logically thinking considering the data we have, king is the super boss.

To support my answer, let's study similar case:

SELECT LEVEL, last_name, department_id, salary

FROM employees

START WITH first_name = 'Lex' AND last_name = 'De Haan'

CONNECT BY PRIOR manager_id = employee_id;

Gives same result as

SELECT LEVEL, last_name, department_id, salary

FROM employees

START WITH first_name = 'Lex' AND last_name = 'De Haan'

CONNECT BY employee_id = PRIOR manager_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.

ANSWER:

SELECT LPAD(last_name, LENGTH(last_name) + (LEVEL-1)*2, '-') "organization chart "

FROM employees

START WITH last_name = (SELECT last_name from employees WHERE 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.

ANSWER:

SELECT LPAD(last_name, LENGTH(last_name) + (LEVEL-1)*2, '-') "organization chart "

FROM employees

START WITH last_name = (SELECT last_name from employees WHERE manager_id IS NULL)

CONNECT BY PRIOR employee_id = manager_id AND last_name != 'De Haan'

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.

ANSWER:

SELECT d_play_list_items.event_id "event id in playlist", d_play_list_items.song_id "song id in playlist", d_play_list_items.comments "comments in playlist", d_track_listings.song_id "song id in tracklist", d_track_listings.cd_number "cd number in tracklist", d_track_listings.track "track in tracklist"
FROM d_play_list_items, d_track_listings;

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

ANSWER:

SELECT d_play_list_items.event_id "event id in playlist", d_play_list_items.song_id "song id in playlist", d_play_list_items.comments "comments in playlist", d_track_listings.song_id "song id in tracklist", d_track_listings.cd_number "cd number in tracklist", d_track_listings.track "track in tracklist"
FROM d_play_list_items, d_track_listings
WHERE d_play_list_items.song_id = d_track_listings.song_id;

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

ANSWER:

```
SELECT  d_songs.title,  d_songs.type_code type, d_types.description
FROM d_songs, d_types
WHERE d_songs.type_code = d_types.code;
```

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

ANSWER:

```
SELECT  d_songs.title,  d_songs.type_code type, d_types.description
FROM d_songs, d_types
WHERE d_songs.type_code = d_types.code AND d_songs.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.

ANSWER:

```
SELECT d_clients.email AS "d_clients - email" , d_clients.phone AS "d_clients - phone" ,
d_clients.last_name AS "d_clients - last_name" , d_clients.first_name AS "d_clients - first_name"  ,
d_clients.client_number AS "d_clients - client_number"  , d_events.id AS "d_events - id",
d_events.name AS "d_events - name", d_events.event_date AS "d_events - event_date",
d_events.description AS "d_events - description", d_events.cost AS "d_events - cost",
d_events.venue_id AS "d_events - venue_id", d_events.package_code AS "d_events - package_code",
d_events.theme_code AS "d_events - theme_code", d_events.client_number AS "d_events -
client_number", d_job_assignments.partner_id AS "d_job_assignments - partner_id",
d_job_assignments.event_id AS "d_job_assignments - event_id", d_job_assignments.job_date AS
"d_job_assignments - job_date", d_job_assignments.status AS "d_job_assignments - status"
FROM d_clients, d_events, d_job_assignments
WHERE d_clients.client_number = d_events.client_number AND d_events.id
=  d_job_assignments.event_id;
```

Group Functions

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

ANSWER:

AVG, COUNT :

· Calculates average value excluding nulls- AVG group function

· Returns the number of rows with non-null values for the expression- COUNT group function

SELECT AVG(salary) || ' is Average Salary of ' || COUNT(salary) || ' employees. This table has primary key employee_id which won''t be null. So, ' || (COUNT(employee_id) - COUNT(salary)) || ' rows are skipped in Average Salary calculation.' "Example"

FROM EMPLOYEES;

(I may use COUNT(*) instead of COUNT(employee_id), if I want to)

Output:

8775 is Average Salary of 20 employees. This table has primary key employee_id which won't be null. So, 0 rows are skipped in Average Salary calculation.

Please note that in this concatenation and minus has same precedence. So () were required.

MAX, MIN:

· Returns minimum value ignoring nulls- MIN group function

· Returns the maximum value ignoring nulls- MAX group function

SELECT 'The maximum of ' || COUNT(salary) || ' salaries in employees table is ' || MAX(salary) ||'. The minimum of '|| COUNT(salary) || ' salaries in employees table is '|| MIN(salary) ||'.' "Example2"

FROM EMPLOYEES;

Output:

The maximum of 20 salaries in employees table is 24000. The minimum of 20 salaries in employees table is 2500.

STDDEV, VARIANCE:

· Used with columns that store numeric data to calculate the spread of data around the mean- VARIANCE group function

· For two sets of data with approximately the same mean, the greater the spread, the greater the standard deviation.- STDDEV group function

SELECT 'The standard deviation of ' || COUNT(salary) || ' salaries in employees table is ' || ROUND(STDDEV(salary), 4) ||'. The variance of '|| COUNT(salary) || ' salaries in employees table is '|| ROUND(VARIANCE(salary), 4) ||'.' "Example3"

FROM EMPLOYEES;

Output:

The standard deviation of 20 salaries in employees table is 5659.6331. The variance of 20 salaries in employees table is 32031447.3684.

If not rounded the values would have been inconvenient to read.

SUM:

· Calculates the sum ignoring null values- SUM group function

SELECT 'The sum of ' || COUNT(salary) || ' salaries in employees table is ' || SUM(salary) ||'.' "Example4"

FROM EMPLOYEES;

Output:

The sum of 20 salaries in employees table is 175500.

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

ANSWER:

SELECT ROUND(AVG(cost),2) as "Average Cost"

FROM d_events;

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

ANSWER:

SELECT TO_CHAR(ROUND(AVG(salary),2), '$999999.99') as "Average Salary"

```sql
FROM f_staffs
WHERE manager_id = 19;
```

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

ANSWER:

```sql
SELECT TO_CHAR(ROUND(SUM(salary),2), '$999999.99') as "Total Salary"
FROM f_staffs
WHERE id in (12, 19);
```

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

ANSWER:

```sql
SELECT MIN(salary) "lowest salary", MAX(hire_date) "most recent hire date", MIN(last_name) "top last name", MAX(last_name) "bottom last name"
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;

ANSWER:

one

7. 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?

ANSWER:

SELECT AVG(NVL(salary, hourly_rate* hrs_worked_in_yr))

This way the null fields beings ignored will also be counted in.

8. 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?

ANSWER:

March 30, 1969

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

ANSWER:

SELECT 'Average of ' || COUNT(order_number) || ' orders is : ' || AVG(NVL(order_total, 0)) as "Average"
FROM f_orders
WHERE order_date BETWEEN TO_DATE('January 1, 2002', 'fmMonth DD, YYYY') AND
TO_DATE('December 21, 2002', 'fmMonth DD, YYYY');

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

ANSWER:

SELECT MAX(hire_date) as "the last"
FROM employees;

11. 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;

ANSWER:

SUM must be be ‘equal or
greater than’ average.

Ex. No. : P-7 **Date:19.11.2024**

Register No.: **221701055** **Name:S Sneha**

COUNT, DISTINCT, NVL

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

ANSWER:

SELECT COUNT(DISTINCT title)

FROM d_songs; gives 6

SELECT COUNT(*)

FROM d_songs; gives 6

SELECT COUNT(id)

FROM d_songs; gives 6

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

ANSWER:

Possible venues count: (4)

SELECT COUNT(DISTINCT loc_type)

FROM d_venues;

Venue types used by events count: (2)

SELECT COUNT(DISTINCT venue_id)

FROM d_events;

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?

ANSWER:

SELECT COUNT(song_id) AS "song with possible duplication", COUNT(distinct cd_number) "cd no. distinct"

FROM d_track_listings;

5 and 4

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

ANSWER:

Count will skip nulls anyhow.

SELECT COUNT(email) "count with email"

FROM d_clients;

3

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?

ANSWER:

The language of question seems to be, the one with mentioned limit as auth_expense_amt are bound by this limit, but the one who don't have limit are free to spend anything. Say, a row has auth_expense_amt has 20, that partner can spend $20, If it is 0 he can't spend anything, but if this field is null, it means he is out of bound .

SELECT (COUNT(*) - COUNT(auth_expense_amt)) "Free from limit count"

FROM d_partners;

COUNT(*) – all row count is 3, and COUNT(auth_expense_amt) is 1. That's why above query says 2 are out of any bound due to auth_expense_amt specified.

COUNT(auth_expense_amt) is 1 because count skips nulls.

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;

ANSWER:

4 and 2

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.

ANSWER:

SELECT TO_CHAR(ROUND(AVG(NVL(auth_expense_amt,100000)),2), '$999999.99')

FROM d_partners;

$166666.67

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.

ANSWER:

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

TRUE, it won't permanently change the column data in table, just in the scope of current query, AVG will get 0.10 in the row where null selling_bonus is encountered.

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.

ANSWER:

_____ **c.** Unique combinations of color and size will appear only once in the result set.

TRUE

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 ;

ANSWER:

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

TRUE

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;

ANSWER:

Above query will say ORA-00934: group function is not allowed here.

If 'desired' result is getting manger ids, under whom, average salary is less than 16000:

SELECT manager_id,AVG(salary)

FROM employees

GROUP BY manager_id

HAVING AVG(salary) <16000;

b. SELECT cd_number, COUNT(title)

FROM d_cds

WHERE cd_number < 93;

ANSWER:

Above query will say ORA-00937: not a single-group group function

If 'desired' result is getting row count, excluding anything greater than or equal to 93:

SELECT COUNT(*)

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;

ANSWER:

Above query will say ORA-00979: **not a GROUP BY expression**

id is primary key of the table, so most likely my intention may not be to group by id here.

Desired may have been grouping by type_code and getting maximum duration

but limiting the input to group by with duration IN('3 min', '6 min', '10 min') and ID < 50

SELECT type_code, MAX(TO_NUMBER(REPLACE(duration,' min',''))) || ' min' as "max duration"

FROM d_songs

WHERE duration IN('3 min', '6 min', '10 min') AND id < 50

```
GROUP BY type_code;
```

d. SELECT loc_type, rental_fee AS Fee

```
FROM d_venues
WHERE   id   <100
GROUP  BY  "Fee"
ORDER BY 2;
```

ANSWER:

1) Above query will say ORA-00904: "Fee": invalid identifier

2) Even if I replace "Fee" alias in group by [don't use alias in group by clause] it will give me error : ORA-00979: not a GROUP BY expression

3) Seems to be question wanted average rental_fee of each loc_type with id less than 100 before grouping.

I assumed rental_fee is a number field:

```
SELECT loc_type, AVG(rental_fee) AS Fee
FROM d_venues
WHERE id <100
GROUP BY loc_type
ORDER BY 2;
```

But above query gives error ORA-01722: invalid number, because rental_fee is a VARCHAR2(50)

4) Now, I need assumptions to view the calculations with same eyes: /hour, /flat fee, /per person.

Let's assume that we are group of 10 people *and event I am organizing occurs for* 5 hours.

```
SELECT loc_type, AVG(
CASE
 WHEN INSTR(rental_fee, '/hour') != 0 THEN  TO_NUMBER(REPLACE(rental_fee,'/hour',''))*5
  WHEN INSTR(rental_fee, '/flat fee') != 0 THEN  TO_NUMBER(REPLACE(rental_fee,'/flat fee',''))
   WHEN INSTR(rental_fee, '/per person') != 0 THEN TO_NUMBER(REPLACE(rental_fee,'/per person',''))*10
 ELSE 0
 END
```

) AS Fee

FROM d_venues

WHERE id <100

GROUP BY loc_type

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);

ANSWER:

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

GROUP BY track;

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.

ANSWER:

a) TRUE

b) FALSE. query will say ORA-00904: "Some Alias": invalid identifier

c) FALSE- GROUP BY clause of SQL statement contains the column name- value of which is used to divide input into subgroups, each subgroup represented as a row in output. The group function- goes in SELECT part. HAVING clause may also contain group function. If I put group function in group by clause I get error 'ORA-00934: group function is not allowed here' e.g. SELECT AVG(salary) FROM employees GROUP BY AVG(salary) HAVING AVG(salary) <16000; will give error.

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

ANSWER:

SELECT ROUND(MAX(AVG(salary)),2) as "Maximum Average of Departments",
ROUND(MIN(AVG(salary)),2) "Minimum Average of Departments"
FROM employees
GROUP BY department_id;

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

ANSWER:

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

Using Set Operators

1. Name the different Set operators?

ANSWER:

UNION

UNION ALL

MINUS

INTERSECT

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.

ANSWER:

```
SELECT employee_id, job_id, hire_date, TO_DATE(NULL)  start_date, department_id
FROM employees
UNION
SELECT employee_id, job_id, TO_DATE(NULL), start_date,  department_id
FROM job_history
ORDER BY employee_id, hire_date, start_date NULLS FIRST;
```

But after reading next problem I changed it to:

SELECT employee_id, job_id, hire_date, department_id

FROM employees

UNION

SELECT employee_id, job_id, start_date, department_id

FROM job_history

ORDER BY employee_id, hire_date;

29 rows returned

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.

ANSWER:

```
SELECT employee_id, job_id, hire_date, department_id
FROM employees
UNION ALL
SELECT employee_id, job_id,  start_date,  department_id
FROM job_history
ORDER BY employee_id, hire_date;
```

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

ANSWER:

```
SELECT DISTINCT employee_id FROM employees
MINUS
SELECT DISTINCT employee_id FROM job_history;
```

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

ANSWER:

```
SELECT DISTINCT employee_id FROM employees
```

INTERSECT SELECT DISTINCT employee_id

FROM job_history;

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.

ANSWER:

```
SELECT employee_id, job_id, NVL(salary, 0) FROM employees
UNION
SELECT employee_id, job_id, 0 FROM job_history
ORDER BY employee_id;
```

Ex. No. : P-8 **Date:19.11.2024**

Register No.: 221701055 **Name:S Sneha**

Fundamentals of Subqueries

1. What is the purpose of using a subquery?

ANSWER:

To find the intermediate information we need to extract information we want. E.g. extracting right part in WHERE/HAVING/FROM clause.

2. What is a subquery?

ANSWER:

An inner query that is nested within an outer query

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

ANSWER:

SELECT song_id

FROM d_play_list_items

WHERE event_id IN(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?

ANSWER:

SELECT id, name

FROM d_events

WHERE cost > (SELECT cost FROM d_events WHERE id = 100);

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

ANSWER:

SELECT track

FROM d_track_listings

WHERE cd_number = (SELECT cd_number FROM d_cds WHERE title = 'Party Music for All Occasions');

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

ANSWER:

SELECT id, name

FROM d_events

WHERE theme_code = (SELECT code FROM d_themes WHERE description = '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?

ANSWER:

SELECT first_name,last_name

FROM f_staffs

WHERE salary > (SELECT salary FROM f_staffs WHERE 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?

ANSWER:

SELECT first_name,last_name

FROM f_staffs

WHERE staff_type != (SELECT staff_type FROM f_staffs WHERE first_name = 'Bob' AND last_name ='Miller');

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

ANSWER:

```
SELECT first_name,last_name
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?

ANSWER:

```
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.

ANSWER: TRUE- this is a guideline, but it doesn't break it.

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

ANSWER:

TRUE

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

ANSWER:

FALSE

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.

ANSWER:

SELECT first_name, last_name

FROM EMPLOYEES

WHERE salary > (SELECT salary FROM employees WHERE last_name = 'Lorentz') AND department_id = (SELECT department_id FROM employees WHERE last_name = 'Abel');

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

ANSWER:

SELECT first_name, last_name

FROM EMPLOYEES

WHERE job_id = (SELECT job_id FROM employees WHERE last_name = 'Rajs') AND hire_date > (SELECT hire_date FROM employees WHERE last_name = 'Davies') ;

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

ANSWER:

SELECT id, name

FROM d_events

WHERE theme_code = (SELECT theme_code FROM d_events WHERE 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?

ANSWER:

SELECT staff_type, MAX(salary)

FROM f_staffs

GROUP BY staff_type

HAVING MAX(salary) < (SELECT MAX(SALARY) FROM f_staffs 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.

ANSWER:

SELECT department_id, TO_CHAR(ROUND(AVG(salary),2),'$999999.99') "Average Salary"

FROM employees

GROUP BY department_id

HAVING AVG(salary) > (SELECT salary from employees WHERE last_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.

ANSWER:

SELECT department_id, TO_CHAR(ROUND(MIN(salary),2),'$999999.99') "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?

ANSWER:

a) Say it is a single row inner query, And I try to use =, != etc :-There are no rows returned here.
Not even the [NULL = NULL |##|
NULL != NULL] could be evaluated here. For this case I will have to use [column IS NULL
|##| COLUMN IS NOT NULL]

b) **Say it is a single row inner query, And I try to use >, >=,<, <= etc:- There are no rows returned here.**

I may have to use NVL to execute comparison with something valid. Based on what is the need of my query, I may use NVL in case a) above also.

c) **Say it is multiple row subquery and I am using IN, ANY, ALL: The items in the collection returned by subquery are ignored which are NULL- in the individual internal comparisons.**

d) **When a multiple-row subquery uses the NOT IN operator (equivalent to <>ALL), if one of the values returned by the inner query is a null value, the entire query returns**: No rows returned

Example to support d:

SELECT commission_pct FROM employees WHERE department_id IS NULL OR department_id = 50

SELECT * FROM employees WHERE commission_pct IN (SELECT commission_pct FROM employees WHERE department_id IS NULL OR department_id = 50)

gives

SELECT * FROM employees WHERE commission_pct NOT IN (SELECT commission_pct FROM employees WHERE department_id IS NULL OR department_id = 50)

SELECT * FROM employees WHERE commission_pct = 0.2

Now execute SELECT * FROM employees WHERE commission_pct NOT IN (SELECT NVL(commission_pct,0) FROM employees WHERE department_id IS NULL OR department_id = 50)

e) **Say it is multiple row subquery and I am using NOT EXISTS / EXISTS: EXISTS and NOT**

EXISTS target presence of one or more rows, rather than its value.

