

**RAJALAKSHMI ENGINEERING
COLLEGE RAJALAKSHMI NAGAR, THANDALAM – 602
105**



**CS23332 DATABASE MANAGEMENT
SYSTEMS LAB**

Laboratory Record Note Book

Name: KRITHIKA B

Year / Branch /Section : II CSE C

University Register No. : 2116230701156

College Roll No. : 230701156

Semester : III

Academic Year : 2024

CS23332 DATABASE MANAGEMENT SYSTEMS

NAME	KRITHIKA B
ROLL No	230701156
DEPT	CSE
SEC	C

SOL 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.ROLLBA CK 3.SAVEPOI NT	Manages the changes made by DML statements. Changes to the data can be grouped together into logical transactions.
DCL	1.GRANT 2.REVOKE	Gives or removes access rights to both the oracle database and the structures within it.

DATA TYPES

1. Character Data types:

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

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

- 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.

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‘.

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. **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.

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

To Add a New Row

INSERT Statement

Syntax

INSERT INTO table_name VALUES (column1 values, column2 values, ..., column 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';

Ex.No.: 1	CREATION OF BASE TABLE AND DML OPERATIONS
Date:	

AIM:

ALGORITHM:

STEP-1: Start.

STEP-2: Create a base Table

Syntax:

CREATE TABLE <table name> (column1 type, column2 type, ...);

STEP-3: Describe the Table structure

Syntax:

DESC <table name>

STEP-4: Add a new row to a Table using INSERT statement.

Syntax:

- INSERT INTO <table name> VALUES (value1, value2..);

- INSERT INTO <table name> (column1, column2..)
VALUES (value1, value2..);

- INSERT INTO <table name> VALUES (&column1, '&column');

STEP-5: Modify the existing rows in the base Table with UPDATE statement.

Syntax:

UPDATE <table name> SET column1=value, column2 = 'value'
WHERE (condition);

STEP-6: Remove the existing rows from the Table using DELETE
statement. Syntax:

DELETE FROM <table name> WHERE <condition>;

STEP-7: Perform a Query using SELECT statement.

Syntax:

```
SELECT [DISTINCT] {*,<column1,..>} FROM <table name>
WHERE <condition>;
```

STEP-8: The truncate command deletes all rows from the table. Only the structure of the table remains.

Syntax:

```
TRUNCATE TABLE <table name>;
```

STEP-9: Alter the existing table using ALTER statement.

Syntax:

Add Column:

```
ALTER TABLE <table name> ADD (column data type
[DEFAULTexpr][,column data type]);
```

Modify Column:

```
ALTER TABLE <table name> MODIFY (column data type
[DEFAULT expr], [,column data type]);
```

Drop Column:

```
ALTER TABLE <table name> DROP COLUMN <column name>;
```

STEP-10: To drop the entire table using DROP statement.

Syntax:

```
DROP TABLE <table name>;
```

STEP-11: Exit.

1. Create MY_EMPLOYEE table with the following structure

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

```
CREATE TABLE MY_EMPLOYEE ( ID NUMBER(4) NOT NULL, Last_name
VARCHAR2(25), First_name VARCHAR2(25), Userid VARCHAR2(25), Salary
```

NUMBER(9,2), CONSTRAINT pk_employee PRIMARY KEY (ID));

2. Add the first and second rows data to MY_EMPLOYEE table from the following sample data.

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

INSERT INTO MY_EMPLOYEE (ID, Last_name, First_name, Userid, Salary) VALUES (1, 'Patel', 'Ralph', 'rpatel', 895); INSERT INTO MY_EMPLOYEE (ID, Last_name, First_name, Userid, Salary) VALUES (2, 'Dancs', 'Betty', 'bdancs', 860);

3. Display the table with values.

SELECT * FROM MY_EMPLOYEE;

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

INSERT INTO MY_EMPLOYEE (ID, Last_name, First_name, Userid, Salary) VALUES (3, 'Biri', 'Ben', NULL, 1100); INSERT INTO MY_EMPLOYEE (ID, Last_name, First_name, Userid, Salary) VALUES (4, 'Newman', 'Chad', NULL, 750);

UPDATE MY_EMPLOYEE SET Userid = LOWER(CONCAT(SUBSTR(First_name, 1, 1), SUBSTR>Last_name, 1, 7))) WHERE ID = 3 OR ID = 4;

5. Delete Betty dancs from MY_EMPLOYEE table.

DELETE FROM MY_EMPLOYEE WHERE First_name = 'Betty' AND Last_name = 'Dancs';

6. Empty the fourth row of the emp table.

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

7. Make the data additions permanent.

COMMIT;

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

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

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

UPDATE MY_EMPLOYEE SET Salary = 1000 WHERE Salary < 900;

Evaluation Procedure	Marks awarded
Query(5)	
Execution (5)	
Viva(5)	
Total (15)	
Faculty Signature	

Ex.No.: 2	DATA MANIPULATIONS	
Date:		

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

```

CREATE TABLE EMPLOYEES ( Employee_id NUMBER(6) NOT NULL, First_Name
VARCHAR2(20), Last_Name VARCHAR2(25) NOT NULL, Email VARCHAR2(25) NOT NULL,
Phone_Number VARCHAR2(20), Hire_date DATE NOT NULL, Job_id VARCHAR2(10) NOT
NULL, Salary NUMBER(8,2), Commission_pct NUMBER(2,2), Manager_id NUMBER(6),
Department_id NUMBER(4), CONSTRAINT pk_employee_id PRIMARY KEY (Employee_id) );
INSERT INTO EMPLOYEES VALUES (101, 'John', 'Doe', 'jdoe@example.com', '1234567890',
TO_DATE('2022-06-15', 'YYYY-MM-DD'), 'IT_PROG', 5000, NULL, 100, 60); INSERT INTO
EMPLOYEES VALUES (102, 'Jane', 'Austin', 'jaustin@example.com', '0987654321',
TO_DATE('2022-08-20', 'YYYY-MM-DD'), 'HR_MAN', 4800, NULL, 101, 70); INSERT INTO
EMPLOYEES VALUES (103, 'Mark', 'Smith', 'msmith@example.com', '1230984567',
TO_DATE('2023-01-10', 'YYYY-MM-DD'), 'SA REP', 4600, 0.10, 100, 80); INSERT INTO
EMPLOYEES VALUES (104, 'Chad', 'Newman', 'cnewman@example.com', '7896541230',
TO_DATE('2021-11-03', 'YYYY-MM-DD'), 'FI_MGR', 6000, NULL, 102, 60); INSERT INTO
EMPLOYEES VALUES (105, 'Betty', 'Austin', 'baustin@example.com', '9874563210',
TO_DATE('2020-12-25', 'YYYY-MM-DD'), 'HR_CLERK', 3900, NULL, 101, 70);

```

Job_id	Not null	Varchar(10)
Salary		Number(8,2)
Commission_pct		Number(2,2)
Manager_id		Number(6)
Department_id		Number(4)

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

```
SELECT Employee_id, First_Name, Last_Name, Salary FROM EMPLOYEES;
```

(b) List out the employees who works under manager 100

```
SELECT Employee_id, First_Name, Last_Name FROM EMPLOYEES WHERE Manager_id = 100;
```

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

```
SELECT First_Name, Last_Name FROM EMPLOYEES WHERE Salary >= 4800;
```

(d) List out the employees whose last name is 'AUSTIN'

```
SELECT First_Name, Last_Name FROM EMPLOYEES WHERE Last_Name = 'AUSTIN';
```

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

```
SELECT First_Name, Last_Name FROM EMPLOYEES WHERE Department_id IN (60, 70, 80);
```

(f) Display the unique Manager_Id.

```
SELECT DISTINCT Manager_id FROM EMPLOYEES;
```

Create an Emp table with the following fields: (EmpNo, EmpName, Job,Basic, DA, HRA,PF,

GrossPay, NetPay) (Calculate DA as 30% of Basic and HRA as 40% of Basic)

```
CREATE TABLE EMP ( EmpNo NUMBER(6), EmpName VARCHAR2(25), Job  
VARCHAR2(20), Basic NUMBER(8,2), DA NUMBER(8,2), HRA NUMBER(8,2), PF  
NUMBER(8,2), GrossPay NUMBER(8,2), NetPay NUMBER(8,2), Department_id  
NUMBER(4) )
```

(a) Insert Five Records and calculate GrossPay and NetPay.

```
INSERT INTO EMP (EmpNo, EmpName, Job, Basic, Department_id) VALUES (1, 'John  
Doe', 'Manager', 5000, 60);  
INSERT INTO EMP (EmpNo, EmpName, Job, Basic, Department_id) VALUES (2, 'Jane  
Austin', 'Clerk', 4000, 70);  
INSERT INTO EMP (EmpNo, EmpName, Job, Basic, Department_id) VALUES (3, 'Mark  
Smith', 'Sales', 3500, 80);  
INSERT INTO EMP (EmpNo, EmpName, Job, Basic, Department_id) VALUES (4, 'Chad  
Newman', 'Manager', 6000, 60);  
INSERT INTO EMP (EmpNo, EmpName, Job, Basic, Department_id) VALUES (5, 'Betty  
Austin', 'HR', 3900, 70);  
UPDATE EMP SET DA = 0.30 * Basic, HRA = 0.40 * Basic, PF = 0.12 * Basic;  
UPDATE EMP SET GrossPay = Basic + DA + HRA;  
UPDATE EMP SET NetPay = GrossPay - PF;
```

(b) Display the employees whose Basic is lowest in each department.

```
SELECT * FROM EMP e WHERE Basic = ( SELECT MIN(Basic) FROM EMP WHERE  
Department_id = e.Department_id );
```

(c) If Net Pay is less than

```
SELECT EmpName, NetPay FROM EMP WHERE NetPay < 7500;
```

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

CREATE TABLE DEPT (ID NUMBER(7), NAME VARCHAR2(25), CONSTRAINT pk_dept PRIMARY KEY (ID));

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

Column name	ID	LAST_NAME	FIRST_NAME	DEPT_ID

Key Type				
Nulls/Unique				
FK table				
FK column				
Data Type	Number	Varchar2	Varchar2	Number
Length	7	25	25	7

CREATE TABLE EMP (ID NUMBER(7), LAST_NAME VARCHAR2(25), FIRST_NAME VARCHAR2(25), DEPT_ID NUMBER(7), CONSTRAINT pk_emp PRIMARY KEY (ID));

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

ALTER TABLE EMP MODIFY LAST_NAME VARCHAR2(50);

4 Create the EMPLOYEES2 table based on the structure of EMPLOYEES table. Include Only the Employee_id, First_name, Last_name, Salary and Dept_id coloumns. Name the columns Id, First_name, Last_name, salary and Dept_id respectively.

CREATE TABLE EMPLOYEES2 AS SELECT Employee_id AS Id, First_Name, Last_Name, Salary, Department_id AS Dept_id FROM EMPLOYEES;

5 Drop the EMP table.

DROP TABLE EMP

6 Rename the EMPLOYEES2 table as EMP.

ALTER TABLE EMPLOYEES2 RENAME TO EMP;

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

COMMENT ON TABLE DEPT IS 'Department Table'; COMMENT ON TABLE EMP IS 'Employees Table'; DESC DEPT; DESC EMP;

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

ALTER TABLE EMP DROP COLUMN First_Name; DESC EMP;

Evaluation Procedure	Marks awarded
Query(5)	
Execution (5)	
Viva(5)	
Total (15)	
Faculty Signature	

Ex.No.: 3	WRITING BASIC SQL SELECT STATEMENTS	
Date:		

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 SOL 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—Suppress the duplicates. Alias—
gives selected columns different headings.

Example: 1

```
SELECT * FROM departments;
```

Example: 2

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

Writing SOL Statements

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

Using Arithmetic Expressions

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

Example:1

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

Example:2

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

The statement is not same as

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

Example:3

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

Example:4

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

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

Example:1

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

Example: 2

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

Concatenation Operator

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

Example:

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

Using Literal Character String

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

Example:

```
SELECT last_name||‘is a’||job_id AS —EMPLOYEES JOB| FROM
```

employees; Eliminating Duplicate Rows

- Using DISTINCT keyword.

Example:

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

Displaying Table Structure

- Using DESC keyword.

Syntax

```
DESC table_name;
```

Example:

```
DESC employees;
```

Find the Solution for the following:

True OR False

1. The following statement executes successfully.

Identify the Errors

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

```
SELECT Employee_id, Last_Name, Salary * 12 AS "ANNUAL SALARY" FROM EMPLOYEES;
```

Queries

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

DESC departments; SELECT * FROM departments;

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

SELECT employee_id, last_name, job_id, hire_date FROM employees;

4. Provide an alias STARTDATE for the hire date.

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

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

SELECT DISTINCT job_id FROM employees;

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

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

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

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

Evaluation Procedure	Marks awarded
Query(5)	
Execution (5)	
Viva(5)	
Total (15)	
Faculty Signature	

Ex.No.: 4	WORKING WITH CONSTRAINTS
Date:	

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

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 employees (employee_id number(6), first_name varchar2(20), ..job_id varchar2(10), CONSTRAINT emp_emp_id_pk PRIMARY KEY (employee_id));

Domain Integrity

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

NOT NULL Constraint

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

Principle of null values:

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

An error is returned for the following statements

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

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

The above statement can be written with CASCADE CONSTRAINT

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

(OR)

```
ALTER TABLE test1 DROP(pk, fk, col1) CASCADE CONSTRAINTS;
```

VIEWING CONSTRAINTS

Query the USER_CONSTRAINTS table to view all the constraints definition and

names. **Example:**

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

Viewing the columns associated with constraints

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

Find the Solution for the following:

1. Add a table-level PRIMARY KEY constraint to the EMP table on the ID column. The constraint should be named at creation. Name the constraint my_emp_id_pk.

```
ALTER TABLE EMP ADD CONSTRAINT my_emp_id_pk PRIMARY KEY  
(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.

```
ALTER TABLE DEPARTMENTS 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.

```
ALTER TABLE EMP ADD DEPT_ID NUMBER(4); ALTER TABLE EMP ADD  
CONSTRAINT my_emp_dept_id_fk FOREIGN KEY (DEPT_ID) REFERENCES  
DEPARTMENTS(dept_id);
```

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

```
ALTER TABLE EMP ADD COMMISSION NUMBER(2,2); ALTER TABLE EMP ADD  
CONSTRAINT chk_commission_gt_zero CHECK (COMMISSION > 0);
```

Evaluation Procedure	Marks awarded
Query(5)	
Execution (5)	
Viva(5)	
Total (15)	
Faculty Signature	

Ex.No.: 5	CREATING VIEWS
Date:	

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 base table exist.

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

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

Example: 1 (Without using Column aliases)

Create a view EMPVU80 that contains details of employees in

department80. **Example 2:**

CREATE VIEW empvu80 AS SELECT employee_id, last_name, salary FROM employees WHERE department_id=80;

Example:1 (Using column aliases)

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

Retrieving data from a view

Example:

SELECT * from salvu50;

Modifying a view

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

Example: (Simple view)

Modify the EMPVU80 view by using CREATE OR REPLACE.

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

Example: (complex view)

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

Rules for performing DML operations on view

- Can perform operations on simple views
- Cannot remove a row if the view contains the following:
 - Group functions
 - Group By clause
 - Distinct keyword
- Cannot modify data in a view if it contains
 - Group functions
 - Group By clause
 - Distinct keyword
 - Columns contain by expressions
- Cannot add data thr‘ a view if it contains
 - Group functions
 - Group By clause
 - Distinct keyword
 - Columns contain by expressions
 - NOT NULL columns in the base table that are not selected by the view

Example: (Using the WITH CHECK OPTION clause)

```
CREATE OR REPLACE VIEW empvu20
AS SELECT *
FROM employees
WHERE department_id=20
```

```
WITH CHECK OPTION CONSTRAINT empvu20_ck;
```

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

Example – (Execute this and note the error)

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

Denying DML operations

Use of WITH READ ONLY option.

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

error. **Try this code:**

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

Find the Solution for the following:

1. Create a view called EMPLOYEE_VU based on the employee numbers, employee names and department numbers from the EMPLOYEES table. Change the heading for the employee name to EMPLOYEE.

```
CREATE VIEW EMPLOYEE_VU AS SELECT Employee_id, First_Name || ' ' || Last_Name
AS EMPLOYEE, Dept_ID FROM EMPLOYEES;
```

2. Display the contents of the EMPLOYEES_VU view.

```
SELECT * FROM EMPLOYEE_VU;
```

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

```
SELECT VIEW_NAME, TEXT FROM USER_VIEWS WHERE VIEW_NAME =
'EMPLOYEE_VU';
```

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

```
SELECT EMPLOYEE, Dept_ID FROM EMPLOYEE_VU;
```

5. Create a view named DEPT50 that contains the employee number, employee last names and department numbers for all employees in department 50. Label the view columns EMPNO, EMPLOYEE and DEPTNO. Do not allow an employee to be reassigned to another department through the view.

```
CREATE VIEW DEPT50 AS SELECT Employee_id AS EMPNO, Last_Name AS  
EMPLOYEE, Dept_ID AS DEPTNO FROM EMPLOYEES WHERE Dept_ID = 50;
```

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

```
DESC DEPT50; SELECT * FROM DEPT50;
```

7. Attempt to reassign Matos to department 80.

```
UPDATE EMPLOYEES SET Dept_ID = 80 WHERE Last_Name = 'Matos';
```

8. Create a view called SALARY_VU based on the employee last names, department names, salaries, and salary grades for all employees. Use the Employees, DEPARTMENTS and JOB_GRADE tables. Label the column Employee, Department, salary, and Grade respectively.

```
CREATE VIEW SALARY_VU AS SELECT E.Last_Name AS Employee, D.dept_name AS  
Department, E.Salary AS Salary, J.Grade_level AS Grade FROM EMPLOYEES E JOIN  
DEPARTMENTS D ON E.Dept_ID = D.dept_id JOIN JOB_GRADE J ON E.Salary  
BETWEEN J.Lowest_sal AND J.Highest_sal;
```

Evaluation Procedure	Marks awarded
Query(5)	
Execution (5)	
Viva(5)	
Total (15)	
Faculty Signature	

Ex.No.: 6	RESTRICTING AND SORTING DATA
------------------	-------------------------------------

Date:	
--------------	--

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
-

Limits the Rows selected

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

Syntax

SELECT-----

FROM-----

WHERE condition;

Example:

```
SELECT employee_id, last_name, job_id, department_id FROM employees WHERE
department_id=90;
```

Character strings and Dates

Character strings and date values are enclosed in single quotation

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

Example:

```
SELECT employee_id, last_name, job_id, department_id FROM
employees WHERE last_name='WHALEN';
```

Comparison Conditions

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

Example:

```
SELECT last_name, salary
FROM employees
WHERE salary<=3000;
```

Other comparison conditions

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

Example:1

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

Example:2

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

Example:3

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

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

Example:4

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

Example:5

ESCAPE option-To have an exact match for the actual % and _ characters

To search for the string that contain _SA_ ‘

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

Test for NULL

- Using IS NULL operator

Example:

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

Logical Conditions

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

Example:1

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

Example:2

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

Example:3

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

Rules of Precedence

Order Evaluated	Operator
1	Arithmetic

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

Example:1

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

Example:2

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

Sorting the rows

Using ORDER BY Clause

ASC-Ascending Order,Default

DESC-Descending order

Example:1

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

Example:2

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

Example:3

Sorting by column alias

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

Example:4

Sorting by Multiple columns

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

Find the Solution for the following:

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

```
SELECT Last_Name, Salary FROM EMPLOYEES WHERE Salary > 12000;
```

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

```
SELECT Last_Name, Department_id FROM EMPLOYEES WHERE Employee_id  
= 176;
```

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

```
SELECT Last_Name, Salary FROM EMPLOYEES WHERE Salary NOT BETWEEN 5000 AND  
12000;
```

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

```
SELECT Last_Name, Job_id, Hire_date FROM EMPLOYEES WHERE Hire_date  
BETWEEN TO_DATE('1998-02-20', 'YYYY-MM-DD') AND TO_DATE('1998-05-01',  
'YYYY-MM-DD') ORDER BY Hire_date;
```

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

```
SELECT Last_Name, Department_id FROM EMPLOYEES WHERE Department_id IN (20,  
50) ORDER BY Last_Name;
```

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

```
SELECT Last_Name AS EMPLOYEE, Salary AS "MONTHLY SALARY" FROM  
EMPLOYEES WHERE Salary BETWEEN 5000 AND 12000 AND Department_id IN (20,  
50) ORDER BY Last_Name;
```

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

```
SELECT Last_Name, Hire_date FROM EMPLOYEES WHERE TO_CHAR(Hire_date,  
'YYYY') = '1994'
```

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

```
SELECT Last_Name, Job_id FROM EMPLOYEES WHERE Manager_id IS NULL
```

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

```
SELECT Last_Name, Salary, Commission_pct FROM EMPLOYEES WHERE Commission_pct IS  
NOT NULL ORDER BY Salary DESC, Commission_pct DESC;
```

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

```
SELECT Last_Name FROM EMPLOYEES WHERE Last_Name LIKE '__a%';
```

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

```
SELECT Last_Name FROM EMPLOYEES WHERE Last_Name LIKE '%a%' AND  
Last_Name LIKE '%e%';
```

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

```
SELECT Last_Name, Job_id, Salary FROM EMPLOYEES WHERE Job_id IN ('SA_REP',  
'ST_CLERK') AND Salary NOT IN (2500, 3500, 7000);
```

Evaluation Procedure	Marks awarded
Query(5)	
Execution (5)	
Viva(5)	
Total (15)	
Faculty Signature	

Ex.No.: 7	USING SET OPERATORS
Date:	

Objectives

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

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

The set operators combine the results of two or more component queries into one result. Queries containing set operators are called *compound queries*.

