INTRODUCTION TO DATABASE MANAGEMENT SYSTEM

OBJECTIVE: To familiarize database Concepts.

THEORY

A database is an organized collection of interrelated data stored together without unnecessary redundancy. The data in the database is isolated, can be shared and concurrently accessed. A Database Management System (DBMS) is a collection of interrelated files and a set of programs that allow users to access those data. The primary goal of a DBMS is to provide an environment that is convenient and efficient to use, in retrieving and storing database information.

Database systems are designed to manage large volumes of information. The management of data involves the definition of structures for the storage of information. In addition, the database system must provide for the safety of the information despite crashes or attempts at unauthorized access.

RELATIONAL DBMS

One relational model is an abstract theory of data that is based on mathematical theory. This model used certain terms and principles which were not familiar in data processing at that time. So a new technology evolved called RDBMS terminology.

Formal Relational Terms	Informal Equivalent
Relation	Table
Tuple	Row, Records
Cardinality	No: of rows
Attribute	Column, Field
Degree	No: of columns
Primary Key	Unique identifier
Domain	Set of legal values

The smallest unit of data in relational model is the individual data values. A domain is a set of all possible data values. Domain is conceptual in value, which may or may not be stored in database as actual set of values.

A relational domain, say D1, D2,.........., Dn consists of a fixed set of distinct attributes say A1,A2,...........An such that each attribute A1 corresponds to exactly one of underlying domain 'D1'. The body consists of a time varying set of attribute value pairs (Ai, Vi) one such pair of

each attribute Ai in heading.
Following are the structured components of a relational model.
Relations: In relational database model, the database is organized in relations. A relation is synonymous to a table.
Attributes: An attribute is a property or characteristics which hold some information about an entity. A customer, for e.g. has attributes like code, name etc.
Tuple: A record/tuple is a complete set of relational fields.

FEATURES AND RESPONSIBILITIES OF RDBMS

RDBMS is software that creates and maintains a database. The tasks related to maintain are:
1. Data abstraction.
2. Control data redundancy/duplication.
3. Support for multiple users.
4. Multiple way of interfacing to the system.
5. Restricting unauthorized access.
6. Enforcing integrity constrains.
7. Backup and recovery.
Examples: Oracle, DB2, Sybase, and SQL Server

ORACLE DATABASE

Oracle Database (commonly referred to as Oracle RDBMS or simply as Oracle) is an object-relational database management system produced and marketed by Oracle Corporation. Users of Oracle databases refer to the server-side memory-structure as the SGA (System Global Area). The SGA typically holds cache information such as data-buffers, SQL commands, and user information. In addition to storage, the database consists of online redo logs (or logs), which hold transactional history. The Oracle DBMS can store and execute stored procedures and functions within it. The Oracle RDBMS stores data logically in the form of tablespaces and physically in the form of data files ("data files"). Oracle database management tracks its computer data storage with the help of information stored in the SYSTEM tablespace. The SYSTEM tablespace contains the data dictionary indexes and clusters. A data dictionary consists of a special collection of tables that contains information about all user-objects in the database. Currently, the version being used is Oracle 12c version 12.1.0.1, where "c" stands for "cloud" to indicate that 12c is "cloud enabled". It features a new multi-tenant option that will help companies to consolidate databases into private or public clouds.

DATABASE LANGUAGES

1. **DATA DEFINITION LANGUAGE (DDL)**
A data base schema is specifies by a set of definitions expressed by a special language called DDL.

2. **DATA MANIPULATION LANGUAGE (DML)**
This language that enable user to access or manipulate data as organized by appropriate data model. There are two types:
    - **Procedural DML :** DML requires a user to specify what data are needed and how to get those data.
    - **Non Procedural DML:** DML requires a user to specify what data are needed without specifying how to get those data.

3. TRANSACTION CONTROL LANGUAGES (TCL)
The subset of SQL which is used to control transactional processing in a database is Transaction Control Languages.

3. DATA CONTROL LANGUAGES (DCL)
A Data Control Language(DCL) is a syntax similar to a computer programming language used to control access to data stored in a database(Authorization)

QUERY: A query is a statement requesting the retrieval of information.

QUERY LANGUAGE: The portion of a DML that involves information retrieval is called a query language.

STRUCTURED QUERY LANGUAGE (SQL)

Structured Query Language (SQL) is the standard language for creation and maintenance of relational database management system. This language allows users to create and modify objects of a relational database. Users can insert, update, delete the data and perform queries for on the data using SQL. SQL is not a case sensitive language. It can be described as a declarative language(Non Procedural Language) because data can be retrieved without specifying the procedures for retrieving it. SQL has a simple structure. The basic structure of an SQL expression consist of three clauses **select**, **from** and **where**. The ***select*** clause corresponds to the projection operation of the relational algebra. It is used to list the attributes desired in the results of a query. The ***from*** clause corresponds to the Cartesian product operation of the relational algebra. The ***where*** clause corresponds the selection predicate of the relational algebra.

OUTCOME: Students are able to understand the fundamental concepts of Database.

EXERCISE NUMBER 1
DATA DEFINITION LANGUAGE, TABLE CREATION, CONSTRAINTS

OBJECTIVE: To execute the Data Definition Language (DDL) commands and creating the table with constraints using RDBMS.

THEORY

DATA DEFINITION LANGUAGE (DDL)

Data Definition Language is a set of SQL statements used to create, modify and delete structure of database objects such as tables, views etc. Examples of DDL statements are

- CREATE
- ALTER
- DROP
- RENAME
- TRUNCATE

1. CREATE

It is used to create a table .

Table Creation Rules:

● Reserved words cannot be used.
● Underscore, numerals, letters are allowed but not blank space.
● Maximum length for the table name is 30 characters.
● 2 different tables should not have same name.
● We should specify a unique column name.
● We should specify proper data type along with width.
● We can include "not null" condition when needed. By default it is 'null'.

Syntax

```
create table <table name>
(
fieldname-1 datatype constraints if any,
fieldname-2 datatype constraints if any,
…….
fieldname-n datatype constraints if any,
);
```

Data types help us to specify the kind of data that can be stored in a table.
Oracle has provided the following data types:

DATA TYPE	DESCRIPTION
CHAR(size)	Fixed-length character data (maximum *size* is 2000)
VARCHAR2(size)	Variable-length character data (maximum *size*

	is 4,000)
NUMBER(p,s)	Number having precision 'p' and scale's'. Precision is the total number of digits and scale is the number of digits to the right of the decimal point.
DATE	Date and time values to the nearest second between January 1, 4712 B.C., and December 31, 9999 A.D.
TIMESTAMP	Date with fractional seconds
LONG	Variable-length character data up to 2 GB
CLOB	Character data up to 4 GB
RAW (size)	Raw binary data of length *size*. (Maximum size is 2000)
LONG RAW	Raw binary data of variable length(up to 2 GB)
BLOB	Binary data up to 4 GB
BFILE	Binary data stored in an external file
ROWID	A base-64 number system representing the unique address of arrow in its table.

DEFAULT Option

The DEFAULT option is used to specify a default value for a column during an insert. This option prevents null values from entering the columns when a row is inserted without a value for the column. The default value can be a literal, an expression, or a SQL function (such as SYSDATE or USER), but the value cannot be the name of another column or a pseudo column (such as NEXTVAL or CURRVAL).The default data type must match the column data type.

SQL INTEGRITY CONSTRAINT

Integrity constraints are used to enforce the business rules associated with the database and prevent the entry of invalid information into tables. SQL allows the definition of constraints on columns and tables.

Constraints can be defined in two ways:

- **Column Constraint**: A constraint specified at the column level. Column-level

constraints are included when the column is defined.
Column-level constraint syntax:

column* [CONSTRAINT *constraint_name*] *constraint_type,

- **Table Constraint**: A constraint specified at the table level. Table-level constraints are defined at the end of the table definition and must refer to the column or columns on which the constraint pertains in a set of parentheses.
  Constraints that apply to more than one column must be defined at the table level.
  Table-level constraint syntax:

  ***column, ...* [CONSTRAINT *constraint_name*] *constraint_type*(*column, ...*),**

TYPES OF CONSTRAINTS

a) Domain Integrity

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

NOT NULL constraint:

The NOT NULL constraint ensures that the column contains no null values.
Columns without the NOT NULL constraint can contain null values by default.
NOT NULL constraints must be defined at the column level.

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.

CHECK Constraint:

CHECK constraint can be defined to allow only a particular range of values. The CHECK constraint defines a condition that each row must satisfy. When the manipulation violates this constraint, the record will be rejected.A single column can have multiple CHECK constraints that refer to the column in its definition. CHECK constraints can be defined at the column level or table level.

Syntax to define a CHECK constraint:

CONSTRAINT constraint_name] CHECK (condition

b) Entity Integrity

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. Entity Integrity can be enforced through indexes, UNIQUE constraints and PRIMARY KEY constraints.

UNIQUE constraint

A UNIQUE key integrity constraint requires that every value in a column or a set of columns (key) be unique—that is, no two rows of a table can have duplicate values in a specified column or a set of columns. The column (or set of columns) included in the definition of the UNIQUE key constraint is called the *unique key*. If the UNIQUE constraint comprises more than one column, that group of columns is called a *composite unique key*.

A null in a column (or in all columns of a composite UNIQUE key) always satisfies a UNIQUE constraint. Maximum combination of columns that a composite unique key can contain is 16.

Syntax to define a Unique key at column level:
[CONSTRAINT constraint_name] UNIQUE

Syntax to define a Unique key at table level:
[CONSTRAINT constraint_name] UNIQUE(column_name)

PRIMARY KEY Constraint

A primary key avoids duplication of rows and does not allow null values. It 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.

Syntax to define a Primary key at column level:
column_name datatype [CONSTRAINT constraint_name] PRIMARY KEY

Syntax to define a Primary key at table level:
[CONSTRAINT constraint_name] PRIMARY KEY (column_name1, [column_name2,...])

- **column_name1, column_name2** are the names of the columns which define the primary Key.
- The syntax within the bracket i.e. [CONSTRAINT constraint_name] is optional.

c) Referential Integrity

A referential integrity rule is a rule defined on a key (a column or set of columns) in one table that guarantees that the values in that key match the values in a key in a related table (the referenced value). It establishes a relationship between two columns in the same table or between different tables having a common column definition. 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.

Referential integrity also includes the rules that dictate what types of data manipulation are allowed on referenced values and how these actions affect dependent values. The rules associated with referential integrity are:

- **Restrict**: Disallows the update or deletion of referenced data.
- **Set to null**: When referenced data is updated or deleted, all associated dependent data is set to **NULL**.
- **Set to default**: When referenced data is updated or deleted, all associated dependent data is set to a default value.
- **Cascade**: When referenced data is updated, all associated dependent data is correspondingly updated. When a referenced row is deleted, all associated dependent rows are deleted.
- **No action**: Disallows the update or deletion of referenced data. This differs from **RESTRICT** in that it is checked at the end of the statement, or at the end of the transaction if the constraint is deferred. (Oracle Database uses No Action as its default action.)

FOREIGN KEY

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

Syntax to define a Foreign key at column level:

[CONSTRAINT constraint_name] REFERENCES referenced_table_name column_name);

Syntax to define a Foreign key at table level:

[CONSTRAINT constraint_name] FOREIGN KEY (column_name) REFERENCES referenced_table_name (column_name);

Foreign key Constraint: Keywords

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

- 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 : Foreign key values are set to null when a row in the parent table is deleted

The foreign key is defined in the child table and the table containing the referenced column is the parent table. The default behavior is called the *restrict rule*, which disallows the update or deletion of referenced data.

Without the ON DELETE CASCADE or the ON DELETE SET NULL options, the row in the parent table cannot be deleted if it is referenced in the child table.

Examples:

```
1.  SQL>  CREATE TABLE DEPT
               (DEPTNO NUMBER (3) PRIMARY KEY,
                DNAME VARCHAR2 (10),
               LOCATION VARCHAR2 (15),
                STARTDATE DATE DEFAULT SYSDATE);
```
Table created.

```
2.  SQL> CREATE TABLE EMP
               (EMPNO NUMBER (5),
               ENAME VARCHAR2 (15),
               JOB CHAR (10) CONSTRAINT UN UNIQUE,
               SAL NUMBER (7, 2),
               DEPTNO NUMBER (3) CONSTRAINT FK1 REFERENCES DEPT
               (DEPTNO));
```
Table created.

```
3.  SQL>  CREATE TABLE STUD
               (SNAME VARCHAR2 (20) NOT NULL,
               ROLLNO NUMBER (10) NOT NULL,
                DOB DATE NOT NULL);
```
Table created.

DESC

This command is used to display the structure of a table or a view.

Syntax:

DESC <table_name>

In the syntax:

<table_name> refers to the name of any existing table or view that is accessible to the user.

Example:

```
SQL> DESCRIBE DEPT;
NAME          NULL                TYPE
---------------- -----------------          --------
DEPTNO      NOT NULL            NUMBER (3)
DNAME                           VARCHAR2 (10)
LOCATION                        VARCHAR2 (15)
STARTDATE                        DATE
SQL> DESC EMP;
NAME          NULL           TYPE
---------------- -----------------    --------
EMPNO                       NUMBER (5)
ENAME                       VARCHAR2 (15)
JOB           NOT NULL      CHAR (10)
SAL                         NUMBER (7, 2)
DEPTNO        NOT NULL      NUMBER (3)
```

2. ALTER TABLE

Alter command is used to:

1. Add a new column to an existing table.
2. Modify an existing column definition
3. Drop a column from an existing table.
4. Rename a column or a table.
5. Add or drop integrity constraint.

2.1. ADD COLUMN IN TABLE

To add a column in a table, the syntax is:

```
ALTER TABLE <tablename> ADD  col_name column definition
```

2.2. ADD MULTIPLE COLUMNS IN TABLE

```
ALTER TABLE <tablename>
      ADD  (column_1 column-definition,
            column_2 column-definition,
      ...
            column_n column_definition);
```

2.3. MODIFY COLUMN IN TABLE

To modify a column in an existing table, the syntax is:

```
ALTER TABLE table_name
```

MODIFY column_name column_type;

2.4. DROP COLUMN IN TABLE

To drop a column in an existing table, syntax is:

ALTER TABLE table_name

DROP COLUMN column_name;

2.5. RENAME COLUMN IN TABLE

To rename a column in an existing table, the syntax is:

ALTER TABLE table_name

RENAME COLUMN old_name to new_name;

2.6. RENAME TABLE

To rename a table, the syntax is:

ALTER TABLE table_name

RENAME TO new_table_name;

2.7. ADD DATA CONSTRAINTS

Syntax: ALTER TABLE table_name add constraint constraint_name primary-key-definition | foreign-key definition | unique-constraint | check-constraint;

2.8. DROP CONSTRAINTS

Syntax: ALTER TABLE table_name drop constraint constraint_name;

Examples:

1. SQL> **ALTER** TABLE EMP **ADD (PHONE_NO NUMBER (7)**);
   Table altered.

```
SQL> DESC EMP;


NAME              NULL             TYPE
----------------  ---------------- --------
EMPNO                              NUMBER (5)
ENAME                              VARCHAR2 (15)
JOB               NOT NULL         CHAR (10)
```

```
SAL                                NUMBER (7, 2)
DEPTNO              NOT NULL       NUMBER (3)
PHONE_NO                           NUMBER (7)
```

2. SQL>ALTER TABLE EMP **ADD (DOB DATE, DOJ DATE)**;
   Table altered.
   SQL> DESC EMP;

```
Name            Null?        Type
------------------ ----------        -------------------------
EMPNO                          NUMBER(5)
ENAME                         VARCHAR2(15)
JOB            NOT NULL       CHAR(10)
SAL                           NUMBER(7,2)
DEPTNO         NOT NULL       NUMBER(3)
PHONE_NO                      NUMBER (7)
DOB                           DATE
DOJ                           DATE
```

3. SQL> ALTER TABLE EMP **MODIFY(PHONE_NO NUMBER (10))**;
   Table altered.
   SQL> **DESC** EMP;

```
NAME            NULL?              TYPE
------------------ ----------        --------------------------
EMPNO                         NUMBER(5)
ENAME                         VARCHAR2(15)
JOB            NOT NULL       CHAR(10)
SAL                           NUMBER(7,2)
DEPTNO         NOT NULL       NUMBER(3)
PHONE_NO                      NUMBER (10)
DOB                           DATE
DOJ                           DATE
```

4. SQL> ALTER TABLE EMP **RENAME COLUMN PHONE_NO TO PHNO**;
   Table altered.
   SQL> DESC EMP;

```
NAME            NULL?              TYPE
------------------ ----------        --------------------------
EMPNO                        NUMBER(5)
ENAME                         VARCHAR2(15)
JOB            NOT NULL      CHAR(10)
```

```
SAL                         NUMBER(7,2)
DEPTNO      NOT NULL       NUMBER(3)
PHNO                       NUMBER (10)
DOB                         DATE
DOJ                         DATE
```

5. SQL>ALTER TABLE EMP **DROP COLUMN PHONE_NO;**
   Table altered.
6. SQL> ALTER TABLE EMP **DROP (DOB, DOJ);**
   Table altered.

```
SQL> DESC EMP;
NAME                          NULL?         TYPE
--------------------------    --------      --------------
EMPNO                                       NUMBER(5)
ENAME                                       VARCHAR2(15)
JOB                  NOT NULL                CHAR(10)
SAL                                          NUMBER(7,2)
DEPTNO               NOT NULL               NUMBER(3)
```

7. SQL> ALTER TABLE EMP **ADD CONSTRAINT PKEY1 PRIMARY KEY (EMPNO);**
   Table altered.
8. SQL> ALTER TABLE EMP **RENAME TO EMP1**;
   Table altered.