Below mentioned two queries gives same result set:

SELECT * FROM wf_countries otter WHERE EXISTS (SELECT NULL FROM dual);

SELECT * FROM wf_countries otter;

And below mentioned gives no data!

SELECT * FROM wf_countries otter WHERE NOT EXISTS (SELECT NULL FROM dual);

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.

ANSWER:

SELECT id, title, duration, artist

FROM d_songs

WHERE type_code IN (SELECT code FROM d_types WHERE description IN ('Jazz', 'Pop'));

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

ANSWER:

SELECT last_name

FROM employees

WHERE salary in (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.

ANSWER:

SELECT last_name

FROM f_staffs

WHERE NVL(salary,0) = (SELECT MIN(NVL(salary,0)) FROM f_staffs);

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 ...

ANSWER:

YEAR is VARCHAR2(4), but results with TO_NUMBER and without it are the same.

SELECT *

FROM d_cds

WHERE TO_NUMBER(year) < (SELECT TO_NUMBER(year) FROM d_cds where title = 'Carpe Diem');

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

WHERE salary___________(SELECT salary ...

ANSWER:

SELECT last_name, salary

FROM employees

WHERE salary < ANY (SELECT salary FROM employees where department_id = (SELECT department_id FROM departments WHERE department_name = 'IT'));

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

WHERE year __________(SELECT year ...

ANSWER:

SELECT title

FROM d_cds

WHERE TO_NUMBER(year) IN (SELECT TO_NUMBER(year) FROM d_cds where title IN ('Carpe Diem', 'Party Music for All Occasions'));

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

WHERE duration ________(SELECT duration ...

ANSWER:

duration is VARCHAR2(20)

SELECT title, duration

FROM d_songs

WHERE TO_NUMBER(REPLACE(duration,' min','')) > ALL (

SELECT TO_NUMBER(REPLACE(duration,' min','')) FROM d_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.

ANSWER:

____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.TRUE. Everything above 8 is ok

____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. **FALSE.** Only 102, 105, 437, and 225 are OK

____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. **FALSE.** Anything below and equal to 65 is OK

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

____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. **FALSE**

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;

ANSWER:

a)

ORA-00934: group function is not allowed here

Remove WHERE MIN(salary)

b)

ORA-00936: missing expression

This is from HAVING

Move the subquery in having and change the sign.

c)

ORA-00923: FROM keyword not found where expected

Put FROM employees in subquery.

d)

But I want minimum salary of department 50, change the where clause in subquery.

e)

But I want minimum salary instead of department no.

Change SELECT of outer query.

Even after correction, there are no results, because 2500 is the lowest salary and that employee is of department_id = 50 J

SELECT department_id, 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.

ANSWER:

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

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

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

FALSE – subquery is executed first

______ d. This query will not execute.

TRUE, it won't execute. This will require multiplerow subquery as in example 3 above.

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.

ANSWER:

SELECT last_name, first_name, department_id, manager_id

FROM employees

WHERE (NVL(department_id,-1), NVL(manager_id,-1)) = (SELECT NVL(department_id,-1), NVL(manager_id,-1) 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.

ANSWER:

SELECT last_name, first_name, department_id, manager_id

FROM employees

WHERE NVL(department_id,-1) = (SELECT NVL(department_id,-1) FROM employees WHERE employee_id = 141)

AND NVL(manager_id,-1) = (SELECT NVL(manager_id,-1) FROM employees WHERE employee_id = 141)

AND employee_id != 141;

Correlated Subqueries

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

ANSWER:

As explained in Multiple-Row Subqueries Practice > Problem 3:

Correlated subquery is executed multiple times once for each intermediate result row from outer query And for each row of the intermediate results inner correlated subquery is executed, if the inner correlated subquery is satisfied, that row becomes part of final results from the whole query. A correlated subquery will get a candidate row from an outer query, execute the inner query using candidate row value, and use values from the inner query to qualify or disqualify the candidate row.

Non-correlated subquery: result/s of inner query are calculated first and reused throughout the execution of 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.

ANSWER:

WITH highs AS (SELECT NVL(department_id,-1), MAX(salary) FROM employees GROUP BY NVL(department_id,-1))

SELECT last_name, department_id, salary

FROM employees

WHERE (NVL(department_id,-1), salary) IN (SELECT * FROM highs);

or

SELECT oe.last_name, oe.department_id, oe.salary

FROM employees oe

WHERE oe.salary = (SELECT MAX(ie.salary) FROM employees ie WHERE NVL(ie.department_id,-1)

= NVL(oe.department_id,-1));

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.

ANSWER:

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

OR

SELECT outer.last_name, outer.department_id, outer.salary
FROM employees outer
WHERE outer.employee_id IN (SELECT DISTINCT inner.manager_id
FROM employees inner
WHERE inner.manager_id IS NOT NULL)

ORDER BY outer.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.

ANSWER:

If I ignore the hint and job_total, which is a very ambiguous requirement here:

WITH max_calc_sal as (SELECT MAX(max_salary)/2 FROM jobs)

SELECT job_title

FROM jobs

WHERE jobs.max_salary > (SELECT * FROM max_calc_sal);

But if I consider hints, I think might be job_total is not a total, but should have been called job_actual_max. Now there is a possibility that a job is there in job table but no employee is there for that job. For such case actual max could be 0.

WITH max_calc_sal AS (SELECT jobs.job_id , jobs.job_title, MAX(NVL(employees.salary,0)) AS job_actual_max FROM employees RIGHT OUTER JOIN jobs ON employees.job_id = jobs.job_id GROUP BY jobs.job_id,jobs.job_title)

SELECT job_title, job_actual_max AS job_total

FROM max_calc_sal

WHERE job_actual_max > (SELECT MAX(job_actual_max)/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.

ANSWER:

· I am on a flight booking site. It shows available flights, say I try to book a ticket and still my transaction is not commenced anywhere, I will be in big trouble.

· I am trying to create a login on a site, it takes my details and say never store my information in registration request table, I will keep on waiting for approval which will never happen.

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.

ANSWER:

Create copy of main table to play around:

CREATE TABLE copy_d_cds

AS (SELECT * FROM d_cds);

see if copy worked well:

DESCRIBE copy_d_cds;

DESCRIBE d_cds;

(clone table lost its primary key constraint and cd_number became nullable.)

see how the copied content looks like.

SELECT * FROM copy_d_cds ;

I see that cd_number 98 will be repeated with my insert statements, but it won't give error, since primary key constrain is lost in copy.

The Explicit insert statements without even missing nullable columns:

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.

<u>ANSWER:</u>

Create copy of main table to play around:

CREATE TABLE copy_d_songs

AS (SELECT * FROM d_songs);

see if copy worked well:

DESCRIBE copy_d_songs;

DESCRIBE d_songs;

(Also, I see that duration & artist is nullable in source table. I could skip these column names in insert into / if I decide to include these columns, I will specify it as NULL in VALUES. I will prefer the later option to be symmetric and avoid missing something by mistake. One more thing the clone table lost its primary key constraint and id became nullable.)

The Explicit insert statements without even missing nullable columns:

INSERT INTO copy_d_songs (id,title,duration,artist,type_code)

VALUES(52,'Surfing Summer',NULL,NULL,12);

INSERT INTO copy_d_songs (id,title,duration,artist,type_code)

VALUES(53,'Victory Victory','5 min',NULL,12);

But, problem specifically says to use implicit insert statement:

INSERT INTO copy_d_songs

VALUES(52,'Surfing Summer',NULL,NULL,12);

INSERT INTO copy_d_songs

VALUES(53,'Victory Victory','5 min',NULL,12);

SELECT * FROM copy_d_songs ;

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

ANSWER:

a) Create copy of main table to play around:

CREATE TABLE copy_d_clients

AS (SELECT * FROM d_clients);

b) see if copy worked well:

DESCRIBE copy_d_clients ;

DESCRIBE d_clients;

(clone table lost its primary key constraint and client_number became nullable.)

SELECT * FROM d_clients ;

SELECT * FROM copy_d_clients ;

c) The Explicit insert statements without even missing nullable columns:

INSERT INTO copy_d_clients(client_number,first_name,last_name,phone,email)

VALUES(6655,'Ayako','Dahish',3608859030,'dahisha@harbor.net');

INSERT INTO copy_d_clients(client_number,first_name,last_name,phone,email)

VALUES(6689,'Nick','Neuville',3608859030,'nnicky@charter.net');

d) verify data:

SELECT * FROM copy_d_clients ;

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

ANSWER:

a) Create copy of main table to play around:

CREATE TABLE copy_d_events

AS (SELECT * FROM d_events);

b) see if copy worked well:

DESCRIBE copy_d_events ;

DESCRIBE d_events;

(

All the constraints are lost in this copy.e.g.:

· clone table lost its primary key constraint and id became nullable.

· 245 and 315 venue_id are not there in d_venues, but it insert will still work. [This loss of foreign key constraint is not mentioned by DESCRIBE]

one more thing:

· cost is not nullable and it needs to be a number. I could either make it nullable, or give some value like 0 to cost. I go with second choice - give value as zero.

)

SELECT * FROM d_events ;

SELECT * FROM copy_d_events ;

c) The Explicit insert statements:

INSERT INTO copy_d_events(id,name,event_date,description,cost,venue_id,package_code,theme_code,client_number) VALUES(110,'Ayako Anniversary',TO_DATE('07-Jul-2004','dd-Mon-yyyy'),'Party for 50, sixties dress, decorations',0,245,79,240,6655);

INSERT INTO copy_d_events(id,name,event_date,description,cost,venue_id,package_code,theme_code,client_number) VALUES(115,'Neuville Sports Banquet',TO_DATE('09-Sep-2004','dd-Mon-yyyy'),'Barbecue at residence, college alumni, 100 people',0,315,87,340,6689);

d)verify data:

SELECT * FROM copy_d_events ;

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.

ANSWER:

Those employees could be Marketing Representative, or Sales Representative. There JOB_ID ends with '_REP'

DESCRIBE rep_email ;

DESCRIBE employees;

Please note, employee_id has precision 6 and scale 0. But id in problem statement has 2, 0

Similarly other fields also have differences.

Expected to see errors like ORA-01438: value larger than specified precision allowed for this column

Luckily, rest of the mismatches still work because data is ok, but for id, I will have alter it:

ALTER TABLE rep_email DROP column id;

ALTER TABLE rep_email ADD id NUMBER(6,0) CONSTRAINT rel_id_pk PRIMARY KEY;

DESCRIBE rep_email ;

SELECT employee_id, first_name, last_name, email

FROM employees

WHERE job_id LIKE '%_REP' ESCAPE '\';

INSERT INTO rep_email(id, first_name, last_name, email_address)

SELECT employee_id, first_name, last_name, email

FROM employees

```
WHERE job_id LIKE '%\_REP' ESCAPE '\';


SELECT * FROM rep_email;
```

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.

<u>ANSWER:</u>

CREATE TABLE copy_f_food_items
AS (SELECT * FROM f_food_items);
DESCRIBE f_food_items;
DESCRIBE copy_f_food_items;
SELECT * FROM f_food_items;
SELECT * FROM copy_f_food_items;

UPDATE copy_f_food_items SET price = 3.75
WHERE LOWER(description) = 'strawberry shake';
UPDATE copy_f_food_items SET price = 1.20
WHERE LOWER(description) = 'fries';

SELECT * FROM copy_f_food_items;

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?)

ANSWER:

```
CREATE TABLE copy_f_staffs
AS ( SELECT * FROM f_staffs);
DESCRIBE f_staffs;
DESCRIBE copy_f_staffs;
SELECT * FROM f_staffs;
SELECT * FROM copy_f_staffs;

UPDATE copy_f_staffs  SET overtime_rate = NVL(overtime_rate, 0) + 0.75
WHERE LOWER(first_name || ' ' || last_name) = 'bob miller';
UPDATE copy_f_staffs  SET overtime_rate = NVL(overtime_rate, 0) + 0.85
WHERE LOWER(first_name || ' ' || last_name) = 'sue doe';

SELECT * FROM copy_f_staffs;
```

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

ANSWER:

f this action would have been done for f_orders, highlighted values would have failed the update due to foreign key constraint.

```
CREATE TABLE copy_f_orders
AS ( SELECT * FROM f_orders);
DESCRIBE f_orders;
DESCRIBE copy_f_orders;
SELECT * FROM f_orders;
SELECT * FROM copy_f_orders;

INSERT INTO copy_f_orders(order_number,order_date,order_total,cust_id,staff_id)
VALUES(5680,TO_DATE('June 12, 2004','fmMonth 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','fmMonth dd, yyyy'),229.31,230,12);

SELECT * FROM copy_f_orders;
```

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_NAME	LAST_NAME	ADDRESS	CITY	STATE	ZIP	PHONE_NUMBER
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

ANSWER:

Yes I will be able to add row, even if it has existing id, since in cloning table as mentioned below, primary key constraint is lost.

CREATE TABLE copy_f_customers
AS (SELECT * FROM f_customers);

DESCRIBE f_customers;

DESCRIBE copy_f_customers;

In copy table, zip is not nullable same as in master table, so the last row will give error while insert.

SELECT * FROM f_customers;

SELECT * FROM copy_f_customers;

INSERT INTO copy_f_customers(id,first_name,last_name,address,city,state,zip,phone_number)
VALUES(145,'Katie','Hernandez','92 Chico Way','Los Angeles','CA',98008,'8586667641');

INSERT INTO copy_f_customers(id,first_name,last_name,address,city,state,zip,phone_number)
VALUES(225,'Daniel','Spode','1923 Silverado','Denver','CO',80219,'7193343523');

INSERT INTO copy_f_customers(id,first_name,last_name,address,city,state,zip,phone_number)
VALUES(230,'Adam','Zurn','5 Admiral Way','Seattle','WA',NULL,'4258879009');

ORA-01400: cannot insert NULL into ("HKUMAR"."COPY_F_CUSTOMERS"."ZIP")

SELECT * FROM copy_f_customers;

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.

ANSWER:

UPDATE copy_f_staffs SET salary = (SELECT salary FROM copy_f_staffs WHERE
LOWER(first_name || ' ' || last_name) = 'bob miller')
WHERE LOWER(first_name || ' ' || last_name) = 'sue doe';

SELECT * from copy_f_staffs;

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

ANSWER:

It should work since all the mandatory columns have values.

INSERT INTO copy_f_staffs(id,first_name,last_name,birthdate,salary,overtime_rate,training,staff_type,manager_id,manager_budget,manager_target) VALUES(25,'Kai','Kim',TO_DATE('03-Nov-1988','fmdd-Mon-yyyy'),6.75,NULL,NULL,'Order Taker',NULL,NULL,NULL);

SELECT * FROM copy_f_staffs;

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.

ANSWER:

UPDATE copy_f_staffs SET manager_id = (SELECT manager_id FROM copy_f_staffs WHERE LOWER(first_name || ' ' || last_name) = 'sue doe')

WHERE LOWER(first_name || ' ' || last_name) = 'kai kim';

SELECT * FROM copy_f_staffs;

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

DELETE from departments

WHERE department_id = 60;

ANSWER:

ORA-02292: integrity constraint

(HKUMAR.EMP_DEPT_FK)

violated - child record found

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.

ANSWER:

SELECT * FROM copy_f_staffs;

DELETE FROM copy_f_staffs

WHERE LOWER(first_name || ' ' || last_name) = 'kai kim';

SELECT * FROM copy_f_staffs;

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.

ANSWER:

CREATE TABLE lesson7_emp

AS (SELECT * FROM employees);

DESCRIBE employees;

DESCRIBE lesson7_emp;

SELECT * FROM employees;

SELECT * FROM lesson7_emp;

SELECT DISTINCT employee_id FROM job_history;

7 rows returned in 0.00 seconds

DELETE FROM lesson7_emp

WHERE employee_id IN (SELECT DISTINCT employee_id FROM job_history) ;

5 row(s) deleted.

DEFAULT Values, MERGE, and Multi-Table Inserts

1. When would you want a DEFAULT value?

ANSWER:

-If no value is given while row creation and I want the field to take some predefined value. For example there may be a created on column, and I want that when a row is created, it gets filled up with current time.

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:

ANSWER:

CREATE TABLE copy_f_promotional_menus
AS (SELECT * FROM f_promotional_menus);

b. Alter the current START_DATE column attributes using:

ANSWER:

ALTER TABLE copy_f_promotional_menus
MODIFY(start_date DATE DEFAULT SYSDATE)

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?