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

The tables used in this lesson are:

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

UNION Operator

Guidelines

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

Example:

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

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

Example:

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

UNION ALL Operator

Guidelines

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

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

Example:

Display the current and previous departments of all employees.

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

INTERSECT Operator

Guidelines

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

Example:

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

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

Example

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

MINUS Operator

Guidelines

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

Example:

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

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

Find the Solution for the following:

1. The HR department needs a list of department IDs for departments that do not contain the job ID ST_CLERK. Use set operators to create this report.

```
SELECT department_id FROM 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.

```
SELECT country_id, country_name FROM COUNTRIES WHERE country_id IN (  
SELECT country_id FROM COUNTRIES MINUS SELECT DISTINCT country_id  
FROM DEPARTMENTS WHERE department_name='HR');
```

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

```
SELECT job_id, department_id FROM EMPLOYEES WHERE department_id = 10 UNION  
ALL SELECT job_id, department_id FROM EMPLOYEES WHERE department_id = 50  
UNION ALL SELECT job_id, department_id FROM EMPLOYEES WHERE department_id  
= 20;
```

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

```
SELECT employee_id, job_id, hire_date FROM EMPLOYEES INTERSECT SELECT
employee_id, job_id, hire_date FROM JOB_HISTORY ORDER BY hire_date ASC;
```

5. The HR department needs a report with the following specifications:

- Last name and department ID of all the employees from the EMPLOYEES table, regardless of whether or not they belong to a department.
- Department ID and department name of all the departments from the DEPARTMENTS table, regardless of whether or not they have employees working in them Write a compound query to accomplish this.

```
SELECT last_name, department_id, NULL AS department_name FROM EMPLOYEES
UNION SELECT NULL AS last_name, department_id, department_name FROM
DEPARTMENTS;
```

Evaluation Procedure	Marks awarded
Query(5)	
Execution (5)	
Viva(5)	
Total (15)	
Faculty Signature	

Ex.No.: 8	WORKING WITH MULTIPLE TABLES	
Date:		

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

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

Additional Search Conditions

Using the AND Operator

Example:

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

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

Qualifying Ambiguous Column Names

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

Find the Solution for the following:

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

```
SELECT e.LAST_NAME, e.DEPARTMENT_ID, d.DEPARTMENT_NAME FROM EMPLOYEES e JOIN DEPARTMENTS d ON e.DEPARTMENT_ID = d.DEPARTMENT_ID;
```

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

```
SELECT DISTINCT e.JOB_ID, d.LOCATION_ID, l.COUNTRY FROM EMPLOYEES e JOIN DEPARTMENTS d ON e.DEPARTMENT_ID = d.DEPARTMENT_ID JOIN LOCATIONS l ON d.LOCATION_ID = l.LOCATION_ID WHERE e.DEPARTMENT_ID = 80;
```

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

```
SELECT e.LAST_NAME, d.DEPARTMENT_NAME, d.LOCATION_ID, l.CITY FROM EMPLOYEES e JOIN DEPARTMENTS d ON e.DEPARTMENT_ID = d.DEPARTMENT_ID JOIN LOCATIONS l ON d.LOCATION_ID = l.LOCATION_ID WHERE e.COMMISSION_PCT IS NOT NULL;
```

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

```
SELECT e.LAST_NAME, d.DEPARTMENT_NAME FROM EMPLOYEES e JOIN DEPARTMENTS d ON e.DEPARTMENT_ID = d.DEPARTMENT_ID WHERE LOWER(e.LAST_NAME) LIKE '%a%';
```

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

```
SELECT e.LAST_NAME, e.JOB_ID, e.DEPARTMENT_ID, d.DEPARTMENT_NAME FROM EMPLOYEES e JOIN DEPARTMENTS d ON e.DEPARTMENT_ID = d.DEPARTMENT_ID JOIN LOCATIONS l ON d.LOCATION_ID = l.LOCATION_ID WHERE l.CITY = 'Toronto';
```

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

```
SELECT e.LAST_NAME AS "Employee", e.EMPLOYEE_ID AS "Emp#", m.LAST_NAME AS "Manager",
m.EMPLOYEE_ID AS "Mgr#" FROM EMPLOYEES e JOIN EMPLOYEES m ON e.MANAGER_ID =
m.EMPLOYEE_ID;
```

7. Modify lab4_6.sql to display all employees including King, who has no manager. Order the results by the employee number.

```
SELECT e.LAST_NAME AS "Employee", e.EMPLOYEE_ID AS "Emp#",
m.LAST_NAME AS "Manager", m.EMPLOYEE_ID AS "Mgr#" FROM EMPLOYEES e
LEFT JOIN EMPLOYEES m ON e.MANAGER_ID = m.EMPLOYEE_ID ORDER BY
e.EMPLOYEE_ID;
```

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

```
SELECT e1.LAST_NAME AS "Employee", e1.DEPARTMENT_ID, e2.LAST_NAME AS
"CoWorkers" FROM EMPLOYEES e1 JOIN EMPLOYEES e2 ON e1.DEPARTMENT_ID =
e2.DEPARTMENT_ID WHERE e1.EMPLOYEE_ID = 106 AND e1.EMPLOYEE_ID <>
e2.EMPLOYEE_ID;
```

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

```
DESCRIBE JOB_GRADES; SELECT e.LAST_NAME, e.JOB_ID,
d.DEPARTMENT_NAME, e.SALARY, jg.GRADE_LEVEL FROM EMPLOYEES e
JOIN DEPARTMENTS d ON e.DEPARTMENT_ID = d.DEPARTMENT_ID JOIN
JOB_GRADES jg ON e.SALARY BETWEEN jg.LOW_SALARY AND
jg.HIGH_SALARY;
```

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

```
SELECT e.LAST_NAME AS "Employee", e.HIRE_DATE AS "Hire Date" FROM
```

```
EMPLOYEES e JOIN EMPLOYEES r ON r.LAST_NAME = 'Davies' WHERE  
e.HIRE_DATE > r.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.

```
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 AND  
e.EMPLOYEE_ID <> m.EMPLOYEE_ID;
```

Evaluation Procedure	Marks awarded
Query(5)	
Execution (5)	
Viva(5)	
Total (15)	
Faculty Signature	

Ex.No.: 9	SUB QUERIES
Date:	

Objectives

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

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

Using a Subquery to Solve a Problem

Who has a salary greater than Abel's?

Main query:

Which employees have salaries greater than Abel's salary?

Subquery:

What is Abel's salary?

Subquery Syntax

`SELECT select_list FROM table WHERE expr operator (SELECT select_list FROM
table);` • The subquery (inner query) executes once before the main query (outer query). •

The result of the subquery is used by the main query.

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

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

- WHERE clause
- HAVING clause
- FROM clause

In the syntax:

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

Note: Comparison conditions fall into two classes: single-row operators ($>$, $=$, \geq , $<$, \neq , \leq) and multiple-row operators (IN, ANY, ALL). statement. The subquery generally executes first, and its output is used to complete the query condition for the main (or outer) query

Using a Subquery

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

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

Guidelines for Using Subqueries

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

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

subqueries. **Types of Subqueries**

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

Single-Row Subqueries

- Return only one row

- Use single-row comparison operators

Example

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

```
SELECT last_name, job_id FROM employees WHERE job_id = (SELECT job_id FROM
employees
```

```
WHERE employee_id = 141);
```

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

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

Using Group Functions in a Subquery

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

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

The HAVING Clause with Subqueries

- The Oracle server executes subqueries first.
- The Oracle server returns results into the HAVING clause of the main query. Displays all the departments that have a minimum salary greater than that of department 50.

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

Example

Find the job with the lowest average salary.

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

What Is Wrong in this Statements?

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

Multiple-Row Subqueries

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

Example

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

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

Using the ANY Operator in Multiple-Row Subqueries

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

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

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

Using the ALL Operator in Multiple-Row Subqueries

```
SELECT employee_id, last_name, job_id, salary
FROM employees
WHERE salary < ALL (SELECT salary FROM employees WHERE job_id =
'IT_PROG') AND job_id <> 'IT_PROG';
Displays employees whose salary is less than the salary of all employees with a job ID of
IT_PROG and whose job is not IT_PROG.
• ALL means more than the maximum, and <ALL means less than the minimum.
```

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

Null Values in a Subquery

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

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

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

Display all employees who do not have any subordinates:

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

Find the Solution for the following:

1. The HR department needs a query that prompts the user for an employee last name. The query then displays the last name and hire date of any employee in the same department as the employee whose name they supply (excluding that employee). For example, if the user enters Zlotkey, find all employees who work with Zlotkey (excluding Zlotkey).

```
SELECT last_name, hire_date FROM employees WHERE department_id = ( SELECT
department_id FROM employees WHERE last_name = 'Zlotkey' ) AND last_name != 'Zlotkey';
```

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

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

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

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

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

```
SELECT last_name, department_id, job_id FROM employees WHERE department_id IN ( SELECT department_id FROM departments WHERE location_id=1700 );
```

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

```
SELECT last_name, salary FROM employees e WHERE EXISTS( SELECT last_name FROM employees m WHERE e.manager_id = m.employee_id AND m.last_name='King' );
```

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

```
SELECT department_id, last_name, job_id FROM employees WHERE department_id = ( 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*.

```
SELECT e.employee_id, e.last_name, e.salary FROM employees e WHERE e.salary >
(SELECT AVG(salary) FROM employees) AND EXISTS ( SELECT * FROM employees e2
WHERE e.department_id = e2.department_id AND e2.last_name LIKE '%u%' );
```

Evaluation Procedure	Marks awarded
Query(5)	
Execution (5)	
Viva(5)	
Total (15)	
Faculty Signature	

Ex.No.: 10	AGGREGATING DATA USING GROUP FUNCTIONS	
Date:		

Objectives

After the completion of this exercise, the students 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 x, ignoring null values
SUM([DISTINCT ALL] n)	Sum values of n, ignoring null values
VARIANCE([DISTINCT ALL] x)	Variance of x, 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_id HAVING MAX(salary)>10000 ;
```

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

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

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

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

Nesting Group Functions

Display the maximum average salary:

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

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

Summary

In this exercise, students should have learned how to:

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

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

Find the Solution for the following:

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

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

TRUE

2. Group functions include nulls in calculations.

True/False

FALSE

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

calculation. True/False

TRUE

The HR department needs the following reports:

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

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

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

```
SELECT job_id, ROUND(MIN(salary)) AS Minimum,  
ROUND(MAX(salary)) AS Maximum, ROUND(SUM(salary)) AS  
Sum, ROUND(AVG(salary)) AS Average FROM employees GROUP  
BY job_id;
```

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

```
SELECT job_id, COUNT(*) AS Number_of_People FROM employees WHERE job_id =  
'Developer' GROUP BY job_id;
```

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

```
SELECT COUNT(DISTINCT manager_id) AS Number_of_Managers FROM employees  
WHERE manager_id IS NOT NULL;
```

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

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

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

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

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

```
SELECT COUNT(*) AS Total_Employees, SUM(CASE WHEN EXTRACT(YEAR FROM hire_date) = 1995 THEN 1 ELSE 0 END) AS Employees_1995, SUM(CASE WHEN EXTRACT(YEAR FROM hire_date) = 1996 THEN 1 ELSE 0 END) AS Employees_1996, SUM(CASE WHEN EXTRACT(YEAR FROM hire_date) = 1997 THEN 1 ELSE 0 END) AS Employees_1997, SUM(CASE WHEN EXTRACT(YEAR FROM hire_date) = 1998 THEN 1 ELSE 0 END) AS Employees_1998 FROM employees;
```

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

```
SELECT job_id, department_id, SUM(salary) AS Total_Salary, AVG(salary) AS Average_Salary FROM employees WHERE department_id IN (20, 50, 80, 90) GROUP BY job_id, department_id ORDER BY department_id, job_id;
```

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

```
SELECT d.department_name AS "Name-Location", d.location_id AS Location, COUNT(e.employee_id) AS "Number of People", ROUND(AVG(e.salary), 2) AS Salary FROM departments d LEFT JOIN employees e ON d.department_id = e.department_id GROUP BY d.department_name, d.location_id;
```

Evaluation Procedure	Marks awarded
Query(5)	
Execution (5)	
Viva(5)	
Total (15)	
Faculty Signature	

Ex.No.: 11	PL SQL PROGRAMS
Date:	

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

PL/SQL procedure successfully completed.

PRINT NUMBERS FROM 1 TO 4 USING WHILE LOOP

SQL> set serveroutput on;

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

1
2
3
4

PL/SQL procedure successfully completed.

PRINT NUMBERS FROM 1 TO 5 USING FOR LOOP

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

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

1
2
3
4
5

PL/SQL procedure successfully completed.

PRINT NUMBERS FROM 1 TO 5 IN REVERSE ORDER USING FOR

LOOP SQL> set serveroutput on;

```
SQL> declare
2 a number:=1;
3 begin
4 for a in reverse 1..5
5 loop
6 dbms_output.put_line (a);
7 end loop;
8 end;
9 /
```

5
4
3
2

1
PL/SQL procedure successfully completed.

TO CALCULATE AREA OF CIRCLE

```
SQL> set serveroutput on;
SQL> declare
2 pi constant number(4,2):=3.14;
3 a number(20);
4 r number(20);
5 begin
6 r:=&r;
7 a:= pi* power(r,2);
8 dbms_output.put_line (' The area of circle is ' || a);
9 end;
```

10 /

Enter value for r: 2

old 6: r:=&r;

new 6: r:=2;

The area of circle is 13

PL/SQL procedure successfully completed.

TO CREATE SACCOUNT TABLE

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

Table created.

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

1 row created.

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

1 row created.

```
SQL> select * from saccount;
```

ACCNO NAME BAL

1 mala 20000

2 kala 30000

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

```
SQL> declare
```

```
2 a_bal number(7);
```

```
3 a_no varchar2(20);
```

```
4 debit number(7):=2000;
```

```
5 minamt number(7):=500;
```

```
6 begin
```

```
7 a_no:=&a_no;
```

```
8 select bal into a_bal from saccount where accno= a_no;
```

```
9 a_bal:= a_bal-debit;
```

```
10 if (a_bal > minamt) then
```

```
11 update saccount set bal=bal-debit where accno=a_no;
```

```
12 end if;
```

```
13 end;
```

```
14
```

```
15 /
```

Enter value for a_no: 1

```
old 7: a_no:=&a_no;
new 7: a_no:=1;
PL/SQL procedure successfully completed.
SQL> select * from saccount;
ACCNO NAME BAL
```

```
-----  
1 mala 18000  
2 kala 30000
```

TO CREATE TABLE SROUTES

```
SQL> create table sroutes ( rno number(5), origin varchar2(20), destination varchar2(20), fare
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 SCALCULATE TABLE

```
SQL> create table scalculate ( radius number(3), area  
number(5,2)); Table created.
```

```
SQL> desc scalculate;
```

Name	Null?	Type
RADIUS	NUMBER(3)	
AREA	NUMBER(5,2)	

```
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

TO CALCULATE FACTORIAL OF A GIVEN NUMBER

```
SQL> set serveroutput on;
SQL> declare
2 f number(4):=1;
3 i number(4);
4 begin
5 i:=&i;
6 while(i>=1)
7 loop
8 f:=f*i;
9 i:=i-1;
10 end loop;
11 dbms_output.put_line('The value is ' || f);
12 end;
13 /
```

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 PL/SQL block to calculate the incentive of an employee whose ID is 110.

```
DECLARE emp_salary employees.salary%TYPE; incentive NUMBER(8,2); BEGIN SELECT
salary INTO emp_salary FROM employees WHERE employee_id = 110; incentive := emp_salary
* 0.1; DBMS_OUTPUT.PUT_LINE('Incentive for Employee ID 110: ' || 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.

```
DECLARE "EmployeeID" NUMBER := 110; BEGIN
DBMS_OUTPUT.PUT_LINE(EmployeeID); END;
```

PROGRAM 3

Write a PL/SQL block to adjust the salary of the employee whose ID 122. Sample table: employees

```
BEGIN UPDATE employees SET salary = salary + 5000 WHERE employee_id = 122;
DBMS_OUTPUT.PUT_LINE('Salary adjusted for Employee ID 122'); END; /
```

PROGRAM 4

Write a PL/SQL block to create a procedure using the "IS [NOT] NULL Operator" and show AND operator returns TRUE if and only if both operands are TRUE.

```
CREATE OR REPLACE PROCEDURE CheckNullAndOperator IS
value1 BOOLEAN := TRUE;
value2 BOOLEAN := TRUE;
BEGIN
IF value1 IS NOT NULL AND value2 IS NOT NULL AND
value1 AND value2 THEN
DBMS_OUTPUT.PUT_LINE('Both conditions are TRUE');
ELSE
DBMS_OUTPUT.PUT_LINE('One or both conditions are FALSE');
END IF;
END;
```

PROGRAM 5

Write a PL/SQL block to describe the usage of LIKE operator including wildcard characters and escape character.