3. DROP TABLE

It will delete the table structure provided the table should be empty.

Example:

SQL> DROP TABLE EMP1;

4. TRUNCATE TABLE

If there is no further use of records stored in a table and the structure has to be retained then the records alone can be deleted.

Syntax:

TRUNCATE TABLE <TABLE NAME>;

Example:

SQL> TRUNCATE TABLE STUD;

5. RENAME

RENAME command is used to rename the objects.

Syntax:

RENAME <OLD_TABLENAME> TO <NEW_TABLENAME>;

Example:

SQL> RENAME DEPT TO DEPT7;

Table renamed.

DOMAIN INTEGRITY

a) NOT NULL Constraint

Example:

```
SQL> CREATE TABLE
              Cust(Custid number(6) NOT NULL,
              Name char(10));
Table created.
SQL> ALTER TABLE Cust modify (Name NOT NULL);
Table altered.
```

b) CHECK CONSTRAINT

i. **Column Level Check Constraint**

Example:

```
SQL> CREATE TABLE
            STUDENT (REGNO NUMBER (6),
            NAME VARCHAR2(10),
            BRANCH  VARCHAR2(5),
            MARK NUMBER (3) CONSTRAINT ST_M_CK
            CHECK (MARK >=0 AND MARK <=100)
            ADDR  VARCHAR2(15) );
Table created.
```

ii. **Table Level Check Constraint:**

```
SQL> CREATE TABLE
            STUD1 (REGNO NUMBER (6),
            NAME VARCHAR2(10),
            BRANCH  VARCHAR2(5),
            MARK NUMBER (3) ,
            ADDR  VARCHAR2(15),
            CONSTRAINT STUD1_MARK_CK  CHECK (MARK >=0 AND MARK
            <=100) );
Table created.
```

iii. **Check Constraint with Alter Command**

```
SQL> ALTER TABLE
            STUDENT ADD CONSTRAINT ST_REG_CK
             CHECK ( LENGTH (REGNO <= 4));
Table altered.
```

ENTITY INTEGRITY

a) Unique key constraint

Example:

```
SQL> CREATE TABLE
          CUST(CUSTID NUMBER(6) CONSTRAINT CUS_U UNIQUE,
           NAME CHAR(10));
Table created.
SQL> ALTER TABLE CUST
           ADD(CONSTRAINT CUST_CUN UNIQUE(CUSTID));
Table altered.
```

b) Primary Key Constraint

i. **Column level Primary Key constraint**

Example:

```
SQL> CREATE TABLE
                 STUD2 (REGNO NUMBER(6) CONSTRAINT STUD1_PK
                 PRIMARY KEY,
                  NAME VARCHAR2(20),
                 AGE NUMBER(2));
Table created.
```

ii. **Table Level Primary Key Constraint**

```
SQL> CREATE TABLE
                 STUD3 (REGNO NUMBER (6),
                     NAME VARCHAR2 (20),
                     AGE NUMBER (2),
                     CONSTRAINT STUD3_REG_PK PRIMARY KEY);
Table created.
```

iii. **Primary Key Constraint with alter command**

```
SQL> CREATE TABLE
                 STUD4 (REGNO NUMBER (6),
                  NAME VARCHAR2 (20),
                 AGE NUMBER (2));

SQL> ALTER TABLE  STUD4 ADD CONSTRAINT STUD4_REG_PK PRIMARY
KEY(REGNO);
```

c) Foreign Key Constraint

a. **Column level foreign key constraint**

```
SQL>CREATE TABLE
                DEPT2  (DEPTNO NUMBER(2) PRIMARY KEY,
                DNAME VARCHAR2(20),
                LOCATION VARCHAR2(15));
SQL>CREATE TABLE EMP4
                (EMPNO NUMBER(3),
                DEPTNO NUMBER(2) REFERENCES DEPT2(DEPTNO),
                DESIGN VARCHAR2(10));
```

b. **Table Level Foreign Key Constraint**

```
SQL>CREATE TABLE DEPT3
                (DEPTNO NUMBER(2) PRIMARY KEY,
                DNAME VARCHAR2(20),
                LOCATION VARCHAR2(15));
SQL>CREATE TABLE EMP5
                (EMPNO NUMBER(3),
                DEPTNO NUMBER(2),
                DESIGN VARCHAR2(10),
                CONSTRAINT EMP5_DEPTNO_FK  FOREIGN KEY(DEPT NO)
                REFERENCES DEPT3(DEPTNO));
```

c. **Table Level Foreign Key Constraints with Alter command**

```
SQL>CREATE TABLE DEPT6
                (DEPTNO NUMBER(2) PRIMARY KEY,
                DNAME VARCHAR2(20),
                LOCATION VARCHAR2(15));
SQL>CREATE TABLE EMP6
                (EMPNO NUMBER(3),
                DEPTNO NUMBER(2),
                DESIGN VARCHAR2(10));
SQL>ALTER TABLE EMP6
        ADD CONSTRAINT EMP6_DEPTNO_FK
         FOREIGN KEY(DEPTNO) REFERENCES DEPT6(DEPTNO);
```

OUTCOME: Students got an idea to use different DDL commands and constraints in oracle **.**

SAMPLE QUESTIONS

1. Create the following tables including the required keys.

   a. **Table Name: Client_master**

   Description: Used to store client information.

Column name	**Data type**	**Size**	**Attributes**
Client_no	Varchar2	6	Primary key. First letter must start with 'c'.
Name	Varchar2	20	Not null
Adrdress1	Varchar2	30	
Address2	Varchar2	30	
City	Varchar2	15	
Pincode	Number	8	
State	Varchar2	15	
Bal_Due	Number	10,2	

   b. **Table Name: Product_master**

   Description: Used to store product information.

Column name	**Data type**	**Size**	**Attributes**
Product_no	Varchar2	6	Primary key. First letter must start with 'p'.
Description	Varchar2	15	Not null
Profit_percent	Number	4,2	
Unit_measure	Varchar2	10	
Qty_on_hand	Number	8	
Reorder_lvl	Number	8	
Sell_price	Number	8,2	
Cost_price	Number	8,2	

   c. **Table Name : Salesman_master**

   Description : Used to store salesman working for the company.

Column name	**Data type**	**Size**	**Attributes**
Salesman_no	Varchar2	6	Primary key. First letter must start with 's'.
Salesman_name	Varchar2	20	Not null
Adrdress1	Varchar2	30	Not null
City	Varchar2	20	
Pincode	Varchar2	8	
State	Varchar2	20	
Sal_amt	Varchar2	8,2	Not null,cannot be zero
Tgt_to_get	Number	6,2	Not null,cannot be zero
Ytd_sales	Number	6,2	Not null
Remarks	Varchar2	60	

d. Table Name: Sales_order
Description : Used to store client's orders.

Column name	**Data type**	**Size**	**Attributes**
Order_no	Varchar2	6	Primary key. First letter must start with 'o'.
Order_date	Date		
Client_no	Varchar2	6	Foreign key references client_no of client_master table
Dely_addr	Varchar2	25	
Salesman_no	Varchar2	6	
Dely_type	Char	1	Deliver: part (P)/full (F), Default 'F'.
Billed_yn	Char	1	
Dely_date	Date		
Order_status	Varchar2	10	Values ('in process','fulfilled','backorder','cancelled')

e. Table Name: Sales_order_details
Description: Used to store client's orders with details of each product ordered.

Column name	**Data type**	**Size**	**Attributes**
Order_no	Varchar2	6	Primary key/foreign key references order_no of the sales_order table
Product_no	Varchar2	6	Primary key/foreign key references product_no of the product_master table
Qty_ordered	Number	8	
Qty_Number	Number	8	
Product_rate	Number	10,2	

EXERCISE NUMBER: 2

DATA MANIPULATION AND DATA CONTROL LANGUAGE

OBJECTIVE: To practice the various DML(Data Manipulation Language) and DCL(Data Control Language) commands and implement them on the database.

THEORY

DATA MANIPULATION LANGUAGE (DML)
Data Manipulation Language is a language that enables user to access or manipulate data organized by the appropriate data model.
DML statements are used to

- Retrieve / fetch information stored in the database.
- Insert new information into the database.
- Delete information from the database.
- Modify / update information stored in the database.

DML statements are

- ➢ SELECT
- ➢ INSERT
- ➢ DELETE
- ➢ UPDATE

1. INSERT COMMAND

This is used to add one or more rows to a table. The values are separated by commas and the data types char and date are enclosed in single quotes. The values must be entered in the same order as they are defined.

Inserting a single row into a table:

Syntax: Insert into <table name> values (value list);

Example: Insert into s values(‘s3’,’sup3’,’blore’,10)

Inserting more than one record using a single insert commands:

Syntax: Insert into <table name> values (&col1, &col2, ….)

Example: Insert into stud values(&reg, ‘&name’, &percentage);

Skipping the fields while inserting:

Insert into <tablename(coln names to which datas to be inserted)> values (list of values);

Other way is to give null while passing the values.

2. SELECT COMMANDS

It is used to retrieve information from the table. It is generally referred to as querying the table. We can either display all columns in a table or only specify column from the table.

Selects all rows from the table

Syntax: Select * from tablename;

Example: Select * from IT;

The retrieval of specific columns from a table:

It retrieves the specified columns from the table

Syntax: Select column_name1, …..,column_namen from table name;

Example: Select empno, empname from emp;

Elimination of duplicates from the select clause:

It prevents retrieving the duplicated values .Distinct keyword is to be used.

Syntax: Select DISTINCT col1, col2 from table name;

Example: Select DISTINCT job from emp;

Select command with where clause:

To select specific rows from a table we include „where" clause in the select command. It can appear only after the „from" clause.

Syntax: Select column_name1, …..,column_namen from table name where condition;

Example: Select empno, empname from emp where sal>4000;

Select command with order by clause:

Syntax: Select column_name1, …..,column_namen from table name where condition order by colmnname;

Example: Select empno, empname from emp order by empno;

Select command to create a table:

Syntax: create table tablename as select * from existing_tablename;
Example:create table emp1 as select * from emp;

Select command to insert records:

Syntax: insert into tablename (select columns from existing_tablename);
Example: Insert into emp1 (select * from emp);

3. UPDATE COMMAND

It is used to alter the column values in a table. A single column may be updated or more than one column could be updated.

Syntax: Update tablename set field=values where condition;

Example: Update emp set sal = 10000 where empno=135;

4. DELETE COMMAND

After inserting row in a table we can also delete them if required. The delete command consists of a from clause followed by an optional where clause.

Syntax: Delete from table where conditions;

Example: Delete from emp where empno=135;

TRANSACTION CONTROL LANGUAGES

The subset of SQL which is used to control transactional processing in a database is Transaction Control Languages.

1. COMMIT

This will commit (write to database) the transactions done by the DML.

```
SQL> commit;
Commit complete.
```

2. ROLLBACK

After inserting, updating or deleting the transactions the user does not want to commit the changes, then the user can rollback the transaction using the command:

```
SQL> rollback;
Rollback complete.
```

It will rollback the transaction and will not commit the changes to the database.

3. SAVEPOINT

```
SQL> savepoint s1;
Savepoint s1 created.
```

It will save the transactions under the name s1.

DATA CONTROL LANGUAGE

A data control language (DCL) is syntax similar to a computer programming language used to control access to data stored in a database.

1. **GRANT -** Used to give privilege to user on object.
   Syntax: Grant privilege on <object_name> to <user name>;
   Example: grant write on employee to user01;
2. **REVOKE -** Used to withdraw the privilege that has been granted to the user.

Syntax: Revoke privilege on <object_name> from <user name>;
Example: revoke write on employee from user01;

OUTCOME: Students are able to work with DDL and DCL commands on data.

SAMPLE QUESTIONS

1. Insert the following data into the tables created in Ex. 1 and define grant and revoke.

   a. **Data for CLIENT_MASTER**

Client_no	**Name**	**Address 1**	**Address 2**	**City**	**Pincode**	**State**	**Bal_due**
C00001	Ivan bayross	Wandon	Worli	Bombay	400054	Maharashtra	15000
C00002	Vandana saitwal	Don Street	Bandra	Madras	780001	Tamil Nadu	0
C00003	Prama dajaguste	Mandon	Dadar	Bombay	400057	Maharashtra	5000
C00004	Basu navindgi	Jerome	Juhu	Bombay	400056	Maharashtra	0
C00005	Ravisreedharan	Dadar	Dadra	Delhi	100001	Delhi	2000
C00006	Rukmini	Rourk	Bandra	Bombay	400057	Maharashtra	0

   b. **Data for PRODUCT_MASTER**

Product_no	**Description**	**Profit_percent**	**unit_measure**	**Qty_on_hand**	**Reorder_lvl**	**Sell_price**	**Cost_price**
P00001	1.44 floppies	5	piece	100	20	525	500
P03453	monitors	6	piece	10	3	12000	11280
P06734	mouse	5	piece	20	5	1050	1000
P07865	1.22 floppies	5	piece	100	20	525	500
P07868	keyboards	2	piece	10	3	3150	3050

P07885	Cd drive	2.5	piece	10	3	5250	5100
P07965	540 HDD	4	piece	10	3	8400	8000
P07975	1.44drive	5	piece	10	3	1050	1000
P08865	1.22drive	5	piece	2	3	1050	1000

c. **Data for SALESMAN_MASTER**

Salesman_no	**Salesman_name**	**Address**	**City**	**Pincode**	**State**	**Sal_amt**	**Tgt_to_get**	**Ytd_sales**	**Remarks**
S00001	Kiran	a/14	Worli	400002	Bombay	3000	100	50	Good
S00002	Maneesh	65	Nariman	400001	Bombay	3000	200	100	Good
S00003	Ravi	p-7	Bandra	400032	Bombay	3000	200	100	Good
S00004	Ashish	a/5	Juhu	400044	Bombay	3000	200	150	Good

d. **Data for SALES_ORDER**

Order_no	**Order_date**	**Client_no**	**Delyaddr**	**Salesman_no**	**Delytype**	**Billed_yn**	**Delivryy_date**	**Order_status**
O19001	30-JUL-12	C00001	Wandon	S00001	F	N	10-JUL-12	In process
O19002	02-AUG-12	C00002	Don Street	S00002	P	N	04-AUG-12	Cancelled
O46865	12-JUL-12	C00003	Mandon	S00003	F	Y	20-JUL-12	Fulfilled
O19003	05-JUN-12	C00001	Jerome	S00001	F	Y	12-JUN-12	Fulfilled
O46866	20-MAY-12	C00004	Dadar	S00002	P	N	22-MAY-12	Cancelled
O19008	28-JUL-12	C00005	Rourk	S00004	F	N	10-AUG-12	In process

e. **Data for SALES_ORDER_DETAILS**

Order_no	Product_no	Qty_ordered	Qty_disp	Product_rate
O19001	P00001	4	4	525
O19001	P07965	2	1	8400
O19001	P07885	2	1	5250
O19002	P00001	10	0	525
O46865	P07868	3	3	3150
O46865	P07885	3	1	5250
O46865	P00001	10	10	525
O46865	P03453	4	4	1050
O19003	P03453	2	2	1050
O19003	P06734	1	1	12000
O46866	P07965	1	0	8400
O46866	P07975	1	0	1050
O19008	P00001	10	5	525
O19008	P07975	5	3	1050

2. Show an example of an insertion in the SALES_ORDER and CLIENT_MASTER relations (in Ex. 1) that violates the referential integrity constraints.
3. Find the list of all clients who stay in ‘Bombay’ or ‘Delhi’.
4. Print the list of clients whose Bal_due is greater than value 10000.
5. Print the information from the sales order table from orders placed in the month of January
6. Display the order information for client no ‘C00001’ and ‘C00002’.
7. Find the products whose selling price is greater than 2000 and less than or equal to 5000.
8. Find the products whose selling price is more than 1500.Calculate a new selling price = original selling price * .15. Rename the new column in the above query as new price.
9. List the names, city and state of clients who are not in the state of ‘Maharashtra’
10. Find all the products whose qty_on_hand is less than reorder level.
11. List the records in the CLIENTMASTER table in descending order based on Client name.
12. Delete the details of clients who lives in Bombay

EXERCISE NUMBER: 3
BUILT-IN FUNCTIONS

OBJECTIVE: To understand the usability of built-in functions in queries.