ANSWER:

As a standard rule, we are doing all operations on copy tables rather than screwing original tables.

a. So first I will create a copy table:

CREATE TABLE copy_f_promotional_menus

AS (SELECT * FROM f_promotional_menus);

DESCRIBE f_promotional_menus;

DESCRIBE copy_f_promotional_menus;

SELECT * FROM f_promotional_menus;

SELECT * FROM copy_f_promotional_menus;

SELECT TO_CHAR(TRUNC(start_date), 'dd-mm-yyyy-hh24:mi:ss'),TO_CHAR(start_date, 'dd-mm-yyyy-hh24:mi:ss') FROM copy_f_promotional_menus;

SELECT TO_CHAR(TRUNC(sysdate), 'dd-mm-yyyy-hh24:mi:ss') FROM dual;

b. Give default vale to start_date:

ALTER TABLE copy_f_promotional_menus

MODIFY(start_date DATE DEFAULT TRUNC(SYSDATE));

c.1 Out of existing rows, say he wants to set start_date to current date for the rows where start_date is 10-Feb-2004

Set to starting of today:

UPDATE copy_f_promotional_menus

SET start_date = TRUNC(SYSDATE)

WHERE TRUNC(start_date) = TO_DATE('10-Feb-2004','dd-Mon-yyyy');

c.2 For new rows, simply skip giving any value to this field or say DEFAULT:

INSERT INTO copy_f_promotional_menus(code,name,start_date,end_date,give_away)
VALUES('115','Back to School part 2',DEFAULT,NULL,'ballpen and highlighter again');
or
INSERT INTO copy_f_promotional_menus(code,name,end_date,give_away)
VALUES('116','Back to School part 3',NULL,'ballpen and highlighter again2');

INSERT INTO copy_f_promotional_menus(code,name,start_date,end_date,give_away)
VALUES('120','New Customer',DEFAULT,TO_DATE('01-Jun-2005','dd-Mon-yyyy'),' 10% discount coupon');

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.

ANSWER:

It seems to be, this cd_number assignment is being done manually, I need not create a sequence for this. If the sequence had to be created, this point would have come after point b below, original table don' have a sequence on this column.

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

ANSWER:

CREATE TABLE manager_copy_d_cds

AS (SELECT * FROM d_cds);

DESCRIBE d_cds;

DESCRIBE manager_copy_d_cds;

SELECT * FROM d_cds;

SELECT * FROM manager_copy_d_cds;

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?

ANSWER:

VALUES(20,'Hello World Here I Am','Middle Earth Records','1998');

INSERT INTO manager_copy_d_cds(cd_number,title,producer,year)
VALUES(97,'Celebrate the Day','R & B Inc.','2003');

INSERT INTO manager_copy_d_cds(cd_number,title,producer,year)
VALUES(99,'Party Music','Old Town Records','2004');

INSERT INTO manager_copy_d_cds(cd_number,title,producer,year)
VALUES(100,'Best of Rock and Roll','Old Town Records','2004');

SELECT * FROM manager_copy_d_cds ;

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?

ANSWER:

To verify merge, first

i) I need to edit some record in manager_copy_d_cds, this should get updated from d_cds to original value.

UPDATE manager_copy_d_cds
SET title = 'hkumar'
WHERE cd_number = 90;

ii) I should delete some record in manager_copy_d_cds which is present in d_cds. This should be recreated after merge.

DELETE FROM manager_copy_d_cds
WHERE cd_number = 91;

SELECT * FROM manager_copy_d_cds ;

MERGE INTO manager_copy_d_cds tgt USING d_cds src
ON (src.cd_number = tgt.cd_number)
WHEN MATCHED THEN UPDATE
SET tgt.title = src.title, tgt.producer = src.producer, tgt.year = src.year
WHEN NOT MATCHED THEN INSERT
VALUES (src.cd_number, src.title, src.producer, src.year);

If () is missing I will get: ORA-00969: missing ON keyword

SELECT * FROM manager_copy_d_cds ;

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

ANSWER:

If I use FISRT / ALL no difference, since there is only one 'WHEN' I am using. In else there are 2 inserts.

INSERT FIRST

WHEN salary > 20000 THEN

INTO special_sal

VALUES(employee_id, salary)

ELSE

INTO sal_history

VALUES(employee_id, hire_date, salary)

INTO mgr_history

VALUES(employee_id, manager_id, salary)

SELECT employee_id, salary, hire_date, manager_id

FROM employees;

39 row(s) inserted.

SELECT COUNT(*) FROM special_sal;

1

SELECT COUNT(*) FROM sal_history;

19

SELECT COUNT(*) FROM mgr_history;

19

Creating Tables

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

ANSWER:

For credits and student_id it could have been precision and scale rather mentioned here. I assume that when 6 is written for student_id it means NUMBER(6,0) and for credits NUMBER(5, 2)

2. Write the syntax to create the grad_candidates table.

ANSWER:

```
CREATE TABLE  graduate_candidates
   ( student_id NUMBER(6,0),
    last_name VARCHAR2(75) CONSTRAINT gcs_last_name_nn NOT NULL ENABLE,
 first_name VARCHAR2(75) CONSTRAINT gcs_first_name_nn NOT NULL ENABLE,
 credits NUMBER(5,2) CONSTRAINT gcs_credits_nn NOT NULL ENABLE,
graduation_date DATE,
 CONSTRAINT gcs_std_id_pk PRIMARY KEY (student_id),
 CONSTRAINT gcs_req_fk FOREIGN KEY(credits) REFERENCES requirements(credits) ENABLE

  );
```

3. Confirm creation of the table using DESCRIBE.

ANSWER:

DESCRIBE graduate_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.

ANSWER:

CREATE TABLE kumar_table

AS (SELECT * FROM graduate_candidates);

see if copy worked well:

DESCRIBE kumar_table;

student_id is nullable and not a primary key here in kumar_table.

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

ANSWER:

INSERT INTO kumar_table (student_id, last_name, first_name, credits, graduation_date)

Values(10,'kumar','he',999.99,NULL);

SELECT LENGTH(credits) FROM kumar_table WHERE student_id = 10;

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.

ANSWER:

--USER_TABLES describes the relational tables owned by the current user. Its columns (except for OWNER) are the same as those in ALL_TABLES.

SELECT * FROM user_tables;

48 rows returned in 0.64 seconds

--USER_CATALOG lists indexes, tables, views, clusters, synonyms, and sequences owned by the current user. Its columns are the same as those in "ALL_CATALOG".

SELECT * FROM user_catalog;

407 rows returned in 0.05 seconds

SELECT DISTINCT(table_type) FROM user_catalog;

3 rows returned in 0.02 seconds

--USER_OBJECTS describes all objects owned by the current user. Its columns are the same as those in "ALL_OBJECTS".

SELECT * FROM user_objects;

117 rows returned in 0.19 seconds

SELECT DISTINCT(object_type) FROM user_objects;

6 rows returned in 0.17 seconds

SELECT * FROM user_cat;

ORA-00942: table or view does not exist

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:

ANSWER:

CREATE TABLE o_jobs AS (SELECT * FROM jobs);

CREATE TABLE o_employees AS (SELECT * FROM employees);

CREATE TABLE o_departments AS (SELECT * FROM departments);

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

ANSWER:

INSERT INTO o_jobs (job_id, job_title, min_salary, max_salary)

VALUES('HR_MAN', 'Human Resources Manager', 4500, 5500);

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

ANSWER:

INSERT INTO o_employees (employee_id, first_name, last_name, email, hire_date, job_id)

VALUES(210, 'Ramon', 'Sanchez', 'RSANCHEZ', SYSDATE, 'HR_MAN');

INSERT INTO o_employees (employee_id, first_name, last_name, email, hire_date, job_id)

VALUES(211, 'Ramon2', 'Sanchez2', 'RSANCHEZ2', SYSDATE, 'HR_MAN');

INSERT INTO o_employees (employee_id, first_name, last_name, email, hire_date, job_id)

VALUES(212, 'Ramon3', 'Sanchez3', 'RSANCHEZ3', SYSDATE, 'HR_MAN');

4. Add Human Resources to the departments table:

ANSWER:

DESCRIBE o_departments;

INSERT INTO o_departments(department_id, department_name)

VALUES (210,'Human Resources');

You will need to know which columns do not allow null values.

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

ANSWER:

There is nothing permanent in this world except change and I do make mistakes, that is why databases are also dynamic in nature and so are the tables could be modified.

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

ANSWER:

CREATE TABLE artists

(artist_id NUMBER(5,0),

first_name VARCHAR2(25) CONSTRAINT ait_first_name_nn NOT NULL ENABLE,

last_name VARCHAR2(30) CONSTRAINT ait_last_name_nn NOT NULL ENABLE,

band_name VARCHAR2(30),

email VARCHAR2(75) CONSTRAINT ait_email_nn NOT NULL ENABLE,

hr_rate NUMBER(8,2) CONSTRAINT ait_hr_rate_nn NOT NULL ENABLE,

song_id NUMBER(5,0) CONSTRAINT ait_song_id_nn NOT NULL ENABLE,

CONSTRAINT ait_id_pk PRIMARY KEY (artist_id)

);

b. INSERT one artist from the d_songs table.

ANSWER:

DESCRIBE artists;

It suggests that,

I must have artist_id, which I could take as 1 since I am inserting one row only, please note there is no sequence present here.

First name and last name I must need, I could get it from artist in d_songs by string manipulation.

band name I may skip but I can temporarily take it same as artist full name.

song id I can take from d_songs very well.

email and hourly rate, d_songs will give no clue about, so, I will have to have two constraints:

```
ALTER TABLE artists
DROP CONSTRAINT ait_email_nn;
ALTER TABLE artists
DROP CONSTRAINT ait_hr_rate_nn;
DESCRIBE artists;
```

Now insert first select from d_songs to artists:

```
INSERT INTO artists (artist_id, first_name, last_name, band_name, email, hr_rate, song_id)
SELECT 1 AS artist_id,
CASE
WHEN artist IS NULL THEN 'first name unknown'
WHEN INSTR(artist,' ') = 0 THEN artist
ELSE SUBSTR(artist,1,INSTR(artist,' ') -1)
END
AS first_name,
CASE
WHEN artist IS NULL THEN 'last name unknown'
WHEN INSTR(artist,' ') = 0 THEN artist
ELSE SUBSTR(artist,INSTR(artist,' '),LENGTH(artist))
END
AS last_name,
artist as band_name,
```

NULL as email,

NULL as hr_rate,

id as song_id

FROM d_songs

WHERE ROWNUM =1 ;

SELECT * FROM artists;

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

ANSWER:

Now leaving song_id blank would require dropping another constraint or I will get error ORA-01400: cannot insert NULL into ("HKUMAR"."ARTISTS"."SONG_ID"):

ALTER TABLE artists

DROP CONSTRAINT ait_song_id_nn;

INSERT INTO artists (artist_id, first_name, last_name, band_name, email, hr_rate, song_id)

VALUES(2,'David','Gray','david''s band','some.email@somedomain.com','999999.99',NULL);

SELECT * FROM artists;

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

1) ALTER TABLE

ANSWER:

ALTER TABLE artists

ADD (specialty VARCHAR2(100), college VARCHAR2(100));

ALTER TABLE artists

MODIFY (specialty VARCHAR2(99), college VARCHAR2(98));

ALTER TABLE artists

DROP COLUMN specialty;

ALTER TABLE artists

RENAME COLUMN college to university;

```
ALTER TABLE artists
  RENAME TO artists_new_name;

ALTER TABLE artists_new_name
MODIFY (university VARCHAR2(98) DEFAULT 'Great College');

ALTER TABLE artists_new_name
SET UNUSED (university);
SELECT column_name FROM user_tab_columns WHERE LOWER(table_name) = 'artists_new_name';

ALTER TABLE artists_new_name
DROP UNUSED COLUMNS;
```

2) DROP TABLE

ANSWER:

```
DROP TABLE artists_new2;
```

3) RENAME TABLE

ANSWER:

```
RENAME artists_new_name TO artists_new2;
```

4) TRUNCATE

ANSWER:

```
TRUNCATE TABLE artists_new2;
SELECT * FROM artists_new2;
```

5) COMMENT ON TABLE

ANSWER:

```
COMMENT ON TABLE artists_new2 IS 'The exercises are reordered to use same table, next I will drop this table';
```

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:

ANSWER:

1. Read the assignment prompt carefully: Ensure you understand what each task is asking for, such as creating a table, inserting data, or querying specific information.
2. Structure your responses clearly: For each term or task, write:
   - The Term or Requirement (e.g., "Create Table").
   - The SQL Statement you used to complete the task.
3. Use proper SQL syntax: Make sure your statements are correct and functional. Test them in a database environment if possible.

Example Format:

Term 1: Create Table

SQL Statement:

sql

Copy code

```
CREATE TABLE Artists (
   ArtistID INT PRIMARY KEY,
   Name VARCHAR(100),
   Genre VARCHAR(50),
   YearStarted INT
);
```

Term 2: Insert Data

SQL Statement:

sql

Copy code

```
INSERT INTO Artists (ArtistID, Name, Genre, YearStarted)
VALUES (1, 'DJ CoolMix', 'Electronic', 2005);
```

Term 3: Select All Artists

SQL Statement:

sql

Copy code

```
SELECT * FROM Artists;
```

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.

ANSWER:

ALTER TABLE o_employees

ADD ("Termination" VARCHAR2(100) DEFAULT TO_CHAR(SYSDATE,'Month ddth, YYYY'));

INSERT INTO o_employees (employee_id, first_name, last_name, email, hire_date, job_id)

VALUES(213, 'Ramon213', 'Sanchez213', 'RSANCHEZ213', SYSDATE, 'HR_MAN');

SELECT "Termination" FROM o_employees WHERE employee_id = 213;

If I wanted it to be 2nd instead of 02nd:

ALTER TABLE o_employees

MODIFY ("Termination" VARCHAR2(100) DEFAULT TO_CHAR(SYSDATE,'fmMonth ddth, YYYY'));

INSERT INTO o_employees (employee_id, first_name, last_name, email, hire_date, job_id)

VALUES(214, 'Ramon214', 'Sanchez214', 'RSANCHEZ214', SYSDATE, 'HR_MAN');

SELECT "Termination" FROM o_employees WHERE employee_id = 214;

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

ANSWER:

ALTER TABLE o_employees

ADD (start_date TIMESTAMP WITH LOCAL TIME ZONE);

default fractional_seconds_precision is 6 here

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

ANSWER:

DESCRIBE o_jobs;

SELECT * FROM o_jobs;

TRUNCATE TABLE o_jobs;

DESCRIBE o_jobs;

SELECT * FROM o_jobs;

Columns are still there, data is gone.

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

ANSWER:

The **DROP TABLE** statement removes the definition of oracle table along with data and indexes. Recovery of a dropped table along with even indexes may be done but it's not guaranteed using FLASHBACK:

DROP TABLE mytable;

It is possible to query what may be restored by command:

SELECT original_name, operation, droptime

FROM user_recyclebin;

FLASHBACK TABLE mytable TO BEFORE DROP;

But if PURGE is used along with DROP TABLE, there is no recyclebin in b/w for sure:

DROP TABLE mytable PURGE;

TRUNCATE TABLE removes all rows and release storage space without possibility of rollback. It will be faster than DELETE. It won't remove columns from table.

TRUNCATE TABLE sometable;

DELETE SQL statement will remove the rows but won't clean storage space. We may use COMMIT or ROLLBACK here and DELETE triggers may also be used unlike TRUNCATE TABLE.

SELECT * FROM o_employees

WHERE employee_id = 100;

1 rows returned in 0.01 seconds

DELETE FROM o_employees
WHERE employee_id = 100;
1 row(s) deleted.

SELECT * FROM o_employees

WHERE employee_id = 100;

no data found

SELECT **versions_operation, versions_starttime, versions_endtime**, employee_id, first_name, last_name, email, phone_number,hire_date,job_id,salary,commission_pct,manager_id, department_id,bonus,"Termination",start_date
FROM o_employees
VERSIONS BETWEEN SCN MINVALUE AND MAXVALUE
WHERE employee_id = 100;

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

ANSWER:

-1. Newly created column is always put at last. But select can be written to return in desired manner, so no issues.

0. Adding new column to table will always give null to pre-existing row's new field, even if, default value is assigned to new column.

1. I can increase precision of a number column.
2. In can increase length of character column.

3. I can decrease precision of number column if: it contains only nulls till now or there is no row in table. Otherwise I will get ORA-00940: invalid ALTER command

4. varchar2 can be decreased to length down to the largest value present currently in all rows.

5. Datatype can be changed altogether if all values in this column are nulls.

6. char can become varchar2 if column contain nulls or the size given is not less than any existing field for that column.

6. varchar2 can become char if column contain nulls or the size given is not less than any existing field for that column.

7. Change in default value is effective to new inserts only not the already present rows.

8. A column containing values can be dropped if this is not referenced as foreign key in further tables. Also, data values in it not recovered after column drop.

9. I can drop only one column at a time. Also, at least one column must remain, I cannot drop last column.

10. Since dropping column may take time, it does modify each row before deleting, I can use SET UNUSED command as a replacement for practical purposes and DROP UNUSED later on.

11. I can rename a column if I want.

7. Add the following comment to the o_jobs table:

"New job description added"

View the data dictionary to view your comments.

ANSWER:

COMMENT ON TABLE o_jobs IS 'New job description added';

SELECT table_name, comments

FROM user_tab_comments WHERE LOWER(table_name) = 'o_jobs';

8. Rename the o_jobs table to o_job_description.

ANSWER:

ALTER TABLE o_jobs

RENAME TO o_job_description;

RENAME o_job_description TO o_job_description2

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.

ANSWER:

CREATE TABLE copy_f_staffs

AS (SELECT * FROM f_staffs);

DESCRIBE f_staffs;

DESCRIBE copy_f_staffs;

SELECT * FROM f_staffs;

SELECT * FROM copy_f_staffs;

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

ANSWER:

DESC copy_f_staffs;

C.Drop the table.

ANSWER:

DROP TABLE copy_f_staffs;

D.Try to select from the table.

ANSWER:

SELECT * FROM copy_f_staffs;

ORA-00942: table or view does not exist

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

ANSWER:

DESCRIBE user_recyclebin ;

Please note droptime is varchar2 here.