```
DECLARE
emp_name employees.first_name%TYPE;
BEGIN
FOR rec IN (SELECT first_name FROM employees WHERE first_name LIKE 'J%')
LOOP
DBMS_OUTPUT.PUT_LINE('Employee name starting with J: ' ||
rec.first_name);
END LOOP;
END;
```

PROGRAM 6

Write a PL/SQL program to arrange the number of two variable in such a way that the smallnumber will store in num_small variable and large number will store in num_large variable.

```
DECLARE
num1 NUMBER := 10;
num2 NUMBER := 5;
num_small NUMBER;
num_large NUMBER;
BEGIN
IF num1 < num2 THEN
num_small := num1;
num_large := num2;
ELSE
num_small := num2;
num_large := num1;
END IF;
DBMS_OUTPUT.PUT_LINE('Small Number: ' ||
num_small || ', Large Number: ' || num_large);
END;
```

PROGRAM 7

Write a PL/SQL procedure to calculate the incentive on a target achieved and display the message either the record updated or not.

```
CREATE OR REPLACE PROCEDURE UpdateIncentive IS
target NUMBER := 100000;
sales NUMBER := 120000;
incentive NUMBER;
BEGIN
IF sales >= target THEN
incentive := sales * 0.1;
DBMS_OUTPUT.PUT_LINE('Incentive updated to ' || incentive);
ELSE
DBMS_OUTPUT.PUT_LINE('Target not met. No incentive.');
END IF;
END;
```

PROGRAM 8

Write a PL/SQL procedure to calculate incentive achieved according to the specific sale limit.

```
CREATE OR REPLACE PROCEDURE CalculateIncentive(sales_limit IN NUMBER) IS incentive  
NUMBER; BEGIN IF sales_limit > 50000 THEN incentive := sales_limit * 0.15; ELSE incentive :=  
sales_limit * 0.1; END IF; DBMS_OUTPUT.PUT_LINE('Incentive: ' || incentive); 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.

```
DECLARE emp_count NUMBER; vacancies NUMBER := 45; BEGIN SELECT COUNT(*) INTO  
emp_count FROM employees WHERE department_id = 50; IF emp_count < vacancies THEN  
DBMS_OUTPUT.PUT_LINE('Vacancies available: ' || (vacancies - emp_count)); ELSE  
DBMS_OUTPUT.PUT_LINE('No vacancies'); 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.

```
DECLARE emp_count NUMBER; dept_id NUMBER := 80; vacancies NUMBER := 45; BEGIN  
SELECT COUNT(*) INTO emp_count FROM employees WHERE department_id = dept_id; IF  
emp_count < vacancies THEN DBMS_OUTPUT.PUT_LINE('Vacancies in Department ' || dept_id || ': ' ||  
(vacancies - emp_count)); ELSE DBMS_OUTPUT.PUT_LINE('No vacancies'); 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.

```
DECLARE CURSOR emp_cursor IS SELECT employee_id, first_name, job_id, hire_date,  
salary FROM employees; BEGIN FOR emp IN emp_cursor LOOP  
DBMS_OUTPUT.PUT_LINE('ID: ' || emp.employee_id || ', Name: ' || emp.first_name || ',  
Job: ' || emp.job_id || ', Hire Date: ' || emp.hire_date || ', Salary: ' || emp.salary); END LOOP;  
END;
```

PROGRAM 12

Write a PL/SQL program to display the employee IDs, names, and department names of all employees.

```
DECLARE CURSOR emp_dept_cursor IS SELECT e.employee_id, e.first_name,  
d.department_name FROM employees e JOIN departments d ON e.department_id =  
d.department_id; BEGIN FOR emp IN emp_dept_cursor LOOP  
DBMS_OUTPUT.PUT_LINE('ID: ' || emp.employee_id || ', Name: ' || emp.first_name || ',  
Dept: ' || emp.department_name); END LOOP; END;
```

PROGRAM 13

Write a PL/SQL program to display the job IDs, titles, and minimum salaries of all jobs.

```
DECLARE CURSOR job_cursor IS SELECT job_id, job_title, min_salary FROM jobs; BEGIN FOR job IN job_cursor LOOP DBMS_OUTPUT.PUT_LINE('Job ID: ' || job.job_id || ', Title: ' || job.job_title || ', Min Salary: ' || job.min_salary); END LOOP; END;
```

PROGRAM 14

Write a PL/SQL program to display the employee IDs, names, and job history start dates of all employees.

```
DECLARE CURSOR job_hist_cursor IS SELECT employee_id, start_date FROM job_history; BEGIN FOR job_hist IN job_hist_cursor LOOP DBMS_OUTPUT.PUT_LINE('Employee ID: ' || job_hist.employee_id || ', Start Date: ' || job_hist.start_date); END LOOP; END;
```

PROGRAM 15

Write a PL/SQL program to display the employee IDs, names, and job history end dates of all employees.

```
DECLARE CURSOR job_hist_cursor IS SELECT employee_id, end_date FROM job_history; BEGIN FOR job_hist IN job_hist_cursor LOOP DBMS_OUTPUT.PUT_LINE('Employee ID: ' || job_hist.employee_id || ', End Date: ' || job_hist.end_date); END LOOP; END;
```

Evaluation Procedure	Marks awarded
PL/SQL Procedure(5)	
Program/Execution (5)	
Viva(5)	
Total (15)	
Faculty Signature	

Ex.No.: 12	WORKING WITH CURSOR, PROCEDURES AND FUNCTIONS
Date:	

AIM:

Create PL/SQL Blocks to perform the Item Transaction Operations using CURSOR, FUNCTION and PROCEDURE.

ALGORITHM:

STEP-1: Start.

STEP-2: Create two tables Item Master and Item Trans.

```
itemmaster(itemid , itemname, stockonhand )
```

```
itemtrans(itemid ,itemname ,dateofpurchase ,quantity)
```

STEP-3: Create a PROCEDURE with id, name and quantity as parameters which make a call to the FUNCTION by passing id, name, dop, and quantity as parameters dop is set as sysdate.

STEP-4: Using FUNCTION fetch each record from the table Item Master using

CURSOR inside a Loop statement,

If Item Master's ItemId is equal to the entered ID value then exit the loop otherwise fetch the next record.

loop

```
    fetch master into masterrec
```

```
    exit when master%notfound
```

```
    if masterrec.itemid=id then
```

```
        exit;
```

```
    end if;
```

```
    end loop;
```

STEP-5: If Itemmaster's itemid = id then,

Add the Itemmaster's stockonhand with the given quantity and update the ItemMaster table and insert the Item information into the ItemTrans table.

STEP-6: Else, if the inputed item is not present in the ItemMaster table then insert the

new Item in both the tables.

STEP-7: Call the Procedure by passing the Item informations which calls the Function.

STEP-8: Exit.

PROCEDURES – SYNTAX

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

FUNCTIONS – SYNTAX

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

CREATING THE TABLE 'ITITEMS' AND DISPLAYING THE CONTENTS

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

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

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

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

```
SQL> select * from ititems;
ITEMID ACTUALPRICEORDID PRODID
```

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

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

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

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

```
SQL> select * from ititems;
ITEMID ACTUALPRICE ORDID PRODID
```

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

PROCEDURE FOR 'IN' PARAMETER – CREATION, EXECUTION

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

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

Procedure created.

```
SQL> exec yyy(103);
Actual price is 4000
PL/SQL procedure successfully completed.
```

PROCEDURE FOR 'OUT' PARAMETER – CREATION, EXECUTION

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

```
SQL> create procedure zzz (a in number, b out number) is identity number;
  2 begin
  3 select ordid into identity from ititems where itemid=a;
  4 if identity<1000 then
  5 b:=100;
  6 end if;
  7 end;
  8 /
```

Procedure created.

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

The value of b is 100

PL/SQL procedure successfully completed.

PROCEDURE FOR 'INOUT' PARAMETER – CREATION, EXECUTION

```
SQL> create procedure itit ( a in out number) is
  2 begin
  3 a:=a+1;
```

```
4 end;
5 /
Procedure created.
```

```
SQL> declare
2 a number:=7;
3 begin
4 itit(a);
5 dbms_output.put_line(_The updated value is _||a);
6 end;
7 /
```

The updated value is 8
PL/SQL procedure successfully completed.

CREATE THE TABLE 'ITTRAIN' TO BE USED FOR FUNCTIONS

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

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

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

```
SQL>select * from
      ittrain; TNO
                  TFARE
-----
1001      550
1002      600
```

PROGRAM FOR FUNCTION AND IT'S EXECUTION

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

Function created.

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

```
SQL> declare
  2  total number;
  3  begin
  4  total:=aaa (1001);
  5  dbms_output.put_line('Train fare is Rs. '||total);
  6  end;
  7  /
```

Train fare is Rs.550

PL/SQL procedure successfully completed.

FACTORIAL OF A NUMBER USING FUNCTION — PROGRAM AND EXECUTION

```
SQL> create function itfact (a number) return number is
  2  fact number:=1;
  3  b number;
  4  begin
  5  b:=a;
  6  while b>0
  7  loop
  8  fact:=fact*b;
  9  b:=b-1;
 10 end loop;
 11 return(fact);
 12 end;
 13 /
```

Function created.

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

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

The factorial of the given number is 5040

PL/SQL procedure successfully completed.

Program 1

FACTORIAL OF A NUMBER USING FUNCTION

```
CREATE OR REPLACE FUNCTION factorial(n NUMBER) RETURN NUMBER IS
result NUMBER := 1;
BEGIN
IF n < 0 THEN
RETURN NULL;
ELSIF n = 0 THEN
RETURN 1;
ELSE
FOR i IN 1..n LOOP
result := result * i;
END LOOP;
END IF;
RETURN result;
END factorial; /
DECLARE
num NUMBER := 5;
fact NUMBER;
BEGIN
fact := factorial(num);
DBMS_OUTPUT.PUT_LINE('Factorial of ' || num || ' is: ' || fact);
END; /
```

Program 2

Write a PL/SQL program using Procedures IN,INOUT,OUT parameters to retrieve the corresponding book information in library

```
CREATE OR REPLACE PROCEDURE get_book_info (
    p_book_id IN NUMBER,
    p_title IN OUT VARCHAR2,
    p_author OUT VARCHAR2,
    p_genre OUT VARCHAR2,
    p_publication_year OUT NUMBER
) IS
BEGIN
    SELECT title, author, genre, publication_year
    INTO p_title, p_author, p_genre, p_publication_year
    FROM books
    WHERE book_id = p_book_id;
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE('No book found with ID: ' || p_book_id);
END get_book_info; /
DECLARE
    book_id NUMBER := 3;
    title VARCHAR2(100) := 'Default Title';
    author VARCHAR2(100);
    genre VARCHAR2(50);
    publication_year NUMBER;
BEGIN
    get_book_info(book_id, title, author, genre, publication_year);
    DBMS_OUTPUT.PUT_LINE('Title: ' || title);
    DBMS_OUTPUT.PUT_LINE('Author: ' || author);
    DBMS_OUTPUT.PUT_LINE('Genre: ' || genre);
    DBMS_OUTPUT.PUT_LINE('Publication Year: ' || publication_year);
END; /
```

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

```
1 declare
2 cursor cenl is select eid,sal from ssempp where dno=11;
3 ecode ssempp.eid%type;
4 esal empp.sal%type;
5 begin
6 open cenl;
7 loop
8 fetch cenl into ecode,esal;
9 exit when cenl%notfound;
10 dbms_output.put_line(' Employee code and employee salary are' || ecode _and'|| esal);
```

```

11 end loop;
12 close cenl;
13* end;

SQL> /
Employee code and employee salary are 1 and 39000
Employee code and employee salary are 5 and 35000
Employee code and employee salary are 6 and 23000

```

PL/SQL procedure successfully completed.

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

```

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

```

The number of rows are 3

Employee record modification successful

PL/SQL procedure successfully completed.

SQL> select * from ssemp;

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

Evaluation Procedure	Marks awarded
PL/SQL Procedure(5)	
Program/Execution (5)	
Viva(5)	
Total (15)	
Faculty Signature	

Ex.No.: 13	WORKING WITH TRIGGER
Date:	<u>TRIGGER</u>

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.

TABLE CREATION:

```
create table employeebonus(empno number(5)constraint emppk primary key, empname
varchar2(25)not null, experience number(2)not null, bonus number(7,2));
```

Table created.

TRIGGER CREATION FOR BONUS CALCULATION:

```
SQL> set serveroutput on
SQL> create or replace trigger employeebonus_tgr
after insert on employeebonus
declare
cursor emp is select * from employeebonus;
emprec employeebonus%rowtype;
begin
```

```

open emp;
loop
fetch emp into emprec;
exit when emp%notfound;
if(emprec.experience<5) then
emprec.bonus:=5000;
elsif(emprec.experience>=5 and emprec.experience<8) then
emprec.bonus:=8000;
else
emprec.bonus:=10000;
end if;
update employeebonus set bonus=emprec.bonus where empno=emprec.empno;
end loop;
close emp;
dbms_output.put_line('Bonus calculated and Updated Sucessfully');
end;
/

```

Trigger created.

TABLE DESCRIPTION:

SQL> desc employeebonus;
Name Null? Type

EMPNO NOT NULL NUMBER(5)
EMPNAME NOT NULL VARCHAR2(25)
EXPERIENCE NOT NULL
NUMBER(2)
BONUS NUMBER(7,2)

RECORD INSERTION:

SQL> insert into employeebonus(empno,empname,experience)
values(&empno,'&empname',&experience);
Enter value for empno: 101
Enter value for empname: murugan Enter
value for experience: 25
old 1: insert into employeebonus(empno,empname,experience)

```
values(&empno,'&empname',&experience)
new 1: insert into employeebonus(empno,empname,experience)
values(101,'murugan',25)
Bonus calculated and Updated Sucessfully
1 row created.
```

RECORD SELECTION:

```
SQL> select * from employeebonus;
EMPNO EMPNAME EXPERIENCE BONUS
-----
101 murugan 25 10000
102 suresh 3 5000
103 akash 7 8000
104 mahesh 2 5000
```

Program 1

Write a code in PL/SQL to develop a trigger that enforces referential integrity by preventing the deletion of a parent record if child records exist.

```
CREATE OR REPLACE TRIGGER prevent_parent_delete BEFORE DELETE ON items
FOR EACH ROW DECLARE child_count NUMBER; BEGIN SELECT COUNT(*) INTO
child_count FROM orders WHERE item_id = :OLD.item_id; IF child_count > 0 THEN
RAISE_APPLICATION_ERROR(-20001, 'Cannot delete item; dependent orders exist.');
END IF; END;
```

Program 2

Write a code in PL/SQL to create a trigger that checks for duplicate values in a specific column and raises an exception if found.

```
CREATE OR REPLACE TRIGGER check_for_duplicates BEFORE INSERT OR UPDATE ON
orders FOR EACH ROW DECLARE duplicate_count NUMBER; BEGIN SELECT COUNT(*) INTO
duplicate_count FROM orders WHERE item_id = :NEW.item_id AND order_id != :NEW.order_id; IF
duplicate_count > 0 THEN RAISE_APPLICATION_ERROR(-20002, 'Duplicate item entry found in
orders.');
```

Program 3

Write a code in PL/SQL to create a trigger that restricts the insertion of new rows if the total of a column's values exceeds a certain threshold.

```
CREATE OR REPLACE TRIGGER restrict_insertion BEFORE INSERT ON orders FOR EACH ROW DECLARE total_quantity NUMBER; BEGIN SELECT SUM(quantity) INTO total_quantity FROM orders; IF (total_quantity + :NEW.quantity) > 500 THEN RAISE_APPLICATION_ERROR(-20003, 'Cannot insert order; total quantity exceeds threshold.');
```

Program 4

Write a code in PL/SQL to design a trigger that captures changes made to specific columns and logs them in an audit table.

```
CREATE OR REPLACE TRIGGER log_changes AFTER UPDATE ON orders FOR EACH ROW BEGIN INSERT INTO audit_log (log_id, table_name, operation, user_id, details) VALUES (audit_log_seq.NEXTVAL, 'orders', 'UPDATE', :NEW.user_id, 'Order ' || :NEW.order_id || ' changed from ' || :OLD.quantity || ' to ' || :NEW.quantity ); END;
```

Program 5

Write a code in PL/SQL to implement a trigger that records user activity (inserts, updates, deletes) in an audit log for a given set of tables.

```
CREATE OR REPLACE TRIGGER log_user_activity AFTER INSERT OR DELETE OR UPDATE ON orders FOR EACH ROW BEGIN INSERT INTO audit_log (log_id, table_name, operation, user_id, details) VALUES (audit_log_seq.NEXTVAL, 'orders', CASE WHEN INSERTING THEN 'INSERT' WHEN UPDATING THEN 'UPDATE' WHEN DELETING THEN 'DELETE' END, NVL(:NEW.user_id, :OLD.user_id), 'User action recorded on order ' || NVL(:NEW.order_id, :OLD.order_id)); END;
```

Program 7

Write a code in PL/SQL to implement a trigger that automatically calculates and updates a running total column for a table whenever new rows are inserted.

```
CREATE OR REPLACE TRIGGER update_running_total
AFTER INSERT ON orders
FOR EACH ROW
BEGIN
UPDATE orders SET running_total = (SELECT SUM(quantity) FROM orders)
WHERE order_id = :NEW.order_id;
```

END;

Program 8

Write a code in PL/SQL to create a trigger that validates the availability of items before allowing an order to be placed, considering stock levels and pending orders.

```
CREATE OR REPLACE TRIGGER validate_item_availability BEFORE INSERT ON orders
FOR EACH ROW DECLARE available_stock NUMBER; BEGIN SELECT stock_level -
pending_orders INTO available_stock FROM items WHERE item_id = :NEW.item_id; IF
:NEW.quantity > available_stock THEN RAISE_APPLICATION_ERROR(-20004,
'Insufficient stock available for the order.');?>
END IF; UPDATE items SET pending_orders =
pending_orders + :NEW.quantity WHERE item_id = :NEW.item_id; END;
```

Evaluation Procedure	Marks awarded
PL/SQL Procedure(5)	
Program/Execution (5)	
Viva(5)	
Total (15)	
Faculty Signature	

Ex.No.: 14	MONGO DB
Date:	

MongoDB is a free and open-source cross-platform document-oriented database. Classified as a NoSQL database, MongoDB avoids the traditional table-based relational database structure in favor of JSON-like documents with dynamic schemas, making the integration of data in certain types of applications easier and faster.

Create Database using mongosh

After connecting to your database using mongosh, you can see which database you are using by typing db in your terminal.

If you have used the connection string provided from the MongoDB Atlas dashboard, you should be connected to the myFirstDatabase database.

Show all databases

To see all available databases, in your terminal type show dbs.

Notice that myFirstDatabase is not listed. This is because the database is empty. An empty database is essentially non-existent.

Change or Create a Database

You can change or create a new database by typing use then the name of the database.

Create Collection using mongosh

You can create a collection using the createCollection() database method.