THEORY

SQL Commands: Function is a group of code that accepts zero or more arguments and both return one or more results. Both are used to manipulate individual data items. Operators differ from functional in that they follow the format of function name (arg..). An argument is a user defined variable or constant. Most operators accept at most 2 arguments while the structure of functions permit to accept 3 or more arguments. Function can be classified into

- **single row function**
- **group function**

Single Row functions
A single row function or scalar function returns only one value for every row queries in table. Single row function can appear in a select command and can also be included in a where clause. The single row function can be broadly classified as,

- Date Function
- Numeric Function
- Character Function
- Conversion Function

The example that follows mostly uses the symbol table "dual". It is a table, which is automatically created by oracle along with the data dictionary.
Date Function
They operate on date values and produce outputs, which also belong to date data type except for months, between, date function returns a number.
DATE FUNCTIONS

1. Add_months

This function returns a date after adding a specified date with specified number of months.
Syntax: Add_months(d,n); wherd-date n-number of months
Example: Select add_months(sysdate,2) from dual;

2. last_day

It displays the last date of that month.
Syntax: last_day (d); where d is date

Example: Select last_day (“1-jun-2009”) from dual;

3. Months_between

It gives the difference in number of months between d1 & d2.
Syntax: month_between (d1,d2); where d1 & d2 -dates
Example: Select month_between (“1-jun-2009”,”1-aug-2009”) from dual;

4. next_day

It returns a day followed the specified date.
Syntax: next_day (d,day);
Example: Select next_day (sysdate,”Wednesday”) from dual

5. round

This function returns the date, which is rounded to the unit specified by the format model.
Syntax :round (d,[fmt]); where d- date, [fmt] – optional. By default date will be rounded to the nearest day
Example: Select round (to_date(“1-jun-2009”,”dd-mm-yy”),‟year‟) from dual;

NUMERICAL FUNCTIONS

Command	**Query**	**Output**
Abs(n)	Select abs(-15) from dual;	15
Ceil(n)	Select ceil(55.67) from dual;	56
Exp(n)	Select exp(4) from dual;	54.59
Floor(n)	Select floor(100.2) from dual;	100
Power(m,n)	Select power(4,2) from dual;	16
Mod(m,n)	Select mod(10,3) from dual;	1
Round(m,n)	Select round(100.256,2) from dual;	100.26
Trunc(m,n)	Select trunc(100.256,2) from dual;	100.23
Sqrt(m,n)	Select sqrt(16) from dual;	4

CHARACTER FUNCTIONS

Command	**Query**	**Output**
initcap(char);	select initcap(‘hello’) from dual;	Hello
lower (char);	select lower (‘HELLO’) from dual;	hello
upper (char);	select upper (“hello‟) from dual;	HELLO
ltrim (char,[set]);	select ltrim („cseit‟, „cse‟) from dual;	it
rtrim (char,[set]);	select rtrim („cseit‟, „it‟) from dual;	cse
replace (char,search string, replace string);	select replace(„jack and jue‟,„j‟,„bl‟) from dual;	black and blue
substr (char,m,n);	select substr („information‟, 3, 4) from dual;	Form

CONVERSION FUNCTION

1. to_char()

Syntax: to_char(d,[format]);

This function converts date to a value of varchar type in a form specified by date format. If format is neglected then it converts date to varchar2 in the default date format.

The different formats are:

Day	Month	Year	Fill Mode	Julian Date
D	MM	YY	FM	J
DD	MON	YYYY		
DDTH		RR		
DAY		RRRR		

Example: select to_char (sysdate, ‟dd-mm-yy‟) from dual;

2. to_date()

Syntax: to_date(d,[format]);

This function converts character to date data format specified in the form character.

Example: select to_date(„aug 15 2009‟,‟mm-dd-yy‟) from dual;

OUTCOME:

Students are able to manipulate different built-in functions in queries.

SAMPLE QUESTIONS

Using the schema created in Ex.1

1. Display the order number and day on which clients placed their order.
2. Display the month (in alphabets) and date which the order must be delivered.
3. Display the order_date in the format 'DD-month-YY'.
4. Find the date, 15 days after today's date.
5. Add two months to the current date and display the date
6. Display the last date of current month
7. Display the number of months between two dates
8. Find the number of days elapsed between today's date and the delivery date of the order placed by the clients.
9. Find the names of all client having 'a' as the second letter in their name
10. Find out the clients who stay in a city whose second letter is 'a'.

EXERCISE NUMBER: 4
AGGREGATE FUNCTIONS

OBJECTIVE: To introduce the concept of aggregate functions for manipulation data items in queries.

THEORY

Aggregate Functions

An Aggregate Functions returns a result based on group of rows.

1. **AVG -** Returns average values.

   Example: select avg (total) from student;

2. **MAX –** Returns the maximum value of an expression

   Example: select max (percentagel) from student;

3. **MIN –** Returns the minimum value of an expression

   Example: select min (marksl) from student;

4. **SUM –** Return sum of values .

   Example: select sum(price) from product;

5. **COUNT -** In order to count the number of rows, count function is used.

   a) **count(*)** – It counts all, inclusive of duplicates and nulls.

      Example: select count(*) from student;

   b) **count(col_name)**– It avoids null value.

      Example: select count(total) from order;

   c) **count(distinct col_name)** – It avoids the repeated and null values.

      Example: select count(distinct ordid) from order;

OUTCOME: Students will able to manipulate data items using aggregate functions.

SAMPLE QUESTIONS

Using the schema created in Ex.1

1. Count the total number of orders.
2. Calculate the average price of all the products
3. Determine the maximum and minimum product prices. Rename the output as max_price and min _price respectively.
4. Count the number of products having price greater than or equal to 1500.

EXERCISE NUMBER: 5

HAVING AND GROUP BY CLAUSES

OBJECTIVE: -To implement the concept of grouping of Data.

THEORY

Grouping Data From Tables: There are circumstances where we would like to apply the aggregate function not only to a single set of tuples, but also to a group of sets of tuples, we specify this wish in SQL using the group by clause. The attribute or attributes given in the group by clause are used to form group. Tuples with the same value on all attributes in the group by clause are placed in one group.

Syntax:
SELECT columnname, columnname
FROM tablename
GROUP BY columnname;
Example: Select max(sell_price), product_no from PRODUCT_MASTER
group by product_no;

At times it is useful to state a condition that applies to groups rather than to tuples. For example we might be interested in only those branches where the average account balance is more than 1200. This condition does not apply to a single tuple, rather itapplies to each group constructed by the GROUP BY clause. To express such Questionry, we use the having clause of SQL. SQL applies predicates in the having may be used.
Syntax:
SELECT columnname, columnname
FROM tablename
GROUP BY columnname;
HAVING searchcondition;
Example:

Select max(sell_price), product_no from PRODUCT_MASTER group by product_no
having profit percent>=3;

OUTCOME
Students can use aggregate functions in grouping of data.
SAMPLE QUESTIONS
Using the schema created in Ex.1

a) Print the description and total qty sold for each product
b) Find the value of each product sold
c) Calculate the average qty sold for each client that has maximum order value of 15000
d) Find out the sum total of all the billed orders for the month of February

EXERCISE NUMBER: 6

SUB QUERIES

OBJECTIVE: To introduce the concept of sub queries.

THEORY

Nesting of queries one within another is known as a nested queries.

Example: select ename, eno, address where salary >(select salary from employee where ename =‟jones‟);

The query within another is known as a sub query. A statement containing sub query is called parent statement. The rows returned by sub query are used by the parent statement. A subquery is used to return data that will be used in the main query as a condition to further restrict the data to be retrieved.

Subqueries can be used with the SELECT, INSERT, UPDATE, and DELETE statements along with the operators like =, <, >, >=, <=, IN, BETWEEN etc.

There are a few rules that subqueries must follow:

- Subqueries must be enclosed within parentheses.
- A subquery can have only one column in the SELECT clause, unless multiple columns are in the main query for the subquery to compare its selected columns.
- An ORDER BY cannot be used in a subquery, although the main query can use an ORDER BY. The GROUP BY can be used to perform the same function as the ORDER BY in a subquery.
- Subqueries that return more than one row can only be used with multiple value operators, such as the IN operator.
- The SELECT list cannot include any references to values that evaluate to a BLOB, ARRAY, CLOB, or NCLOB.
- A subquery cannot be immediately enclosed in a set function.
- The BETWEEN operator cannot be used with a subquery; however, the BETWEEN operator can be used within the subquery.

a. ***Subqueries with the SELECT Statement****:*

Subqueries are most frequently used with the SELECT statement. The basic syntax is as follows:

```
SELECT column_name [, column_name ]
FROM   table1 [, table2 ]
WHERE  column_name OPERATOR
(SELECT column_name [, column_name ]
    FROM table1 [, table2 ]
[WHERE])
```

Subqueries that return several values

Example: select ename, eno, from employee where salary <any (select salary from employee where deptno =10‟);

Correlated subquery

Example: select * from emp x where x.salary > (select avg(salary) from emp where deptno =x.deptno);

b. ***Subqueries with the INSERT Statement***

Subqueries also can be used with INSERT statements. The INSERT statement uses the data returned from the subquery to insert into another table. The selected data in the subquery can be modified with any of the character, date or number functions.The basic syntax is as follows:

```
INSERT INTO table_name [(column1 [, column2 ])]
        SELECT [*|column1 [, column2 ]
        FROM table1 [, table2 ]
[ WHERE VALUE OPERATOR ]
```

c. Subqueries with the UPDATE Statement

The subquery can be used in conjunction with the UPDATE statement. Either single or multiple columns in a table can be updated when using a subquery with the UPDATE statement.The basic syntax is as follows:

```
UPDATE table SET column_name = new_value
[ WHERE OPERATOR [ VALUE ]
(SELECT COLUMN_NAME
 FROM TABLE_NAME)
[ WHERE)]
```

d. Subqueries with the DELETE Statement

The subquery can be used in conjunction with the DELETE statement like with any other statements mentioned above. The basic syntax is as follows:

```
DELETE FROM TABLE_NAME
[ WHERE OPERATOR [ VALUE ]
(SELECT COLUMN_NAME
  FROM TABLE_NAME)
[ WHERE)]
```

OUTCOME: Students got an idea to embed two or more queries together.

SAMPLE QUESTIONS

Using the schema created in Ex.1

1. Find the product_no and description of non-moving products ie, products not being sold.
2. Find the customer name, address1, address2,city and pincode for the client who has placed order_no ‘O19001’.
3. Find the client_names who have placed orders before the month of May ’96.
4. Find out if the product ‘1.44 Drive’ has been ordered by any client and print the client_no,name to whom it was sold.
5. Find the names of clients who have placed orders worth Rs 10000 or more.

EXERCISE NUMBER: 7
JOINS

OBJECTIVE: To understand the use of joining two or more tables in queries.

THEORY

JOINS

The SQL **Joins** clause is used to combine records from two or more tables in a database. A JOIN is a means for combining fields from two tables by using values common to each.

Different Types of JOIN

a) CROSS JOIN(CARTESIAN JOIN)

The cross join returns the Cartesian product of the sets of records from the two or more joined tables which results potentially very large number of rows. The row count of the result is equal to the number of rows in the first table times the number of rows in the second table. Each row is a combination of the rows of the first and second table.

Syntax: SELECT <attribute> FROM Table1 CROSS JOIN Table2

SELECT <attribute> FROM Table1 ,Table2

Example: SELECT * FROM Course CROSS JOIN Student; or SELECT * FROM Course, Student;

Table : Course

NAME	COURSE NUMBER
John	ITEC 122
Cindy	ITEC 120
Reenu	ITEC 121

Table :Student

SNAME	PHONE NUMBER	GPA
John	6391111	2.0
David	8311111	3.0
Cindy	7311111	4.0

OUTPUT

NAME	COURSE NUMBER	SNAME	PHONE NUMBER	GPA
John	ITEC 122	John	6391111	2.0
John	ITEC 122	David	8311111	3.0
John	ITEC 122	Cindy	7311111	4.0
Cindy	ITEC 120	John	6391111	2.0
Cindy	ITEC 120	David	8311111	3.0
Cindy	ITEC 120	Cindy	7311111	4.0

Reenu	ITEC 121	John	6391111	2.0
Reenu	ITEC 121	David	8311111	3.0
Reenu	ITEC 121	Cindy	7311111	4.0

b) NATURAL JOIN

A NATURAL JOIN is a JOIN operation that creates an implicit join clause based on the common columns in the two tables being joined. Common columns are columns that have the same name in both tables. After Cross Join, pick up only rows with the same (or matching) common column values. Show the common column value only once.

Syntax: SELECT <attribute> FROM Table1 NATURAL JOIN Table2
Example: SELECT * FROM Course NATURAL JOIN Student;

OUTPUT

NAME	**COURSE NUMBER**	**PHONE NUMBER**	**GPA**
John	ITEC 122	6391111	2.0
Cindy	ITEC 120	7311111	4.0

c) INNER JOIN(SIMPLE JOIN)
The most frequently used and important of the joins is the **INNER JOIN**. They are also referred to as an EQUIJOIN. It returns the matching rows from the table that are being joined. However, show the common matched column values for both tables.

Syntax: SELECT <attribute> FROM Table1 INNERJOIN Table2 ON Table1.attribute=Table2.attribute.
Example: SELECT * FROM Course INNER JOIN Student ON Course.Name= Student. Name;

NAME	**COURSE NUMBER**	**NAME**	**PHONE NUMBER**	**GPA**
John	ITEC 122	John	6391111	2.0
Cindy	ITEC 120	Cindy	7311111	4.0

d) OUTER JOIN

i. LEFT OUTER JOIN

The SQL **LEFT JOIN** returns all rows from the left table, even if there are no matches in the right table. This means that if the ON clause matches 0 (zero) records in right table, the join will still return a row in the result, but with NULL in each column from right table.This means that a left join returns all the values from the left table, plus matched values from the right table or NULL in case of no matching join predicate.

Syntax: SELECT <attribute> FROM Table1 LEFT OUTER JOIN Table2 ON Table1.attribute=Table2.attribute.

SELECT <attribute> FROM Table Name WHERE Table1.attribute(+)=Table2.attribute.

Example: SELECT * FROM Course LEFT OUTER JOIN Student ON Course.Name= Student. Name;

OUTPUT:

NAME	COURSE NUMBER	NAME	PHONE NUMBER	GPA
John	ITEC 122	John	6391111	2.0
Cindy	ITEC 120	Cindy	7311111	4.0
Reenu	ITEC 121	NULL	NULL	NULL

ii. RIGHT OUTER JOIN

The SQL **RIGHT JOIN** returns all rows from the right table, even if there are no matches in the left table. This means that a right join returns all the values from the right table, plus matched values from the left table or NULL in case of no matching join predicate.