SELECT * FROM
  (SELECT * FROM user_recyclebin ORDER BY droptime DESC)
  WHERE ROWNUM <= 100;

SELECT object_name,droptime FROM user_recyclebin WHERE LOWER(original_name) =

'copy_f_staffs';

SELECT object_name FROM user_recyclebin WHERE LOWER(original_name) = 'copy_f_staffs' AND droptime = '2016-11-02:20:14:25';

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”.

ANSWER:

SELECT * FROM "BIN$QF30ctmEV7jgU81jFJDpGA==$0";

b. Undrop the table.

ANSWER:

FLASHBACK TABLE copy_f_staffs TO BEFORE DROP;

c. Describe the table.

ANSWER:

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;

ANSWER:

SELECT * FROM copy_f_staffs;

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

ANSWER:

UPDATE copy_f_staffs

SET salary = 12

WHERE first_name = 'Sue' AND last_name = 'Doe';

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

ANSWER:

SELECT * FROM copy_f_staffs;

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

ANSWER:

UPDATE copy_f_staffs

SET salary = 2

WHERE first_name = 'Sue' AND last_name = 'Doe';

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

ANSWER:

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.

ANSWER:

```
SELECT versions_operation, versions_starttime,  versions_endtime, id, first_name, last_name,
birthdate, salary,overtime_rate,training,staff_type,manager_id, manager_budget,manager_target
FROM copy_f_staffs
VERSIONS BETWEEN SCN MINVALUE AND MAXVALUE
WHERE id = 12;
```

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.

ANSWER:

```
UPDATE copy_f_staffs
SET salary = (SELECT salary
FROM copy_f_staffs
VERSIONS BETWEEN SCN MINVALUE AND MAXVALUE
WHERE first_name = 'Sue' AND last_name = 'Doe' AND versions_operation IS NULL AND
versions_starttime IS NULL)
WHERE first_name = 'Sue' AND last_name = 'Doe';
```

Now,

```
SELECT * FROM copy_f_staffs;
SELECT versions_operation, versions_starttime,  versions_endtime, id, first_name, last_name, birthdate,
salary,overtime_rate,training,staff_type,manager_id, manager_budget,manager_target
FROM copy_f_staffs
VERSIONS BETWEEN SCN MINVALUE AND MAXVALUE


WHERE id = 12;
```

Ex. No. : 14 **Date:19.11.2024**

Register No.: 221701055 **Name:S Sneha**

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?

ANSWER:

Database can be as reliable as the data in it, and database rules are implemented as Constraint to maintain data integrity. For example these constraints may prohibit deletion of a table or some row when insertion, updation or deletion is executed. Type of constraints:

· PRIMARY KEY Constraint

· UNIQUE Constraint

· FOREIGN KEY Constraint

· CHECK Constraint with condition applied on the column/columns (they work at row level)

· NOT NULL Constraint (implemented at row level using special CHECK Constraint having condition IS NOT NULL for single column)

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

ANSWER:

· Constraints referring to more than one column are defined at Table Level
· NOT NULL constraint must be defined at column level as per ANSI/ISO SQL standard.
· If word CONSTRAINT is used in a CREATE TABLE statement, I must specify constraint name. Also, that is why, Table level constraint must be user-named.

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

ANSWER:

· If a constraint is violated in a SQL statement execution, it is easy to identify the cause with user-named constraints.

· It is easy to alter names/drop constraint.

· Handling production issues may be faster with user-named constraints

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.

ANSWER:

Global Fast Foods global_locations Table						
NAME	TYPE	DataType	LENGTH	PRECISION	SCALE	NULLABLE
id	pk	**NUMBER**	**6**	**0**		No
name		**VARCHAR2**	**50**			
date_opened		**DATE**				No
address		**VARCHAR2**	**50**			No
city		**VARCHAR2**	**30**			No
zip_postal_code		**VARCHAR2**	**12**			
phone		**VARCHAR2**	**20**			
email	uk	**VARCHAR2**	**75**			
manager_id		**NUMBER**	**6**	**0**		
emergency_contact		**VARCHAR2**	**20**			

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

ANSWER:

Global Fast Foods global_locations Table						
NAME	TYPE	DataType	LENGTH	PRECISION	SCALE	NULLABLE
id	pk	NUMBER	6	0		No
name		VARCHAR2	50			Yes
date_opened		DATE				No
address		VARCHAR2	50			No
city		VARCHAR2	30			No
zip_postal_code		VARCHAR2	12			Yes

phone		VARCHAR2	20			Yes
email	uk	VARCHAR2	75			Yes
manager_id		NUMBER	6	0		Yes
emergency_contact		VARCHAR2	20			Yes

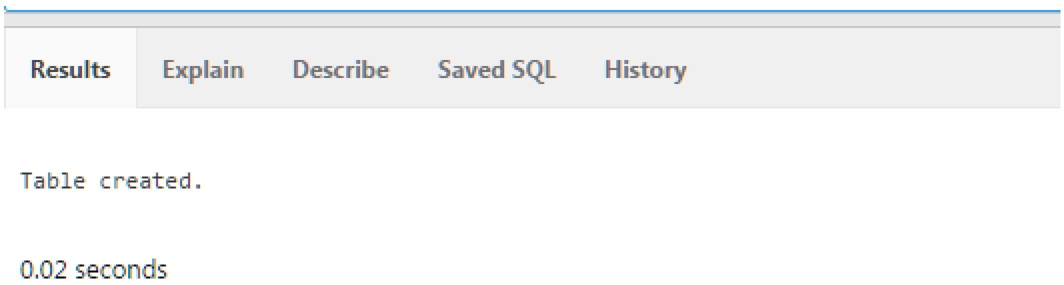
6. Write the CREATE TABLE statement for the Global Fast Foods locations table to define the constraints at the column level.

ANSWER:

```
CREATE TABLE  f_global_locations
( id NUMBER(6,0) CONSTRAINT f_gln_id_pk PRIMARY KEY ,
name VARCHAR2(50),
date_opened DATE CONSTRAINT f_gln_dt_opened_nn NOT NULL ENABLE,
address VARCHAR2(50) CONSTRAINT f_gln_add_nn NOT NULL ENABLE,
city VARCHAR2(30)  CONSTRAINT f_gln_city_nn NOT NULL ENABLE,
zip_postal_code VARCHAR2(12),
phone VARCHAR2(20),
email VARCHAR2(75) CONSTRAINT f_gln_email_uk UNIQUE,
manager_id NUMBER(6,0),
emergency_contact VARCHAR2(20)
);
```

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

ANSWER:



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

ANSWER:

DESCRIBE f_global_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	
phone	varchar2	15			X	
email	varchar2	80			X	
manager_id	number	4			X	
contact	varchar2	40			X	

ANSWER:

CREATE TABLE f_global_locations

(id NUMBER(6,0) CONSTRAINT f_gln_id_pk PRIMARY KEY ,

name VARCHAR2(50),

date_opened DATE CONSTRAINT f_gln_dt_opened_nn NOT NULL ENABLE,

address VARCHAR2(50) CONSTRAINT f_gln_add_nn NOT NULL ENABLE,

city VARCHAR2(30) CONSTRAINT f_gln_city_nn NOT NULL ENABLE,

zip_postal_code VARCHAR2(12),

phone VARCHAR2(20),

email VARCHAR2(75) ,

manager_id NUMBER(6,0),

emergency_contact VARCHAR2(20),

CONSTRAINT f_gln_email_uk UNIQUE(email)

PRIMARY KEY, FOREIGN KEY, and CHECK Constraints

1. What is the purpose of a

- PRIMARY KEY
- FOREIGN KEY
- CHECK CONSTRAINT

ANSWER:

a) Uniquely identify each row in table.

b) Referential integrity constraint links back parent table's primary/unique key to child table's column.

c) Explicitly define condition to be met by each row's fields. This condition must be returned as true or unknown.

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

ANSWER:

animal_id NUMBER(6) **- PRIMARY KEY**

name VARCHAR2(25)

license_tag_number NUMBER(10)**- UNIQUE**

admit_date DATE**- NOT NULL**

adoption_id NUMBER(5),

vaccination_date DATE- **NOT NULL**

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

ANSWER:

```
CREATE TABLE  animals
( animal_id  NUMBER(6,0) CONSTRAINT anl_anl_id_pk PRIMARY KEY ,
name VARCHAR2(25),
license_tag_number NUMBER(10,0) CONSTRAINT anl_l_tag_num_uk UNIQUE,
admit_date  DATE CONSTRAINT anl_adt_dat_nn NOT NULL ENABLE,
adoption_id   NUMBER(5,0),
vaccination_date  DATE CONSTRAINT anl_vcc_dat_nn NOT NULL ENABLE

);

DESCRIBE animals;

SELECT *
FROM user_constraints
WHERE LOWER(table_name) = 'animals';
```

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

ANIMAL_ I D	NA M E	LICENSE_TAG_NUM BE R	ADMIT_DA T E	ADOPTION_ I D	VACCINATION_D AT E
101	Spot	35540	10-Oct-2004	205	12-Oct-2004

ANSWER:

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.

ANSWER:

Note: I tried below mentioned queries with d_songs(id) and later find replace all for the sake of problem statement.

If there is a value in animals.adoption_id, which is not present as primary key/unique key in parent table, it will give error, so first fix it:

UPDATE animals

SET adoption_id = (SELECT id FROM adoptions WHERE ROWNUM = 1);

or

UPDATE animals

```
SET adoption_id   =  NULL;
SELECT * FROM animals;
```

Adding foreign key using column level statement:

```
ALTER TABLE animals
MODIFY ( adoption_id   NUMBER(5,0) CONSTRAINT anl_adopt_id_fk  REFERENCES adoptions(id)
ENABLE );
```

Verify that constraint is generated:

```
SELECT *
FROM user_constraints
WHERE LOWER(table_name) = 'animals' AND constraint_type = 'R';
```

Now drop it:

```
ALTER TABLE animals
DROP CONSTRAINT anl_adopt_id_fk  ;
```

Adding foreign key using table level statement:

```
ALTER TABLE  animals ADD CONSTRAINT anl_adopt_id_fk  FOREIGN KEY (adoption_id)
        REFERENCES  adoptions(id) ENABLE;
```

Verify that constraint is generated:

```
SELECT *
FROM user_constraints
WHERE LOWER(table_name) = 'animals' AND constraint_type = 'R';
```

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

ANSWER:

ALTER TABLE animals
ADD CONSTRAINT anl_adopt_id_fk FOREIGN KEY (adoption_id)
REFERENCES adoptions (id) ENABLE ;

Gives:

SELECT delete_rule
FROM user_constraints
WHERE LOWER(table_name) = 'animals' AND constraint_type = 'R';

Assume, adoptions has a row with id 500 and this row is referenced in animals. If I try:

DELETE FROM adoptions WHERE id= 500;

I will get error:

ORA-02292: integrity constraint (HKUMAR.ANL_ADOPT_ID_FK) violated - child record found

a. ON DELETE CASCADE

ANSWER:

Recreate the row in adoptions with adoptions.id 500 again, refer it in animals.adoption_id.

ALTER TABLE animals
DROP CONSTRAINT anl_adopt_id_fk ;

ALTER TABLE animals
ADD CONSTRAINT anl_adopt_id_fk FOREIGN KEY (adoption_id)
REFERENCES adoptions(id) ON DELETE CASCADE ENABLE ;

SELECT delete_rule
FROM user_constraints
WHERE LOWER(table_name) = 'animals' AND constraint_type = 'R';

DELETE FROM adoptions WHERE id= 500;

But in fact child table also lost the row referring to this parent row:

SELECT * FROM animals;

b. ON DELETE SET NULL

ANSWER:

ALTER TABLE animals

DROP CONSTRAINT anl_adopt_id_fk ;

ALTER TABLE animals

ADD CONSTRAINT anl_adopt_id_fk FOREIGN KEY (adoption_id)

REFERENCES adoptions(id) ON DELETE SET NULL ENABLE ;

SELECT delete_rule

FROM user_constraints

WHERE LOWER(table_name) = 'animals' AND constraint_type = 'R';

DELETE FROM adoptions WHERE id= 500;

SELECT * FROM animals;

Value in animals.adoption_id where 500 adoptions.id from parent was referred is now set to NULL;

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

ANSWER:

I cannot specify check constraint for a view however in this case I could use WITH CHECK OPTION clause

- I am restricted to columns from self table and fields in self row.
- I cannot use subqueries and scalar subquery expressions.
- I cannot call functions that are not deterministic e.g. CURRENT_DATE, CURRENT_TIMESTAMP,

DBTIMEZONE, LOCALTIMESTAMP, SESSIONTIMEZONE, SYSDATE, SYSTIMESTAMP, UID, USER, and USERENV

CREATE TABLE hemant2
(somecolumnname2 DATE CHECK (somecolumnname2 > SYSDATE)
);

Says ORA-02436: date or system variable wrongly specified in CHECK constraint

But below mentioned works:

CREATE TABLE hemant2
(somecolumnname2 DATE CHECK (somecolumnname2 > TO_DATE('03-Nov-2016','DD-Mon-YYYY'))
);

SELECT *
FROM user_constraints
WHERE LOWER(table_name) = 'hemant2';

- I cannot call user defined functions
- I cannot dereference a REF column e.g. using the DEREF function
- Nested table columns or attributes are not allowed
- pseudocolumns CURRVAL, NEXTVAL, LEVEL, or ROWNUM are not allowed
- I can't use Date constants that are not fully specified

This last statement is ambiguous, better is to convert a date with fully specified format in check constraint. I give one example for this:

In Application Express 4.0.2.00.09:

SELECT SYSDATE, TO_CHAR(SYSDATE, 'dd-month-yyyy') from dual;

SELECT TO_DATE('11/04/2016'), TO_CHAR(TO_DATE('11/04/2016'), 'dd-month-yyyy') from dual;

So, I got to know that in Application Express 4.0.2.00.09 default format is 'mm/dd/yyyy'

But

CREATE TABLE hemant3
(somecolumnname2 DATE CHECK (somecolumnname2 > TO_DATE('11/04/2016'))
);

Says: ORA-01843: not a valid month

But Below mentioned works:

```
CREATE TABLE hemant3
(somecolumnname2 DATE CHECK (somecolumnname2  >  TO_DATE('11/04/2016','mm/dd/yyyy' ))
);
```

Application Express 5.0.3.00.03

```
SELECT SYSDATE, TO_CHAR(SYSDATE, 'dd-month-yyyy') from dual;
```

```
SELECT TO_DATE('03-Nov-2016'),  TO_CHAR(TO_DATE('03-Nov-2016'), 'dd-month-yyyy') from dual;
```

So, I got to know that in Application Express 5.0.3.00.03 default format is 'dd-Mon-yyyy'

As opposed to previous version of apex, here, below mentioned works!

```
CREATE TABLE hemant3
(somecolumnname2 DATE CHECK (somecolumnname2  >  TO_DATE('03-Nov-2016'))
);
```

Also, below mentioned works:

```
CREATE TABLE hemant5
(somecolumnname2 DATE CHECK (somecolumnname2  >  TO_DATE('11/04/2016','mm/dd/yyyy' ))
);
```

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.

ANSWER:

```
CREATE TABLE copy_d_clients
AS ( SELECT * FROM d_clients);
DESCRIBE copy_d_clients ;
DESCRIBE d_clients;
SELECT * FROM d_clients ;
SELECT * FROM copy_d_clients ;
SELECT *
FROM user_constraints
WHERE LOWER(table_name) IN ( 'd_clients', 'copy_d_clients');
```

```
CREATE TABLE copy_d_events
AS ( SELECT * FROM d_events);
DESCRIBE copy_d_events ;
DESCRIBE d_events;
SELECT * FROM d_events ;
```

```
SELECT * FROM copy_d_events ;
SELECT *
FROM user_constraints
WHERE LOWER(table_name) IN ( 'd_events', 'copy_d_events');
```

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?

ANSWER:

- **ADD (uses modify clause to add not null on a column though)**
- **DROP**
- **ENABLE/DISABLE**

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?

ANSWER:

ALTER TABLE copy_d_clients
ADD CONSTRAINT copy_d_clt_client_number_pk PRIMARY KEY (client_number);

SELECT *

FROM user_constraints

WHERE LOWER(table_name) = 'copy_d_clients' and constraint_type = 'P';

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?

ANSWER:

ALTER TABLE copy_d_events

ADD CONSTRAINT copy_d_eve_client_number_fk FOREIGN KEY (client_number)

REFERENCES copy_d_clients (client_number) ENABLE;

SELECT *

FROM user_constraints

WHERE LOWER(table_name) = 'copy_d_events' and constraint_type = 'R';

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

ANSWER:

SELECT constraint_name, constraint_type, table_name

FROM user_constraints

WHERE table_name = UPPER('copy_d_events') ;

SELECT chld.table_name "Subject", chldcols.column_name "Subject Column Name", chld.constraint_name "constraint_name in Subject", chld.constraint_type "constraint_type in Subject", prnt.table_name "Parent of FK", prntcols.column_name "Parent's Column Name", prnt.constraint_name "Parent PK"

FROM user_constraints chld LEFT OUTER JOIN user_constraints prnt ON chld.r_constraint_name = prnt.constraint_name

LEFT OUTER JOIN user_cons_columns chldcols ON chld.constraint_name = chldcols.constraint_name

LEFT OUTER JOIN user_cons_columns prntcols ON prnt.constraint_name = prntcols.constraint_name

WHERE chld.table_name = UPPER('copy_d_events');

SELECT constraint_name, constraint_type, table_name

FROM user_constraints

WHERE table_name = UPPER('copy_d_clients') ;

SELECT chld.table_name "Subject", chldcols.column_name "Subject Column Name", chld.constraint_name "constraint_name in Subject", chld.constraint_type "constraint_type in Subject", prnt.table_name "Parent of FK", prntcols.column_name "Parent's Column Name", prnt.constraint_name "Parent PK"

FROM user_constraints chld LEFT OUTER JOIN user_constraints prnt ON chld.r_constraint_name = prnt.constraint_name

LEFT OUTER JOIN user_cons_columns chldcols ON chld.constraint_name = chldcols.constraint_name

LEFT OUTER JOIN user_cons_columns prntcols ON prnt.constraint_name = prntcols.constraint_name

WHERE chld.table_name = UPPER('copy_d_clients');

a. The constraint name for the primary key in the copy_d_clients table is_________________.