Insert Documents

```
insertOne()
```

```
db.posts.insertOne({
```

```
    title: "Post Title 1",
```

```
    body: "Body of post.",
```

```
    category: "News",
```

```
    likes: 1,
```

```
    tags: ["news", "events"],
```

```
date: Date()  
})
```

EXERCISE 18

Structure of 'restaurants' collection:

```
{  
  "address": {  
    "building": "1007",  
    "coord": [ -73.856077, 40.848447 ],  
    "street": "Morris Park Ave",  
    "zipcode": "10462"  
  },  
  "borough": "Bronx",  
  "cuisine": "Bakery",  
  
  "grades": [  
    { "date": { "$date": 1393804800000 }, "grade": "A", "score": 2 },  
    { "date": { "$date": 1378857600000 }, "grade": "A", "score": 6 },  
    { "date": { "$date": 1358985600000 }, "grade": "A", "score": 10 },  
    { "date": { "$date": 1322006400000 }, "grade": "A", "score": 9 },  
    { "date": { "$date": 1299715200000 }, "grade": "B", "score": 14 }  
  ],  
  "name": "Morris Park Bake Shop",  
  "restaurant_id": "30075445"  
  
}
```

1. Write a MongoDB query to find the restaurant Id, name, borough and cuisine for those restaurants which prepared dish except 'American' and 'Chinees' or restaurant's name begins with letter 'Wil'.

```
db.restaurants.find( { $or: [ { cuisine: { $nin: ["American", "Chinese"] } }, { name: /^Wil/ } ] },  
{ 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..

```
db.restaurants.find( { grades: { $elemMatch: { grade: "A", score: 11, date: ISODate("2014-08-  
11T00:00:00Z") } } }, { 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".

```
db.restaurants.find( { "grades.1.grade": "A", "grades.1.score": 9, "grades.1.date": ISODate("2014-08-11T00:00:00Z") }, { 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..

```
db.restaurants.find( { "address.coord.1": { $gt: 42, $lte: 52 } }, { restaurant_id: 1, name: 1, address: 1, "address.coord": 1 } )
```

5. Write a MongoDB query to arrange the name of the restaurants in ascending order along with all the columns.

```
db.restaurants.find().sort({ name: 1 })
```

6. Write a MongoDB query to arrange the name of the restaurants in descending along with all the columns.

```
db.restaurants.find().sort({ name: -1 })
```

7. Write a MongoDB query to arranged the name of the cuisine in ascending order and for that same cuisine borough should be in descending order.

```
db.restaurants.find().sort({ cuisine: 1, borough: -1 })
```

8. Write a MongoDB query to know whether all the addresses contains the street or not.

```
db.restaurants.find( { "address.street": { $exists: true } } )
```

9. Write a MongoDB query which will select all documents in the restaurants collection where the coord field value is Double.

```
db.restaurants.find({ "address.coord": { $type: "double" } })
```

10. Write a MongoDB query which will select the restaurant Id, name and grades for those restaurants which returns 0 as a remainder after dividing the score by 7.

```
db.restaurants.find( { "grades.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.

```
db.restaurants.find( { name: /mon/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.

```
db.restaurants.find( { name: /^Mad/ }, { 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.

```
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.

```
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.

```
db.restaurants.find({ "grades.score": { $lt: 5 }, borough: { $in: ["Manhattan", "Brooklyn"] } })
```

16. Write a MongoDB query to find the restaurants that have at least one grade with a score of less than 5 and that are located in the borough of Manhattan or Brooklyn, and their cuisine is not American.

```
db.restaurants.find(  
  { "grades.score": { $lt: 5 }, borough: { $in: ["Manhattan", "Brooklyn"] }, 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.

```
db.restaurants.find( { "grades.score": { $lt: 5 }, borough: { $in: ["Manhattan", "Brooklyn"] },  
cuisine: { $nin: ["American", "Chinese"] } } )
```

18. Write a MongoDB query to find the restaurants that have a grade with a score of 2 and a grade with a score of 6.

```
db.restaurants.find({ grades: { $all: [ { $elemMatch: { score: 2 } }, { $elemMatch: { score: 6 } } ] } })
```

19. Write a MongoDB query to find the restaurants that have a grade with a score of 2 and a grade with a score of 6 and are located in the borough of Manhattan.

```
db.restaurants.find({ grades: { $all: [ { $elemMatch: { score: 2 } }, { $elemMatch: { score: 6 } } ] },  
borough: "Manhattan" })
```

20. Write a MongoDB query to find the restaurants that have a grade with a score of 2 and a grade with a score of 6 and are located in the borough of Manhattan or Brooklyn.

```
db.restaurants.find({ grades: { $all: [ { $elemMatch: { score: 2 } }, { $elemMatch: { score: 6 } } ] },  
borough: { $in: ["Manhattan", "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 American.

```
db.restaurants.find({ grades: { $all: [ { $elemMatch: { score: 2 } }, { $elemMatch: { score: 6 } } ] }, borough: { $in: ["Manhattan", "Brooklyn"] }, 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.

```
db.restaurants.find({ grades: { $all: [ { $elemMatch: { score: 2 } }, { $elemMatch: { score: 6 } } ] }, borough: { $in: ["Manhattan", "Brooklyn"] }, cuisine: { $nin: ["American", "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.

```
db.restaurants.find({ grades: { $elemMatch: { score: { $in: [2, 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/MV5BMTU3NjE5NzYtYTYYyNS00MDVmLWIwYjgtMmYwYWIxZDYYNzU2XkEyXkFqcGdeQXVyNzQzNzQxNzI@._V1_SY1000_SX677_AL_.jpg',
  title: 'The Great Train Robbery',
  fullplot: "Among the earliest existing films in American cinema - notable as the first film that presented a narrative story to tell - it depicts a group of cowboy outlaws who hold up a train and rob the passengers. They are then pursued by a Sheriff's posse. Several scenes have color included - all hand tinted."
}
```

```
languages: [ 'English' ],
released: ISODate("1903-12-01T00:00:00.000Z"),
directors: [ 'Edwin S. Porter' ],
rated: 'TV-G',
awards: { wins: 1, nominations: 0, text: '1 win.' },
lastupdated: '2015-08-13 00:27:59.177000000',
year: 1903,
imdb: { rating: 7.4, votes: 9847, id: 439 },
countries: [ 'USA' ],
type: 'movie',
tomatoes: {
viewer: { rating: 3.7, numReviews: 2559, meter: 75 },
fresh: 6,
critic: { rating: 7.6, numReviews: 6, meter: 100 },
rotten: 0,
lastUpdated: ISODate("2015-08-08T19:16:10.000Z")
}
```

1. Find all movies with full information from the 'movies' collection that released in the year 1893.

```
db.movies.find({ year: 1893 })
```

2. Find all movies with full information from the 'movies' collection that have a runtime greater than 120 minutes.

```
db.movies.find({ runtime: { $gt: 120 } })
```

3. Find all movies with full information from the 'movies' collection that have "Short" genre.

```
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.

```
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.

```
db.movies.find({ countries: "USA" })
```

6. Retrieve all movies from the 'movies' collection that have complete information and are rated as "UNRATED".

```
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.

```
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.

```
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.

```
db.movies.find({ "tomatoes.viewer.rating": { $gt: 4 } })
```

10. Retrieve all movies from the 'movies' collection that have received an award.

```
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.

```
db.movies.find({ "awards.nominations": { $gte: 1 } }, { 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".

```
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 })
```

13. Retrieve all movies with title, languages, released, directors, writers, countries from the 'movies' collection in MongoDB that released on May 9, 1893.

```
db.movies.find({ released: new Date("1893-05-09") }, { 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.

```
db.movies.find({ title: /scene/i }, { title: 1, languages: 1, released: 1, directors: 1, writers: 1, countries: 1 })
```

Evaluation Procedure	Marks awarded
----------------------	---------------

PL/SQL Procedure(5)	
Program/Execution (5)	
Viva(5)	
Total (15)	
Faculty Signature	

Ex.No.: 15	OTHER DATABASE OBJECTS
Date:	

OTHER DATABASE OBJECTS

Objectives

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

- Create, maintain, and use sequences
- Create and maintain indexes

Database Objects

Many applications require the use of unique numbers as primary key values. You can either build code into the application to handle this requirement or use a sequence to generate unique numbers.

If you want to improve the performance of some queries, you should consider creating an index. You

can also use indexes to enforce uniqueness on a column or a collection of columns.

You can provide alternative names for objects by using synonyms.

What Is a Sequence?

A sequence:

- Automatically generates unique numbers
- Is a sharable object
- Is typically used to create a primary key value
- Replaces application code
- Speeds up the efficiency of accessing sequence values when cached in memory

The CREATE SEQUENCE Statement Syntax

Define a sequence to generate sequential numbers automatically:

```
CREATE SEQUENCE sequence
[INCREMENT BY n]
[START WITH n]
[{:MAXVALUE n | 
NOMAXVALUE}] [{:MINVALUE
n | NOMINVALUE}] [{:CYCLE |
NOCYCLE}]
[{:CACHE n | NOCACHE}];
```

In the syntax:

sequence is the name of the sequence generator

INCREMENT BY *n* specifies the interval between sequence numbers where *n* is an integer (If this clause is omitted, the sequence increments by 1.)

START WITH *n* specifies the first sequence number to be generated (If this clause is omitted, the sequence starts with 1.)

MAXVALUE *n* specifies the maximum value the sequence can generate

NOMAXVALUE specifies a maximum value of 10^{27} for an ascending sequence and -1 for a descending sequence (This is the default option.)

MINVALUE *n* specifies the minimum sequence value

NOMINVALUE specifies a minimum value of 1 for an ascending sequence and - (10^{26}) for a descending sequence (This is the default option.)

CYCLE | NOCYCLE specifies whether the sequence continues to generate values after reaching its maximum or minimum value (NOCYCLE is the default option.)

CACHE *n* | NOCACHE specifies how many values the Oracle server preallocates and keep in memory (By default, the Oracle server caches 20 values.)

Creating a Sequence

- Create a sequence named DEPT_DEPTID_SEQ to be used for the primary key of the DEPARTMENTS table.
- Do not use the CYCLE option.

EXAMPLE:

```
CREATE SEQUENCE
dept_deptid_seq INCREMENT BY
10
START WITH 120
MAXVALUE
9999 NOCACHE
NOCYCLE;
```

Confirming Sequences

- Verify your sequence values in the USER_SEQUENCES data dictionary table.
- The LAST_NUMBER column displays the next available sequence number if NOCACHE is specified.

EXAMPLE:

```
SELECT sequence_name, min_value, max_value, increment_by, last_number
```

NEXTVAL and CURRVAL Pseudocolumns

- NEXTVAL returns the next available sequence value. It returns a unique value every time it is referenced, even for different users.
- CURRVAL obtains the current sequence value.
- NEXTVAL must be issued for that sequence before CURRVAL contains a value.

Rules for Using NEXTVAL and CURRVAL

You can use NEXTVAL and CURRVAL in the following contexts:

- The SELECT list of a SELECT statement that is not part of a subquery
- The SELECT list of a subquery in an INSERT statement
- The VALUES clause of an INSERT statement
- The SET clause of an UPDATE statement

You cannot use NEXTVAL and CURRVAL in the following contexts:

- The SELECT list of a view
- A SELECT statement with the DISTINCT keyword
- A SELECT statement with GROUP BY, HAVING, or ORDER BY clauses
- A subquery in a SELECT, DELETE, or UPDATE statement
- The DEFAULT expression in a CREATE TABLE or ALTER TABLE statement

Using a Sequence

- Insert a new department named ‘Support’ in location ID 2500.
- View the current value for the DEPT_DEPTID_SEQ sequence.

EXAMPLE:

```
INSERT INTO departments(department_id, department_name, location_id)
VALUES (dept_deptid_seq.NEXTVAL, 'Support', 2500);
```

```
SELECT dept_deptid_seq.CURRVAL FROM dual;
```

The example inserts a new department in the DEPARTMENTS table. It uses the DEPT_DEPTID_SEQ sequence for generating a new department number as follows:

You can view the current value of the sequence:

```
SELECT dept_deptid_seq.CURRVAL FROM dual;
```

Removing a Sequence

- Remove a sequence from the data dictionary by using the DROP SEQUENCE statement.
- Once removed, the sequence can no longer be referenced.

EXAMPLE:

```
DROP SEQUENCE dept_deptid_seq;
```

What is an Index?

An index:

- Is a schema object
- Is used by the Oracle server to speed up the retrieval of rows by using a pointer
- Can reduce disk I/O by using a rapid path access method to locate data quickly
- Is independent of the table it indexes
- Is used and maintained automatically by the Oracle server

How Are Indexes Created?

- Automatically: A unique index is created automatically when you define a PRIMARY KEY or UNIQUE constraint in a table definition.
- Manually: Users can create nonunique indexes on columns to speed up access to the rows.

Types of Indexes

Two types of indexes can be created. One type is a unique index: the Oracle server automatically creates this index when you define a column in a table to have a PRIMARY KEY or a UNIQUE key constraint. The name of the index is the name given to the constraint.

The other type of index is a nonunique index, which a user can create. For example, you can create a FOREIGN KEY column index for a join in a query to improve retrieval speed.

Creating an Index

- Create an index on one or more columns.
- Improve the speed of query access to the LAST_NAME column in the EMPLOYEES table.

```
CREATE INDEX index
  ON table (column[, column]...);
```

EXAMPLE:

```
CREATE INDEX
  emp_last_name_idx ON
  employees(last_name);
```

In the syntax:

index is the name of the index

table is the name of the table

column is the name of the column in the table to be indexed

When to Create an Index

You should create an index if:

- A column contains a wide range of values
- A column contains a large number of null values
- One or more columns are frequently used together in a WHERE clause or a join condition
- The table is large and most queries are expected to retrieve less than 2 to 4 percent of the rows

When Not to Create an Index

It is usually not worth creating an index if:

- The table is small
- The columns are not often used as a condition in the query
- Most queries are expected to retrieve more than 2 to 4 percent of the rows in the table
- The table is updated frequently
- The indexed columns are referenced as part of an Expression

Confirming Indexes

- The USER_INDEXES data dictionary view contains the name of the index and its uniqueness.
- The USER_IND_COLUMNS view contains the index name, the table name, and the column name.

EXAMPLE:

```
SELECT ic.index_name, ic.column_name, ic.column_position  
col_pos, ix.uniqueness FROM user_indexes ix, user_ind_columns ic  
WHERE ic.index_name = ix.index_name  
AND ic.table_name = 'EMPLOYEES';
```

Removing an Index

- Remove an index from the data dictionary by using the DROP INDEX command.
- Remove the UPPER_LAST_NAME_IDX index from the data dictionary.
- To drop an index, you must be the owner of the index or have the DROP ANY INDEX privilege.

```
DROP INDEX upper_last_name_idx;
```

```
DROP INDEX index;
```

Find the Solution for the following:

1. Create a sequence to be used with the primary key column of the DEPT table. The sequence should start at 200 and have a maximum value of 1000. Have your sequence increment by ten numbers. Name the sequence DEPT_ID_SEQ.

```
CREATE SEQUENCE DEPT_ID_SEQ INCREMENT BY 10 START WITH 200  
MAXVALUE 1000 NOCYCLE;
```

2. Write a query in a script to display the following information about your sequences: sequence name, maximum value, increment size, and last number

```
SELECT sequence_name, max_value, increment_by, last_number FROM user_sequences;
```

3. Write a script to insert two rows into the DEPT table. Name your script lab12_3.sql. Be sure to use the sequence that you created for the ID column. Add two departments named Education and Administration. Confirm your additions. Run the commands in your script.

```
INSERT INTO DEPT (ID, DEPARTMENT_NAME) VALUES (DEPT_ID_SEQ.NEXTVAL, 'Education');  
INSERT INTO DEPT (ID, DEPARTMENT_NAME) VALUES (DEPT_ID_SEQ.NEXTVAL, 'Administration');  
SELECT * FROM DEPT;
```

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

```
CREATE INDEX emp_dept_id_idx ON EMP(DEPT_ID);
```

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

```
SELECT ic.index_name, ic.column_name, ic.column_position AS col_pos, ix.uniqueness FROM  
user_indexes ix JOIN user_index_columns ic ON ic.index_name = ix.index_name WHERE ic.table_name = 'EMP';
```

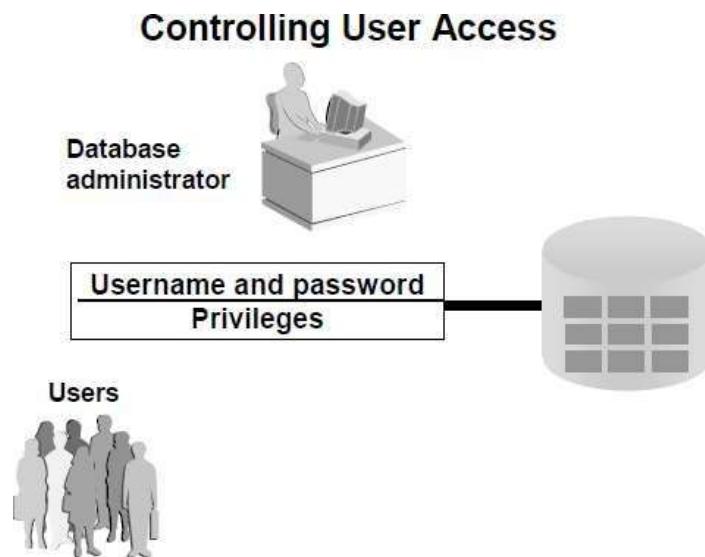
Evaluation Procedure	Marks awarded
PL/SQL Procedure(5)	
Program/Execution (5)	
Viva(5)	
Total (15)	
Faculty Signature	

Ex.No.: 16	CONTROLLING USER ACCESS
Date:	

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?

The user should be given the CREATE SESSION privilege. This is a system privilege.

2. What privilege should a user be given to create tables?

The user should be given the CREATE TABLE privilege.

3. If you create a table, who can pass along privileges to other users on your table?

Only the owner of the table (the user who created the table) can pass along privileges to other users on that table.

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?

You should create a role with the necessary privileges and then grant this role to each user.

5. What command do you use to change your password?

ALTER USER username IDENTIFIED BY new_password;

6. Grant another user access to your DEPARTMENTS table. Have the user grant you query access to his or her DEPARTMENTS table.

GRANT SELECT ON departments TO other_user;
GRANT SELECT ON departments TO original_user;

7. Query all the rows in your DEPARTMENTS table.

SELECT * FROM departments;

8. Add a new row to your DEPARTMENTS table. Team 1 should add Education as department number 500. Team 2 should add Human Resources department number 510. Query the other team's table.

INSERT INTO departments (department_id, department_name) VALUES (500, 'Education');
INSERT INTO departments (department_id, department_name) VALUES (510, 'Human Resources');

9. Query the USER_TABLES data dictionary to see information about the tables that you own.

SELECT * FROM other_team_user.departments;

10. Revoke the SELECT privilege on your table from the other team.

```
REVOKE SELECT ON departments FROM other_team_user;
```

11. Remove the row you inserted into the DEPARTMENTS table in step 8 and save the changes.

```
DELETE FROM departments WHERE department_id = 500; COMMIT;  
DELETE FROM departments WHERE department_id = 510; COMMIT;
```

Evaluation Procedure	Marks awarded
PL/SQL Procedure(5)	
Program/Execution (5)	
Viva(5)	
Total (15)	
Faculty Signature	

Logistics Management System

A MINI-PROJECT BY:

KRITHIKA B 230701156

KOMMANA SAI LEELA YUKTHA 230701154

in partial fulfillment of the award of the degree

OF

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE AND ENGINEERING



RAJALAKSHMI ENGINEERING COLLEGE, CHENNAI

An Autonomous Institute

CHENNAI

NOVEMBER 2024

BONAFIDE CERTIFICATE

Certified that this project report "**Logistics Management System**" is a Bonafide work of "**KRITHIKA B (230701156) & KOMMANA SAI LEELA YUKTHA (230701154)**".