Syntax: SELECT <attribute> FROM Table1 RIGHT OUTER JOIN Table2 ON Table1.attribute=Table2.attribute.

SELECT <attribute> FROM Table Name WHERE Table1.attribute=(+)Table2.attribute.

Example: SELECT * FROM Course RIGHT OUTER JOIN Student ON Course.Name= Student. Name;

OUTPUT

NAME	COURSE NUMBER	NAME	PHONE NUMBER	GPA
John	ITEC 122	John	6391111	2.0

Cindy	ITEC 120	Cindy	7311111	4.0
NULL	NULL	David	8311111	3.0

iii. FULL OUTER JOIN

The SQL **FULL OUTER JOIN** combines the results of both left and right outer joins. The joined table will contain all records from both tables, and fill in NULLs for missing matches on either side.

Syntax: SELECT <attribute> FROM Table1 FULL JOIN Table2 on Table1.attribute=Table2.attribute
Example: SELECT * FROM Course FULL JOIN Student ON Course.Name= Student. Name;

OUTPUT:

NAME	COURSE NUMBER	NAME	PHONE NUMBER	GPA
John	ITEC 122	John	6391111	2.0
NULL	NULL	David	8311111	3.0
Cindy	ITEC 120	Cindy	7311111	4.0
Reenu	ITEC 121	NULL	NULL	NULL

OUTCOME:

By using joins, students can relate different tables using a single query.

SAMPLE QUESTIONS

Using the schema created in Ex.1

1. Find out the products and their quantities that will have to be delivered in the current month.
2. Find the product_no and description of constantly sold i.e, rapidly moving products.
3. Find the names of clients who have purchased ‘CD Drive’.
4. List the product_no and order_no of customers having qty_ordered less than 5 from the sales_order_details table for the product ‘1.44 Floppies’.
5. Find the products and their quantities for the orders placed by ‘Ivan bayross’ and ‘Vandana Saitwal’.
6. Find the products and their quantities for the orders placed by the client_no ‘C00001’ and ‘C00002’.

EXERCISE NUMBER: 8
VIEWS

OBJECTIVE: To create and manipulate various database objects of the table using views.

THEORY

VIEWS

A view is the tailored presentation of data contained in one or more table and can also be said as restricted view to the data's in the tables. A view is a “virtual table” or a “stored query” which takes the output of a query and treats it as a table. The table upon which a view is created is called as base table.

A view is a logical table based on a table or another view. A view contains no data of its own 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. The view is stored as a SELECT statement in the data dictionary

Advantages of a view:

a. Additional level of table security.
b. Hides data complexity.
c. Simplifies the usage by combining multiple tables into a single table.
d. Provides data's in different perspective.

Creating and dropping view:

Syntax:

```
CREATE [OR REPLACE] VIEW <VIEW NAME> [COLUMN ALIAS NAMES] AS
<QUERY> [WITH <OPTIONS> CONDITIONS];
DROP VIEW <VIEW NAME>;
```

Example:

```
CREATE OR REPLACE VIEW empview AS SELECT * FROM emp;
DROP VIEW empview;
```

OUTCOME:

Students are able to create and manipulate various database objects of the table using views.

SAMPLE QUESTIONS

1. Create a view based on Client_master table and display all the rows.
    a. Insert record into newly created view.
    b. Delete from view a client whose name is Ivan Bayross.
    c. Update the name of client as Kavitha whose name is Rukmini
2. Create a view which contains details of clients whose lives in Bombay.
3. Create a view for retrieving order number and day on which clients placed their order.

EXERCISE NUMBER: 9
INDEXES IN SQL

OBJECTIVE: To create indexes on table data and implement it**.**

THEORY

Indexes are special lookup tables that the database search engine can use to speed up data retrieval. Simply put, an index is a pointer to data in a table. An index in a database is very similar to an index in the back of a book.For example, if you want to reference all pages in a book that discuss a certain topic, you first refer to the index, which lists all topics alphabetically and are then referred to one or more specific page numbers.

An index helps speed up SELECT queries and WHERE clauses, but it slows down data input, with UPDATE and INSERT statements. Indexes can be created or dropped with no effect on the data.Creating an index involves the CREATE INDEX statement, which allows you to name the index, to specify the table and which column or columns to index, and to indicate whether the index is in ascending or descending order.

Indexes can also be unique, similar to the UNIQUE constraint, in that the index prevents duplicate entries in the column or combination of columns on which there's an index.

Creating an index for a table

Single Column Indexes:

A single-column index is one that is created based on only one table column.

Syntax: CREATE INDEX index filename ON tablename(Column name);
Example: The SQL statement below creates an index named "PIndex" on the "LastName" column in the "Persons" table:

```
 CREATE INDEX  client_indx ON client_master (client_no);
```

***Composite Indexs*:**

A composite index is an index on two or more columns of a table.
Syntax: CREATE INDEX indexfilename ON tablename (column name,columnname);
Example: The SQL statement below creates an index named "PIndex" on the "LastName" and FirstName columns in the "Persons" table:

```
CREATE INDEX PIndex ON Persons (LastName, FirstName)
```

Whether to create a single-column index or a composite index, take into consideration the column(s) that you may use very frequently in a query's WHERE clause as filter conditions. Should there be only one column used, a single-column index should be the choice. Should there be two or more columns that are frequently used in the WHERE clause as filters, the composite index would be the best choice.

Unique Indexes:

Unique indexes are used not only for performance, but also for data integrity. A unique index does not allow any duplicate values to be inserted into the table.

Syntax: CREATE UNIQUE INDEX indexfilename ON tablename(columnname);

Implicit Indexes:

Implicit indexes are indexes that are automatically created by the database server when an object is created. Indexes are automatically created for primary key constraints and unique constraints.

Dropping Indexes:

An indexes can be dropped by using the DROP INDEX command.

Syntax: DROP INDEX index filename;

When should indexes be avoided?

Although indexes are intended to enhance a database's performance, there are times when they should be avoided. The following guidelines indicate when the use of an index should be reconsidered:

- Indexes should not be used on small tables.
- Tables that have frequent, large batch update or insert operations.
- Indexes should not be used on columns that contain a high number of NULL values.
- Columns that are frequently manipulated should not be indexed.

OUTCOME:

Students are able to create Index for the tables .

SAMPLE QUESTIONS

Using the schema created in Ex.1

1. Create An Index On Table CLIENT MASTER for Field CLIENT_NO.
2. Create A Composite Index on the SALES_ORDER_DETAILS TABLE FOR COLUMN, ORDER_NO, and PRODUCT_NO.
3. Create a UNIQUE INDEX ON The Table CLIENT_MASTER for a Column CLIENT_NAME.
4. Drop Index on table CLIENT_MASTER.

EXERCISE NUMBER: 10
SEQUENCES IN SQL

OBJECTIVE: To create Sequences on table data and implement it.

THEORY

Sequence is an object in Oracle database, which is used by multiple users to generate unique numbers. Sequence is typically used to generate primary keys like account number, employee number etc., where uniqueness and sequence matter.

In order to use a sequence, first it is to be created using CREATE SEQUENCE command. Then pseudo columns NEXTVAL and CURRVAL are used to retrieve unique values from sequence.

CREATING SEQUENCES:

Syntax:

```
CREATE SEQUENCE sequencename
 [INCREMENT BY integer]
 [START WITH integer]
 [MAXVALUE integer | NOMAXVALUE]
 [MINVALUE integer | NOMINVALUE]
 [CYCLE | NOCYCLE]
 [CACHE integer value/NOCACHE];
```

KEYWORDS

INCREMENT BY- Increment by specifies the interval between sequence numbers. It can be positive or negative value but not zero. IF the clause is omitted default value is 1

MIN VALUE- Specifies the sequences minimum value

NOMINVALUE- specifies a minimum value of 1 for an ascending sequence and –(10)^26 for a descending sequence.

MAX VALUE- specifies maximum of (10)^27 for an ascending sequence or -1 for a descending sequence.

START WITH- specifies the first sequence number to be generated. The default for an ascending sequence is the sequence minimum value and for descending it is the maximum value.

CYCLE- specifies that if the maximum value exceeds the set limit, sequence will restart its cycle from the begining.

NO CYCLE specifies that if sequence exceeds **maxvalue** an error will be thrown.

CACHE-cache specifies how many value of the sequence oracle pre-allocates and keep in memory for faster access. The minimum value for the parameter is 2.

NOCACHE-nocache specifies that the value of the sequence are not pre allocated. If the cache or noncache clause is omitted oracle caches 20 sequence numbers by default.

Example:

The following command creates a sequence called ROLLNO to generate roll numbers for students.

CREATE SEQUENCE rollno

START WITH 100

INCREMENT BY 1;

The above sequence starts generating numbers at 100 and increments the number by 1 every time the number is taken. The following two pseudo columns are used to access the next and current value of the sequence.

NEXTVAL
This pseudo column will yield the next value from sequence and automatically increments the value of the sequence.

CURRVAL
This returns the value that is taken by most recent NEXTVAL. This cannot be used unless NEXTVAL is first called.

Examples:

- SELECT rollno.NEXTVAL FROM DUAL;
  NEXTVAL
  100

As the sequence starts with 100, first NEXTVAL returned 100. And it also increments the sequence by 1. See the next example below.

- SELECT rollno.NEXTVAL FROM DUAL;
  NEXTVAL
  101

CURRVAL pseudo column returns the current value of the sequence, which is the value returned by most recent NEXTVAL. In the following example CURRVAL returns 101 since that is the most recent value returned by NEXTVAL.

- SELECT rollno.CURRVAL FROM DUAL;
  CURRVAL
  101

CURRVAL is used to reuse the values returned by most recent NEXTVAL. The real usage of sequence is in inserting rows into table. The following INSERT command will use rollno sequence to get next available roll number for a new student.

- INSERT INTO STUDENTS VALUES (rollno.NEXTVAL , ...);

DROPPING A SEQUENCE

DROP SEQUENCE command is used to drop a sequence.

Syntax: DROP SEQUENCE sequencename;

OUTCOME: Students are able to implement sequences on table data.

SAMPLE QUESTIONS

1. Create a table for the following schema
   Department (Dept_id: Number, Depart_ Name: String, Manager_id: Number)
   Create a sequence that can be used with the primary key column of the Department table.The sequence should start at 200 and have a maximum value of 1,000 with interval value of 10. Name the sequence DEPT_ID_SEQ.
2. To test your sequence, insert two rows in the Department table using the sequence that you created for the ID column.
3. Drop the sequence which generated from Q:1

EXERCISE NUMBER: 11

PL/SQL CONTROL STRUCTURES

OBJECTIVE: To create PL/SQL programs to implement various types of control structure.

THEORY

PL/SQL is a language used in all oracle products. It has procedural features which enable a user to call SQL statements.

The procedural features of PL/SQL are:

1. Variables
2. Control statements such as IF THEN, FOR loop, WHILE loop.
3. Sub programs.
4. Exceptions.

PL/SQL is used to store programs, procedures, packages, forms & reports. It is different from other languages as it does not have conventional input and output statements (read and write).

This input is taken mainly by from the table and output is put in the table.

Features of PL/SQL are:

1. Block structures.

PL/SQL is a blocked structured language. Each block is supposed to perform one typical unit of job. The block can be a named block(procedures, functions) or an anonymous block.

Each block is having 3 parts:-

Declarative part, an executable part and an exception handling part.

2. Control structures.

The data is processed using control statements. These are conditional IF-THEN-ELSE structure, FOR loop, WHILE loop, LOOP, EXIT, WHEN and for changing GO TO statements.

3. Modularity.

Modularity means dividing the application to manageable well defined blocks. PL/SQL provides three ways of implementing modularity.

a) PL/SQL block.
b) Procedures.
c) Functions.

4. Data Abstraction.

PL/SQL supports data abstraction. Data abstraction means essential properties of data are provided to the user while ignoring unnecessary details.

5) Cursor

Oracle uses temporary work area for storing output of an SQL statement. There are two types of cursors.

a) Implicit.
b) Explicit.

6) Error Handling

PL/SQL has given the facility of exception for handling abnormal conditions. Exceptions if occurred during program execution can be handled by Exception handler.

FUNDAMENTALS OF PL/SQL

PL/SQL is made of character set, data types, expressions etc. The basic elements which make PL/SQL are:

a) Lexical units.
b) Data types.
c) User defined types.
d) Data type conversions.
e) Declaration.
f) Naming conversions.
g) Scope and visibility.
h) Assignments.
i) Expression and comparisons

PL/SQL CONTROL STRUCTURES

Control Structures are used to modify the flow of programs. The control statements are classified into the following categories.

- ➢ Conditional control –Branching
- ➢ Iterative control – Looping

PL/SQL programming language provides following types of decision-making statements.

Statement	Description
IF - THEN statement	The IF statement associates a condition with a sequence of statements enclosed by the keywords THEN and END IF. If the condition is true, the statements get executed and if the condition is false or NULL then the IF statement does nothing.
IF-THEN-ELSE statement	IF statement adds the keyword ELSE followed by an alternative sequence of statement. If the condition is false or NULL , then only the alternative sequence of statements get executed. It ensures that either of the sequence of statements is executed.
IF-THEN-ELSIF statement	It allows you to choose between several alternatives.
Case statement	Like the IF statement, the CASE statement selects one sequence of statements to execute. However, to select the sequence, the CASE statement uses a selector rather than multiple Boolean expressions. A selector is

	an expression whose value is used to select one of several alternatives.
Searched CASE statement	The searched CASE statement has no selector, and it's WHEN clauses contain search conditions that yield Boolean values.
nested IF-THEN-ELSE	You can use one IF-THEN or IF-THEN-ELSIF statement inside another IF-THEN or IF-THEN-ELSIF statement(s).

BRANCHING in PL/SQL:

1. **Syntax for IF-THEN statement is:**

```
IF condition THEN
statement1;
statement2;
END IF;
```

Example:

```
DECLARE
  a NUMBER(2) := 10;
BEGIN
  a:= 10;
    IF( a < 20 ) THEN
         DBMS_OUTPUT.PUT_LINE('a is less than 20 ' );
   END IF;
         DBMS_OUTPUT.PUT_LINE('value of a is : ' || a);
END;
/
```

2. **IF-THEN-ELSE STATEMENT**

Syntax:

```
IF condition THEN
statement1;
ELSE
statement2;
END IF;
```

Example:

```
DECLARE
  a NUMBER(2) := 10;
BEGIN
  a:= 10;
```

```
IF( a < 20 ) THEN
       DBMS_OUTPUT.PUT_LINE('a is less than 20 ' );
       ELSE
             DBMS_OUTPUT.PUT_LINE('a is not less than 20 ' );
          END IF;
          DBMS_OUTPUT.PUT_LINE('value of a is : ' || a);
       END;
       /
```

3. **IF-THEN-ELSIF STATEMENTS**

Syntax:

```
IF condition1 THEN
statement1;
ELSIF condition2 THEN
statement2;
ELSIF condition3 THEN
statement3;
ELSE
statement;
END IF;
```

Example:

```
DECLARE
   a NUMBER(3) := 100;
BEGIN
   IF ( a = 10 ) THEN
      DBMS_OUTPUT.PUT_LINE('Value of a is 10' );
   ELSIF ( a = 20 ) THEN
      DBMS_OUTPUT.PUT_LINE('Value of a is 20' );
   ELSIF ( a = 30 ) THEN
      DBMS_OUTPUT.PUT_LINE('Value of a is 30' );
   ELSE
       DBMS_OUTPUT.PUT_LINE('None of the values is matching');
   END IF;
  DBMS_OUTPUT.PUT_LINE('Exact value of a is: '|| a );
END;
/
```

4. **NESTED IF**

Syntax:

```
IF condition THEN
statement1;
```

```
ELSE
        IF condition THEN
        statement2;
        ELSE
        statement3;
         END IF;
 END IF;
```

SELECTION IN PL/SQL (Sequential Controls)

5. **SIMPLE CASE**

Syntax:

```
CASE SELECTOR
WHEN Expr1 THEN statement1;
WHEN Expr2 THEN statement2;
:
ELSE
Statement  n;
END CASE;
```

Example:

```
DECLARE
  grade CHAR(1) := 'A';
BEGIN
  CASE grade
            WHEN 'A' THEN DBMS_OUTPUT.PUT_LINE('Excellent');
            WHEN 'B' THEN DBMS_OUTPUT.PUT_LINE('Very good');
            WHEN 'C' THEN DBMS_OUTPUT.PUT_LINE('Well done');
            WHEN 'D' THEN DBMS_OUTPUT.PUT_LINE('You passed');
            WHEN 'F' THEN DBMS_OUTPUT.PUT_LINE('Better try again');
            ELSE DBMS_OUTPUT.PUT_LINE('No such grade');
  END CASE;
END;
/
```

6. **SEARCHED CASE:**

Syntax:

```
CASE
WHEN searchcondition1 THEN
statement1; WHEN searchcondition2
THEN statement2;
::
```

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

Example:

```
DECLARE
  grade char(1) := 'B';
BEGIN
  CASE
    WHEN GRADE = 'A' THEN DBMS_OUTPUT.PUT_LINE('Excellent');
    WHEN GRADE = 'B' then dbms_output.put_line('Very good');
    when grade = 'C' then dbms_output.put_line('Well done');
    when grade = 'D' then dbms_output.put_line('You passed');
    when grade = 'F' then dbms_output.put_line('Better try again');
    else dbms_output.put_line('No such grade');
  end case;
END;
/
```

ITERATIONS IN PL/SQL

Iterative control Statements are used when we want to repeat the execution of one or more statements for specified number of times.