ANSWER:

COPY_D_CLT_CLIENT_NUMBER_PK

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

ANSWER:

ALTER TABLE copy_d_clients

DROP CONSTRAINT COPY_D_CLT_CLIENT_NUMBER_PK ;

I get message:

ORA-02273: this unique/primary key is referenced by some foreign keys

ALTER TABLE copy_d_clients

DROP CONSTRAINT COPY_D_CLT_CLIENT_NUMBER_PK **CASCADE** ;

SELECT constraint_name, constraint_type, table_name

FROM user_constraints

WHERE table_name = UPPER('copy_d_events') AND constraint_type = 'R';

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

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

ANSWER:

INSERT INTO

copy_d_events(client_number,id,name,event_date,description,cost,venue_id,package_code,theme_code)

VALUES(7125,140,'Cline Bas Mitzvah',TO_DATE('15-Jul-2004','dd-Mon-yyyy'),'Church and Private Home formal',4500,105,87,77);

ORA-02291: integrity constraint (HKUMAR.COPY_D_EVE_CLIENT_NUMBER_FK) violated - parent key not found

SELECT * FROM copy_d_clients WHERE client_number = 7125;

no data found

That is why I got above error ORA-02291.

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.

ANSWER:

ALTER TABLE copy_d_clients

DISABLE CONSTRAINT COPY_D_CLT_CLIENT_NUMBER_PK ;

I get message ORA-02297: cannot disable constraint (HKUMAR.COPY_D_CLT_CLIENT_NUMBER_PK) **- dependencies exist.**

ALTER TABLE copy_d_clients

DISABLE CONSTRAINT COPY_D_CLT_CLIENT_NUMBER_PK **CASCADE**;

Table altered.

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

ANSWER:

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

1 row(s) inserted.

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

ANSWER:

ALTER TABLE copy_d_clients
ENABLE CONSTRAINT COPY_D_CLT_CLIENT_NUMBER_PK CASCADE;

ORA-00933: SQL command not properly ended

ALTER TABLE copy_d_clients
ENABLE CONSTRAINT COPY_D_CLT_CLIENT_NUMBER_PK ;

In case of enable I don’t have option like CASCADE.

10. If you wanted to enable the foreign-key column and reestablish the referential integrity between these two tables, what must be done?

ANSWER:

ALTER TABLE copy_d_events
ENABLE CONSTRAINT COPY_D_EVE_CLIENT_NUMBER_FK;

ORA-02298: cannot validate (HKUMAR.COPY_D_EVE_CLIENT_NUMBER_FK) - parent keys not found

So first I need to fix the row with client_number to a valid value/null.
But since client_number is not nullable, I will either have to delete invalid row or update that row.

DELETE FROM copy_d_events WHERE

client_number NOT IN (SELECT client_number FROM copy_d_clients);

1 row(s) deleted.

ALTER TABLE copy_d_events
ENABLE CONSTRAINT COPY_D_EVE_CLIENT_NUMBER_FK;
Table altered.

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

ANSWER:

Generally to make bulk operations fast, where my input data is diligently sanitized and I am sure, it is safe to save some time in this clumsy process.

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

ANSWER:

Queries are same as in point 2,3, 4 above.

I can check value of CONSTRAINT_TYPE in all_constraints/user_constraints view.

· C - Check constraint

Sub-case - if I see SEARCH_CONDITION something like "FIRST_NAME" IS NOT NULL , its a NOT NULL constraint.

· P - Primary key

· R - Referential integrity (fk)

· U - Unique key

Ex. No. : 15 **Date:19.11.2024**

Register No.: 221701055 **Name:S Sneha**

Creating Views

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

ANSWER:

· Restrict access and display selective columns

· Reduce complexity of queries from other internal systems. So, providing a way to view same data in a different manner.

· Let the app code rely on views and allow the internal implementation of tables to be modified later.

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.

ANSWER:

The subquery:

SELECT d_songs.id, d_songs.title "Song Title", d_songs.artist

from d_songs INNER JOIN d_types ON d_songs.type_code = d_types.code

where d_types.description = 'New Age';

Now the view:

CREATE VIEW view_d_songs AS

SELECT d_songs.id, d_songs.title "Song Title", d_songs.artist from d_songs INNER JOIN d_types ON d_songs.type_code = d_types.code

where d_types.description = 'New Age';

But second execution of this SQL statement will say:

ORA-00955: name is already used by an existing object

If I want to recreate the view/ create it first time

CREATE OR REPLACE VIEW view_d_songs AS

SELECT d_songs.id, d_songs.title "Song Title", d_songs.artist

from d_songs INNER JOIN d_types ON d_songs.type_code = d_types.code

where d_types.description = 'New Age';

Verify results again:

SELECT * FROM view_d_songs ;

3. SELECT * FROM view_d_songs. What was returned?

ANSWER:

The result is same as that of

SELECT d_songs.id, d_songs.title "Song Title", d_songs.artist

from d_songs INNER JOIN d_types ON d_songs.type_code = d_types.code

where d_types.description = 'New Age';

SELECT * FROM view_d_songs ;

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

ANSWER:

CREATE OR REPLACE VIEW view_d_songs AS

SELECT d_songs.id, d_songs.title "Song Title", d_songs.artist, d_songs.type_code

```
from d_songs INNER JOIN d_types ON d_songs.type_code = d_types.code
where d_types.description = 'New Age';
```

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.

ANSWER:

```
CREATE OR REPLACE VIEW view_d_events_pkgs AS
SELECT evt.name "Name of Event", TO_CHAR(evt.event_date, 'dd-Month-yyyy')  "Event date",
thm.description "Theme description"
FROM  d_events  evt INNER JOIN d_themes  thm  ON evt.theme_code = thm.code
WHERE evt.event_date <= ADD_MONTHS(SYSDATE,1);
SELECT * FROM view_d_events_pkgs ;
```

6. 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.

ANSWER:

DESCRIBE employees;

suggests:

Salary is a nullable field, I don't want to miss nulls in average/min/max calculation.

There may be some employees without department mentioned since it is nullable. I want to miss such records in my calculations.

SELECT department_id FROM departments WHERE department_id NOT IN (SELECT NVL(department_id,0) FROM employees);

Suggests:

There may be a department for which there is no record in employees table.

CREATE OR REPLACE VIEW view_min_max_avg_dpt_salary ("Department Id", "Department Name", "Max Salary", "Min Salary", "Average Salary") AS

SELECT dpt.department_id, dpt.department_name, MAX(NVL(emp.salary,0)), MIN(NVL(emp.salary,0)), ROUND(AVG(NVL(emp.salary,0)),2)

FROM departments dpt LEFT OUTER JOIN employees emp ON dpt.department_id = emp.department_id

GROUP BY (dpt.department_id, dpt.department_name);

SELECT * FROM view_min_max_avg_dpt_salary;

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.

ANSWER:

```
CREATE TABLE copy_d_songs
AS ( SELECT * FROM d_songs);
DESCRIBE copy_d_songs;
DESCRIBE d_songs;
SELECT * FROM d_songs;
SELECT * FROM copy_d_songs;
```

```
CREATE TABLE copy_d_events
AS ( SELECT * FROM d_events);
DESCRIBE copy_d_events ;
DESCRIBE d_events;
SELECT * FROM d_events ;
SELECT * FROM copy_d_events ;
```

```
CREATE TABLE copy_d_cds
AS ( SELECT * FROM d_cds);
DESCRIBE copy_d_cds;
DESCRIBE d_cds;
SELECT * FROM d_cds;
SELECT * FROM copy_d_cds ;
```

```
CREATE TABLE copy_d_clients
AS ( SELECT * FROM d_clients);
```

DESCRIBE copy_d_clients ;

DESCRIBE d_clients;

SELECT * FROM d_clients ;

SELECT * FROM copy_d_clients ;

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.

ANSWER:

USER_UPDATABLE_COLUMNS describes columns in a join view that can be updated by the current user, subject to appropriate privileges.

SELECT owner, table_name, column_name, updatable,insertable, deletable

FROM user_updatable_columns WHERE LOWER(table_name) = 'copy_d_songs';

SELECT owner, table_name, column_name, updatable,insertable, deletable

FROM user_updatable_columns WHERE LOWER(table_name) = 'copy_d_events';

SELECT owner, table_name, column_name, updatable,insertable, deletable

FROM user_updatable_columns WHERE LOWER(table_name) = 'copy_d_cds';

SELECT owner, table_name, column_name, updatable,insertable, deletable

FROM user_updatable_columns WHERE LOWER(table_name) = 'copy_d_clients';

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.

ANSWER:

CREATE OR REPLACE VIEW view_copy_d_songs AS
SELECT *
FROM copy_d_songs;

SELECT * FROM view_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

ANSWER:

INSERT INTO view_copy_d_songs(id,title,duration,artist,type_code)
VALUES(88,'Mello Jello','2 min','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.

ANSWER:

CREATE OR REPLACE VIEW read_copy_d_cds AS
SELECT *
FROM copy_d_cds
WHERE year = '2000'
WITH READ ONLY ;

SELECT * FROM read_copy_d_cds;

5. Using the read_copy_d_cds view, execute a DELETE FROM read_copy_d_cds WHERE cd_number = 90;

ANSWER:

ORA-42399: cannot perform a DML operation on a read-only view

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.

ANSWER:

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.

ANSWER:

DELETE FROM read_copy_d_cds

WHERE year = '2000';

8. Use the read_copy_d_cds view to delete cd_number 90 from the underlying copy_d_cds table.

ANSWER:

DELETE FROM read_copy_d_cds

WHERE cd_number = 90;

9. Use the read_copy_d_cds view to delete year 2001 records.

ANSWER:

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?

ANSWER:

Only the one in problem 7 above, not the one in 8 and 9

11.What are the restrictions on modifying data through a view?

ANSWER:

For simple views, all DML's are OK, but for complex views:

Delete restricted if it contains:

Group functions

GROUP BY CLAUSE

DISTINCT

pseudocolumn ROWNUM Keyword

Modify restricted if it contains:

Group functions

GROUP BY CLAUSE

DISTINCT

pseudocolumn ROWNUM Keyword

Column defined by expressions

INSERT restricted if it contains:

Group functions

GROUP BY CLAUSE

DISTINCT

pseudocolumn ROWNUM Keyword

Column defined by expressions

Does not include NOT NULL columns in the base table.

12. What is Moore's Law? Do you consider that it will continue to apply indefinitely? Support your opinion with research from the internet.

ANSWER:

It roughly predicted that computing power nearly doubles every year. But Moore also said in 2005 that as per nature of exponential functions, this trend may not continue forever.

13. What is the "singularity" in terms of computing?

ANSWER:

Is the hypothesis that the invention of artificial superintelligence will abruptly trigger runaway technological growth, resulting in unfathomable changes to human civilization.

3 Reasons To Believe The Singularity Is Near as per Greg Satell on Forbes:

- We're Going Beyond Moore's Law
- Robots Are Doing Human Jobs

- We’re Editing Genes

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.

ANSWER:

CREATE OR REPLACE VIEW view_copy_d_songs AS

SELECT title, artist

FROM copy_d_songs;

SELECT * FROM view_copy_d_songs;

2. Issue a DROP view_copy_d_songs. Execute a SELECT * statement to verify that the view has been deleted.

ANSWER:

DROP VIEW view_copy_d_songs;

SELECT * FROM view_copy_d_songs;

ORA-00942: table or view does not exist

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.

ANSWER:

SELECT * FROM

(SELECT last_name, salary FROM employees ORDER BY salary DESC)

```
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.

ANSWER:

There may be some employees without department mentioned since it is nullable. I want to miss such records in my calculations. This is achieved in dptmx in-line view itself. Also a department without an employee is also taken in.

```
SELECT empm.last_name, empm.salary, dptmx.department_id
FROM
(SELECT dpt.department_id, MAX(NVL(emp.salary,0)) max_dpt_sal
FROM departments dpt LEFT OUTER JOIN employees emp ON dpt.department_id =
emp.department_id
GROUP BY dpt.department_id) dptmx LEFT OUTER JOIN employees empm ON dptmx.department_id
= empm.department_id
WHERE NVL(empm.salary,0) = dptmx.max_dpt_sal;
```

5. Create a query that will return the staff members of Global Fast Foods ranked by salary from lowest to highest.

ANSWER:

So, it means lowest has rank one:

```
SELECT ROWNUM,last_name, salary
FROM
(SELECT * FROM f_staffs ORDER BY SALARY);
```

Indexes and Synonyms

1. What is an index and what is it used for?

ANSWER:

Definition: These are schema objects which make retrieval of rows from table faster.

- They are meant to be efficient way to find data in database. I may like to drop an index if, queries in application are not using some index or say it is not speeding up the queries or may be table is very small. An index provides direct and fast access to row in table.

· I should create an index if the table is large and most queries are expected to retrieve less than 2 to 4 percent of the rows.

· I should create an index if one or more columns are frequently used together in a join condition.

Purpose: An index provides direct and fast access to row in table. They provide indexed path to locate data quickly, so hereby reduce necessity of heavy disk input/output operations.

Track usage of index:

Look into what indexes employees table has:

SELECT ucm.index_name, ucm.column_name, ucm.column_position, uix.uniqueness
FROM user_indexes uix INNER JOIN user_ind_columns ucm ON uix.index_name = ucm.index_name
WHERE ucm.table_name = 'EMPLOYEES';

Start monitoring an index:

ALTER INDEX emp_id_pk MONITORING USAGE;

Note down column values in V$OBJECT_USAGE:

SELECT * FROM v$object_usage WHERE index_name = 'EMP_ID_PK';

Run a statement which may be using the index:

SELECT * from employees where employee_id = 100;

Note down column values in V$OBJECT_USAGE:

SELECT * FROM v$object_usage WHERE index_name = 'EMP_ID_PK';

Stop monitoring an index:

ALTER INDEX emp_id_pk NOMONITORING USAGE;

2. What is a ROWID, and how is it used?

ANSWER:

Indexes use ROWID's (base 64 string representation of the row address containing block identifier, row location in the block and the database file identifier) which is the fastest way to access any particular row.

3. When will an index be created automatically?

ANSWER:

For primary/unique keys: Although unique index can be created manually, but preferred should be by using unique/primary constraint in the table. So, it means that primary key/unique key use already existing unique index but if index is not present already, it is created while applying unique/primary key constraint.

Oracle also creates index automatically for LOB storage, xmltype and materialized view.

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.

ANSWER:

Creating index (non-unique):

CREATE INDEX d_tlg_cd_number_fk_i

on d_track_listings (cd_number);

Verify by SQL statement:

SELECT ucm.index_name, ucm.column_name, ucm.column_position, uix.uniqueness

FROM user_indexes uix INNER JOIN user_ind_columns ucm ON uix.index_name = ucm.index_name

WHERE ucm.table_name = 'D_TRACK_LISTINGS' AND column_name = 'CD_NUMBER';

Verify by object browser

Or search in object browser

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.

ANSWER:

SELECT ucm.index_name, ucm.column_name, ucm.column_position, uix.uniqueness

FROM user_indexes uix INNER JOIN user_ind_columns ucm ON uix.index_name =

ucm.index_name

WHERE ucm.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.

ANSWER:

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.

ANSWER:

CREATE PUBLIC SYNONYM dj_tracks FOR d_track_listings;

ORA-01031: insufficient privileges

Means I don't have CREATE PUBLIC SYNONYM privilege

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.

ANSWER:

Read this as last_name

What All indexes this table has right now:

SELECT ucm.index_name, ucm.column_name, ucm.column_position, uix.uniqueness

FROM user_indexes uix INNER JOIN user_ind_columns ucm ON uix.index_name = ucm.index_name

WHERE ucm.table_name = 'D_PARTNERS';

Create index:

CREATE INDEX d_ptr_last_name_idx

ON d_partners(LOWER(last_name));

Start monitoring an index:

ALTER INDEX d_ptr_last_name_idx MONITORING USAGE;

Note down column values in V$OBJECT_USAGE:

SELECT * FROM v$object_usage WHERE index_name = 'D_PTR_LAST_NAME_IDX';

Run a statement which may be using the index:

SELECT *

FROM d_partners

WHERE LOWER(last_name) = 'something';

Note down column values in V$OBJECT_USAGE:

SELECT * FROM v$object_usage WHERE index_name = 'D_PTR_LAST_NAME_IDX';

Stop monitoring an index:

ALTER INDEX d_ptr_last_name_idx NOMONITORING USAGE;

If I want case insensitive search rather than upper or lower confusion:

CREATE INDEX d_ptr_last_name2_idx

ON d_partners(NLSSORT (last_name, 'NLS_SORT=BINARY_CI'));

```
ALTER INDEX d_ptr_last_name2_idx MONITORING USAGE;

SELECT * FROM v$object_usage WHERE index_name = 'D_PTR_LAST_NAME2_IDX';
```

Just making sure, the previous index is not used in execution plan:

```
DROP INDEX d_ptr_last_name_idx;

SELECT *
FROM d_partners
WHERE NLSSORT (last_name, 'NLS_SORT=BINARY_CI') = NLSSORT ('pLuMb',
'NLS_SORT=BINARY_CI');