Submitted for the Practical Examination held on _____ 26.11.2024 _____

Signature

Mrs. K MahesMeena

Assistant Professor,

Computer Science and Engineering,
Rajalakshmi Engineering College (Autonomous),
Thandalam, Chennai-602 105

Internal Examiner

External Examiner

LOGISTICS MANAGEMENT SYSTEM MINI PROJECT

ABSTARCT

Efficient logistics management is essential for businesses to maintain smooth operations and meet customer demands. This mini project presents a comprehensive solution to optimize inventory control, supplier coordination, and order processing. The system leverages real-time stock tracking with automatic low-stock alerts to ensure businesses can prevent stockouts or overstocking, thereby maintaining a balanced inventory.

The project integrates a centralized platform for supplier information, simplifying communication and restocking processes. The order management feature streamlines order placement and fulfillment, linking each order to specific customers and providing accurate tracking of payments and delivery statuses. These functionalities reduce errors and improve overall process efficiency.

Comprehensive inventory logs record all stock changes, including sales, restocking, and adjustments, ensuring transparency and accountability. Additionally, the system provides detailed analytics and reporting, offering insights into sales trends, inventory levels, and critical alerts. This empowers businesses to make informed, data-driven decisions.

By addressing common logistical challenges, this mini project aims to enhance operational efficiency, improve customer satisfaction, and support businesses in staying competitive in dynamic markets.

TABLE OF CONTENTS

1. INTRODUCTION:

- 1.1 INTRODUCTION
- 1.2 IMPLEMENTATION
- 1.3 SCOPE OF THE PROJECT
- 1.4 WEBSITE FEATURES

2. SYSTEM SPECIFICATION :

- 2.1 HARDWARE SPECIFICATION
- 2.2 SOFTWARE SPECIFICATION

3. ER DIAGRAM

4. SAMPLE CODE :

- 4.1 LOGIN PAGE
- 4.2 DASHBOARD
- 4.3 APP
- 4.4 CUSTOMER DATA
- 4.5 PRIMARY
- 4.6 PRODUCTS MODEL
- 4.7 MODULE INFO

5. SNAPSHOTS :

- 5.1 LOGIN PAGE
- 5.2 LOGIN SUCCESSFULLY
- 5.3 HOME PAGE
- 5.4 ENTERING ORDERS
- 5.5 ADDING PRODUCTS
- 5.6 LOGOUT SUCCESSFUL

6. CONCLUSION

7. REFERENCES

INTRODUCTION

1. INTRODUCTION

The project streamlines logistics management by offering businesses a centralized platform for inventory tracking, supplier coordination, and order processing. Users can monitor stock levels in real-time with alerts, ensuring optimal inventory control. Supplier details are centralized for seamless restocking, while orders are accurately tracked with payment and delivery status updates. Comprehensive logs and detailed analytics empower businesses to make informed decisions, improve efficiency, and enhance customer satisfaction.

1.1 IMPLEMENTATION

The **LOGISTICS MANAGEMENT** project discussed here is implemented using the concepts of **JAVA SWINGS** and **MYSQL**.

1.2 SCOPE OF THE PROJECT

The logistics management system is designed for businesses to register and create accounts, allowing them to manage all their inventory and order data in one organized platform. This ensures streamlined operations, saves time, and promotes a sense of professionalism.

1.3 WEBSITE FEATURES

- Registering and login page.
- Income data chart .
- Dashboard showing possible actions.
- Overview of all the products .

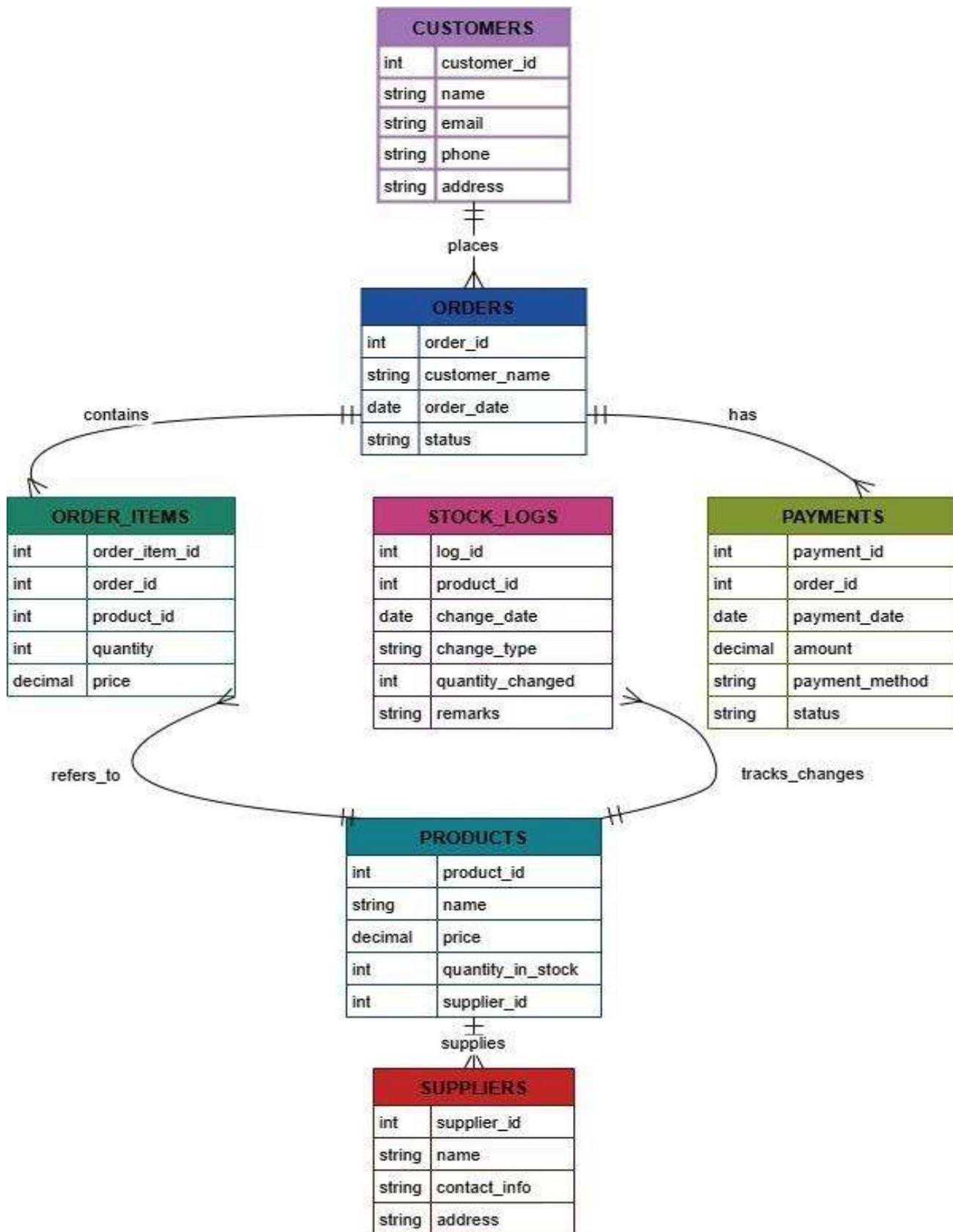
SYSTEM SPECIFICATIONS

2.1 HARDWARE SPECIFICATIONS:

PROCESSOR : Intel i5
MEMORY SIZE : 4GB(Minimum)
HARD DISK : 500 GB of free space

2.2 SOFTWARE SPECIFICATIONS:

PROGRAMMING LANGUAGE : Java, MySQL
FRONT-END : Java
BACK-END : MySQL
OPERATING SYSTEM : Windows 11



SAMPLE CODE

4.1 LOGIN PAGE

```
3 package com.mycompany.yt_inventorymanagementsystem_javafx;
4
5 import java.net.URL;
6 import java.sql.Connection;
7 import java.sql.PreparedStatement;
8 import java.sql.ResultSet;
9 import java.util.ResourceBundle;
10 import javafx.fxml.FXML;
11 import javafx.fxml.FXMLLoader;
12 import javafx.fxml.Initializable;
13 import javafx.scene.Parent;
14 import javafx.scene.Scene;
15 import javafx.scene.control.Alert;
16 import javafx.scene.control.Button;
17 import javafx.scene.control.PasswordField;
18 import javafx.scene.control.TextField;
19 import javafx.scene.input.MouseEvent;
20 import javafx.scene.layout.AnchorPane;
21 import javafx.stage.Stage;
22 import javafx.stage.StageStyle;
23
24 /**
25  * FXML Controller class
26  *
27  * @author USER
28 */
29 public class LoginController implements Initializable {
30
31     @FXML
32     private Button close_btn;
33
34     @FXML
35     private Button login_btn;
36
37     @FXML
38     private AnchorPane main_form;
39
40     @FXML
41     private PasswordField password_txt;
42
43     @FXML
44     private TextField username_txt;
45
46     // DATABASE TOOLS
47     private Connection con;
48     private PreparedStatement prepare;
49     private ResultSet result;
50     private double x = 0;
51     private double y = 0;
52
53     /**
54      * Initializes the controller class.
55      */
56     @Override
57     public void initialize(URL url, ResourceBundle rb) {
```

```
60
61     // LOGIN METHOD
62     public void loginAdmin() {
63         String sql = "select * from users where username = ? and password = ?";
64
65         // Try to establish a connection
66         con = databaseHandler.connectDb();
67
68         if (con == null) {
69             showAlert(Alert.AlertType.ERROR, "Database Connection Error", "Failed to connect to the
database.");
70             return;
71         }
72
73         try {
74             prepare = con.prepareStatement(sql);
75             prepare.setString(1, username_txt.getText());
76             prepare.setString(2, password_txt.getText());
77
78             result = prepare.executeQuery();
79
80             if (username_txt.getText().isEmpty() || password_txt.getText().isEmpty()) {
81                 showAlert(Alert.AlertType.ERROR, "Input Error", "Please fill all blank fields.");
82             } else {
83                 if (result.next()) {
84                     // If login is successful, store username for later use
85                     GetData.username = username_txt.getText();
86
87                     // Show success alert
88                     showAlert(Alert.AlertType.INFORMATION, "Login Success", "Successfully logged
in!");
89
90                     // Hide the login window
91                     login_btn.getScene().getWindow().hide();
92
93                     // Load the dashboard scene
94                     loadDashboard();
95                 } else {
96                     // If username/password is wrong
97                     showAlert(Alert.AlertType.ERROR, "Login Failed", "Wrong Username or Password.");
98                 }
99             }
100         } catch (Exception e) {
101             e.printStackTrace();
102             showAlert(Alert.AlertType.ERROR, "Error", "An error occurred: " + e.getMessage());
103         }
104     }
105
106     // Helper method to show alerts
107     private void showAlert(Alert.AlertType type, String title, String content) {
108         Alert alert = new Alert(type);
109         alert.setTitle(title);
110         alert.setHeaderText(null);
111         alert.setContentText(content);
112         alert.showAndWait();
113     }
114
115     // Load the dashboard
116     private void loadDashboard() {
117         try {
```

```

121         // Drag functionality for moving the window
122         root.setOnMousePressed((MouseEvent event) -> {
123             x = event.getSceneX();
124             y = event.getSceneY();
125         });
126
127         root.setOnMouseDragged((MouseEvent event) -> {
128             stage.setX(event.getScreenX() - x);
129             stage.setY(event.getScreenY() - y);
130         });
131
132         stage.initStyle(StageStyle.TRANSPARENT);
133         stage.setScene(scene);
134         stage.show();
135     } catch (Exception e) {
136         e.printStackTrace();
137         showAlert(Alert.AlertType.ERROR, "Error", "Unable to load the dashboard: " +
138             e.getMessage());
139     }
140 }
141
142     // Close the application
143     public void close() {
144         System.exit(0);
145     }
146 }
```

4.2 Dashboard

```

5 package com.mycompany.yt_inventorymanagementsystem_javafx;
6
7 import de.jensd.fx.glyphs.fontawesome.FontAwesomeIconView;
8 import java.io.File;
9 import java.net.URL;
10 import java.sql.Connection;
11 import java.sql.PreparedStatement;
12 import java.sql.ResultSet;
13 import java.sql.Statement;
14 import java.util.ArrayList;
15 import java.util.Date;
16 import java.util.HashMap;
17 import java.util.List;
18 import java.util.Optional;
19 import java.util.ResourceBundle;
20 import javafx.collections.FXCollections;
21 import javafx.collections.ObservableList;
22 import javafx.collections.transformation.FilteredList;
23 import javafx.collections.transformation.SortedList;
24 import javafx.event.ActionEvent;
25 import javafx.fxml.FXML;
26 import javafx.fxml.FXMLLoader;
27 import javafx.fxml.Initializable;
28 import javafx.scene.Parent;
29 import javafx.scene.Scene;
30 import javafx.scene.chart.AreaChart;
31 import javafx.scene.chart.BarChart;
32 import javafx.scene.chart.XYChart;
33 import javafx.scene.control.Alert;
34 import javafx.scene.control.Button;
35 import javafx.scene.control.ButtonType;
36 import javafx.scene.control.ComboBox;
```

```
39 import javafx.scene.control.SpinnerValueFactory;
40 import javafx.scene.control.TableColumn;
41 import javafx.scene.control.TableView;
42 import javafx.scene.control.TextField;
43 import javafx.scene.control.cell.PropertyValueFactory;
44 import javafx.scene.image.Image;
45 import javafx.scene.image.ImageView;
46 import javafx.scene.input.MouseEvent;
47 import javafx.scene.layout.AnchorPane;
48 import javafx.stage.FileChooser;
49 import javafx.stage.Stage;
50 import javafx.stage.StageStyle;
51 import net.sf.jasperreports.engine.JRException;
52 import net.sf.jasperreports.engine.JasperCompileManager;
53 import net.sf.jasperreports.engine.JasperFillManager;
54 import net.sf.jasperreports.engine.JasperPrint;
55 import net.sf.jasperreports.engine.JasperReport;
56 import net.sf.jasperreports.engine.design.JasperDesign;
57 import net.sf.jasperreports.engine.xml.JRXmlLoader;
58 import net.sf.jasperreports.view.JasperViewer;
59
60 /**
61  * FXML Controller class
62  *
63  * @author USER
64  */
65 public class DashboardController implements Initializable {
66
67     /**
68      * Initializes the controller class.
69      */
70     @FXML
71     private Button addProduct_add_btn;
72
73     @FXML
74     private TextField addProduct_brand_txt;
75
76     @FXML
77     private FontAwesomeIconView addProduct_btn;
78
79     @FXML
80     private Button addProduct_delete_btn;
81
82     @FXML
83     private ImageView addProduct_imgView;
84
85     @FXML
86     private Button addProduct_imort_btn;
87
88     @FXML
89     private TextField addProduct_price_txt;
90
91     @FXML
92     private TextField addProduct_productId_txt;
93
94     @FXML
95     private TextField addProduct_productName_txt;
96
97     @FXML
```

```
101     private Button addProduct_reset_btn;
102
103     @FXML
104     private TextField addProduct_search_txt;
105
106     @FXML
107     private ComboBox<?> addProduct_status_combo;
108
109     @FXML
110     private TableView<ProductsModel> addProduct_tableView;
111
112     @FXML
113     private TableColumn<ProductsModel, String> addProduct_tbvCol_brand;
114
115     @FXML
116     private TableColumn<ProductsModel, String> addProduct_tbvCol_price;
117
118     @FXML
119     private TableColumn<ProductsModel, String> addProduct_tbvCol_productID;
120
121     @FXML
122     private TableColumn<ProductsModel, String> addProduct_tbvCol_productName;
123
124     @FXML
125     private TableColumn<ProductsModel, String> addProduct_tbvCol_status;
126
127     @FXML
128     private TableColumn<ProductsModel, String> addProduct_tbvCol_type;
129
130     @FXML
131     private Button addProduct_update_btn;
132
133     @FXML
134     private Button addProducts_btn;
135
136     @FXML
137     private AnchorPane addProducts_form;
138
139     @FXML
140     private Button close_btn;
141
142     @FXML
143     private Label home_availableProducts_lbl;
144
145     @FXML
146     private Button home_btn;
147
148     @FXML
149     private AnchorPane home_form;
150
151     @FXML
152     private AreaChart<?, ?> home_incomeChart;
153
154     @FXML
155     private Label home_numberOfOrders_lbl;
156
157     @FXML
158     private BarChart<?, ?> home_orderChart;
159
160     @FXML
```

```
164     private Button logout;
165
166     @FXML
167     private AnchorPane main_form;
168
169     @FXML
170     private FontAwesomeIconView minimize_btn;
171
172     @FXML
173     private Button orders_add_btn;
174
175     @FXML
176     private TextField orders_amount_txt;
177
178     @FXML
179     private Label orders_balance_lbl;
180
181     @FXML
182     private ComboBox<?> orders_brandName_combo;
183
184     @FXML
185     private Button orders_btn;
186
187     @FXML
188     private TableColumn<CustomerData, String> orders_col_brand;
189
190     @FXML
191     private TableColumn<CustomerData, String> orders_col_price;
192
193     @FXML
194     private TableColumn<CustomerData, String> orders_col_productName;
195
196     @FXML
197     private TableColumn<CustomerData, String> orders_col_quantity;
198
199     @FXML
200     private TableColumn<CustomerData, String> orders_col_type;
201
202     @FXML
203     private AnchorPane orders_form;
204
205     @FXML
206     private Button orders_pay_btn;
207
208     @FXML
209     private ComboBox<?> orders_productName_combo;
210
211     @FXML
212     private ComboBox<?> orders_productType_combo;
213
214     @FXML
215     private Spinner<Integer> orders_quantity_spinner;
216
217     @FXML
218     private Button orders_receipt_btn;
219
220     @FXML
221     private Button orders_reset_btn;
222
223     @FXML
```

```
227     private Label orders_total_lbl;
228
229     @FXML
230     private Label username_lbl;
231
232     // DATABASE TOOLS
233     private Connection conn;
234     private PreparedStatement prepare;
235     private Statement statement;
236     private ResultSet resultSet;
237
238
239     public void homeDisplayTotalOrders(){
240 /**
241 //         Date date = new Date();
242 //         java.sql.Date sqlDate = new java.sql.Date(date.getTime());
243 //         prepare.setString(8, String.valueOf(sqlDate));
244
245     Date date = new Date();
246     java.sql.Date sqlDate = new java.sql.Date(date.getTime());
247
248 /**
249     String sql = "SELECT COUNT(id) AS count_of_id FROM customer_receipt";
250     String sql = "SELECT COUNT(id) FROM customer_receipt WHERE date = '"+sqlDate+"' ";
251
252     conn = databaseHandler.connectDb();
253
254     int countOrders = 0;
255
256     try {
257
258         prepare = conn.prepareStatement(sql);
259         resultSet = prepare.executeQuery();
260
261 /**
262         if(resultSet.next()){
263             countOrders = resultSet.getInt("count_of_id");
264             countOrders = resultSet.getInt("COUNT(id)");
265         }
266
267         home_numberOfOrders_lbl.setText(String.valueOf(countOrders));
268
269     } catch (Exception e) {e.printStackTrace();}
270
271 }
272
273     public void homeTotalIncome(){
274
275         String sql = "SELECT SUM(total) AS total_income FROM customer_receipt";
276
277         conn = databaseHandler.connectDb();
278
279         double totalIncome = 0;
280
281         try {
282
283             prepare = conn.prepareStatement(sql);
284             resultSet = prepare.executeQuery();
285
286             if(resultSet.next()){
287                 totalIncome = resultSet.getDouble("total_income");
288             }
289         }
290     }
291 }
```

```
289     } catch (Exception e) {e.printStackTrace();}
290
291 }
292
293 public void homeAvailableProducts(){
294
295     String sql = "SELECT COUNT(id) AS ava_products FROM product WHERE status = 'Available'";
296
297     conn = databaseHandler.connectDb();
298
299     int availableProducts = 0;
300
301     try {
302
303         prepare = conn.prepareStatement(sql);
304         resultSet = prepare.executeQuery();
305
306         if(resultSet.next()){
307             availableProducts = resultSet.getInt("ava_products");
308         }
309
310         home_availableProducts_lbl.setText(String.valueOf(availableProducts));
311
312     } catch (Exception e) {e.printStackTrace();}
313
314 }
315
316
317 public void homeIncomeChart(){
318
319     home_incomeChart.getData().clear();
320
321 //     String sql = "SELECT date, SUM(total) FROM customer_receipt GROUP BY date GROUP BY
322 //     TIMESTAMP(date) ASC LIMIT 6";
323
324     String sql = "SELECT date, SUM(total) FROM customer_receipt GROUP BY date ORDER BY
325     TIMESTAMP(date) ASC LIMIT 6;";
326
327     conn = databaseHandler.connectDb();
328
329     try {
330
331         XYChart.Series chart = new XYChart.Series();
332
333         prepare = conn.prepareStatement(sql);
334         resultSet = prepare.executeQuery();
335
336         while(resultSet.next()){
337             chart.getData().add(new XYChart.Data(resultSet.getString(1), resultSet.getInt(2)));
338         }
339
340         home_incomeChart.getData().add(chart);
341
342     } catch (Exception e) {e.printStackTrace();}
343
344
345 public void homeOrdersChart(){
346
347     home_orderChart.getData().clear();
```

```

349
350     /*
351      SELECT date, SUM(total): We ask for two things: the date and the sum of income for each date.
352      The SUM(total) adds up all income values for each date, so we get the total income on that day.
353      FROM customer_receipt: This tells the database to look in the customer_receipt table.
354      GROUP BY date: It groups the data by date, so it only shows one total amount per date.
355      ORDER BY TIMESTAMP(date) ASC: Orders the data by date from oldest to newest.
356      LIMIT 6: Only shows the last 6 days (or dates) of data.
357     */
358
359     conn = databaseHandler.connectDb();
360
361     try {
362
363         XYChart.Series chart = new XYChart.Series<>(); // This creates an empty series, which is
364         like a container that will hold all the data points for the chart.
365
366         prepare = conn.prepareStatement(sql);
367         resultSet = prepare.executeQuery();
368
369         while(resultSet.next()){
370             chart.getData().add(new XYChart.Data(resultSet.getString(1), resultSet.getInt(2)));
371
372             /*
373             resultSet.next(): Moves to the next row of data.
374             resultSet.getString(1): Gets the date from the first column.
375             resultSet.getInt(2): Gets the total income for that date from the second column.
376             */
377
378         }
379
380         home_orderChart.getData().add(chart);
381
382     } catch (Exception e) {e.printStackTrace();}
383
384     public void addProductsAdd() {
385
386         String sql = "INSERT INTO product (product_id, type, brand, product_name, price, status, image,
387         date) "
388             + "VALUES(?,?,?,?,?,?)";
389
390         conn = databaseHandler.connectDb();
391
392         try {
393
394             Alert alert;
395
396             if (addProduct_productId_txt.getText().isEmpty()
397                 || addProduct_productType_combo.getSelectionModel().getSelectedItem() == null
398                 || addProduct_brand_txt.getText().isEmpty()
399                 || addProduct_productName_txt.getText().isEmpty()
400                 || addProduct_price_txt.getText().isEmpty()
401                 || addProduct_status_combo.getSelectionModel().getSelectedItem() == null
402                 || GetData.path == "") {
403
404                 alert = new Alert(Alert.AlertType.ERROR);
405                 alert.setTitle("Error Message");
406                 alert.setHeaderText(null);