There are three types of loops in PL/SQL:

- Simple Loop
- While Loop
- For Loop

7. **SIMPLE LOOP Syntax:**

```
LOOP
statement1;
EXIT [ WHEN Condition];
END LOOP;
```

Example:

```
Declare
A number:=10;
Begin
Loop
a := a+25;
exit when a=250;
end loop;
```

```
dbms_output.put_line(to_char(a));
end;
/
```

8. **WHILE LOOP Syntax**

```
WHILE condition
LOOP statement1;
statement2;
END LOOP;
```

Example:

```
Declare
i number:=0;
j number:=0;
begin
while i<=100 Loop
j := j+i;
i :=   i+2;
end loop;
dbms_output.put_line(“the value of j is‟ ||j);
end;
/
```

9. **FOR LOOP**

   Syntax:

```
FOR counter IN [REVERSE]
LowerBound..UpperBound
 LOOP
statement1;
statement2;
END LOOP;
```

Example:

```
Begin
For I in 1..2
 Loop
Update emp set field = value where
condition;
End loop;
End;
/
```

OUTCOME: Students are able to implement PL/SQL programs.

SAMPLE QUESTIONS

1. Write a PL/SQL block which will accept product_no from user and subtract an amount of 200 from cost price if the cost price has minimum of Rs.3000 after the subtraction .The process is to be performed on product_master table (use exceptions to handle standard error conditions .Don't use cursor definition).
2. Write a PL/SQL block to calculate the area and circumference of a circle for the radius varying from lower to upper values. Lower and upper values are accepted from the user . Store radius, circumference ,area in the table named circle_area.
3. Write PL/SQL block to invert o given no:

   a) Consider no as a string
   b) Consider no as integer.
4. Write PL/SQL block to check whether the given number is prime or not?
5. Write PL/SQL block to accept a product_no and update the Sell_price of that product by 2 .Display appropriate message based on the existence of the record in the product _master table
6. Write a PL/SQL block to find the square, cube, and double of a number inputted with a substitution variable, and print the results
7. Write a PL/SQL block to find out if a year is a leap year.
8. Write a PL/SQL block to print all odd numbers between 1 and 10 using a basic loop
9. Write a PL/SQL block to display the Palindrome numbers between 1 and 100.

EXERCISE NUMBER: 12
CURSORS

OBJECTIVE: To implement cursors for various operations.

THEORY

A cursor is a temporary work area created in system memory when a SQL statement is executed. It contains information on a select statement and the rows of data accepted by it. There are two types of cursors in PL/SQL

IMPLICIT CURSORS

These are created by default when DML statements like INSERT, UPDATE and DELETE statements are executed. They are also created when a SELECT statement that returns just one row is executed.

In PL/SQL, the most recent implicit cursor referred to as the **SQL cursor**, always has the attributes like %FOUND, %ISOPEN, %NOTFOUND, and %ROWCOUNT.

The following table provides the description of the most used attributes:

Attribute	**Description**
%FOUND	Returns TRUE if an INSERT, UPDATE, or DELETE statement affected one or more rows or a SELECT INTO statement returned one or more rows. Otherwise, it returns FALSE.
%NOTFOUND	The logical opposite of %FOUND. It returns TRUE if an INSERT, UPDATE, or DELETE statement affected no rows, or a SELECT INTO statement returned no rows. Otherwise, it returns FALSE.
%ISOPEN	Always returns FALSE for implicit cursors, because Oracle closes the SQL cursor automatically after executing its associated SQL statement.
%ROWCOUNT	Returns the number of rows affected by an INSERT, UPDATE, or DELETE statement, or returned by a SELECT INTO statement.

EXPLICIT CURSORS

They must be created when we are executing a SELECT statement that returns more than one row. Even though the cursor stores multiple records, only one record can be processed at a time which is called current row. On fetching, the current row position moves to next row. An explicit cursor should be defined in the declaration section of the PL/SQL Block.

Syntax for creating a cursor is as given below:

CURSOR cursor_name IS select_statement;

- cursor_name – A suitable name for the cursor.
- select_statement – A select query which returns multiple rows.

Working with an explicit cursor involves four steps:

- Declaring the cursor for initializing in the memory
- Opening the cursor for allocating memory
- Fetching the cursor for retrieving data
- Closing the cursor to release allocated memory

Declaring the Cursor

Declaring the cursor defines the cursor with a name and the associated SELECT statement

Syntax:

```
DECLARE
  CURSOR <cursor_name> IS
  SELECT statement;
```

Opening the Cursor

Opening the cursor allocates memory for the cursor and makes it ready for fetching the rows returned by the SQL statement into it.

Syntax to open a cursor is:

OPEN < cursor_name>;

Fetching the Cursor

Fetching the cursor involves accessing one row at a time.

Syntax to fetch records from a cursor is:

```
FETCH <cursor_name> INTO  <record_name>;
                OR
FETCH <cursor_name> INTO <variable_list>;
```

Closing the Cursor

Closing the cursor means releasing the allocated memory.

Syntax to close a cursor is:

CLOSE <cursor_name>;

General Form of using an explicit cursor is:

```
        DECLARE
                Variables;
                Records;
                Create a cursor
        BEGIN
```

```
            Open cursor;
            FETCH cursor;
            PROCESS records;
            CLOSE cursor;
      END;
```

Using Loops with Explicit Cursors:
Oracle provides three types of cursors namely SIMPLE LOOP, WHILE LOOP and FOR LOOP. These loops can be used to process multiple rows in the cursor.
Cursor with a FOR Loop:
When using FOR LOOP we need not declare a record or variables to store the cursor values, need not open, fetch and close the cursor. These functions are accomplished by the FOR LOOP automatically.
General Syntax for using FOR LOOP:

```
      FOR record_name IN cusror_name
      LOOP
        process the row...
      END LOOP;
```

Example:
1. Cursor using Simple loop

```
SQL> SET SERVEROUTPUT ON;
SQL>DECLARE
            CURSOR C1 IS
            SELECT DEPTNO,DNAME FROM DEPT;
            V_DNO DEPT.DEPTNO%TYPE;
            V_DNAME DEPT.DNAME%TYPE;
       BEGIN
            OPEN C1;
            DBMS_OUTPUT.PUT_LINE('DNO DNAME');
            LOOP
                  FETCH C1 INTO V_DNO,V_DNAME;
                  EXIT WHEN C1%NOTFOUND;
                  DBMS_OUTPUT.PUT_LINE(V_DNO||' '||V_DNAME);
            END LOOP;
            CLOSE C1;
      END;
      /
PL/SQL procedure successfully completed.
```

2. Cursor using Simple loop and record variable

```
SQL>DECLARE
                     CURSOR C1 IS
                     SELECT DNAME,COUNT(*) AS COUNTEMP
                     FROM EMPLOYEE,DEPT
                     WHERE DEPT.DEPTNO=EMPLOYEE.DEPTNO
                     GROUP BY DNAME;
                    V_REC C1%ROWTYPE;
           BEGIN
                    OPEN C1;
                              DBMS_OUTPUT.PUT_LINE(' DNAME  COUNT OF EMP');
                    LOOP
                              FETCH C1 INTO V_REC;
                              EXIT WHEN C1%NOTFOUND;
                    DBMS_OUTPUT.PUT_LINE(V_REC.DNAME||'    '||V_REC.COUNTEMP);
                    END LOOP;
                    CLOSE C1;
          END;
PL/SQL procedure successfully completed.
```

3. **Cursor using FOR LOOP**

```
SQL>DECLARE
                     CURSOR C1 IS
                     SELECT DNAME,COUNT(*) AS COUNTEMP
                     FROM EMPLOYEE,DEPT
                     WHERE DEPT.DEPTNO=EMPLOYEE.DEPTNO
                     GROUP BY DNAME;
                    BEGIN
                              DBMS_OUTPUT.PUT_LINE(' DNAME  COUNT OF EMP');
                    FOR V_CURVAR IN C1
                    LOOP
                              DBMS_OUTPUT.PUT_LINE(V_CURVAR.DNAME||'  '||
                              V_CURVAR.COUNTEMP);
                    END LOOP;
                    END;
```

PL/SQL procedure successfully completed.

OUTPUT:

```
dname  count of emp
cse          4
it           1
ece          4
ae           1
```

OUTCOME: Students are able to implement cursors for various operations.

SAMPLE QUESTIONS

1. Create a PL/SQL block that declares a cursor. Display the Eid ,Ename, and salary of employees who joined in a particular year(accept year from the user).(use open fetch)
2. Create a PL/SQL block to display the Eid , Ename ,Company_id of employees who lives in Bombay. (for loop)
3. Create a PL/SQL block to compute the average salary of employees and display the name and salary which are above and below average salary.
4. Create a PL/SQL block to find out the details of employee having second largest salary.
5. List the names of employees who works in the same company as that of Sunil. (use parameterised cursor)
6. Create a PL/SQL block to increase the salary of the employee in Company_id 10. The salaries increases 15% for employees whose salary is less than 10000 & 10% for the employees whose salary is greater than 10000.

EXERCISE NUMBER: 13

PROCEDURE AND FUNCTIONS

OBJECTIVE: To develop procedures and function for various operations.

THEORY
PL/SQL PROCEDURES
A stored procedure is a named PL/SQL block which performs one or more specific task. This is similar to a procedure in other programming languages. A procedure has a header and a body. The header consists of the name of the procedure and the parameters or variables passed to the procedure. The body consists or declaration section, execution section and exception section similar to a general PL/SQL Block.

Declarative Part
It is an optional part. However, the declarative part for a subprogram does not start with the DECLARE keyword. It contains declarations of types, cursors, constants, variables, exceptions, and nested subprograms. These items are local to the subprogram and cease to exist when the subprogram completes execution.

Executable Part
This is a mandatory part and contains statements that perform the designated action.

Exception-handling
This is again an optional part. It contains the code that handles run-time errors.

Advantages:
- ➢ Procedures promote reusability and maintainability. Once validated, they can be used in number of applications. If the definition changes, only the procedure are affected, this greatly simplifies maintenance.
- ➢ Modularized program development:
    - Group logically related statements within blocks.
    - Nest sub-blocks inside larger blocks to build powerful programs.
    - Break down a complex problem into a set of manageable well defined logical modules and implement the modules with blocks.

CREATING A PROCEDURE

A procedure is created with the CREATE OR REPLACE PROCEDURE statement.

Syntax for the CREATE OR REPLACE PROCEDURE statement is as follows:

```
CREATE [OR REPLACE] PROCEDURE procedure_name
[(parameter_name [IN | OUT | IN OUT] type [, ...])]
{IS | AS}
BEGIN
  < procedure_body >
END procedure_name;
```

- **procedure-name** specifies the name of the procedure.
- [OR REPLACE] option allows modifying an existing procedure.
- **Parameter_name:** It is the name of the argument to the procedure. Parenthesis can be omitted if no arguments are present.
- **IN:** Specifies that a value for the argument must be specified when calling the procedure ie., used to pass values to a sub-program. This is the default parameter.
- **OUT:** Specifies that the procedure passes a value for this argument back to it's calling environment after execution ie. used to return values to a caller of the sub-program.
- **IN OUT:** Specifies that a value for the argument must be specified when calling the procedure and that procedure passes a value for this argument back to its calling environment after execution.
- ***procedure-body*** contains the executable part.
- The **AS** keyword is used instead of the IS keyword for creating a standalone procedure.

EXECUTING A STANDALONE PROCEDURE

A standalone procedure can be called in two ways:

a. Using the EXECUTE keyword

   EXECUTE [or EXEC] procedure_name;

b. Calling the name of the procedure from a PL/SQL block

PL/SQL FUNCTIONS

A function is a named PL/SQL Block which is similar to a procedure. The major difference between a procedure and a function is, a function must always return a value, but a procedure may or may not return a value.

CREATING A FUNCTION

A standalone function is created using the CREATE FUNCTION statement.

Syntax:

```
CREATE [OR REPLACE] FUNCTION function_name
 [(parameter_name [IN | OUT | IN OUT] type [, ...])]
 RETURN return_datatype
 {IS | AS}
 BEGIN
   < function_body >
 END [function_name];
```

Where,

- function-name specifies the name of the function.
- [OR REPLACE] option allows modifying an existing function.
- The optional parameter list contains name, mode and types of the parameters.
- IN represents that value will be passed from outside
- OUT represents that this parameter will be used to return a value outside of the procedure.
- The function must contain a return statement.
- RETURN clause specifies that data type you are going to return from the function.
- function-body contains the executable part.
- The AS keyword is used instead of the IS keyword for creating a standalone function.

CALLING A FUNCTION

To call a function we need to pass the required parameters along with function name and if function returns a value then we can store returned value.

Examples:

Tables used:

```
SQL> select * from ititems;
ITEMID   ACTUALPRICE     ORDID      PRODID
-------  -----------     --------   ---------
101      2000            500        201
102      3000            1600       202
103      4000            600        202
```

Program for general procedure – selected record‘s price is incremented by 500 , executing the procedure created and displaying the updated table

```
SQL> create procedure itsum(identity number, total number) is price number;
     null_price exception;
     begin
     select actualprice into price from ititems where itemid=identity;
```

```
if price is null then
raise null_price;
else
update ititems set actualprice=actualprice+total where itemid=identity;
end if;
exception
when null_price then
dbms_output.put_line('price is null');
end;
/
```

Procedure created.

```
SQL> exec itsum(101, 500);
```

PL/SQL procedure successfully completed.

```
SQL> select * from ititems;

ITEMID   ACTUALPRICE     RDID       PRODID

-------  -----------     --------   ---------

101      2000            500        201

102      3000            1600       202

103      4000            600        202
```

PROGRAM FOR FUNCTION AND IT'S EXECUTION

```
SQL> create function trainfn (trainnumber number) return number is
        trainfunction ittrain.tfare % type;
        begin
        select tfare into trainfunction from ittrain where tno=trainnumber;
        return(trainfunction);
        end;
        /
```

Function created.

```
SQL>  declare
        total number;
        begin
        total:=trainfn (1001);
        dbms_output.put_line('Train fare is Rs. '||total);
        end;
          /
```

OUTPUT

```
Train fare is Rs.550
```

PL/SQL procedure successfully completed.

OUTCOME: Students are able to implement procedures and function for various operations.

SAMPLE QUESTIONS

1. Write procedure to describe the names of all the clients having the specified character as the i^{th} letter in the names. From the main program read the values for the character and i.
2. Write a function to count number of orders and display the value in main program.
3. Write a procedure to find minimum and maximum product price .Display the value in the main program.
4. Write a function to find the sum total of all billed orders for a specific month .The month name is accepted from the main program and the result is also displayed in the main program.
5. Write procedure to find the details of orders placed by customers whose name contain ‘a’ as the second letter.
6. Write a procedure to find the product having the ith maximam sell prize from the product_ master table .

EXERCISE NUMBER: 14
TRIGGER AND EXCEPTIONS

OBJECTIVE: To create triggers for various events such as insertion, updation, Deletion and to resolve runtime errors using Exception Handling.