SELECT * FROM v$object_usage WHERE index_name = 'D_PTR_LAST_NAME2_IDX';

ALTER INDEX d_ptr_last_name2_idx NOMONITORING USAGE;
```

9. Create a synonym for the D_TRACK_LISTINGS table. Confirm that it has been created by querying the data dictionary.

ANSWER:

See problem 7 above for creation part too.

CREATE SYNONYM dj_tracks FOR d_track_listings;

ORA-00955: name is already used by an existing object

CREATE SYNONYM dj_tracks2 FOR d_track_listings;

Synonym created.

SELECT * FROM user_synonyms WHERE table_NAME = UPPER('d_track_listings');

10.Drop the synonym that you created in question

ANSWER:

DROP SYNONYM dj_tracks2;

Synonym dropped.

SELECT * FROM user_synonyms WHERE table_NAME = UPPER('d_track_listings');

Ex. No. : 16 **Date:19.11.2024**

Register No.: 221701055 **Name:S Sneha**

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.

<u>Types of Indexes</u>

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.

<u>Creating an Index</u>

• 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*]...);

<u>EXAMPLE:</u>

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*;

<u>Find the Solution for the following:</u>

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.

<u>ANSWER:</u>

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

<u>ANSWER:</u>

SELECT FROM sequence_name, max_value, increment_by, last_number user_sequences;

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.

<u>ANSWER:</u>

INSERT INTO dept VALUES (dept_id_seq.nextval, 'Education'); INSERT INTO dept VALUES (dept_id_seq.nextval, 'Administration');

4. Create a nonunique index on the foreign key column (DEPT_ID) in the EMP table.

ANSWER:

CREATE INDEX emp_dept_id_idx ON emp (dept_id);

5. Display the indexes and uniqueness that exist in the data dictionary for the EMP table.

ANSWER:

SELECT FROM WHERE index_name, table_name, uniqueness user_indexes table_name = 'EMP';

Ex. No. : 17 **Date:19.11.2024**

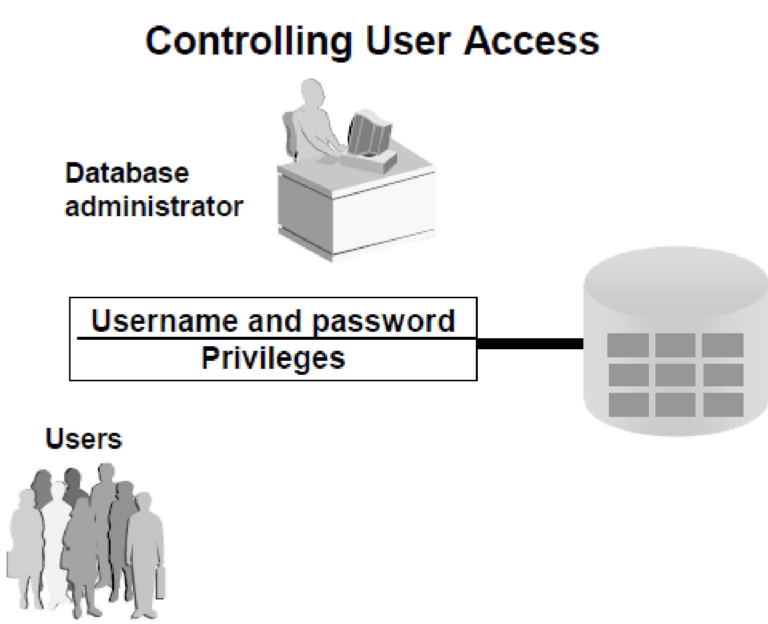
Register No.: 221701055 **Name: S Sneha**

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?

ANSWER:

The CREATE SESSION system privilege

---

2. What privilege should a user be given to create tables?

ANSWER:

The CREATE TABLE privilege

---

3. If you create a table, who can pass along privileges to other users on your table?

ANSWER:

You can, or anyone you have given those privileges to by using the WITH GRANT OPTION.

---

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?

ANSWER:

Create a role containing the system privileges and grant the role to the users

---

5. What command do you use to change your password?

ANSWER:

The ALTER USER statement.

6. Grant another user access to your DEPARTMENTS table. Have the user grant you query access to his or her DEPARTMENTS table.

ANSWER:

Team 2 executes the GRANT statement. GRANT select ON departments TO <user1>; Team 1 executes the GRANT statement. GRANT select ON departments TO <user2>; WHERE user1 is the name of team 1 and user2 is the name of team 2.

7 Query all the rows in your DEPARTMENTS table.

ANSWER:

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.

ANSWER:

Team 1 executes this INSERT statement. INSERT INTO departments(department_id, department_name) VALUES (200, 'Education'); COMMIT; Team 2 executes this INSERT statement. INSERT INTO departments(department_id, department_name) VALUES (210, 'Administration'); COMMIT;

9. Query the USER_TABLES data dictionary to see information about the tables that you own.

ANSWER:

Team 1 creates a synonym named team2. CREATE SYNONYM team2 FOR <user2>.DEPARTMENTS; Team 2 creates a synonym named team1. CREATE SYNONYM team1 FOR <user1>. DEPARTMENTS;

10. Revoke the SELECT privilege on your table from the other team.

ANSWER:

Team 1 executes this SELECT statement. SELECT * FROM team2; Team 2 executes this SELECT statement. SELECT * FROM team1;

11. Remove the row you inserted into the DEPARTMENTS table in step 8 and save the changes.

ANSWER:

DELETE FROM DEPARTMENTS WHERE DepartmentID = 10;

PL/SQL

Ex. No. : 18 **Date:19.11.2024**

Register No.: 221701055 **Name: S Sneha**

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.

ANSWER:

```
DECLARE
  incentive   NUMBER(8,2);
BEGIN
  SELECT salary * 0.12 INTO incentive
  FROM employees
  WHERE employee_id = 110;
DBMS_OUTPUT.PUT_LINE('Incentive  = ' || TO_CHAR(incentive));
```

END;

PROGRAM 2

Write a PL/SQL block to show an invalid case-insensitive reference to a quoted and without quoted user-defined identifier.

ANSWER:

```
DECLARE
  "WELCOME" varchar2(10) := 'welcome'; -- identifier with quotation
BEGIN
  DBMS_Output.Put_Line("Welcome"); --reference to the identifier with quotation and different
case
END;
```

PROGRAM 3

Write a PL/SQL block to adjust the salary of the employee whose ID 122.

Sample table: employees

ANSWER:

```
DECLARE
  salary_of_emp  NUMBER(8,2);
```

```
 PROCEDURE approx_salary (
   emp       NUMBER,
   empsal IN OUT NUMBER,
   addless     NUMBER
 ) IS
 BEGIN
   empsal := empsal + addless;
 END;

BEGIN
 SELECT salary INTO salary_of_emp
 FROM employees
 WHERE employee_id = 122;

 DBMS_OUTPUT.PUT_LINE
  ('Before invoking procedure, salary_of_emp: ' || salary_of_emp);

 approx_salary (100, salary_of_emp, 1000);

 DBMS_OUTPUT.PUT_LINE
  ('After invoking procedure, salary_of_emp: ' || salary_of_emp);
```

```
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.

ANSWER:

```
CREATE OR REPLACE PROCEDURE pri_bool(
  boo_name   VARCHAR2,
  boo_val    BOOLEAN
) IS
BEGIN
  IF boo_val IS NULL THEN
    DBMS_OUTPUT.PUT_LINE( boo_name || ' = NULL');
  ELSIF boo_val = TRUE THEN
    DBMS_OUTPUT.PUT_LINE( boo_name || ' = TRUE');
  ELSE
    DBMS_OUTPUT.PUT_LINE( boo_name || ' = FALSE');
  END IF;
END;
/
```

Copy

Now call the procedure pri_bool:

PL/SQL Code:

```
DECLARE
  PROCEDURE pri_m_and_n (
    m  BOOLEAN,
    n  BOOLEAN
  ) IS
  BEGIN
   pri_bool ('m', m);
   pri_bool ('n', n);
   pri_bool ('m AND n', m AND n);
 END pri_m_and_n;

BEGIN
DBMS_OUTPUT.PUT_LINE('------------- FOR m and n both FALSE ---------------------');
 pri_m_and_n (FALSE, FALSE);
DBMS_OUTPUT.PUT_LINE('------------- FOR m TRUE AND n FALSE ---------------------');
 pri_m_and_n (TRUE, FALSE);
DBMS_OUTPUT.PUT_LINE('------------- FOR m FALSE AND n TRUE ---------------------');
 pri_m_and_n (FALSE, TRUE);
DBMS_OUTPUT.PUT_LINE('------------- FOR m TRUE AND n TRUE ---------------------');
```

```
 pri_m_and_n (TRUE, TRUE);
DBMS_OUTPUT.PUT_LINE('------------- FOR m TRUE AND n NULL ---------------------');
 pri_m_and_n (TRUE, NULL);
DBMS_OUTPUT.PUT_LINE('------------- FOR m FALSE AND n NULL---------------------');
 pri_m_and_n (FALSE, NULL);
DBMS_OUTPUT.PUT_LINE('------------- FOR m NULL AND n TRUE ---------------------');
 pri_m_and_n (NULL, TRUE);
DBMS_OUTPUT.PUT_LINE('------------- FOR m NULL AND n FALSE ---------------------');
 pri_m_and_n (NULL, FALSE);
END;
/
```

PROGRAM 5

Write a PL/SQL block to describe the usage of LIKE operator including wildcard characters and escape character.

ANSWER:

```
DECLARE

  PROCEDURE pat_match (

    test_string   VARCHAR2,
```

```
    pattern      VARCHAR2
  ) IS
  BEGIN
    IF test_string LIKE pattern THEN
      DBMS_OUTPUT.PUT_LINE ('TRUE');
    ELSE
      DBMS_OUTPUT.PUT_LINE ('FALSE');
    END IF;
  END;
BEGIN
  pat_match('Blweate', 'B%a_e');
  pat_match('Blweate', 'B%A_E');
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.

ANSWER:

DECLARE

num_small NUMBER := 8;

```
num_large NUMBER := 5;

num_temp NUMBER;

BEGIN



IF num_small > num_large THEN

num_temp := num_small;

num_small := num_large;

num_large := num_temp;

END IF;



DBMS_OUTPUT.PUT_LINE ('num_small = '||num_small);

DBMS_OUTPUT.PUT_LINE ('num_large = '||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.

ANSWER:

```
DECLARE
  PROCEDURE test1 (
```

```
  sal_achieve  NUMBER,
  target_qty  NUMBER,
  emp_id NUMBER
)
IS
  incentive   NUMBER := 0;
  updated  VARCHAR2(3) := 'No';
BEGIN
  IF sal_achieve > (target_qty + 200) THEN
    incentive := (sal_achieve - target_qty)/4;

    UPDATE emp
    SET salary = salary + incentive
    WHERE employee_id = emp_id;

    updated := 'Yes';
  END IF;
```

```
    DBMS_OUTPUT.PUT_LINE (
      'Table updated?  ' || updated || ', ' ||
      'incentive = ' || incentive || '.'
    );
  END test1;
BEGIN
  test1(2300, 2000, 144);
  test1(3600, 3000, 145);
END;
/
```

PROGRAM 8

Write a PL/SQL procedure to calculate incentive achieved according to the specific sale limit.

ANSWER:

```
DECLARE

 PROCEDURE test1 (sal_achieve  NUMBER)

 IS

  incentive  NUMBER := 0;

 BEGIN

  IF sal_achieve > 44000 THEN

   incentive := 1800;
```

```
   ELSIF sal_achieve > 32000 THEN
     incentive := 800;
   ELSE
     incentive := 500;
   END IF;
 DBMS_OUTPUT.NEW_LINE;
   DBMS_OUTPUT.PUT_LINE (
     'Sale achieved : ' || sal_achieve || ', incentive : ' || incentive || '.'
   );
  END test1;
BEGIN
  test1(45000);
  test1(36000);
  test1(28000);
END;
/
```

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.

ANSWER:

```
SET SERVEROUTPUT ON

DECLARE

   tot_emp NUMBER;

BEGIN

   SELECT Count(*)

   INTO   tot_emp

   FROM   employees e

          join departments d

            ON e.department_id = d.department_id

   WHERE  e.department_id = 50;



   dbms_output.Put_line ('The employees are in the department 50: '

                         ||To_char(tot_emp));
```

```
    IF tot_emp >= 45 THEN
      dbms_output.Put_line ('There are no vacancies in the department 50.');
    ELSE
      dbms_output.Put_line ('There are some vacancies in department 50.');
    END IF;
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.

ANSWER:

```
SET SERVEROUTPUT ON

DECLARE

   tot_emp NUMBER;

BEGIN

   SELECT Count(*)

   INTO   tot_emp

   FROM   employees e

          join departments d

            ON e.department_id = d.department_id

   WHERE  e.department_id = 50;
```

```
   dbms_output.Put_line ('The employees are in the department 50: '
                        ||To_char(tot_emp));


   IF tot_emp >= 45 THEN
     dbms_output.Put_line ('There are no vacancies in the department 50.');
   ELSE
     dbms_output.Put_line ('There are some vacancies in department 50.');
   END IF;
END;
/
```

PROGRAM 11

Write a PL/SQL program to display the employee IDs, names, job titles, hire dates, and salaries of all employees.

ANSWER:

```
DECLARE

v_employee_idemployees.employee_id%TYPE;

v_full_nameemployees.first_name%TYPE;

v_job_idemployees.job_id%TYPE;
```

```
v_hire_dateemployees.hire_date%TYPE;

v_salaryemployees.salary%TYPE;

  CURSOR c_employees IS

    SELECT employee_id, first_name || ' ' || last_name AS full_name, job_id, hire_date,
salary

    FROM employees;

BEGIN

  DBMS_OUTPUT.PUT_LINE('Employee ID | Full Name | Job Title | Hire Date |
Salary');

  DBMS_OUTPUT.PUT_LINE('---------------------------------------------------------------
');

  OPEN c_employees;

  FETCH c_employees INTO v_employee_id, v_full_name, v_job_id, v_hire_date,
v_salary;

  WHILE c_employees%FOUND LOOP

    DBMS_OUTPUT.PUT_LINE(v_employee_id || '       | ' || v_full_name || ' | ' || v_job_id
|| ' | ' || v_hire_date || ' | ' || v_salary);

    FETCH c_employees INTO v_employee_id, v_full_name, v_job_id, v_hire_date,
v_salary;

  END LOOP;
```

```
  CLOSE c_employees;

END;

/
```

PROGRAM 12

Write a PL/SQL program to display the employee IDs, names, and department names of all employees.

ANSWER:

```
DECLARE
  CURSOR emp_cursor IS
    SELECT e.employee_id, e.first_name, m.first_name AS manager_name
    FROM employees e
    LEFT JOIN employees m ON e.manager_id = m.employee_id;
emp_recordemp_cursor%ROWTYPE;
BEGIN
  OPEN emp_cursor;
  FETCH emp_cursor INTO emp_record;
  WHILE emp_cursor%FOUND LOOP
```

```
    DBMS_OUTPUT.PUT_LINE('Employee ID: ' || emp_record.employee_id);

    DBMS_OUTPUT.PUT_LINE('Employee Name: ' || emp_record.first_name);

    DBMS_OUTPUT.PUT_LINE('Manager Name: ' || emp_record.manager_name);

    DBMS_OUTPUT.PUT_LINE('--------------------------');

    FETCH emp_cursor INTO emp_record;

  END LOOP;

  CLOSE emp_cursor;

END;

/
```

PROGRAM 13

Write a PL/SQL program to display the job IDs, titles, and minimum salaries of all jobs.

ANSWER:

```
DECLARE
  CURSOR job_cursor IS
    SELECT job_id, job_title, min_salary
    FROM jobs;
job_recordjob_cursor%ROWTYPE;
BEGIN
  OPEN job_cursor;
```

```
  FETCH job_cursor INTO job_record;

  WHILE job_cursor%FOUND LOOP

    DBMS_OUTPUT.PUT_LINE('Job ID: ' || job_record.job_id);

    DBMS_OUTPUT.PUT_LINE('Job Title: ' || job_record.job_title);

    DBMS_OUTPUT.PUT_LINE('Minimum Salary: ' || job_record.min_salary);

    DBMS_OUTPUT.PUT_LINE('-------------------------');

    FETCH job_cursor INTO job_record;

  END LOOP;

  CLOSE job_cursor;

END;

/
```

PROGRAM 14

Write a PL/SQL program to display the employee IDs, names, and job history start dates of all employees.

ANSWER:

```
DECLARE

v_employee_idemployees.employee_id%TYPE;

v_first_nameemployees.first_name%TYPE;

v_end_datejob_history.end_date%TYPE;

  CURSOR c_employees IS
```

```
  SELECT e.employee_id, e.first_name, jh.end_date
  FROM employees e
  JOIN job_history jh ON e.employee_id = jh.employee_id;
BEGIN
 OPEN c_employees;
 FETCH c_employees INTO v_employee_id, v_first_name, v_end_date;
 WHILE c_employees%FOUND LOOP
  DBMS_OUTPUT.PUT_LINE('Employee ID: ' || v_employee_id);
  DBMS_OUTPUT.PUT_LINE('Employee Name: ' || v_first_name);
  DBMS_OUTPUT.PUT_LINE('End Date: ' || v_end_date);
  DBMS_OUTPUT.PUT_LINE('----------------------');
  FETCH c_employees INTO v_employee_id, v_first_name, v_end_date;
 END LOOP;
 CLOSE c_employees;
END;
```

PROGRAM 15

Write a PL/SQL program to display the employee IDs, names, and job history end dates of all employees.

ANSWER:

DECLARE

v_employee_idemployees.employee_id%TYPE;

```
v_first_nameemployees.first_name%TYPE;

v_end_datejob_history.end_date%TYPE;

  CURSOR c_employees IS

    SELECT e.employee_id, e.first_name, jh.end_date

    FROM employees e

    JOIN job_history jh ON e.employee_id = jh.employee_id;

BEGIN

  OPEN c_employees;

  FETCH c_employees INTO v_employee_id, v_first_name, v_end_date;

  WHILE c_employees%FOUND LOOP

    DBMS_OUTPUT.PUT_LINE('Employee ID: ' || v_employee_id);

    DBMS_OUTPUT.PUT_LINE('Employee Name: ' || v_first_name);

    DBMS_OUTPUT.PUT_LINE('End Date: ' || v_end_date);

    DBMS_OUTPUT.PUT_LINE('----------------------');

    FETCH c_employees INTO v_employee_id, v_first_name, v_end_date;

  END LOOP;

  CLOSE c_employees;

END;
```

Ex. No. : p-8 **Date:19.11.2024**

Register No.: 221701055 **Name: S Sneha**

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;

<u>CREATING THE TABLE 'ITITEMS' AND DISPLAYING THE CONTENTS</u>

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;

```
null_price exception;
begin
select actualprice into price from ititems where itemid=identity;
if price is null then
raise null_price;
else
update ititems set actualprice=actualprice+total where itemid=identity;
end if;
exception
when null_price then
dbms_output.put_line('price is null');
end;
/
Procedure created.