```

```

409         } else {
410
411             // CHECK IF THE PRODUCT ID IS ALREADY EXIST
412             String checkData = "SELECT product_id FROM product WHERE product_id = '" +
413                 + addProduct_productId_txt.getText() + "'";
414
415             statement = conn.createStatement();
416             resultSet = statement.executeQuery(checkData);
417
418             if (resultSet.next()) {
419
420                 alert = new Alert(Alert.AlertType.ERROR);
421                 alert.setTitle("Error Message");
422                 alert.setHeaderText(null);
423                 alert.setContentText("Product ID: " + addProduct_productId_txt.getText() + " was
already exist!");
424                 alert.showAndWait();
425             } else {
426
427                 prepare = conn.prepareStatement(sql);
428                 prepare.setString(1, addProduct_productId_txt.getText());
429                 prepare.setString(2, (String)
addProduct_productType_combo.getSelectionModel().getSelectedItem());
430                 prepare.setString(3, addProduct_brand_txt.getText());
431                 prepare.setString(4, addProduct_productName_txt.getText());
432                 prepare.setString(5, addProduct_price_txt.getText());
433                 prepare.setString(6, (String)
addProduct_status_combo.getSelectionModel().getSelectedItem());
434
435                 String uri = GetData.path;
436                 uri = uri.replace("\\", "\\\\");
437                 prepare.setString(7, uri);
438
439                 Date date = new Date();
440                 java.sql.Date sqlDate = new java.sql.Date(date.getTime());
441                 prepare.setString(8, String.valueOf(sqlDate));
442
443                 prepare.executeUpdate();
444
445             /*
446             Difference Between executeUpdate() and executeQuery():
447             executeUpdate():
448
449                 This method is used for modifying the database. It is typically used with SQL
statements such as INSERT, UPDATE, DELETE, and CREATE. These types of statements alter the contents or
structure of the database but do not return data in the form of a result set.
450                 It returns an integer representing the number of rows affected by the operation.
451             executeQuery():
452
453                 This method is used for retrieving data from the database. It is typically used
with SELECT statements, where the query returns a result set (data retrieved from the database).
454                 It returns a ResultSet object, which contains the data retrieved by the query.
455                 */
456                 // TO BECOME UPDATED YOUR TABLEVIEW
457                 addProductsShowListData();
458
459                 // CLEAR THE FIELDS
460                 addProductsReset();
461
462             }

```

```

466         e.printStackTrace();
467     }
468
469 }
470
471 private String[] listType = {"Snacks", "Drinks", "Dessert", "Gadgets", "Personal Product",
472 "Others"};
473
474 public void addProductsListTypes() {
475
476     List<String> listT = new ArrayList<>();
477
478     for (String data : listType) {
479         listT.add(data);
480     }
481
482     ObservableList listData = FXCollections.observableArrayList(listT);
483     addProduct_productType_combo.setItems(listData);
484 }
485
486 private String[] listStatus = {"Available", "Not Available"};
487
488 public void addProductsListStatus() {
489     List<String> listS = new ArrayList<>();
490
491     for (String data : listStatus) {
492         listS.add(data);
493     }
494
495     ObservableList listData = FXCollections.observableArrayList(listS);
496     addProduct_status_combo.setItems(listData);
497 }
498
499 public void addproductsSelectedItemsToInputFields() {
500
501     ProductsModel productsData = addProduct_tableView.getSelectionModel().getSelectedItem(); //  

502     TableView<ProductsModel> addProduct_tableView; nisa return wenneth ProductsModel object ekak  

503     int num = addProduct_tableView.getSelectionModel().getSelectedIndex();
504
505     if ((num - 1) < -1) {
506         return;
507     } // This condition checks if the selected index (num) is invalid. If no row is selected (i.e.,  

508     num is -1), the method returns and exits without doing anything. The num - 1 < -1 check prevents trying  

509     to access an invalid row.
510
511     addProduct_productId_txt.setText(String.valueOf(productsData.getProductId())); //  

512     addProductsShowListData() Read this method to understand how the variable of the ProductsModel.java  

513     class matched to the column of the table
514
515     addProduct_productType_combo.getSelectionModel().clearSelection();
516     addProduct_brand_txt.setText(productsData.getBrand());
517     addProduct_productName_txt.setText(productsData.getProductName());
518     addProduct_price_txt.setText(String.valueOf(productsData.getPrice()));
519     addProduct_status_combo.getSelectionModel().clearSelection();
520
521     String uri = "file:" + productsData.getImage();
522
523     image = new Image(uri, 115, 128, false, true);
524     addProduct_imgView.setImage(image);
525
526 }

```

```
the path is empty, by clicking a row in the table, the path of the image can be set to the path of
GetData.java.
521     }
522
523     public void addProductsUpdate() {
524
525         String uri = GetData.path;
526         uri = uri.replace("\\", "\\\\");
527
528         Date date = new Date();
529         java.sql.Date sqlDate = new java.sql.Date(date.getTime());
530
531         String sql = "UPDATE product SET type = ''"
532             + addProduct_productType_combo.getSelectionModel().getSelectedItem()
533             + ', brand = \'' + addProduct_brand_txt.getText()
534             + '\', product_name = \'' + addProduct_productName_txt.getText()
535             + '\', price = \'' + addProduct_price_txt.getText()
536             + '\', status = \'' + addProduct_status_combo.getSelectionModel().getSelectedItem()
537             + '\', image = \'' + uri
538             + '\', date = \'' + sqlDate
539             + '\' WHERE product_id = \'' + addProduct_productId_txt.getText() + '\'';
540
541         conn = databaseHandler.connectDb();
542
543         try {
544
545             Alert alert;
546
547             if (addProduct_productId_txt.getText().isEmpty()
548                 || addProduct_productType_combo.getSelectionModel().getSelectedItem() == null
549                 || addProduct_brand_txt.getText().isEmpty()
550                 || addProduct_productName_txt.getText().isEmpty()
551                 || addProduct_price_txt.getText().isEmpty()
552                 || addProduct_status_combo.getSelectionModel().getSelectedItem() == null
553                 || GetData.path == "") {
554
555                 alert = new Alert(Alert.AlertType.ERROR);
556                 alert.setTitle("Error Message");
557                 alert.setHeaderText(null);
558                 alert.setContentText("Please fill all blank fields");
559                 alert.showAndWait();
560
561             } else {
562
563                 alert = new Alert(Alert.AlertType.CONFIRMATION);
564                 alert.setTitle("Confirmation Message");
565                 alert.setHeaderText(null);
566                 alert.setContentText("Are you sure you want to UPDATE Product ID: " +
567                     addProduct_productId_txt.getText() + " ?");
568
569                 Optional<ButtonType> option = alert.showAndWait();
570
571                 if (option.get().equals(ButtonType.OK)) {
572
573                     statement = conn.createStatement();
574                     statement.executeUpdate(sql);
575
576                     alert = new Alert(Alert.AlertType.INFORMATION);
577                     alert.setTitle("Information Message");
578                     alert.setHeaderText(null);
579
580                 }
581             }
582         }
583     }
584 }
```

```
581             // TO BECOME UPDATED YOUR TABLEVIEW
582             addProductsShowListData();
583
584             // CLEAR THE FIELDS
585             addProductsReset();
586         }
587     }
588
589 } catch (Exception e) {
590     e.printStackTrace();
591 }
592 }
593
594 public void addProductsDelete() {
595
596     String sql = "DELETE FROM product WHERE product_id = '" + addProduct_productId_txt.getText() +
597     "'";
598     conn = databaseHandler.connectDb();
599
600     try {
601
602         Alert alert;
603
604         if (addProduct_productId_txt.getText().isEmpty()
605             || addProduct_productType_combo.getSelectionModel().getSelectedItem() == null
606             || addProduct_brand_txt.getText().isEmpty()
607             || addProduct_productName_txt.getText().isEmpty()
608             || addProduct_price_txt.getText().isEmpty()
609             || addProduct_status_combo.getSelectionModel().getSelectedItem() == null
610             || GetData.path == "") {
611
612             alert = new Alert(Alert.AlertType.ERROR);
613             alert.setTitle("Error Message");
614             alert.setHeaderText(null);
615             alert.setContentText("Please fill all blank fields");
616             alert.showAndWait();
617
618         } else {
619
620             alert = new Alert(Alert.AlertType.CONFIRMATION);
621             alert.setTitle("Confirmation Message");
622             alert.setHeaderText(null);
623             alert.setContentText("Are you sure you want to DELETE Product ID: " +
624             addProduct_productId_txt.getText() + " ?");
625
626             Optional<ButtonType> option = alert.showAndWait();
627
628             if (option.get().equals(ButtonType.OK)) {
629
630                 statement = conn.createStatement();
631                 statement.executeUpdate(sql);
632
633                 alert = new Alert(Alert.AlertType.INFORMATION);
634                 alert.setTitle("Information Message");
635                 alert.setHeaderText(null);
636                 alert.setContentText("Successfully Deleted");
637                 alert.showAndWait();
638
639             // TO BECOME UPDATED YOUR TABLEVIEW
640             addProductsShowListData();
641
642         }
643
644     }
645
646 }
```

```
642         }
643     }
644 } catch (Exception e) {
645     e.printStackTrace();
646 }
647 }
648
649 public void addProductsReset() {
650
651     addProduct_productId_txt.setText("");
652     addProduct_productType_combo.getSelectionModel().clearSelection();
653     addProduct_brand_txt.setText("");
654     addProduct_productName_txt.setText("");
655     addProduct_price_txt.setText("");
656     addProduct_status_combo.getSelectionModel().clearSelection();
657     addProduct_imgView.setImage(null);
658     GetData.path = "";
659 }
660
661 private Image image;
662
663 public void addProductsImportImage() {
664
665     FileChooser open = new FileChooser(); // This creates a new FileChooser object, which is a
JavaFX component that opens a dialog window to let the user browse and choose files from their
computer.
666     open.setTitle("Open Image File");
667     open.getExtensionFilters().add(new FileChooser.ExtensionFilter("Image File", "*jpg", "*png"));
// This line adds a filter to the FileChooser, so only image files with the extensions .jpg and .png
are shown in the file browser. The user will only be able to select these types of image files.
668
669     File file = open.showOpenDialog(main_form.getScene().getWindow()); // This is the method that
opens the file dialog window, allowing the user to browse and select an image file.
670
671     /*
672      --What is a Modal Window?--
673      A modal window is a type of window that temporarily blocks the interaction with the parent
window (the main application window) until the modal window is closed. This means:
674
675      While the modal window (in this case, the file dialog) is open, the user cannot interact with
the main window (e.g., click buttons, type in text fields) until they either select a file or close the
dialog.
676      This is helpful to ensure that the user completes the current action (choosing an image) before
interacting with the rest of the application.
677
678      ---Why Specify the Parent Window?--
679      The argument main_form.getScene().getWindow() is used to indicate the parent window of the file
dialog, which ties the file dialog to the main application window. Here's why this is important:
680
681      Blocking interaction with the main window:
682
683      When the file dialog is modal, it prevents the user from interacting with the parent window
until they complete the task in the dialog (choosing a file or closing the dialog). This ensures the
user focuses on selecting an image before doing anything else in the app.
684      Dialog behavior and position:
685
686      By specifying the parent window, the file dialog will appear centered relative to the main
window. It also ensures that when you minimize or move the main window, the file dialog will minimize
or move along with it.
```

```

690     If the user could interact with the main window while the file dialog was open, they might
691     trigger unintended behaviors (such as clicking other buttons or opening additional dialogs), which
692     could lead to confusion or errors in the application.
693
694     /*
695      if (file != null) {
696
697         GetData.path = file.getAbsolutePath(); // On Windows, an absolute path might look like
698         // this: C:\Users\John\Pictures\image.jpg
699
700     }
701 }
702
703 public void addProductsSearch() {
704
705     FilteredList<ProductsModel> filter = new FilteredList<>(addProductList, e -> true);
706
707     addProduct_search_txt.textProperty().addListener((Observable, oldValue, newValue) -> {
708
709         filter.setPredicate(predicateProductData -> {
710
711             if (newValue == null || newValue.isEmpty()) {
712                 return true;
713             }
714
715             String searchKey = newValue.toLowerCase();
716
717             if (predicateProductData.getProductId().toString().contains(searchKey)) {
718                 return true;
719             } else if (predicateProductData.getType().toLowerCase().contains(searchKey)) {
720                 return true;
721             } else if (predicateProductData.getBrand().toLowerCase().contains(searchKey)) {
722                 return true;
723             } else if (predicateProductData.getProductName().toLowerCase().contains(searchKey)) {
724                 return true;
725             } else if (predicateProductData.getPrice().toString().contains(searchKey)) {
726                 return true;
727             } else if (predicateProductData.getStatus().toLowerCase().contains(searchKey)) {
728                 return true;
729             } else {
730                 return false;
731             }
732         });
733     });
734
735     SortedList<ProductsModel> sortList = new SortedList<>(filter);
736
737     sortList.comparatorProperty().bind(addProduct_tableView.comparatorProperty());
738     addProduct_tableView.setItems(sortList);
739
740 }
741
742 public ObservableList<ProductsModel> addProductListDataGet() {
743
744     ObservableList<ProductsModel> productList = FXCollections.observableArrayList();
745
746     String sql = "SELECT * FROM product";

```

```

750
751     prepare = conn.prepareStatement(sql);
752     resultSet = prepare.executeQuery();
753     ProductsModel proData;
754
755     while (resultSet.next()) {
756
757         proData = new ProductsModel(
758             resultSet.getInt("product_id"),
759             resultSet.getString("type"),
760             resultSet.getString("brand"),
761             resultSet.getString("product_name"),
762             resultSet.getDouble("price"),
763             resultSet.getString("status"),
764             resultSet.getString("image"),
765             resultSet.getDate("date")
766         );
767
768         productList.add(proData);
769     }
770
771 } catch (Exception e) {
772     e.printStackTrace();
773 }
774
775 return productList;
776
777 }
778
779 private ObservableList<ProductsModel> addProductList;
780 //    public void addProductsShowListData(){
781 //
782 //        addProductList = addProductListDataGet();
783 //
784 //        addProduct_tbvCol_productID.setCellValueFactory(new PropertyValueFactory<>("product_id")); // table columns name of database
785 //        addProduct_tbvCol_type.setCellValueFactory(new PropertyValueFactory<>("type"));
786 //        addProduct_tbvCol_brand.setCellValueFactory(new PropertyValueFactory<>("brand"));
787 //        addProduct_tbvCol_productName.setCellValueFactory(new PropertyValueFactory<>("product_name"));
788 //        addProduct_tbvCol_price.setCellValueFactory(new PropertyValueFactory<>("price"));
789 //        addProduct_tbvCol_status.setCellValueFactory(new PropertyValueFactory<>("status"));
790 //
791 //        addProduct_tableView.setItems(addProductList);
792 //    }
793
794 public void addProductsShowListData() { // show on table
795
796     addProductList = addProductListDataGet();
797
798     addProduct_tbvCol_productID.setCellValueFactory(new PropertyValueFactory<>("productId")); // productId is the variable name of the ProductsModel class
799     addProduct_tbvCol_type.setCellValueFactory(new PropertyValueFactory<>("type")); // this does not immediately "put all the data related to the product ID at once." Instead, what it does is link the productId field from each product in your list to the Product ID column of the TableView
800     addProduct_tbvCol_brand.setCellValueFactory(new PropertyValueFactory<>("brand"));
801     addProduct_tbvCol_productName.setCellValueFactory(new PropertyValueFactory<>("productName"));
802     addProduct_tbvCol_price.setCellValueFactory(new PropertyValueFactory<>("price"));
803     addProduct_tbvCol_status.setCellValueFactory(new PropertyValueFactory<>("status"));
804