THEORY

A trigger is a stored procedure that defines an action that the database automatically takes when some database-related event such as Insert, Update or Delete occur.

TRIGGER VS. PROCEDURE

TRIGGER	PROCEDURES
These are named PL/SQL blocks.	These are named PL/SQL blocks.
These are invoked automatically.	User as per need invokes these.
These can't take parameters.	These can take parameters.
These are stored in database.	These are stored in database.

TYPES OF TRIGGERS

The various types of triggers are as follows:

- Row Level Triggers
- Statement Level Triggers
- Before Triggers
- After Triggers

Row Level Triggers

Row level triggers execute once for each row in a transaction. The commands of row level triggers are executed on all rows that are affected by the command that enables the trigger. Row level triggers are created using the FOR EACH ROW clause in the CREATE TRIGGER command.

Statement Level Triggers

Statement level triggers are triggered only once for each transaction. For example when an UPDATE command update 15 rows, the commands contained in the trigger are executed only once, and not with every processed row. Statement level trigger are the default types of trigger created via the CREATE TRIGGER command.

Before Trigger

When a trigger is defined, we can specify whether the trigger must occur before or after the triggering event i.e. INSERT, UPDATE, or DELETE commands. BEFORE trigger execute the trigger action before the triggering statement. These types of triggers are commonly used in the following situation.

BEFORE triggers are used when the trigger action should determine whether or not the triggering statement should be allowed to complete. For example: To prevent deletion on Sunday, for this we have to use Statement level before trigger on DELETE statement.

BEFORE triggers are used to derive specific column values before completing a triggering INSERT or UPDATE statement.

After Trigger

AFTER trigger executes the trigger action after the triggering statement is executed.

AFTER triggers are used when we want the triggering statement to complete before

executing the trigger action.

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.

Row Level Trigger vs. Statement Level Trigger:

Row Level Trigger	Statement Level Trigger
These are fired for each row affected by the DML statement.	These are fired once for the statement instead of the no of rows modified by it.

Syntax:

```
CREATE [OR REPLACE ] TRIGGER <trigger_name>
        {BEFORE | AFTER | INSTEAD OF }
        {INSERT [OR] | UPDATE [OR] | DELETE}
        [OF col_name]
        ON table_name
        [REFERENCING OLD AS o NEW AS n]
        [FOR EACH ROW]
        WHEN (condition)
```

```
DECLARE
Declaration-statements
BEGIN
Executable-statements
EXCEPTION
Exception-handling-statements
END;
```

In the syntax:

- CREAT E [OR REPLACE] TRIGGER trigger_name: Creates or replaces an existing trigger with the *trigger_name*.
- {BEFORE | AFTER | INSTEAD OF} : This specifies when the trigger would be executed. The INST EAD OF clause is used for creating trigger on a view.
- {INSERT [OR] | UPDAT E [OR] | DELET E}: T his specifies the DML operation.
- [OF col_name]: T his specifies the column name that would be updated.
- [ON table_name]: T his specifies the name of the table associated with the trigger.
- [REFERENCING OLD AS o NEW AS n]: This allows to refer new and old values for various DML statements, like INSERT, UPDATE and DELETE.
- [FOR EACH ROW]: This specifies a row level trigger, i.e., the trigger would be executed for each row being affected. Otherwise the trigger will execute just once when the SQL statement is executed, which is called a table level trigger.
- WHEN (condition): This provides a condition for rows for which the trigger would fire. This clause is valid only for row level triggers.

Example: Create a trigger that insert current user into a username column of an existing table

```
SQL> CREATE TABLE itstudent4(name varchar2(15),username varchar2(15));
 Table created.
SQL> CREATE OR REPLACE TRIGGER itstudent4_tr
            before insert on itstudent4
            for each row
            declare
            name varchar2(20);
            begin
            select user into name from dual;
            :new.username:=name;
            end;
          /
```

Trigger created.

Output:

```
SQL> insert into itstudent4 values('&name','&username');
```

```
Enter value for name: akbar
Enter value for username: ranjani
old  1: insert into itstudent4 values('&name','&username')
new  1: insert into itstudent4 values('akbar','ranjani')
1 row created.

SQL> /
Enter value for name: suji
Enter value for username: priya
old  1: insert into itstudent4 values('&name','&username')
new  1: insert into itstudent4 values('suji','priya')
1 row created.
SQL> select * from itstudent4;
NAME          USERNAME
--------------- ---------------
akbar         SCOTT
suji          SCOTT
```

PL/SQL EXCEPTION

An error condition during a program execution is called an exception in PL/SQL. PL/SQL supports programmers to catch such conditions using **EXCEPTION** block in the program and an appropriate action is taken against the error condition.

Syntax for Exception Handling

The syntax for exception handling is as follows. The default exception will be handled using *WHEN others THEN*:

```
DECLARE
  <declarations section>
BEGIN
  <executable command(s)>
EXCEPTION
  <exception handling goes here >
  WHEN exception1 THEN
     exception1-handling-statements
  WHEN exception2  THEN
    exception2-handling-statements
  WHEN exception3 THEN
    exception3-handling-statements
  ........
  WHEN others THEN
```

```
    exception3-handling-statements
END;
```

RAISING EXCEPTIONS

Exceptions are raised by the database server automatically whenever there is any internal database error, but exceptions can be raised explicitly by the programmer by using the command **RAISE**.

Syntax of raising an exception:

```
DECLARE
        exception_name EXCEPTION;
BEGIN
         IF condition THEN
                  RAISE exception_name;
         END IF;
EXCEPTION
        WHEN exception_name THEN
        statement;
END;
```

TYPES OF EXCEPTIONS

There are two types of exceptions:

- System-defined exceptions
- User-defined exceptions

System-defined Exceptions

System exceptions are automatically raised by Oracle, when a program violates a RDBMS rule. There are some system exceptions which are raised frequently, so they are pre-defined and given a name in Oracle which are known as Named System Exceptions.

For example: NO_DATA_FOUND and ZERO_DIVIDE are called Named System exceptions.

Named system exceptions are:
1) not declared explicitly,
2) Raised implicitly when a predefined Oracle error occurs,
3) Caught by referencing the standard name within an exception-handling routine.

Exception Name	**Reason**	**Error Number**
CURSOR_ALREADY_OPEN	When you open a cursor that is already open.	ORA-06511

INVALID_CURSOR	When you perform an invalid operation on a cursor like closing a cursor, fetch data from a cursor that is not opened.	ORA-01001
NO_DATA_FOUND	When a SELECT...INTO clause does not return any row from a table.	ORA-01403
TOO_MANY_ROWS	When you SELECT or fetch more than one row into a record or variable.	ORA-01422
ZERO_DIVIDE	When you attempt to divide a number by zero.	ORA-01476

User-Defined Exceptions

We can explicitly define exceptions based on business rules. These are known as user-defined exceptions.

Steps to be followed to use user-defined exceptions:
• They should be explicitly declared in the declaration section.
• They should be explicitly raised in the Execution Section.
• They should be handled by referencing the user-defined exception name in the exception section.

RAISE_APPLICATION_ERROR ( )

RAISE_APPLICATION_ERROR is a built-in procedure in Oracle which is used to display the user-defined error messages along with the error number whose range is in between -20000 and -20999. RAISE_APPLICATION_ERROR raises an exception but does not handle it.

Syntax to use this procedure is:

RAISE_APPLICATION_ERROR (error_number, error_message);

• The Error number must be between -20000 and -20999
• The Error_message is the message you want to display when the error occurs.

Steps to be followed to use RAISE_APPLICATION_ERROR procedure:
1. Declare a user-defined exception in the declaration section.
2. Raise the user-defined exception based on a specific business rule in the execution section.
3. Finally, catch the exception and link the exception to a user-defined error number in RAISE_APPLICATION_ERROR.

Examples:

System-defined exception

1. SQL> DECLARE

```
v_dnam dept.dname%type;
v_dn dept.deptno%type;
begin
v_dn:=&dno;
select dname into v_dnam from dept where deptno=v_dn;
dbms_output.put_line('dept name is '||v_dnam);
exception
when no_data_found then
dbms_output.put_line('invalid dept no');
end;
/
```

2. SQL>**DECLARE**

```
n number;
m number;
quo number;
begin
n:=&n;
m:=&m;
quo:=n/m;
dbms_output.put_line('quo is: '||quo);
exception
when zero_divide then
dbms_output.put_line('divide by zero error');
end;
/
```

User-defined Exception

3. SQL> **DECLARE**

```
v_count number;
v_dn dept.deptno%type;
e_invaliddeptno exception;
begin
v_dn:=&dno;
select count(*) into v_count from dept where deptno=v_dn;
if v_count=0 then
raise e_invaliddeptno;
end if;
dbms_output.put_line('dept no is '||v_dn);
exception
when e_invaliddeptno then
dbms_output.put_line('no such dept no exists');
end;
/
```

SQL> CREATE TRIGGER EMPTRIG

```
 before insert or update or delete
 on emp
for each row
 begin
 raise_application_error(-20010,'You cannot do manipulation');
 end; /
```

OUTCOME: Students are able to Created triggers for various events such as insertion, updation, deletion etc. and resolve runtime errors using Exception handling.

SAMPLE QUESTIONS

1. Create a trigger to control the insertion operation on the product_master table .Insertion is possible if profit_percent of the current entry is greater than the maximum profit_percent of the values available else it should give an error message.
2. Create a trigger to control the updation operation on product-master table which allow the updation when profit_percent value given now is greater than the earlier value else it should given error message .
3. Create a trigger on sales_order table which allow deletion if order_date is less than 12-JUL-12.

EXERCISE NUMBER: 15
PACKAGES

OBJECTIVE: To understand encapsulation of procedures and functions.

THEORY

A package is an encapsulated collection of related program objects (for example, procedures, functions, variables, constants, cursors, and exceptions) stored together in the database. A package has two components: package specification and package body or definition.

Package Specification

A package specification declares the type, memory variables, constants, exceptions, cursors, subprograms that are available for use. It contains all information about the content of the package, but excludes the code for the subprograms. All objects placed in the specification are called **public** objects. Any subprogram not in the package specification but coded in the package body is called a **private** object.

Package Specification Syntax

```
CREATE [OR REPLACE] PACKAGE <package_name>
   IS | AS
   [variable_declaration ...]
   [constant_declaration ...]
   [exception_declaration ...]
   [cursor_specification ...]
         [PROCEDURE [Schema..] procedure_name
                  [ (parameter {IN,OUT,IN OUT} datatype [,parameter]) ]
         ]
         [FUNCTION [Schema..] function_name
                  [ (parameter {IN,OUT,IN OUT} datatype [,parameter]) ]
                  RETURN return_datatype
         ]
END [package_name];
```

Package Body

A package body fully defines cursors, functions, procedures and thus implements specifications. The package body has the codes for various methods declared in the package specification and other private declarations, which are hidden from code outside the package.

The CREATE PACKAGE BODY statement is used for creating the package body.

Package Body Syntax

```
CREATE [OR REPLACE] PACKAGE BODY package_name
   IS | AS
   [private_variable_declaration ...]
   [private_constant_declaration ...]
         BEGIN
                  [initialization_statement]
                  [PROCEDURE [Schema..] procedure_name
                           [ (parameter [,parameter]) ]
                           IS | AS
                                    variable declarations;
                                    constant declarations;
                           BEGIN
                                    statement(s);
                           EXCEPTION
                                    WHEN ...
                           END
                  ]
                  [FUNCTION  [Schema..] function_name
                                    [ (parameter [,parameter]) ]
                                    RETURN return_datatype

                           IS | AS
                                    variable declarations;
                                    constant declarations;
                           BEGIN
                                    statement(s);
                           EXCEPTION
                                    WHEN ...
                           END
                  ]

         [EXCEPTION
                  WHEN built-in_exception_name_1 THEN
                           User defined statement (action) will be taken;
]
END;
/
```

Example:

```
SQL> CREATE or REPLACE PACKAGE pkg1
             AS
           PROCEDURE pro1
                     (no in number);
           FUNCTION fun1
                     (no in number)
                     RETURN varchar2;
           END;
/
```

Package created.

```
SQL>  CREATE or REPLACE PACKAGE BODY pkg1 AS
           PROCEDURE pro1(no in number)
                     IS
           temp emp%ROWTYPE;
                     BEGIN
                               SELECT * INTO temp FROM emp WHERE empno = no;
           dbms_output.put_line (temp.empno||'     '||
                          temp.ename||'   '||
                          temp.deptno||'   '||
                          temp.sal||'   ');
                     END;

           FUNCTION fun1(no in number) return varchar2
                     IS
                     name varchar2(20);
                     BEGIN
                               SELECT ename INTO name FROM emp WHERE empno = no;
                               RETURN name;
                     END;
END;
/
```

Package body created.

```
SQL> set serveroutput on;
SQL> DECLARE
      no number := &no;
       name1 varchar2(20);
    BEGIN
       dbms_output.put_line('Procedure Result');
       pkg1.pro1(no);
```

```
     dbms_output.put_line('Function Result');
    name1 := pkg1.fun1(no);
    dbms_output.put_line(name1);
  END;
  /
Enter value for no: 1
old   2:        no number := &no;
new   2:        no number := 1;
Procedure Result
1     arun   101   5000
Function Result
arun

PL/SQL procedure successfully completed.
```

OUTCOME: Students are able to reuse the stored procedures and functions.

SAMPLE QUESTIONS

1. Create a package with following functions or procedures
   a. Write a procedure to find the area of circle of given radius.
   b. Write a procedure to find the area of a right angled triangle
   c. Write a function to find the area of a triangle by giving a,b,c.
   d. Write a function to find the volume of cylinder.

2. Create package in oracle which contains procedure or function to do the following operation on product_master table
   a) Find the highest profit-percent item in the table.
   b) To display the details of items whose sell price is greater than a given value

3. Create a package in oracle which contains procedure or function to do the following
   a. Factorial of a given number
   b. To find the square of a given number.
   c. To find the nth term of the Fibonacci series
   d. To check whether the given number is prime or not.

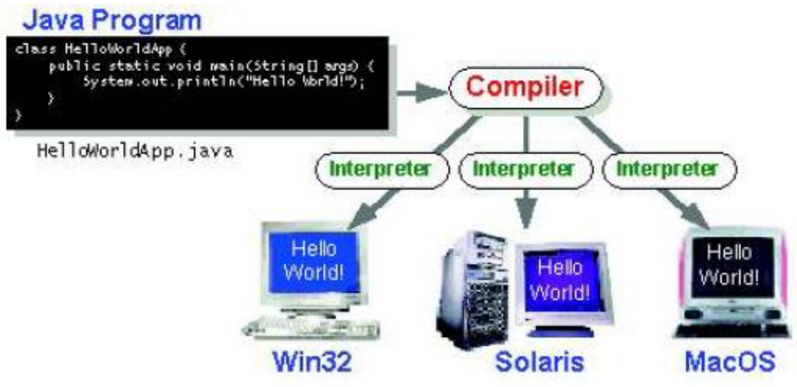
EXERCISE NUMBER: 16
INTRODUCTION TO JAVA & NETBEANS

OBJECTIVE: To familiarize the basic concepts of java programming and Netbeans

THEORY

The target of Java is to write a program once and then run this program on multiple operating systems.Java has the following properties:

- **Platform independent**: Java programs use the Java virtual machine as abstraction and do not access the operating system directly. This makes Java programs highly portable. A Java program (which is standard complaint and follows certain rules) can run unmodified on all supported platforms, e.g. Windows or Linux.
- **Object-orientated programming language**: Except the primitive data types, all elements in Java are objects.
- **Strongly-typed programming language**: Java is strongly-typed, e.g. the types of the used variables must be pre-defined and conversion to other objects is relatively strict, e.g. must be done in most cases by the programmer.
- **Interpreted and compiled language**: Java source code is transferred into the bytecode format which does not depend on the target platform. These bytecode instructions will be interpreted by the Java Virtual machine (JVM). The JVM contains a so called Hotspot-Compiler which translates performance critical bytecode instructions into native code instructions.



```
class HelloWorldApp {
    public static void main(String[] args) {
        System.out.println("Hello World!");
    }
}
```

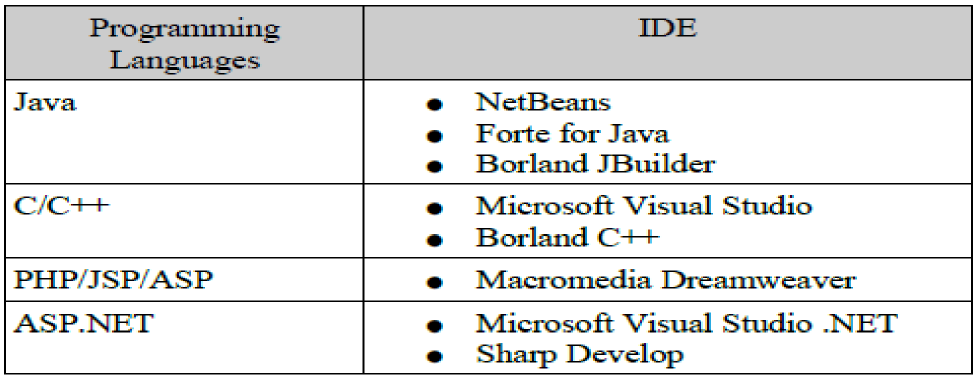
- **Automatic memory management**: Java manages the memory allocation and de-allocation for creating new objects. The program does not have direct access to the memory. The so-called garbage collector deletes automatically objects to which no active pointer exists.