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

SQL> select * from ititems;
 ITEMID   ACTUALPRICE     ORDID     PRODID
---------       -----------       ---------    ---------
     101          2500              500        201
     102          3000             1600        202
     103         4000               600        202
```

<u>PROCEDURE FOR 'IN' PARAMETER – CREATION, EXECUTION</u>

```
SQL> set serveroutput on;

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

```
begin
select actualprice into price from ititems where itemid=a;
dbms_output.put_line('Actual price is ' || price);
if price is null then
dbms_output.put_line('price is null');
end if;
end;
/
Procedure created.

SQL> exec yyy(103);
Actual price is 4000
PL/SQL procedure successfully completed.
```

PROCEDURE FOR 'OUT' PARAMETER – CREATION, EXECUTION

```
SQL> set serveroutput on;

SQL> create procedure zzz (a in number, b out number) is identity number;
begin
select ordid into identity from ititems where itemid=a;
if identity<1000 then
 b:=100;
end if;
end;
/
Procedure created.

SQL> declare
a number;
```

```
b number;
begin
zzz(101,b);
dbms_output.put_line('The value of b is '|| b);
end;
/
The value of b is 100
PL/SQL procedure successfully completed.
```

<u>PROCEDURE FOR ‘INOUT’ PARAMETER – CREATION, EXECUTION</u>

```
SQL> create procedure itit ( a in out number) is
begin
a:=a+1;
end;
/
Procedure created.
```

```
SQL> declare
a number:=7;
begin
itit(a);
dbms_output.put_line(‘The updated value is ‘||a);
end;
/
The updated value is 8
PL/SQL procedure successfully completed.
```

<u>CREATE THE TABLE ‘ITTRAIN’ TO BE USED FOR FUNCTIONS</u>

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

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

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

SQL>select * from ittrain;
     TNO      TFARE
   ---------    ------------
    1001        550
    1002        600
```

PROGRAM FOR FUNCTION AND IT'S EXECUTION

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

Function created.

SQL> set serveroutput on;

SQL> declare
```

```
total number;
begin
total:=aaa (1001);
dbms_output.put_line('Train fare is Rs. '||total);
end;
/
```

Train fare is Rs.550

PL/SQL procedure successfully completed.

Program 1

FACTORIAL OF A NUMBER USING FUNCTION

ANSWER:

```
declare
-- it gives the final answer after computation
fac number :=1;

-- given number n
-- taking input from user
n number := &1;

-- start block
begin

-- start while loop
while n > 0 loop

-- multiple with n and decrease n's value
fac:=n*fac;
n:=n-1;
end loop;
-- end loop

-- print result of fac
dbms_output.put_line(fac);
```

-- end the begin block

end;

INPUT:

5

OUTPUT:

120

Program 2

Write a PL/SQL program using Procedures IN,INOUT,OUT parameters to retrieve the corresponding book information in library

ANSWER:

```
-- Create the procedure
CREATE OR REPLACE PROCEDURE Get_Book_Info(
   p_book_id IN NUMBER,
   p_title OUT VARCHAR2,
   p_author OUT VARCHAR2,
   p_published_year IN OUT NUMBER
) AS
BEGIN
   -- Retrieve book details based on BOOK_ID
   SELECT TITLE, AUTHOR, PUBLISHED_YEAR
   INTO p_title, p_author, p_published_year
   FROM BOOKS
   WHERE BOOK_ID = p_book_id;

   -- Optional: Update published year for demonstration
   p_published_year := p_published_year + 1;

EXCEPTION
   WHEN NO_DATA_FOUND THEN
      -- Handle the case where no book is found
      p_title := NULL;
      p_author := NULL;
      p_published_year := NULL;
      DBMS_OUTPUT.PUT_LINE('No book found with the given BOOK_ID.');
```

```
END;
/
```

Ex. No. : 19 **Date:19.11.2024**

Register No.: 221701055 **Name: S Sneha**

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));
Table created.

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

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

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

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

zzz 13 15500

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

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

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

SQL> delete from itempls where ename='xxx';
delete from itempls where ename='xxx'
            *
ERROR at line 1:
ORA-20010: You cannot do manipulation
ORA-06512: at "STUDENT.ITTRIGG", line 2
ORA-04088: error during execution of trigger 'STUDENT.ITTRIGG'

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

```
update itempls set eid=15 where ename='yyy'
       *
ERROR at line 1:
ORA-20010: You cannot do manipulation
ORA-06512: at "STUDENT.ITTRIGG", line 2
ORA-04088: error during execution of trigger 'STUDENT.ITTRIGG'
```

TO DROP THE CREATED TRIGGER

```
SQL> drop trigger ittrigg;

Trigger dropped.
```

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

```
SQL> create trigger ittriggs before insert or update of salary on itempls for each row
  2  declare
  3  triggsal itempls.salary%type;
  4  begin
  5  select salary into triggsal from itempls where eid=12;
  6  if(:new.salary>triggsal or :new.salary<triggsal) then
  7  raise_application_error(-20100,'Salary has not been changed');
  8  end if;
  9  end;
 10  /

Trigger created.

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

insert into itempls values ('bbb',16,45000)
*
ERROR at line 1:
ORA-04098: trigger 'STUDENT.ITTRIGGS' is invalid and failed re-validation

SQL> update itempls set eid=18 where ename='zzz';
update itempls set eid=18 where ename='zzz'
*
ERROR at line 1:
ORA-04298: trigger 'STUDENT.ITTRIGGS' is invalid and failed re-validation

Cursor for loop

- Explicit cursor
- Implicit cursor

TO CREATE THE TABLE 'SSEMPP'

SQL> create table ssempp(eid number(10), ename varchar2(20), job varchar2(20), sal number (10),dnonumber(5));
Table created.

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

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

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

SQL> insert into ssempp values(6,'vijay','lecturer',18000,11);

1 row created.

SQL> insert into ssempp values(3,'nila','professor',60000,12);

1 row created.

SQL> select * from ssempp;

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

EXTRA PROGRAMS

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

```
SQL> set serveroutput on;
SQL> declare
begin
for emy in (select eid,ename from ssempp)
loop
dbms_output.put_line('Employee id and employee name  are '|| emy.eid ‘and’|| emy.ename);
end loop;
end;
/
Employee id and employee name are 1 and nala
Employee id and employee name are 2 and kala
Employee id and employee name are 5 and ajay
Employee id and employee name are 6 and vijay
Employee id and employee name are 3 and nila

PL/SQL procedure successfully completed.
```

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

```
SQL> set serveroutput on;
SQL> declare
cursor cem is select eid,ename,sal,dno from ssempp where dno=11;
begin
```

```
--open cem;
for rem in cem
loop
update ssempp set sal=rem.sal+5000 where eid=rem.eid;
end loop;
--close cem;
end;
/
```

PL/SQL procedure successfully completed.

SQL> select * from ssempp;

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

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

```
declare
cursor cenl is select eid,sal from ssempp where dno=11;
ecode ssempp.eid%type;
esal empp.sal%type;
begin
open cenl;
```

```
loop
fetch cenl into ecode,esal;
exit when cenl%notfound;
dbms_output.put_line(' Employee code and employee salary are' || ecode ‘and’|| esal);
end loop;
close cenl;
end;

SQL> /
Employee code and employee salary are 1 and 39000
Employee code and employee salary are 5 and 35000
Employee code and employee salary are 6 and 23000

PL/SQL procedure successfully completed.
```

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

```
SQL> declare
county number;
begin
update ssempp set sal=sal+10000 where job='lecturer';
county:= sql%rowcount;
if county > 0 then
dbms_output.put_line('The number of rows are '|| county);
end if;
if sql %found then
dbms_output.put_line('Employee record modification successful');
else if sql%notfound then
```

```
 12  dbms_output.put_line('Employee record is not found');
 13  end if;
 14  end if;
 15  end;
 16  /
The number of rows are 3

Employee record modification successful

PL/SQL procedure successfully completed.

SQL> select * from ssempp;

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

PROGRAMS

TO DISPLAY HELLO MESSAGE

```
SQL> set serveroutput on;
SQL> declare
  2  a varchar2(20);
  3  begin
  4  a:='Hello';
```

```
  5  dbms_output.put_line(a);
  6  end;
  7  /
Hello

PL/SQL procedure successfully completed.
```

TO INPUT A VALUE FROM THE USER AND DISPLAY IT

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

PL/SQL procedure successfully completed.
```

GREATEST OF TWO NUMBERS

```
SQL> set serveroutput on;

SQL> declare
  2  a number(7);
```

```
  3  b number(7);
  4  begin
  5  a:=&a;
  6  b:=&b;
  7  if(a>b) then
  8  dbms_output.put_line (' The grerater of the two is'|| a);
  9  else
 10  dbms_output.put_line (' The grerater of the two is'|| b);
 11  end if;
 12  end;
 13  /
Enter value for a: 5
old   5: a:=&a;
new   5: a:=5;
Enter value for b: 9
old   6: b:=&b;
new   6: b:=9;
The grerater of the two is9

PL/SQL procedure successfully completed.
```

GREATEST OF THREE NUMBERS

```
SQL> set serveroutput on;

SQL> declare
  2  a number(7);
  3  b number(7);
  4  c number(7);
  5  begin
```

```
  6  a:=&a;
  7  b:=&b;
  8  c:=&c;
  9  if(a>b and a>c) then
 10  dbms_output.put_line (' The greatest of the three is ' || a);
 11  else if (b>c) then
 12  dbms_output.put_line (' The greatest of the three is ' || b);
 13  else
 14  dbms_output.put_line (' The greatest of the three is ' || c);
 15  end if;
 16  end if;
 17  end;
 18  /
Enter value for a: 5
old   6: a:=&a;
new   6: a:=5;
Enter value for b: 7
old   7: b:=&b;
new   7: b:=7;
Enter value for c: 1
old   8: c:=&c;
new   8: c:=1;
The greatest of the three is 7

PL/SQL procedure successfully completed.
```

PRINT NUMBERS FROM 1 TO 5 USING SIMPLE LOOP

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

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

PL/SQL procedure successfully completed.
```

PRINT NUMBERS FROM 1 TO 4 USING WHILE LOOP

```
SQL> set serveroutput on;

SQL> declare
a number:=1;
begin
while(a<5)
loop
dbms_output.put_line (a);
```

```
  7  a:=a+1;
  8  end loop;
  9  end;
 10  /
1
2
3
4
PL/SQL procedure successfully completed.
```

PRINT NUMBERS FROM 1 TO 5 USING FOR LOOP

```
SQL> set serveroutput on;

SQL> declare
  2  a number:=1;
  3  begin
  4  for a in 1..5
  5  loop
  6  dbms_output.put_line (a);
  7  end loop;
  8  end;
  9  /
1
2
3
4
5
PL/SQL procedure successfully completed.
```

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

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

```
a number:=1;
begin
for a in reverse 1..5
loop
dbms_output.put_line (a);
end loop;
end;
/
5
4
3
2
1
PL/SQL procedure successfully completed.
```

TO CALCULATE AREA OF CIRCLE

```
SQL> set serveroutput on;
SQL> declare
pi constant number(4,2):=3.14;
a number(20);
r number(20);
begin
r:=&r;
a:= pi* power(r,2);
dbms_output.put_line (' The area of circle is ' || a);
end;
/
Enter value for r: 2
old   6: r:=&r;
new   6: r:=2;
The area of circle is 13
```

PL/SQL procedure successfully completed.

TO CREATE SACCOUNT TABLE

```
SQL> create table saccount  ( accno number(5), name varchar2(20), bal number(10));
Table created.
SQL> insert into saccount values ( 1,'mala',20000);
1 row created.
SQL> insert into saccount values (2,'kala',30000);
1 row created.
SQL> select * from saccount;

   ACCNO NAME                        BAL
--------- -------------------- ---------
        1 mala                     20000
        2 kala                     30000
SQL> set serveroutput on;
SQL> declare
a_bal number(7);
a_no varchar2(20);
debit number(7):=2000;
minamt number(7):=500;
begin
a_no:=&a_no;
select bal into a_bal from saccount where accno= a_no;
a_bal:= a_bal-debit;
if (a_bal > minamt) then
update  saccount  set bal=bal-debit where accno=a_no;
end if;
end;

/
```

```
Enter value for a_no: 1
old   7: a_no:=&a_no;
new   7: a_no:=1;

PL/SQL procedure successfully completed.
SQL> select * from saccount;
  ACCNO NAME                   BAL
--------- -------------------- ---------
        1 mala                 18000
        2 kala                 30000
```

<u>TO CREATE TABLE SROUTES</u>

```
SQL> create table sroutes ( rno number(5), origin varchar2(20), destination varchar2(20), fare numbe
r(10), distance number(10));
Table created.
SQL> insert into sroutes values ( 2, 'chennai', 'dindugal', 400,230);
1 row created.
SQL> insert into sroutes values ( 3, 'chennai', 'madurai', 250,300);
1 row created.
SQL> insert into sroutes values ( 6, 'thanjavur', 'palani', 350,370);
1 row created.
SQL> select * from sroutes;

      RNO ORIGIN               DESTINATION               FARE  DISTANCE
--------- -------------------- -------------------- --------- ---------
        2 chennai              dindugal                   400       230
        3 chennai              madurai                    250       300
        6 thanjavur            palani                     350       370

SQL> set serveroutput on;
```

```
SQL> declare
  2  route sroutes.rno % type;
  3  fares sroutes.fare % type;
  4  dist sroutes.distance % type;
  5  begin
  6  route:=&route;
  7  select fare, distance into fares , dist from sroutes where rno=route;
  8  if (dist < 250) then
  9  update sroutes set fare=300 where rno=route;
 10  else if dist between 250 and 370 then
 11  update sroutes set fare=400 where rno=route;
 12  else if (dist > 400) then
 13  dbms_output.put_line('Sorry');
 14  end if;
 15  end if;
 16  end if;
 17  end;
 18  /
Enter value for route: 3
old   6: route:=&route;
new   6: route:=3;

PL/SQL procedure successfully completed.

SQL> select * from sroutes;

      RNO ORIGIN               DESTINATION                FARE  DISTANCE
--------- -------------------- -------------------- --------- ---------
        2 chennai              dindugal                   400       230
```

```
3 chennai        madurai           400    300
6 thanjavur      palani            350    370
```

TO CREATE 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  /
```

```
PL/SQL procedure successfully completed.