```

```
808     public void ordersAdd() {
809
810         customerId();
811         String sql = "INSERT INTO customer (customer_id, type, brand, productName, quantity, price,
812         date) "
813             + "VALUES(?, ?, ?, ?, ?, ?)";
814
815         conn = databaseHandler.connectDb();
816
817         try {
818
819             String checkData = "SELECT * FROM product WHERE product_name = '"
820                 + orders_productName_combo.getSelectionModel().getSelectedItem() + "'";
821
822             double priceData = 0;
823
824             statement = conn.createStatement();
825             resultSet = statement.executeQuery(checkData);
826
827             if (resultSet.next()) {
828                 priceData = resultSet.getDouble("price"); // if there are more prices more than 1 then
829                 what happen?????
830             }
831
832             double totalPdata = (priceData * qty);
833
834             Alert alert;
835             if (orders_productType_combo.getSelectionModel().getSelectedItem() == null
836                 || orders_brandName_combo.getSelectionModel().getSelectedItem() == null
837                 || orders_productName_combo.getSelectionModel().getSelectedItem() == null
838                 || totalPdata == 0) {
839                 alert = new Alert(Alert.AlertType.ERROR);
840                 alert.setTitle("Error Message");
841                 alert.setHeaderText(null);
842                 alert.setContentText("Please choose product first");
843                 alert.showAndWait();
844             } else {
845
846                 prepare = conn.prepareStatement(sql);
847                 prepare.setInt(1, customer_Id);
848                 prepare.setString(2, (String)
849                     orders_productType_combo.getSelectionModel().getSelectedItem());
850                     prepare.setString(3, (String)
851                     orders_brandName_combo.getSelectionModel().getSelectedItem());
852                     prepare.setString(4, (String)
853                     orders_productName_combo.getSelectionModel().getSelectedItem());
854                     prepare.setString(5, String.valueOf(qty));
855
856                     prepare.setString(6, String.valueOf(totalPdata));
857
858                     Date date = new Date();
859                     java.sql.Date sqlDate = new java.sql.Date(date.getTime());
860                     prepare.setString(7, String.valueOf(sqlDate));
861
862                     prepare.executeUpdate();
863
864                     ordersShowListData();
865                     ordersDisplayTotal();
866             }
867         }
868     }
869 }
```

```
866         e.printStackTrace();
867     }
868 }
869 }
870 }
871 public void ordersPay() {
872
873     customerId();
874     String sql = "INSERT INTO customer_receipt (customer_id, total, amount, balance, date) "
875             + "VALUES(?, ?, ?, ?, ?)" // WHERE customer_id = '"+ customer_Id +'" // change this I
876
877     conn = databaseHandler.connectDb();
878
879     try {
880
881         Alert alert;
882         if (totalP > 0) { // || orders_amount_txt.getText().isEmpty() || amount == 0
883
884             alert = new Alert(Alert.AlertType.CONFIRMATION);
885             alert.setTitle("Confirmation Message");
886             alert.setHeaderText(null);
887             alert.setContentText("Are you sure?");
888             Optional<ButtonType> optional = alert.showAndWait();
889
890             if (optional.get().equals(ButtonType.OK)) {
891
892                 prepare = conn.prepareStatement(sql);
893
894                 prepare.setInt(1, customer_Id); // I change this
895                 prepare.setString(2, String.valueOf(totalP));
896                 prepare.setString(3, String.valueOf(amount));
897                 prepare.setString(4, String.valueOf(balance));
898
899                 Date date = new Date();
900                 java.sql.Date sqlDate = new java.sql.Date(date.getTime());
901                 prepare.setString(5, String.valueOf(sqlDate));
902
903                 prepare.executeUpdate();
904
905                 alert = new Alert(Alert.AlertType.INFORMATION);
906                 alert.setTitle("Information Message");
907                 alert.setHeaderText(null);
908                 alert.setContentText("Successfully Payed!");
909                 alert.showAndWait();
910
911 //                 ordersShowListData();
912
913 //                 totalP = 0;
914                 balance = 0;
915                 amount = 0;
916
917                 orders_balance_lbl.setText("Rs 0.0");
918                 orders_total_lbl.setText("Rs 0.0");
919                 orders_amount_txt.setText("");
920
921             } else {
922                 return;
923             }
924
925     } else {
```

```

929         alert.setHeaderText(null);
930         alert.setContentText("Invalid : 3");
931         alert.showAndWait();
932     }
933 }
934 } catch (Exception e) {
935     e.printStackTrace();
936 }
937 }
938 }
939 }
940 }
941 public void ordersReceipt() throws JRException{
942     try {
943         customerId();
944
945 //        customerId();
946
947         HashMap hash = new HashMap();
948         hash.put("inventoryP", customer_Id);
949
950         if(totalP == 0){
951
952             Alert alert = new Alert(Alert.AlertType.ERROR);
953             alert.setTitle("Error Message");
954             alert.setHeaderText(null);
955             alert.setContentText("Invalid :3"); // kalin order eka pay karala iwara nam den totalP
 = 0 ema nisa receipt ekak print kranna denne aye order ekak daddi
956             alert.showAndWait();
957
958         }else{
959
960             JasperDesign jDesign =
961             JRXmlLoader.load("C:\\\\Users\\\\krith\\\\Downloads\\\\YT_InventoryManagementSystem_JavaFx-
main\\\\YT_InventoryManagementSystem_JavaFx-main\\\\report.jrxml");
962             JasperReport jReport = JasperCompileManager.compileReport(jDesign);
963             JasperPrint jPrint = JasperFillManager.fillReport(jReport, hash, conn);
964
965             JasperViewer.viewReport(jPrint, false);
966
967         }
968
969     } catch (Exception e) {e.printStackTrace();}
970 }
971 }
972 }
973 public void ordersReset(){
974
975     customerId();
976
977     String sql = "DELETE FROM customer WHERE customer_id = '" + customer_Id + "'";
978
979     conn = databaseHandler.connectDb();
980
981     try {
982         if(!orders_tableView.getItems().isEmpty()){
983
984             Alert alert = new Alert(Alert.AlertType.CONFIRMATION);
985             alert.setTitle("Confirmation Message");

```

```
989             if(option.get().equals(ButtonType.OK)){
990
991                 statement = conn.createStatement();
992                 statement.executeUpdate(sql);
993
994                 ordersShowListData();
995
996                 totalP = 0;
997                 balance = 0;
998                 amount = 0;
999
1000                orders_balance_lbl.setText("Rs 0.0");
1001                orders_total_lbl.setText("Rs 0.0");
1002                orders_amount_txt.setText("");
1003
1004            }
1005
1006        }
1007    } catch (Exception e) {e.printStackTrace();}
1008
1009 }
1010
1011 private double amount;
1012 private double balance;
1013
1014 public void ordersAmount() {
1015
1016     Alert alert;
1017
1018     if (!orders_amount_txt.getText().isEmpty()) {
1019
1020         amount = Double.parseDouble(orders_amount_txt.getText());
1021
1022         if (totalP > 0) {
1023             if (amount >= totalP) {
1024                 balance = (amount - totalP);
1025                 orders_balance_lbl.setText("Rs " + String.valueOf(balance));
1026             } else {
1027                 alert = new Alert(Alert.AlertType.ERROR);
1028                 alert.setHeaderText(null);
1029                 alert.setContentText("Enter Enough Amount");
1030                 alert.showAndWait();
1031
1032                 orders_amount_txt.setText("");
1033             }
1034         } else {
1035
1036             alert = new Alert(Alert.AlertType.ERROR);
1037             alert.setHeaderText(null);
1038             alert.setContentText("Invalid :3");
1039             alert.showAndWait();
1040
1041         }
1042     } else {
1043
1044         alert = new Alert(Alert.AlertType.ERROR);
1045         alert.setHeaderText(null);
1046         alert.setContentText("Amount Field is Empty");
1047         alert.showAndWait();
1048     }
1049 }
```

```

1052
1053     private double totalP;
1054
1055     public void ordersDisplayTotal() {
1056         customerId();
1057
1058         String sql = "SELECT SUM(price) FROM customer WHERE customer_id = '" + customer_Id + "'"
1059         "; // mehid ee cus id of customer table is 1, and cus id of customer_receipt is 0, customerId() ekedi
1060         // customer table eke customer id eka select krala gannwa, eka thamay methana thiennen "+customer_Id+
1061         // lesa.
1062
1063         conn = databaseHandler.connectDb();
1064
1065         try {
1066
1067             prepare = conn.prepareStatement(sql);
1068             resultSet = prepare.executeQuery();
1069
1070             while (resultSet.next()) {
1071                 totalP = resultSet.getDouble("SUM(price)");
1072             }
1073
1074             orders_total_lbl.setText("Rs " + String.valueOf(totalP));
1075
1076         } catch (Exception e) {
1077             e.printStackTrace();
1078         }
1079
1080         private String[] OrderslistType = {"Snacks", "Drinks", "Dessert", "Gadgets", "Personal
1081         Product", "Others"};
1082
1083         public void OrdersListTypes() {
1084             // orders_productType_combo.getSelectionModel().clearSelection();
1085             List<String> listT = new ArrayList<>();
1086
1087             for (String data : OrderslistType) {
1088                 listT.add(data);
1089             }
1090
1091             ObservableList listData = FXCollections.observableArrayList(listT);
1092             orders_productType_combo.setItems(listData);
1093
1094             ordersListBrands();
1095         }
1096
1097         public void ordersListBrands() { // this method call after type selected, but when selected
1098             // brand name combo we want call again this method, because we want to show product name combo list
1099             // Clear the product name combo box when a new type is selected
1100             orders_productName_combo.getSelectionModel().clearSelection();
1101             orders_productName_combo.getItems().clear();
1102
1103             // Check if brand selection exists before executing the query
1104             if (orders_productType_combo.getSelectionModel().getSelectedItem() == null) {
1105                 return; // Exit the method if no brand is selected
1106             }

```

```

1110             + " ' and status = 'Available'" ; // GROUP BY brand
1111
1112     conn = databaseHandler.connectDb();
1113
1114     try {
1115
1116         prepare = conn.prepareStatement(sql);
1117         resultSet = prepare.executeQuery();
1118
1119         ObservableList listData = FXCollections.observableArrayList();
1120
1121         while (resultSet.next()) {
1122             listData.add(resultSet.getString("brand"));
1123         }
1124
1125         orders_brandName_combo.setItems(listData);
1126
1127         ordersListProductName();
1128     } catch (Exception e) {
1129         e.printStackTrace();
1130     }
1131
1132 }
1133
1134 public void ordersListProductName() {
1135
1136     // Check if brand selection exists before executing the query
1137     if (orders_brandName_combo.getSelectionModel().getSelectedItem() == null) {
1138         return; // Exit the method if no brand is selected
1139     }
1140
1141     String sql = "SELECT * FROM product WHERE brand = "
1142         + orders_brandName_combo.getSelectionModel().getSelectedItem() + "'";
1143
1144     conn = databaseHandler.connectDb();
1145
1146     try {
1147
1148         prepare = conn.prepareStatement(sql);
1149         resultSet = prepare.executeQuery();
1150
1151         ObservableList listData = FXCollections.observableArrayList();
1152
1153         while (resultSet.next()) {
1154             listData.add(resultSet.getString("product_name"));
1155         }
1156
1157         orders_productName_combo.setItems(listData);
1158
1159     //     while(resultSet.next()){
1160     //         customerData = new CustomerData(
1161     //             resultSet.getInt("customer_id"),
1162     //             resultSet.getString("type"),
1163     //             resultSet.getString("brand"),
1164     //             resultSet.getString("productName"),
1165     //             resultSet.getInt("quantity"),
1166     //             resultSet.getDouble("price"),
1167     //             resultSet.getDate("date"));
1168     //     }
1169     //     listData.add(customerData);

```

```

1173         }
1174     }
1175
1176     private SpinnerValueFactory<Integer> spinner;
1177
1178     public void ordersSpinner() {
1179
1180         spinner = new SpinnerValueFactory.IntegerSpinnerValueFactory(0, 20, 0);
1181         orders_quantity_spinner.setValueFactory(spinner);
1182
1183         /*
1184
1185         --- ordersSpinner() Method ---
1186
1187         This method is used to set up a Spinner for ordering quantities, allowing the user to
1188         select a number between 0 and 20. Here's a detailed explanation of each part of this method:
1189
1190         1. Spinner Initialization:
1191
1192             spinner = new SpinnerValueFactory.IntegerSpinnerValueFactory(0, 20, 0);
1193
1194             SpinnerValueFactory.IntegerSpinnerValueFactory(0, 20, 0) creates an integer-based spinner
1195             with a minimum value of 0, a maximum value of 20, and an initial/default value of 0.
1196             The SpinnerValueFactory helps manage the range and value of the spinner for quantity
1197             selection.
1198
1199
1200         2. Assigning the Spinner to the orders_quantity_spinner:
1201
1202             orders_quantity_spinner.setValueFactory(spinner);
1203
1204             Here, orders_quantity_spinner is a spinner UI element (likely created in the FXML file or
1205             elsewhere in your JavaFX UI code).
1206             This line assigns the spinner factory as the value factory for orders_quantity_spinner.
1207             This means that the spinner's value range (0 to 20) and starting value (0) are now applied to
1208             orders_quantity_spinner.
1209
1210
1211         3. Placement in Initialization or Button Click:
1212
1213
1214     private int qty;
1215
1216     public void ordersShowSpinnerValue() {
1217         qty = orders_quantity_spinner.getValue();
1218     }
1219
1220     public ObservableList<CustomerData> ordersListData() {
1221         customerId(); // pay button eke explain ekata meka yodaganna
1222         ObservableList<CustomerData> listData = FXCollections.observableArrayList();
1223         String sql = "SELECT * FROM customer WHERE customer_id = '" + customer_Id + "' ";
1224