- **Multithreaded:** these capabilities of java provide capability to single program to perform several tasks simultaneously. Multithreading is very useful for developing applications like animation, GUI, and networks. Unlike other programming languages multithreading is integrated in Java. In other programming languages, you have to call operating systems specific procedures to perform the task of multithreading.
- **Distributed:** Using java programs simultaneous processing can be done on multiple computer on Internet. Java provides strong networking features to write distribute programs.
- **Secure:** Design of java is having multiple layers of security that ensures proper access of private data and have control over access of disk files.

Java is case sensitive, e.g. variables called myValue and myvalue are treated as different variables.

NetBeans IDE

IDE stands for Integrated Development Environment. An IDE is a programming environment integrated into a software application that provides a GUI builder, a text or code editor, a compiler and/or interpreter and a debugger. IDE programs vary and each one of them usually targeted to certain programming language, the following table shows well-known IDE programs:



Programming Languages	IDE
Java	• NetBeans • Forte for Java • Borland JBuilder
C/C++	• Microsoft Visual Studio • Borland C++
PHP/JSP/ASP	• Macromedia Dreamweaver
ASP.NET	• Microsoft Visual Studio .NET • Sharp Develop

NetBeans is a software development platform written in Java. NetBeans is the best offered IDE for us as Java learners, because it is free compared to JBuilder, more advanced and stable compared to Forte. NetBeans is cross-platform and runs on Microsoft Windows, Mac OS X, Linux, Solaris and other platforms supporting a compatible JVM. The NetBeans Platform is a framework for simplifying the development of Java Swing desktop applications.

OUTCOME: Students are able to understand fundamentals of java programming and Netbeans IDE.

EXERCISE NUMBER: 17
DESIGN GUI USING SWING COMPONENTS

OBJECTIVE: To design GUI using SWING Components and handle events using event handling techniques.

THEORY

Swing is the principal GUI toolkit for the Java programming language. It is a part of the JFC (Java Foundation Classes), which is an API for providing a graphical user interface for Java programs. It is completely written in Java. It is built on the top of AWT (Abstract Windowing Toolkit) API and entirely written in java.Unlike AWT, Java Swing provides platform-independent and lightweight components.The javax.swing package provides classes for java swing API such as JButton, JTextField, JTextArea, JRadioButton, JCheckbox, JMenu, JColorChooser etc.

Difference between AWT and Swing

Java AWT	**Java Swing**
AWT components are **platform-dependent**.	Java swing components are **platform-independent**.
AWT components are **heavyweight**.	Swing components are **lightweight**.
AWT **doesn't support pluggable look and feel**.	Swing **supports pluggable look and feel**.
AWT provides **less components** than Swing.	Swing provides **more powerful components**such as tables, lists, scrollpanes, colorchooser, tabbedpane etc.

Commonly used Methods of Component class

The methods of Component class are widely used in java swing that are given below.

Method	**Description**
public void add(Component c)	add a component on another component.
public void setSize(int width,int height)	sets size of the component.
public void setLayout(LayoutManager m)	sets the layout manager for the component.
public void setVisible(boolean b)	sets the visibility of the component. It is by default false.

JLabel Class

A JLabel just displays text. It has no built-in response to user mouse or keyboard actions. In most common uses, you just create it and add it into a container. The most common constructor for a JLabel is the following.

- **new JLabel(String lbl)**

If you want to change a label during program execution, you can send it a setText() message.

- **void setText(String lbl)**

JButton Class

A JButton is useful when you want to provide the user with the capability of performing an action at any time. The button is constructed with a label that suggests the nature of the action.

- **new JButton(String lbl)**

The action performed by a JButton is defined in the actionPerformed() method of an action listener registered with the JButton.

- **void addActionListener(ActionListener l)**
- **void setText(String lbl)**

JTextField Class

A JTextField is used to entry and display of short single-line text. The text that it initially displays can be specified as a constructor parameter. It can also be constructed with no initial text, with the text added later.

- **new JTextField(String txt)**
- **new JTextField()**
- **void setText(String txt)**

For user text entry, a JTextField usually responds to the user when the user hits the return key. The response to the user is implemented in the actionPerformed() method of an action listener registered with the JTextField. In the actionPerformed() method, you can obtain the text entered by the user with the getText() method.

- **void addActionListener(ActionListener l)**
- **String getText()**

JTextArea Class

A JTextArea can be used to display and edit longer multiline text. You can construct a text area either with text specified by a constructor parameter, or you can start off with no text and add it later.

- **new JTextArea()**
- **new JTextArea(String str)**
- **void setText(String txt)**
- **void append(String txt)**

You can use the code setText("") to clear the text in a text area.

Unless you are doing something sophisticated, you will normally process the text in a text area after the user has taken some other action such as clicking on a button or selecting a menu item. The response would then be implemented in the actionPerformed() method of an action listener registered with the button or menu item. The text can be obtained with the getText() method.

JSlider Class

A JSlider is used to allow the user to select a value from a range of numerical values by moving a tab along a track. A JSlider is normall oriented horizontally, but can be oriented vertically with an extra constructor parameter. Other parameters provide the minimum, maximum, and starting values for the slider.

- **new JSlider(int min, int max, int strt)**
- **new JSlider(JSlider.VERTICAL, int min, int max, int strt)**

To respond to user actions with a JSlider, you create a ChangeListener, implementing the response in its StateChanged() method. The response usually uses the
JSlider's getValue() method to get the current value of the slider.

- **void addChangeListener(ChangeListener l)**
- **int getValue()**

JList Class

A JList displays a list of options that a user can select. Unlike a JComboBox, all of the options are displayed at all times.

You can do some very fancy things with a JList, such as dynamically changing the list of values that it presents. However, the most common use has fixed selection values. The easiest way to construct a JList with fixed selection values is to declare an initialized array of objects (usually strings) and pass it as the JList constructor argument.

- **new JList()**
- **new JList(Object[] listData)**
- **new JList(Vector listData)**

Once a JList is constructed, you need to register a selection listener to respond to user selections. The response is implemented in the listener's valueChanged() method. In that method, you can determine the selected value by sending the JList a getSelectedValue() message.

- **void addListSelectionListener(ListSelectionListener l)**
- **Object getSelectedValue()**

Menu Classes

There are three main classes used in building graphical user interface menus: JMenuItem, JMenu, and JMenuBar. JMenuItem objects are added into a JMenu objects to form a menu. Several JMenu objects can be added into a JMenuBar to form a menu bar. The menu bar is then used as a parameter for the setJMenuBar() method of the JApplet or JFrame class.

JMenuItem Class

A JMenuItem is constructed in one of two ways.

- **new JMenuItem(String text)**
- **new JMenuItem(Action action)**

For the first constructor, an ActionListener is later added to the JMenuItem. The parameter in the second constructor includes an ActionListener.

- **void addActionListener(ActionListener listener)**

JMenu Class

A JMenu is a JMenuItem that can hold a group of JMenuItem objects.

- **new JMenu(String text)**

JMenuItem objects are added to a JMenu with the add() method. Action objects can also be added to a JMenu. When this is done, a JMenuItem is created automatically.

- **void add(<u>JMenuItem</u> menuItem)**
- **void add(<u>Action</u> action)**

The fact that JMenu is a subclass of JMenuItem means that a JMenu can be added to another JMenu. This allows you to construct nested menus.

JMenuBar Class

The JMenuBar class has a simple constructor.

- **new JMenuBar()**Once you have constructed a JMenuBar, you then add JMenu objects to it.

- **void add(<u>JMenu</u> menu)**

The final step in constructing menus is adding a JMenuBar to either a JApplet or a JFrame with the setJMenuBar() method. The JMenuBar is passed as a parameter.

EVENT HANDLING

Event Handling is an important part of GUI based applications. Any program that uses GUI application is event driven.Event describes change of state of any object.
Event handling has three main components.

1. Event source: An object that generates an event.
2. Event listener: Listens to an event and notifies the handler.
3. Event handler: handles a specific event.

Events are generated by event sources. A mouse click, Window closed, key typed etc. are examples of events.All java events are sub-classes of java.awt.AWTEvent class.
Java has two types of events:

1. Low-Level Events: Low-level events represent direct communication from user. A low level event is a key press or a key release, a mouse click, drag, move or release, and so on. Following are low level events.

Event	Description
ComponentEvent	Indicates that a component object (e.g. Button, List, TextField) is moved, resized,
	rendered invisible or made visible again.
FocusEvent	Indicates that a component has gained or lost the input focus.
KeyEvent	Generated by a component object (such as TextField) when a key is pressed, released or typed.
MouseEvent	Indicates that a mouse action occurred in a component. E.g.

	mouse is pressed,
	releases, clicked (pressed and released), moved or dragged.
ContainerEvent	Indicates that a container's contents are changed because a component was added or removed.
WindowEvent	Indicates that a window has changed its status. This low level event is generated by a Window object when it is opened, closed, activated, deactivated, iconified, deiconified or when focus is transferred into or out of the Window.

2. High-Level Events: High-level (also called as semantic events) events encapsulate the meaning of a user interface component. These include following events.

Event	Description
ActionEvent	Indicates that a component-defined action occurred. This high-level event is generated by a component (such as Button) when the component-specific action occurs (such as being pressed).
AdjustmentEvent	The adjustment event is emitted by Adjustable objects like scrollbars.
ItemEvent	Indicates that an item was selected or deselected. This high-level event is generated by an ItemSelectable object (such as a List) when an item is selected or deselected by the user.
TextEvent	Indicates that an object's text changed. This high-level event is generated by an object (such as TextComponent) when its text changes.

The following table lists the events, their corresponding listeners and the method to add the listener to the component.

Event	Event Source	Event Listener	Method to add listener to event source
Low-level Events			
ComponentEvent	Component	ComponentListener	addComponentListener()
FocusEvent	Component	FocusListener	addFocusListener()
KeyEvent	Component	KeyListener	addKeyListener()

MouseEvent	Component	MouseListener
		MouseMotionListener
			addMouseListener()
			addMouseMotionListener()

ContainerEvent	Container	ContainerListene	addContainerListener()
WindowEvent	Window	WindowListener	addWindowListener()
High-level Events			
ActionEvent	Button List MenuItem TextField	ActionListener	addActionListener()
ItemEvent	Choice CheckBox CheckBoxMenuItem List	ItemListener	addItemListener()
AdjustmentEvent	Scrollbar	AdjustmentListener	addAdjustmentListener()
TextEvent	TextField TextArea	TextListener	addTextLIstener()

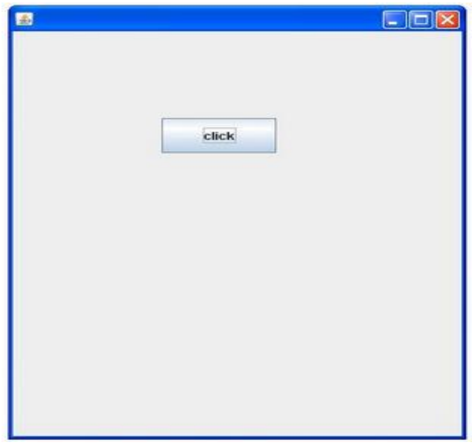
Java Swing Simple Example

This example create one button and add it on the JFrame object inside the main() method.

File: FirstSwingExample.java

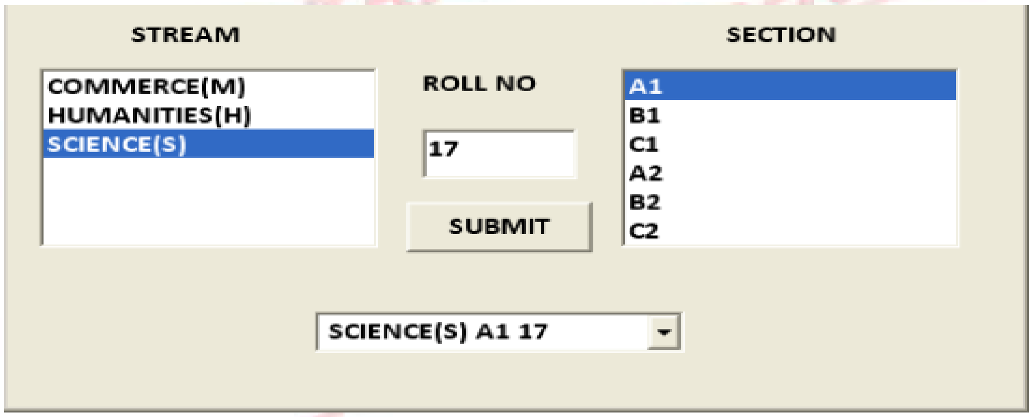
```
import javax.swing.*;
public class FirstSwingExample {
public static void main(String[] args) {
JFrame f=new JFrame();//creating instance of JFrame
JButton b=new JButton("click");//creating instance of JButton
b.setBounds(130,100,100, 40);//x axis, y axis, width, height
f.add(b);//adding button in JFrame
f.setSize(400,500);//400 width and 500 height
f.setLayout(null);//using no layout managers
f.setVisible(true);//making the frame visible
}
}
```

OUTPUT

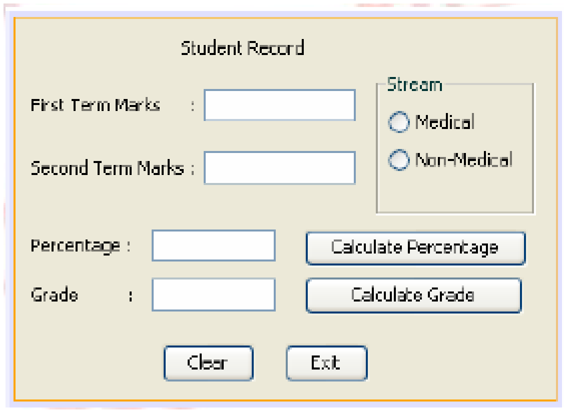


SAMPLE QUESTIONS

1. Create an application that receive a number through a jTextField1 and Print the sum of the individual digits when the submit button is pressed.
2. Create an application that implement a simple arithmetic calculator. Perform appropriate validations.
3. Community Works Department has proposed the following layout to obtain the details of volunteers for a community work project running in your school. The user types his/her roll number and selects his/her stream and section from the jList provided. Upon clicking the submit" button, the selected contents of two lists and rollno are merged in the order <stream><section><Rollno> and added to the combo box. Write a java code to implement the above metioned functionality.



4. Write a GUI program for number conversion from decimal to binary, octal and hexadecimal when the user clicks on “Calculate”.
5. A programmer is required to develop a student record. The school offers two different streams, medical and non-medical, with different grading criteria.
   The following is the data entry screen used to calculate percentage and grade.



i. Write code to prevent the user from entering negative value in First Term Marks text field.
ii. Write the code for the “Clear‟ button to set the default choice to “Medical”.
iii. Write the code to close the form when “Exit‟ button is clicked.
iv. Write the code for the “Calculate Percentage‟ button to calculate the percentage and display in text field “Percentage‟, after finding the total marks of first term and second term (assuming that both marks are out of 100).