SQL> select * from scalculate;
  RADIUS      AREA
--------- ---------
        3     28.26
        4     50.24
        5      78.5
        6    113.04
        7    153.86
```

<u>TO CALCULATE FACTORIAL OF A GIVEN NUMBER</u>

```
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  /
Enter value for i: 5
old   5: i:=&i;
new   5: i:=5;
The value is 120
```

PL/SQL procedure successfully completed.

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.

ANSWER:

```
-- Create the 'departments' table

CREATE TABLE departments (

department_id NUMBER PRIMARY KEY,

department_nameVARCHAR2(50)

);


-- Create the 'employees' table with a foreign key reference

CREATE TABLE employees (

employee_id NUMBER PRIMARY KEY,

first_nameVARCHAR2(50),

last_nameVARCHAR2(50),

department_id NUMBER,

   CONSTRAINT fk_department FOREIGN KEY (department_id) REFERENCES
departments (department_id)
```

```
);


-- Create a trigger to enforce referential integrity
CREATE OR REPLACE TRIGGER prevent_parent_deletion
BEFORE DELETE ON departments
FOR EACH ROW
DECLARE
v_count NUMBER;
BEGIN
   -- Check if there are any associated child records
   SELECT COUNT(*) INTO v_count FROM employees WHERE department_id =
:OLD.department_id;


   -- If child records exist, raise an error
   IF v_count> 0 THEN
      RAISE_APPLICATION_ERROR(-20001, 'Cannot delete department with
associated employees.');
   END IF;
END;
```

/

Copy

Let's see how the trigger functions:

PL/SQL Code:

```
-- Insert sample department and employee records

INSERT INTO departments (department_id, department_name) VALUES (1, 'Sales');

INSERT INTO employees (employee_id, first_name, last_name, department_id)
VALUES (1, 'Ben', 'Turner', 1);


-- Try to delete the department with associated employees

DELETE FROM departments WHERE department_id = 1; -- This will raise an error


-- Delete the employee first

DELETE FROM employees WHERE employee_id = 1;


-- Now, delete the department

DELETE FROM departments WHERE department_id = 1; -- This will work
```

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.

ANSWER:

-- Create the 'products' table

CREATE TABLE products (

product_id NUMBER PRIMARY KEY,

```
product_nameVARCHAR2(50)
);
-- Create a trigger to check for duplicate values
CREATE OR REPLACE TRIGGER prevent_duplicates
BEFORE INSERT ON products
FOR EACH ROW
DECLARE
v_count NUMBER;
BEGIN
   -- Check if the new product_name already exists
   SELECT COUNT(*) INTO v_count FROM products WHERE product_name =
:NEW.product_name;
      -- If duplicate value found, raise an error
   IF v_count> 0 THEN
      RAISE_APPLICATION_ERROR(-20001, 'Product name already exists.');
   END IF;
END;
/
```

Copy

Let's see how trigger can be used:

PL/SQL Code:

-- Insert a product

INSERT INTO products (product_id, product_name) VALUES (1, 'Widget');

-- Try to insert a product with a duplicate name

INSERT INTO products (product_id, product_name) VALUES (2, 'Widget'); -- This will raise an error

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.

ANSWER:

```
-- Create the orders table

CREATE TABLE orders (

order_id NUMBER PRIMARY KEY,

customer_id NUMBER,

order_amount NUMBER

);



-- Create a trigger to restrict total order amount

CREATE OR REPLACE TRIGGER check_order_amount

BEFORE INSERT ON orders

FOR EACH ROW
```

```
DECLARE
total_amount NUMBER;
max_threshold NUMBER := 10000; -- Change this to your desired threshold
BEGIN
    -- Calculate the current total order amount for the customer
    SELECT NVL(SUM(order_amount), 0) INTO total_amount
    FROM orders
    WHERE customer_id= :NEW.customer_id;

    -- Check if inserting the new row will exceed the threshold
    IF total_amount+ :NEW.order_amount>max_threshold THEN
        RAISE_APPLICATION_ERROR(-20001, 'Total order amount exceeds the
threshold.');
    END IF;
END;
/
```

Copy

Let's see how the trigger can be functions:

PL/SQL Code:

```
-- Inserting rows that don't exceed the threshold

INSERT INTO orders (order_id, customer_id, order_amount) VALUES (1, 101, 5000);

INSERT INTO orders (order_id, customer_id, order_amount) VALUES (2, 101, 3000);

INSERT INTO orders (order_id, customer_id, order_amount) VALUES (3, 102, 8000);


-- Attempting to insert a row that would exceed the threshold

-- This should raise an error and prevent the insertion

BEGIN

   INSERT INTO orders (order_id, customer_id, order_amount) VALUES (4, 102,
5000);

EXCEPTION

   WHEN OTHERS THEN

      DBMS_OUTPUT.PUT_LINE('Error: ' || SQLERRM);

END;

/
```

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.

ANSWER:

```
-- Create the employees table

CREATE TABLE employees (
```

```
employee_id NUMBER PRIMARY KEY,

employee_nameVARCHAR2(100),

salary NUMBER

);


-- Create the salary_audit table to store changes

CREATE TABLE salary_audit (

audit_id NUMBER PRIMARY KEY,

employee_id NUMBER,

old_salary NUMBER,

new_salary NUMBER,

change_date TIMESTAMP

);


-- Create a sequence for generating unique audit IDs

CREATE SEQUENCE seq_salary_audit START WITH 1 INCREMENT BY 1;


-- Create a trigger to capture changes in salary
```

```
CREATE OR REPLACE TRIGGER salary_change_audit

AFTER UPDATE ON employees

FOR EACH ROW

WHEN (NEW.salary<>OLD.salary) -- Only capture changes in the salary column

DECLARE

v_audit_id NUMBER;

BEGIN

   -- Generate a unique audit ID

   SELECT seq_salary_audit.NEXTVAL INTO v_audit_id FROM DUAL;



   -- Insert the change details into the audit table

   INSERT INTO salary_audit (audit_id, employee_id, old_salary, new_salary,
change_date)

   VALUES (v_audit_id, :OLD.employee_id, :OLD.salary, :NEW.salary,
SYSTIMESTAMP);

END;

/
```

Copy

Let's see how the trigger functions:

PL/SQL Code:

```
-- Inserting a sample employee record

INSERT INTO employees (employee_id, employee_name, salary)

VALUES (1, 'John Doe', 50000);



-- Updating the salary of the employee

UPDATE employees SET salary = 60000 WHERE employee_id = 1;
```

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.

ANSWER:

```
-- Create Employee table

CREATE TABLE Employee (

emp_id NUMBER PRIMARY KEY,

emp_nameVARCHAR2(100),

emp_salary NUMBER

);

-- Create Audit_Log table

CREATE TABLE Audit_Log (

log_id NUMBER PRIMARY KEY,

table_nameVARCHAR2(100),

activity_typeVARCHAR2(20),
```

```
activity_date TIMESTAMP,
user_idVARCHAR2(50)
);
CREATE SEQUENCE Audit_Log_Seq START WITH 1 INCREMENT BY 1;
CREATE OR REPLACE TRIGGER Employee_Audit_Trigger
AFTER INSERT OR UPDATE OR DELETE ON Employee
FOR EACH ROW
DECLARE
v_activity_typeVARCHAR2(20);
BEGIN
   IF INSERTING THEN
v_activity_type := 'INSERT';
   ELSIF UPDATING THEN
v_activity_type := 'UPDATE';
   ELSIF DELETING THEN
v_activity_type := 'DELETE';
   END IF;
   INSERT INTO Audit_Log (log_id, table_name, activity_type, activity_date, user_id)
```

```
   VALUES (Audit_Log_Seq.NEXTVAL, 'Employee', v_activity_type,
SYSTIMESTAMP, USER);

END;

-- Insert a new employee

INSERT INTO Employee (emp_id, emp_name, emp_salary)

VALUES (1, 'John Doe', 50000);

-- Update an employee's salary

UPDATE Employee SET emp_salary = 55000 WHERE emp_id = 1;

-- Delete an employee

DELETE FROM Employee WHERE emp_id = 1;

SELECT * FROM Audit_Log;
```

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.

ANSWER:

```
-- Create Sales table

CREATE TABLE Sales (

sale_id NUMBER PRIMARY KEY,

sale_date DATE,

amount NUMBER,

running_total NUMBER
```

```
);


-- Create Trigger

CREATE OR REPLACE TRIGGER Update_Running_Total

BEFORE INSERT ON Sales

FOR EACH ROW

BEGIN

IF :NEW.running_total IS NULL THEN

    SELECT NVL(MAX(running_total), 0) + :NEW.amount

INTO :NEW.running_total

    FROM Sales;

  ELSE

:NEW.running_total := :NEW.running_total + :NEW.amount;

  END IF;

END;

/
```

Copy

Let's see how the trigger functions

PL/SQL Code:

```
-- Insert sample data

INSERT INTO Sales (sale_id, sale_date, amount) VALUES (1, TO_DATE('2023-08-01', 'YYYY-MM-DD'), 100);

INSERT INTO Sales (sale_id, sale_date, amount) VALUES (2, TO_DATE('2023-08-02', 'YYYY-MM-DD'), 200);

INSERT INTO Sales (sale_id, sale_date, amount) VALUES (3, TO_DATE('2023-08-03', 'YYYY-MM-DD'), 150);


-- Query the Sales table to see the running total

SELECT * FROM Sales;
```

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.

ANSWER:

```
-- Create Products table
CREATE TABLE Products (
product_id NUMBER PRIMARY KEY,
product_nameVARCHAR2(100),
stock_quantity NUMBER
);

-- Create Orders table
CREATE TABLE Orders (
order_id NUMBER PRIMARY KEY,
```

```
product_id NUMBER,
order_quantity NUMBER
);

-- Create Trigger to validate availability before placing an order
CREATE OR REPLACE TRIGGER Validate_Order_Availability
BEFORE INSERT ON Orders
FOR EACH ROW
DECLARE
v_current_stock NUMBER;
v_pending_orders NUMBER;
BEGIN
   -- Get current stock for the product
   SELECT stock_quantity INTO v_current_stock
   FROM Products
   WHERE product_id= :NEW.product_id;

   -- Get total quantity of pending orders for the product
   SELECT NVL(SUM(order_quantity), 0) INTO v_pending_orders
   FROM Orders
   WHERE product_id= :NEW.product_id;

   -- Calculate total available quantity (stock - pending orders)
   IF v_current_stock - v_pending_orders - :NEW.order_quantity< 0 THEN
      RAISE_APPLICATION_ERROR(-20001, 'Insufficient stock for the order');
   END IF;
```

END;

/

Copy

Let’s see how the trigger functions

PL/SQL Code:

```
-- Insert sample data into Products table
INSERT INTO Products (product_id, product_name, stock_quantity)
VALUES (1, 'Product A', 100);

-- Attempt to place an order with insufficient stock
INSERT INTO Orders (order_id, product_id, order_quantity)
VALUES (1, 1, 150);
-- This should fail due to insufficient stock
-- Place an order within available stock
INSERT INTO Orders (order_id, product_id, order_quantity)
VALUES (2, 1, 50);
-- This should succeed
-- Query the Orders table to see the placed orders
SELECT * FROM Orders;
```

MONGO DB

Ex. No. : 20 **Date:19.11.2024**

Register No.: 221701055 **Name: S Sneha**

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

ANSWER:

use myDatabase

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

ANSWER:

show dbs

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

ANSWER:

use myNewDatabase

You can change or create a new database by typing use then the name of the database.

Create Collection using mongosh

ANSWER:

```
db.createCollection("myCollection");
```

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:19.11.2024**

Register No.: 221701055 **Name: S Sneha**

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'.

ANSWER:

```
db.restaurants.find(
{$or: [
  {name: /^Wil/},
  {"$and": [
       {"cuisine" : {$ne :"American "}},
       {"cuisine" : {$ne :"Chinees"}}
   ]}
]}
,{"restaurant_id" : 1,"name":1,"borough":1,"cuisine" :1}
);
```

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..

ANSWER:

```
db.restaurants.find(
```

```
{ "grades.1.date": ISODate("2014-08-11T00:00:00Z"),
  "grades.1.grade":"A" ,
  "grades.1.score" : 9
},
 {"restaurant_id" : 1,"name":1,"grades":1}
);
```

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".

ANSWER:

```
db.restaurants.find(
  { "grades.1.date": ISODate("2014-08-11T00:00:00Z"),
    "grades.1.grade":"A" ,
    "grades.1.score" : 9
  },
   {"restaurant_id" : 1,"name":1,"grades":1}
);
```

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..**

ANSWER:

```
db.restaurants.find(
    {
      "address.coord.1": {$gt : 42, $lte : 52}
    },
      {"restaurant_id" : 1,"name":1,"address":1,"coord":1}
  );
```

5. **Write a MongoDB query to arrange the name of the restaurants in ascending order along with all the columns.**

ANSWER:

```
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.**

ANSWER:

```
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.

ANSWER:

```
db.restaurants.find().sort(
                {"cuisine":1,"borough" : -1,}
               );
```

8. Write a MongoDB query to know whether all the addresses contains the street or not.

ANSWER:

```
db.restaurants.find(
            {"address.street" :
               { $exists : true }
            }
          );
```

9. **Write a MongoDB query which will select all documents in the restaurants collection where the coord field value is Double.**

<u>ANSWER:</u>

```
db.restaurants.find(
        {"address.coord" :
          {$type : 1}
        }
       );
```

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.**

<u>ANSWER:</u>

```
db.restaurants.find(
         {"grades.score" :
           {$mod : [7,0]}
         },
           {"restaurant_id" : 1,"name":1,"grades":1}
```

```
);
```

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.**

ANSWER:

```
db.restaurants.find(
        { name :
          { $regex : "mon.*", $options: "i" }
        },
          {
            "name":1,
            "borough":1,
            "address.coord":1,
            "cuisine" :1
          }
        );
```

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.

ANSWER:

```
db.restaurants.find(
        { name :
          { $regex : /^Mad/i, }
        },
          {
            "name":1,
            "borough":1,
            "address.coord":1,
            "cuisine" :1
           }
        );
```

13 **Write a MongoDB query to find the restaurants that have at least one grade with a score of less than 5.**

ANSWER:

```
db.restaurants.find({ "grades.score": { $lt: 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.

ANSWER:

```
db.restaurants.find({ "grades.score": { $lt: 5 }, "borough": "Manhattan" })
```

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.

ANSWER:

```
db.restaurants.find({
 $and: [
   {
     $or: [
       {borough: "Manhattan"},
       {borough: "Brooklyn"}
     ]
   },
   {
```

```
    "grades.score": { $lt: 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.**

ANSWER:

```
db.restaurants.find({
 $and: [
{ $or: [{ borough: "Manhattan" }, { borough: "Brooklyn" }] },
{ "grades.score": { $lt: 5 } },
{ cuisine: { $ne: "American" } }
 ]
})
```

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.**

ANSWER:

```
db.restaurants.find({
   $and: [
      {
         $or: [
            {borough: "Manhattan"},
            {borough: "Brooklyn"}
         ]
      },
      {
         $nor: [
            {cuisine: "American"},
            {cuisine: "Chinese"}
         ]
      },
      {
grades: {
            $elemMatch: {
score: { $lt: 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**

ANSWER:

```
db.restaurants.find({
  $and: [
    {"grades.score": 2},
    {"grades.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.

ANSWER:

```
db.restaurants.find({
  $and: [
```

```
  {"grades.score": 2},
  {"grades.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.

ANSWER:

```
db.restaurants.find({
 $and: [
  { "grades": { $elemMatch: { score: 2 } } }, // Match a grade with score 2
  { "grades": { $elemMatch: { score: 6 } } } // Match a grade with score 6
 ],
 borough: { $in: ["Manhattan", "Brooklyn"] }   // Match borough of Manhattan or Brooklyn
});
```

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 America.

ANSWER:

```
db.restaurants.find({
 $and: [
   {
     $or: [
{ borough: "Manhattan" },
{ borough: "Brooklyn" }
     ]
   },
   {
     $or: [
{ "grades.score": 2 },
{ "grades.score": 6 }
     ]
   },
   {
```

```
cuisine: { $ne: "American" }
   }
 ]
})
```

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.**

ANSWER:

```
db.restaurants.find({
  $and: [
    {
      $or: [
{ "grades.score": 2 },
{ "grades.score": 6 }
      ]
    },
    {
      $or: [
```

```
{ borough: "Manhattan" },
{ borough: "Brooklyn" }
      ]
    },
    {
      $nor: [
{ cuisine: "American" },
{ cuisine: "Chinese" }
      ]
    }
  ]
})
```

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.

ANSWER:

```
db.restaurants.find({ $or: [ { "grades.score": 2 }, { "grades.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.

ANSWER:

db.movies.find({ year: 1893 })

2.Find all movies with full information from the 'movies' collection that have a runtime greater than 120 minutes.

ANSWER:

db.movies.find({ "runtime": { $gt: 120 } })

3. Find all movies with full information from the 'movies' collection that have "Short" genre.

ANSWER:

db.movies.find({ "genres": "Short" })

4. Retrieve all movies from the 'movies' collection that were directed by "William K.L. Dickson" and include complete information for each movie.

ANSWER:

db.movies.find({ directors: "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.

ANSWER:

db.movies.find({ countries: "USA" })

6. Retrieve all movies from the 'movies' collection that have complete information and are rated as "UNRATED".

ANSWER:

```
db.movies.find({ rated: "UNRATED" })
```

7. Retrieve all movies from the 'movies' collection that have complete information and have received more than 1000 votes on IMDb.

ANSWER:

```
db.movies.find({ "imdb.votes": { $gt: 1000 } })
```

8. Retrieve all movies from the 'movies' collection that have complete information and have an IMDb rating higher than 7.

ANSWER:

```
db.movies.find({ "imdb.rating": { $gt: 7 } })
```

9. Retrieve all movies from the 'movies' collection that have complete information and have a viewer rating higher than 4 on Tomatoes.

ANSWER:

```
db.movies.find({

  "tomatoes.viewer.rating": { $gt: 4 }

})
```

10. Retrieve all movies from the 'movies' collection that have received an award.

ANSWER:

```
db.movies.find(
{'awards.wins': {$gt: 0}
})
```

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.

ANSWER:

```
db.movies.find({

  "awards.nominations": { $gt: 0 }

}, {
```

```
  "title": 1,
  "languages": 1,
  "released": 1,
  "directors": 1,
  "writers": 1,
  "awards": 1,
  "year": 1,
  "genres": 1,
  "runtime": 1,
  "cast": 1,
  "countries": 1
})
```

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".

ANSWER:

```
db.movies.find({
  "cast": "Charles Kayser"
}, {
  "title": 1,
  "languages": 1,
  "released": 1,
  "directors": 1,
  "writers": 1,
  "awards": 1,
  "year": 1,
  "genres": 1,
  "runtime": 1,
  "cast": 1,
  "countries": 1
})
```

Copy

13. Retrieve all movies with title, languages, released, directors, writers, countries from the 'movies' collection in MongoDB that released on May 9, 1893.

ANSWER:

```
db.movies.find({
released: ISODate("1893-05-09T00:00:00.000Z")
},
{
title: 1,
languages: 1,
released: 1,
directors: 1,
writers: 1,
countries: 1
})
```

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.

ANSWER:

```
db.movies.find(

{ title: { $regex: /scene/i } },

{ title: 1, languages: 1, released: 1, directors: 1, writers: 1, countries: 1 }

)
```