```

```
1228     CustomerData customerData;
1229     prepare = conn.prepareStatement(sql);
1230     resultSet = prepare.executeQuery();
1231
1232     while (resultSet.next()) {
1233         customerData = new CustomerData(
1234             resultSet.getInt("customer_id"),
1235             resultSet.getString("type"),
1236             resultSet.getString("brand"),
1237             resultSet.getString("productName"),
1238             resultSet.getInt("quantity"),
1239             resultSet.getDouble("price"),
1240             resultSet.getDate("date"));
1241
1242         listData.add(customerData);
1243     }
1244
1245 } catch (Exception e) {
1246     e.printStackTrace();
1247 }
1248
1249     return listData;
1250 }
1251
1252 private ObservableList<CustomerData> orderList;
1253
1254 public void ordersShowListData() {
1255
1256     orderList = ordersListData();
1257
1258     orders_col_type.setCellValueFactory(new PropertyValueFactory<>("type"));
1259     orders_col_brand.setCellValueFactory(new PropertyValueFactory<>("brand"));
1260     orders_col_productName.setCellValueFactory(new PropertyValueFactory<>("productName"));
1261     orders_col_quantity.setCellValueFactory(new PropertyValueFactory<>("quantity"));
1262     orders_col_price.setCellValueFactory(new PropertyValueFactory<>("price"));
1263
1264     orders_tableView.setItems(orderList);
1265
1266     ordersDisplayTotal();
1267 }
1268
1269 private int customer_Id;
1270
1271 public void customerId() {
1272
1273     String customerId = "SELECT * FROM customer";
1274     conn = databaseHandler.connectDb();
1275
1276     try {
1277
1278         prepare = conn.prepareStatement(customerId);
1279         resultSet = prepare.executeQuery();
1280
1281         int checkId = 0;
1282
1283         while (resultSet.next()) {
1284
1285             // GET LAST CUSTOMER ID
1286             customer_Id = resultSet.getInt("customer_id");
1287
1288             if (checkId < customer_Id) {
1289                 checkId = customer_Id;
1290             }
1291
1292         }
1293
1294         customer_Id = checkId;
1295
1296     } catch (SQLException e) {
1297         e.printStackTrace();
1298     }
1299
1300     System.out.println("Customer ID: " + customer_Id);
1301
1302     return customer_Id;
1303 }
1304
1305 public void ordersDisplayTotal() {
1306
1307     int total = 0;
1308
1309     for (CustomerData data : orderList) {
1310
1311         total += data.getPrice() * data.getQuantity();
1312
1313     }
1314
1315     System.out.println("Total Order Value: " + total);
1316
1317 }
1318
1319 public void ordersDisplayCustomer() {
1320
1321     int id = customerId();
1322
1323     String query = "SELECT * FROM customer WHERE customer_id = ? ";
1324
1325     PreparedStatement statement = conn.prepareStatement(query);
1326
1327     statement.setInt(1, id);
1328
1329     ResultSet resultSet = statement.executeQuery();
1330
1331     if (resultSet.next()) {
1332
1333         String name = resultSet.getString("name");
1334         String address = resultSet.getString("address");
1335         String city = resultSet.getString("city");
1336         String state = resultSet.getString("state");
1337         String zip = resultSet.getString("zip");
1338
1339         System.out.println("Customer Name: " + name);
1340         System.out.println("Customer Address: " + address);
1341         System.out.println("Customer City: " + city);
1342         System.out.println("Customer State: " + state);
1343         System.out.println("Customer Zip: " + zip);
1344
1345     }
1346
1347 }
1348
1349 public void ordersDisplayProduct() {
1350
1351     int id = customerId();
1352
1353     String query = "SELECT * FROM product WHERE customer_id = ? ";
1354
1355     PreparedStatement statement = conn.prepareStatement(query);
1356
1357     statement.setInt(1, id);
1358
1359     ResultSet resultSet = statement.executeQuery();
1360
1361     if (resultSet.next()) {
1362
1363         String name = resultSet.getString("product_name");
1364         String type = resultSet.getString("type");
1365         String brand = resultSet.getString("brand");
1366         String quantity = resultSet.getString("quantity");
1367         String price = resultSet.getString("price");
1368
1369         System.out.println("Product Name: " + name);
1370         System.out.println("Product Type: " + type);
1371         System.out.println("Product Brand: " + brand);
1372         System.out.println("Product Quantity: " + quantity);
1373         System.out.println("Product Price: " + price);
1374
1375     }
1376
1377 }
1378
1379 public void ordersDisplayBrand() {
1380
1381     int id = customerId();
1382
1383     String query = "SELECT * FROM brand WHERE customer_id = ? ";
1384
1385     PreparedStatement statement = conn.prepareStatement(query);
1386
1387     statement.setInt(1, id);
1388
1389     ResultSet resultSet = statement.executeQuery();
1390
1391     if (resultSet.next()) {
1392
1393         String name = resultSet.getString("brand_name");
1394         String type = resultSet.getString("type");
1395         String quantity = resultSet.getString("quantity");
1396         String price = resultSet.getString("price");
1397
1398         System.out.println("Brand Name: " + name);
1399         System.out.println("Brand Type: " + type);
1400         System.out.println("Brand Quantity: " + quantity);
1401         System.out.println("Brand Price: " + price);
1402
1403     }
1404
1405 }
1406
1407 public void ordersDisplayType() {
1408
1409     int id = customerId();
1410
1411     String query = "SELECT * FROM type WHERE customer_id = ? ";
1412
1413     PreparedStatement statement = conn.prepareStatement(query);
1414
1415     statement.setInt(1, id);
1416
1417     ResultSet resultSet = statement.executeQuery();
1418
1419     if (resultSet.next()) {
1420
1421         String name = resultSet.getString("type_name");
1422         String type = resultSet.getString("type");
1423         String quantity = resultSet.getString("quantity");
1424         String price = resultSet.getString("price");
1425
1426         System.out.println("Type Name: " + name);
1427         System.out.println("Type Type: " + type);
1428         System.out.println("Type Quantity: " + quantity);
1429         System.out.println("Type Price: " + price);
1430
1431     }
1432
1433 }
```

```

1291         String checkData = "SELECT * FROM customer_receipt";
1292
1293         statement = conn.createStatement();
1294         resultSet = statement.executeQuery(checkData);
1295
1296         while (resultSet.next()) {
1297             checkId = resultSet.getInt("customer_id");
1298         }
1299
1300         if (customer_Id == 0) {
1301             customer_Id += 1;
1302         } else if (checkId == customer_Id) {
1303             customer_Id += 1;
1304         }
1305
1306         /*
1307
1308             above block checks if a unique customer_Id should be assigned:
1309
1310             if(customer_Id == 0) checks if there were no customer_id records in the customer table
1311             and increments customer_Id by 1.
1312             else if(checkId == customer_Id) checks if the last customer_id from customer_receipt
1313             matches customer_Id. If they are the same, it increments customer_Id by 1 to avoid duplication.
1314
1315             */
1316         } catch (Exception e) {
1317             e.printStackTrace();
1318         }
1319     public void switchForm(ActionEvent event) {
1320
1321         if (event.getSource() == home_btn) {
1322
1323             home_form.setVisible(true);
1324             addProducts_form.setVisible(false);
1325             orders_form.setVisible(false);
1326
1327             home_btn.setStyle("-fx-background-color:linear-gradient(to bottom right, #269e70,
1328 #969635);");
1329             addProducts_btn.setStyle("-fx-background-color:transparent;");
1330             orders_btn.setStyle("-fx-background-color:transparent;");
1331
1332             homeDisplayTotalOrders();
1333             homeTotalIncome();
1334             homeAvailableProducts();
1335             homeIncomeChart();
1336             homeOrdersChart();
1337
1338         } else if (event.getSource() == addProducts_btn) {
1339
1340             home_form.setVisible(false);
1341             addProducts_form.setVisible(true);
1342             orders_form.setVisible(false);
1343
1344             home_btn.setStyle("-fx-background-color:transparent;");
1345             addProducts_btn.setStyle("-fx-background-color:linear-gradient(to bottom right,
1346 #269e70, #969635);");
1347             orders_btn.setStyle("-fx-background-color:transparent;");
```

```
1350         addProductsSearch();
1351
1352     } else if (event.getSource() == orders_btn) {
1353
1354         home_form.setVisible(false);
1355         addProducts_form.setVisible(false);
1356         orders_form.setVisible(true);
1357
1358         home_btn.setStyle("-fx-background-color:transparent;");
1359         addProducts_btn.setStyle("-fx-background-color:transparent;");
1360         orders_btn.setStyle("-fx-background-color:linear-gradient(to bottom right, #269e70,
1361 #969635);");
1362
1363         orders_productType_combo.getSelectionModel().clearSelection();
1364         orders_brandName_combo.getSelectionModel().clearSelection();
1365         orders_productName_combo.getSelectionModel().clearSelection();
1366
1367         ordersShowListData();
1368         OrdersListTypes();
1369         ordersListBrands();
1370         ordersListProductName();
1371         ordersSpinner();
1372
1373     }
1374
1375     public void defaultNav(){
1376         home_btn.setStyle("-fx-background-color:linear-gradient(to bottom right, #269e70,
1377 #969635);"); // software eka run karala home page eka penne, ema nisa home_btn eka click unama watena
1378         pata watila tyena one muladima
1379     }
1380
1381     private double x = 0;
1382     private double y = 0;
1383
1384     public void logout() {
1385         try {
1386             Alert alert = new Alert(Alert.AlertType.CONFIRMATION);
1387             alert.setTitle("Confirmation Message");
1388             alert.setHeaderText(null);
1389             alert.setContentText("Are you sure want to logout?");
1390             Optional<ButtonType> option = alert.showAndWait();
1391
1392             if (option.get().equals(ButtonType.OK)) {
1393
1394                 main_form.getScene().getWindow().hide();
1395
1396                 // LINK YOUR LOGIN FORM
1397                 Parent root = FXMLLoader.load(getClass().getResource("/fxml/Login.fxml"));
1398                 Stage stage = new Stage();
1399                 Scene scene = new Scene(root);
1400
1401                 root.setOnMousePressed((MouseEvent event) -> {
1402                     x = event.getSceneX();
1403                     y = event.getSceneY();
1404                 });
1405
1406                 root.setOnMouseDragged((MouseEvent event) -> {
1407                     stage.setX(event.getScreenX() - x);
1408                     stage.setY(event.getScreenY() - y);
1409                 });
1410             }
1411         }
1412     }
1413 }
```

```
1410
1411         root.setOnMouseReleased((MouseEvent event) -> {
1412             stage.setOpacity(1);
1413         });
1414
1415         stage.initStyle(StageStyle.TRANSPARENT);
1416
1417         stage.setScene(scene);
1418         stage.show();
1419     } else {
1420         return;
1421     }
1422
1423     } catch (Exception e) {
1424         e.printStackTrace();
1425     }
1426 }
1427
1428 public void setUsername(){
1429     username_lbl.setText(GetData.username);
1430 }
1431
1432 public void minimize() {
1433     Stage stage = (Stage) main_form.getScene().getWindow();
1434     stage.setIconified(true);
1435 }
1436
1437 public void close() {
1438     System.exit(0);
1439 }
1440
1441 @Override
1442 public void initialize(URL url, ResourceBundle rb) {
1443
1444     setUsername();
1445     defaultNav();
1446
1447     homeDisplayTotalOrders();
1448     homeTotalIncome();
1449     homeAvailableProducts();
1450     homeIncomeChart();
1451     homeOrdersChart();
1452
1453     // TO SHOW THE DATA ON TABLEVIEW
1454     addProductsShowListData();
1455     addProductsListTypes();
1456     addProductsListStatus();
1457
1458     ordersShowListData();
1459     OrdersListTypes();
1460     ordersListBrands();
1461     ordersListProductName();
1462     ordersSpinner();
1463 }
1464
1465 }
```

```

import java.io.IOException;
import javafx.scene.input.MouseEvent;
import javafx.stage.StageStyle;

/**
 * JavaFX App
 */
public class App extends Application {

    private static Scene scene;
    private static Parent root;
    private double x = 0;
    private double y = 0;

    @Override
    public void start(Stage stage) throws IOException {
        Parent root = FXMLLoader.load(getClass().getResource("/fxml/Login.fxml"));
        scene = new Scene(root, 623, 439); // Specify the fxml folder

        root.setOnMousePressed((MouseEvent event) -> {
            x = event.getSceneX();
            y = event.getSceneY();
        });

        root.setOnMouseDragged((MouseEvent event) -> {
            stage.setX(event.getScreenX() - x);
            stage.setY(event.getScreenY() - y);

            stage.setOpacity(0.8);
        });

        root.setOnMouseReleased((MouseEvent event) -> {
            stage.setOpacity(1);
        });

        stage.initStyle(StageStyle.TRANSPARENT);

        stage.setScene(scene);
        stage.show();
    }

    static void setRoot(String fxml) throws IOException {
        scene.setRoot(loadFXML("fxml/" + fxml)); // Specify the fxml folder
    }

    private static Parent loadFXML(String fxml) throws IOException {
        FXMLLoader fxmlLoader = new FXMLLoader(App.class.getResource("//" + fxml + ".fxml")); // Include leading slash for resources
        return fxmlLoader.load();
    }

    public static void main(String[] args) {
        launch();
    }
}

```

4.4 Customer data

```
package com.mycompany.yt_inventorymanagementsystem_javafx;
```

```
import java.util.Date;
```

```
*  
* @author USER  
*/  
public class CustomerData {  
  
    private Integer customerId;  
    private String type;  
    private String brand;  
    private String productName;  
    private Integer quantity;  
    private Double price;  
    private Date date;  
  
    public CustomerData(Integer customerId, String type, String brand, String productName, Integer quantity, Double price, Date date) {  
        this.customerId = customerId;  
        this.type = type;  
        this.brand = brand;  
        this.productName = productName;  
        this.quantity = quantity;  
        this.price = price;  
        this.date = date;  
    }  
  
    public Integer getCustomerId() {  
        return customerId;  
    }  
  
    public String getType() {  
        return type;  
    }  
  
    public String getBrand() {  
        return brand;  
    }  
  
    public String getProductName() {  
        return productName;  
    }  
  
    public Integer getQuantity() {  
        return quantity;  
    }  
  
    public Double getPrice() {  
        return price;  
    }  
  
    public Date getDate() {  
        return date;  
    }  
}
```

4.5 Primary

```
5 package com.mycompany.yt_inventorymanagementsystem_javafx;
6
7 import java.io.IOException;
8 import javafx.fxml.FXML;
9
10 public class PrimaryController {
11
12     @FXML
13     private void switchToSecondary() throws IOException {
14         App.setRoot("secondary");
15     }
16 }
```

4.6 Products model

```
package com.mycompany.yt_inventorymanagementsystem_javafx;

import java.sql.Date;

/**
 *
 * @author USER
 */
public class ProductsModel {

    private Integer productId;
    private String type;
    private String brand;
    private String productName;
    private Double price;
    private String status;
    private String image;
    private Date date; // It represents only the date (year, month, and day) without any time information (hours, minutes, seconds, or milliseconds).

    public ProductsModel(Integer productId, String type, String brand, String productName, Double price, String status, String image, Date date) {
        this.productId = productId;
        this.type = type;
        this.brand = brand;
        this.productName = productName;
        this.price = price;
        this.status = status;
        this.image = image;
        this.date = date;
    }

    public Integer getProductId() {
        return productId;
    }

    public String getType() {
        return type;
    }

    public String getBrand() {
        return brand;
    }

    public String getProductName() {
```

```

public Double getPrice() {
    return price;
}

public String getStatus() {
    return status;
}

public String getImage() {
    return image;
}

public Date getDate() {
    return date;
}

}

```

4.7 Module info :

```

module com.mycompany.yt_inventorymanagementsystem_javafx {
    requires javafx.controls;
    requires javafx.fxml;
    requires java.sql;
    requires de.jensd.fx.glyphs.fontawesome;
    requires java.base;
    requires jasperreports;

    opens com.mycompany.yt_inventorymanagementsystem_javafx to javafx.fxml;
    exports com.mycompany.yt_inventorymanagementsystem_javafx;
}

```



Logistics Management System

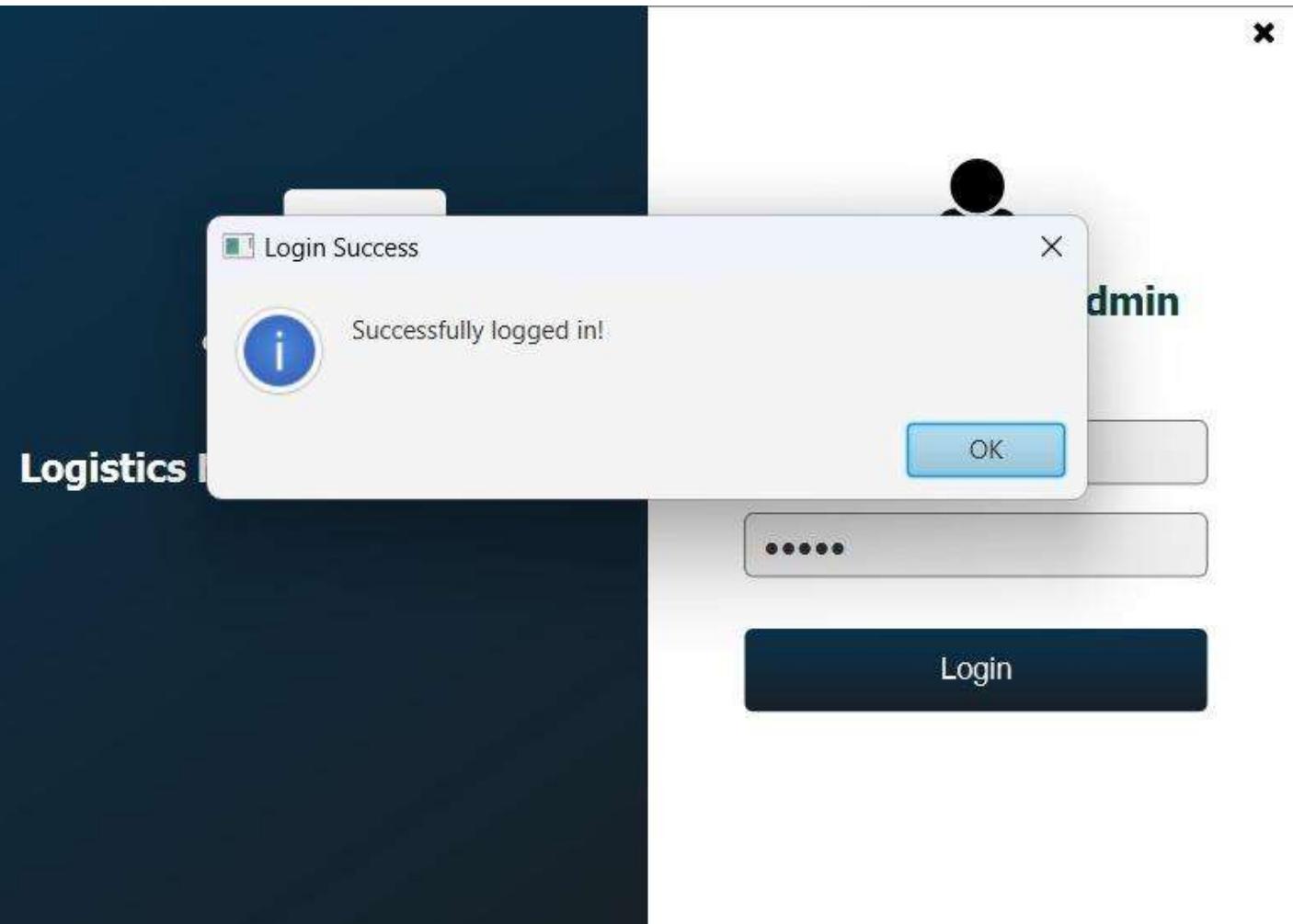


Welcome Back Admin

A light gray rectangular input field with a placeholder character 'I' inside.A light gray rectangular input field with the word "Password" inside.

Login





Welcome krithika

-  Home
-  Add Products
-  Orders

 Sign Out

Today's Number of Orders 0

Total Income Rs 987.56

Available Products 0

Income Data Chart



Order Data Chart



5.4 entering orders

The screenshot displays the Logistics Management System interface. On the left sidebar, there is a user profile icon and the text "Welcome krithika". Below the sidebar are three main navigation buttons: "Home", "Add Products", and "Orders", with "Orders" being highlighted by a green gradient background. At the bottom left is a "Sign Out" button with a right-pointing arrow icon.

The central area shows a table with columns: Type, Brand, Product Name, Quantity, and Price. A single row is present with the values: Snacks, grb, chips, 2, and 20.0.

To the right of the table, there is a form for adding a new product. The fields are: Product Type (dropdown menu showing "Snacks"), Brand Name (dropdown menu showing "grb"), Product Name (dropdown menu showing "chips"), and Quantity (input field showing "1"). Below these fields is a blue "Add" button. Further down, the total amount is displayed as "Total: Rs 20.0". There are also input fields for "Amount" and "Balance", both currently showing "Rs 0.0". At the bottom right are two buttons: a blue "Pay" button and a purple "Reset" button.

5.5 adding products

The screenshot shows the 'Logistics Management System' interface. On the left sidebar, there is a user icon, a welcome message 'Welcome krithika', and navigation links for 'Home', '+ Add Products' (highlighted in green), 'Orders', and 'Sign Out'. The main content area has a search bar at the top right. In the center, there is an 'Import' button above a form with fields for 'Product ID', 'Product Type' (dropdown: Choose), 'Brand' (input: [empty]), 'Product Name' (input: [empty]), 'Status' (dropdown: Choose), and 'Price' (input: [empty]). Below the form are four buttons: 'Add' (blue), 'Update' (green), 'Reset' (purple), and 'Delete' (red). To the right of the form is a table with columns: Product ID, Type, Brand, Product Name, Price, and Status. The table contains four rows of data:

Product ID	Type	Brand	Product Name	Price	Status
1	Snacks	grb	chips	10.0	Available
2	Drinks	coke	soda	50.0	Not Available
3	Dessert	grb	mysoreoak	250.09	Not Available
4	Personal Pro...	ntg	ntg	12.0	Available

5.6 logout successful

The screenshot shows the 'Logistics Management System' interface after a successful logout. The left sidebar remains the same. The main content area now features a confirmation dialog box in the center. The dialog title is 'Confirmation Message' and the message is 'Are you sure want to logout?'. It has 'OK' and 'Cancel' buttons. To the right of the dialog, there is a table with columns: Type, Brand, Product Name, Quantity, and Price. One row is visible: Snacks, grb, chips, 2, 20.0. To the right of the table is a form for adding a new item. The form includes dropdowns for 'Product Type' (Snacks), 'Brand Name' (grb), and 'Product Name' (chips), and an input field for 'Quantity' (1). A large blue 'Add' button is positioned below the form. At the bottom right, there are labels for 'Total: Rs 20.0', 'Amount:' (input field), 'Balance: Rs 0.0', a blue 'Pay' button, and a purple 'Reset' button.

Conclusion

With the help of our project, businesses can efficiently manage their logistics operations by tracking inventory, coordinating with suppliers, and processing orders seamlessly. The system organizes all critical data in one centralized platform, accessible through their accounts, saving time and ensuring a more streamlined, professional approach to logistics management.

Reference

1. <https://www.javatpoint.com/java-tutorial>
2. <https://www.wikipedia.org/>
3. <https://www.w3schools.com/sql/>
4. <https://www.codecademy.com/learn/learn-sql>