EXERCISE NUMBER: 18
APPLETS

OBJECTIVE: To embed java programs in html file and also to draw different shapes.

THEORY
A Java applet is a special kind of Java program that a browser enabled with Java technology can download from the internet and run. An applet is typically embedded inside a web page and runs in the context of a browser. An applet must be a subclass of java.applet.Applet class. The Applet class provides the standard interface between the applet and the browser environment.

java.applet.Applet class
For creating any applet java.applet.Applet class must be inherited. It provides 4 life cycle methods of applet.

1. **public void init():** is used to initialized the Applet. It is invoked only once.
2. **public void start():** is invoked after the init() method or browser is maximized. It is used to start the Applet.
3. **public void stop():** is used to stop the Applet. It is invoked when Applet is stop or browser is minimized.
4. **public void destroy():** is used to destroy the Applet. It is invoked only once.

java.awt.Component class
The Component class provides 1 life cycle method of applet.

1. **public void paint(Graphics g):** is used to paint the Applet. It provides Graphics class object that can be used for drawing oval, rectangle, arc etc.

There are two ways to run an applet
1. By html file.
2. By appletViewer tool (for testing purpose).

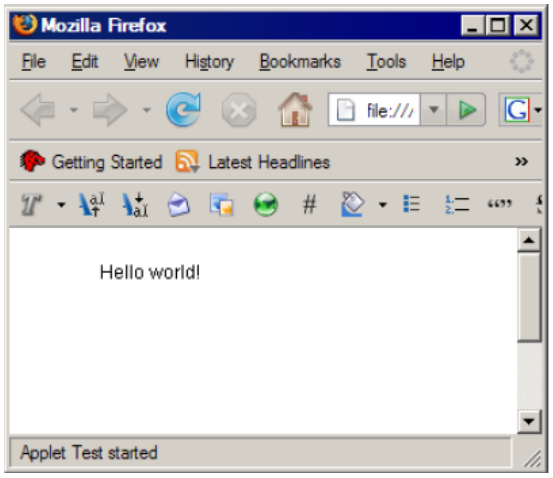
Simple example of Applet by html file:
To execute the applet by html file, create an applet and compile it. After that create an html file and place the applet code in html file. Now click the html file.

Example:

```
//First.java
import java.applet.Applet;
import java.awt.Graphics;
public class First extends Applet{
public void paint(Graphics g){
g.drawString("Hello World!",50,25);
}
}
```

```
//myapplet.html
<html>
<body>
<applet code="First.class" width="400" height="200">
</applet>
</body>
</html>
```

OUTPUT


Mozilla Firefox
File Edit View History Bookmarks Tools Help
file:///
Getting Started Latest Headlines

```
Hello world!
```

Applet Test started

Simple example of Applet by appletviewer tool:

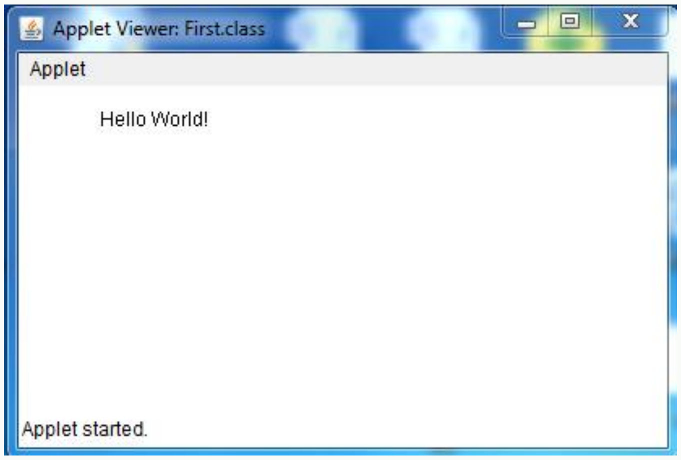
To execute the applet by appletviewer tool, create an applet that contains applet tag in comment and compile it. After that run it by: appletviewer First.java. Now Html file is not required but it is for testing purpose only.

```
//First.java
/*
<applet code="First.class" width="400" height="200">
</applet>
*/
import java.applet.Applet;
import java.awt.Graphics;
public class First extends Applet{
public void paint(Graphics g){
g.drawString("Hello World!",50,25);
}
}
```

To execute the applet by appletviewer tool, write in command prompt:

```
c:\>javac First.java
c:\>appletviewer First.java
```

OUTPUT



```
Hello World!
```

Applet started.

Steps for doing applet programs.

1. Import the package java.applet.*
2. Write the java program file
3. Compile the java file.
4. Embed the java program file in html file.
5. Run the html file in web browser or in applet viewer.

Displaying Graphics in Applet

java.awt.Graphics class provides many methods for graphics programming.

Commonly used methods of Graphics class:

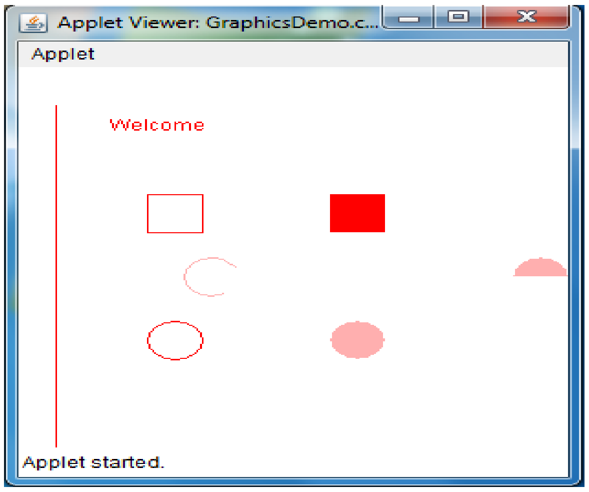
1. **public abstract void drawString(String str, int x, int y):** is used to draw the specified string.
2. **public void drawRect(int x, int y, int width, int height):** draws a rectangle with the specified width and height.
3. **public abstract void fillRect(int x, int y, int width, int height):** is used to fill rectangle with the default color and specified width and height.
4. **public abstract void drawOval(int x, int y, int width, int height):** is used to draw oval with the specified width and height.
5. **public abstract void fillOval(int x, int y, int width, int height):** is used to fill oval with the default color and specified width and height.
6. **public abstract void drawLine(int x1, int y1, int x2, int y2):** is used to draw line between the points(x1, y1) and (x2, y2).
7. **public abstract boolean drawImage(Image img, int x, int y, ImageObserver observer):** is used draw the specified image.
8. **public abstract void drawArc(int x, int y, int width, int height, int startAngle, int arcAngle):** is used draw a circular or elliptical arc.
9. **public abstract void fillArc(int x, int y, int width, int height, int startAngle, int arcAngle):** is used to fill a circular or elliptical arc.
10. **public abstract void setColor(Color c):** is used to set the graphics current color to the specified color.
11. **public abstract void setFont(Font font):** is used to set the graphics current font to the specified font.

EXAMPLE:

```
import java.applet.Applet;
import java.awt.*;
public class GraphicsDemo extends Applet{
public void paint(Graphics g){
g.setColor(Color.red);
g.drawString("Welcome",50, 50);
g.drawLine(20,30,20,300);
g.drawRect(70,100,30,30);
g.fillRect(170,100,30,30);
g.drawOval(70,200,30,30);
g.setColor(Color.pink);
g.fillOval(170,200,30,30);
g.drawArc(90,150,30,30,30,270);
g.fillArc(270,150,30,30,0,180);
}
}
```

```
//myapplet.html
<html>
<body>
<applet code="GraphicsDemo.class" width="300" height="300">
</applet>
</body>
</html>
```

OUTPUT



OUTCOME: Students are able to create different shapes and run java programs from remote locations.

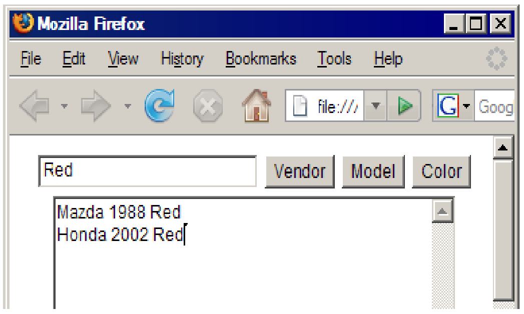
SAMPLE QUESTIONS

1. Write a Java Applet that has 5 int fields, named x1, x2, y1, y2 and radius. Add two buttons to your Applet named "line" and "circle". The behavior of these buttons should be as shown in table below.

Button	**Functionality**
Line	Draws line from (`x1`, `y1`) to (`x2`, `y2`)
Circle	Draws circle centered on (`x1`, `y1`) and has the radius of the value entered in `radius` text field

2. Write Java Applet that implements cars table. Your applet should have three buttons, a text field and a text area (TextArea class is like TextField but have multiple lines). It should look like figure below. The following table shows how should this Applet behave.(Implement database using arrays)

Button	**Behavior**
Vendor	Performs search using car's vendor and gives full details about each matching car in the text area
Model	Similar to "By Vendor", but performs search using model
Color	Similar to "By Vendor", but performs search using color



Red

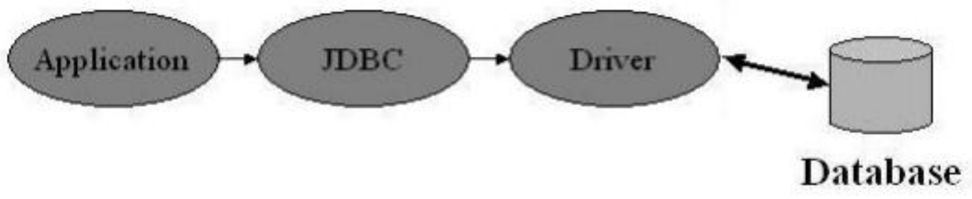
Mazda 1988 Red
Honda 2002 Red

3. Write an applet that contains three buttons OK, CANCEL and HELP and one textfield. if OK is pressed shown on the status bar-"OK is pressed" and the textfield should turn red. When CANCEL is pressed -shown on the status bar-"CANCEL is pressed "and text field should turn green. When HELP is pressed shown on the status bar-"HELP is pressed" and the text field should turn yellow.
4. Write an applet that draw a human face.
5. Create a conversion applet which accepts value in one unit and converts it to another. The input and output unit is selected from a list. Perform conversion to and from Feet, Inches, Centimeters, Meters and Kilometers.

EXERCISE NUMBER: 19
JDBC CONNECTIVITY

OBJECTIVE: To understand the concept of database connectivity and thereby perform database GUI applications.

THEORY
The **JDBC (Java Database Connectivity)** API defines interfaces and classes for writing database applications in Java by making database connections. Using JDBC you can send SQL, PL/SQL statements to almost any relational database. JDBC is a Java API for executing SQL statements and supports basic SQL functionality. It provides RDBMS access by allowing you to embed SQL inside Java code. Because Java can run on a thin client, applets embedded in Web pages can contain downloadable JDBC code to enable remote database access.
JDBC Architecture



Java application calls the JDBC library. JDBC loads a driver which talks to the database

In general, to process any SQL statement with JDBC, you follow these steps:

1. Configuring a JDBC development environment
2. Establishing a connection.
3. Create a statement.
4. Execute the query.
5. Process the ResultSet object.
6. Close the connection.

1. **Configuring a JDBC development environment**
   First of all, you should download the JDBC Driver for your DBMS. For this class, you can use “ojdbc6.jar” provided on class web-site. The ojdbc6.jar is a JDBC driver for Oracle Database 11g.
2. **Establishing a Connection**
   In this step of the jdbc connection process, we load the driver class by calling Class.forName() with the Driver class name as an argument. Once loaded, the Driver class creates an instance of itself. Example for loading JDBC driver.

```
String driverName = "oracle.jdbc.driver.OracleDriver"; // for Oracle
    // String driverName = "com.mysql.jdbc.Driver"; //for MySql
  try {
        // Load the JDBC driver
        Class.forName(driverName);
     }catch (ClassNotFoundException e) {
     // Could not find the database driver
      System.out.println("ClassNotFoundException : "+e.getMessage());
     }
```

The JDBC DriverManager class defines objects which can connect Java applications to a JDBC driver. DriverManager is considered the backbone of JDBC architecture. DriverManager class manages the JDBC drivers that are installed on the system. Its getConnection() method is used to establish a connection to a database. It uses a username, password, and a jdbc url to establish a connection to the database and returns a connection object. A jdbc Connection represents a session/connection with a specific database. Within the context of a Connection, SQL, PL/SQL statements are executed and results are returned. An application can have one or more connections with a single database, or it can have many connections with different databases.

Example for JDBC connection.

```
String serverName = "192.168.0.1";
String portNumber = "1521";
String sid = "orcl";
String username="r570";
String password="r570";
String url = "jdbc:oracle:thin:@" + serverName + ":" + portNumber + ":" + sid; // for Oracle
 try {
   // Create a connection to the database
connection =DriverManager.getConnection(url, username, password);
} catch (SQLException e) { // Could not connect to the database
  System.out.println(e.getMessage());
 }
```

3. **Creating statements & executing queries**

A Statement is an interface that represents a SQL statement. You execute Statement objects, and they generate ResultSet objects, which is a table of data representing a database result set. You need a Connection object to create a Statement object.

There are three different kinds of statements.

- **Statement:** Used to implement simple SQL statements with no parameters
- **PreparedStatement**: (Extends Statement.) Used for precompiling SQL statements that might contain input parameters.
- **CallableStatement:** (Extends PreparedStatement.) Used to execute stored procedures that may contain both input and output parameters. Creating Statement Object:

Example for creating Statement Object

```
Statement stmt =null;

try{
   stmt = connection.createStatement();
...
}
catch(SQLException e){
...
}
```

Example for Creating Prepared Statement Object:

```
PreparedStatement pstmt = null;
try {
   String SQL = "Update Employees SET age = ? WHERE id = ?";
   pstmt = connection.prepareStatement(SQL);
   . . .
}
catch (SQLException e) {
   . . .
}
```

To execute a query, call an execute method from Statement such as the following:

execute: Use this method for DDLcommands.
executeQuery: Returns one ResultSet object.Usethis method if you are using SELECT statements.
executeUpdate: Returns an integer representing the number of rows affected by the SQL statement. Use this method if you are using INSERT,DELETE, or UPDATE SQL statements.

Example for executeQuery

```
String country=”D”;

Statement stmt = null;

String query = " SELECT * FROM CITY WHERE country='”+country+”'”;

try {

 stmt = connection.createStatement();

ResultSet rs = stmt.executeQuery(query);

while (rs.next())

 { String name = rs.getString(1); // or rs.getString("NAME");

String coun= rs.getString(2);

String province = rs.getString(3);

int population = rs.getInt(4);

}

stmt.close();

} catch (SQLException e ) {

System.out.println(e.getMessage());

}
```

The re.next() return „true‟ until there is no more result.

Example for executeUpdate

```
String population=”1705000”;
String cityName=”Hamburg”;

String province=”Hamburg”;

Statement stmt = null;

try {

stmt = connection.createStatement();
String sql = "UPDATE CITY SET population='"+ population +"' WHERE NAME='"+
cityName +"' AND PROVINCE='"+ province +"'";

stmt.executeUpdate(sql);

stmt.close();

} catch (SQLException e ) { System.out.println(e.getMessage()); }
```

OUTCOME: Students are able to perform database GUI application using java as front end and oracle as back end.

SAMPLE QUESTIONS

1. Design a GUI for user registration form. Perform the following validations:
   - Password should be minimum 6 characters containing atleast one uppercase letter one digit and one symbol.
   - Confirm password and password fields should match

   If above conditions are met, display "Registration Successful" otherwise "Registration Failed" after the user clicks the Submit button. Also display the details into another page when user clicks "Display Button".

2. Write a GUI program to acces the details ofemployees such as name, designation and basic pay from a database file.Calculate the total salary by adding basic pay,50% of basic pay as DA and 20 % of basic pay as HRA.Sort the content of database based on the total salary and display the salary statement of all employees.

3. Draw a simple GUI with necessary controls for performing four basic arithmetic operations addition,subtraction,multiplication and division.Connect the GUI to a database so that the database can store all operands ,operations and results of last five arithmetic operations that have been performed.Write the code for handling the events associated with each control and connecting the database.