Index

Sr. No.	List of Practical
1.	**SQL Queries:** □ Design and Develop SQL DDL statements which demonstrate the use of SQL objects such as Table, View, Index, Sequence, Synonym, different constraints etc. □ Write at least 10 SQL queries on the suitable database application using SQL DML statements. Note: Instructor will design the queries which demonstrate the use of concepts like Insert, Select, Update, Delete with operators, functions, and set operator etc.
2.	**SQL Queries – all types of Join, Sub-Query and View:** Write at least10 SQL queries for suitable database application using SQL DML statements. Note: Instructor will design the queries which demonstrate the use of concepts like all types of Join ,Sub-Query and View
3.	**Unnamed PL/SQLcode block: Use of Control structure and Exception handling is mandatory.** Suggested Problem statement: Consider Tables: 1. Borrower (Roll_no, Name, Date_of_Issue, Name_of_Book, Status) 2. Fine (Roll_no, Date, Amt) □ Accept Roll_no and Name_of_Book from user. □ Check the number of days (from Date_of_Issue). □ If days are between 15 to 30 then fine amount will be Rs 5per day. □ If no. of days>30, per day fine will be Rs 50 per day and for days less than 30, Rs. 5 per day. □ After submitting the book, status will change from I to R. □ If condition of fine is true, then details will be stored into fine table. □ Also handles the exception by named exception handler or user define exception handler. **OR** □ MongoDB – Aggregation and Indexing: Design and Develop MongoDB Queries using aggregation and indexing with suitable example using MongoDB. □ MongoDB – Map-reduce operations: Implement Map-reduce operation with suitable example using MongoDB.
4.	**6. Cursors: (All types: Implicit, Explicit, Cursor FOR Loop, Parameterized Cursor)** Write a PL/SQL block of code using parameterized Cursor that will merge the data available in the newly created table N_Roll_Call with the data available in the table

	O_Roll_Call. If the data in the first table already exists in the second table then that data should be skipped. Note: Instructor will frame the problem statement for writing PL/SQL block using all types of Cursors in line with above statement**.**
5.	**Database Connectivity:** Write a program to implement MySQL/Oracle database connectivity with any front end language to implement Database navigation operations (add, delete, edit etc.)

Assignment No. 1

Title: Design and Develop SQL DDL statements which demonstrate the use of SQL objects such as Table, View , Index, Sequence, Synonym

Objectives: To study SQL DDL statements

Theory: SQL – Structured Query Language

Data Definition in SQL

Creating Tables

Syntax:-

Create table<table name>

(colume_name 1 datatype size(),

colume_name 2 datatype size(),

….

colume_name n datatype size());

e.g. Create table student with the following fields(name,roll,class,branch)

Create table student
(name char(20),

Roll number(5),

Class char(10),

Branch char(15));

A table from a table

- **Syntax :**

  CREATE TABLE <TableName> (<ColumnName>, <Columnname>) AS SELECT <ColumnName>, <Columnname> FROM <TableName>;

  - If the source table contains the records, then new table is also created with the same records present in the source table.

- If you want only structure without records then select statement must have condition. Syntax:

  CREATE TABLE <TableName> (<ColumnName>, <Columnname>) AS SELECT <ColumnName>, <Columnname> FROM <TableName> WHERE 1=2; (Or)

  CREATE TABLE <TableName> (<ColumnName>, <Columnname>) AS SELECT <ColumnName>, <Columnname> FROM <TableName> WHERE ColumnName =NULL;

Constraints

The definition of a table may include the specification of integrity constraints. Basically two types of constraints are provided: column constraints are associated with a single column whereas table constraints are typically associated with more than one column. A constraint can be named. It is advisable to name a constraint in order to get more meaningful information when this constraint is violated due to, e.g., an insertion of a tuple that violates the constraint. If no name is specified for the constraint, Oracle automatically generates a name of the pattern SYS C<number>.Rules are enforced on data being stored in a table, are called **Constraints**.

Both the Create table & Alter Table SQL can be used to write SQL sentences that attach constraints.

Basically constraints are of three types

1) Domain
   - Not Null
   - Check
2) Entity
   - Primary Key
   - Unique
3) Referential
   - Foreign key
4) Not Null:-Not null constraint can be applied at column level only.

We can define these constraints

1) at the time of table creation Syntax :

   CREATE TABLE <tableName> (<ColumnName> datatype(size) NOT NULL,

   <ColumnName> datatype(size),….

   );

2) After the table creation

   ALTER TABLE <tableName> Modify(<ColumnName> datatype(size) NOT NULL);

   Check constraints

   - Can be bound to column or a table using CREATE TABLE or ALTER TABLE command.

- Checks are performed when write operation is performed .
- Insert or update statement causes the relevant check constraint.
- Ensures the integrity of the data in tables.

Syntax :

- Check constraints at column level

Syntax :

CREATE TABLE <tableName>

(<ColumnName>datatype(size)CHECK(columnName

condition),<columnname datatype(size));

CREATE TABLE <tableName>

(<ColumnName> datatype(size) CONSTRAINT <constraint_name>

CHECK (columnName condition),..

);

- Check constraints at table level

Syntax :

CREATE TABLE <tableName>

(<ColumnName> datatype(size),
<ColumnName> datatype(size),
CONSTRAINT <constraint_name> CHECK (columnName condition),..);

- Check constraints at table level

Syntax :

CREATE TABLE

<tableName>

(<ColumnName>

datatype(size),

<ColumnName> datatype(size),….,

CHECK (columnName condition));

After table creation

Alter table tablename

Add constraints constraintname ckeck(condition)

The PRIMARY KEY Constraint

A primary key is one or more column(s) in a table used to uniquely identify each row in the table.

- A table can have only one primary key.
- Can not be left blank

- Data must be UNIQUE.
- Not allows null values
- Not allows duplicate values.
- Unique index is created automatically if there is a primary key.

Primary key constraint defined at column level

Syntax:

CREATE TABLE <TableName>

(<ColumnName1> <DataType>(<Size>)PRIMARY KEY,<columnname2

<datatype(<size>),…..);

- Primary key constraint defined at Table level

Syntax:

CREATE TABLE <TableName>

(<ColumnName1> <DataType>(<Size>) ,…, PRIMARY

KEY(<ColumnName1> <ColumnName2>));

- key constraint defined at Table level

Syntax:

CREATE TABLE <TableName>

(<ColumnName1> <DataType>(<Size>) <columnname2

datatype<(size)<,<columnname3 datatype<size>constraint constraintname

PRIMARY KEY(<ColumnName1>));

After table creation

Alter table tablename

Add(constraint constraintname primary key(columnname));

The Unique Key Constraint

- The unique column constraint permits multiple entries of NULL into the column.
- Unique key not allowed duplicate values
- Unique index is automatically created.
- Table can have more than one unique key.

- UNIQUE constraint defined at column level

  Syntax :

  Create table tablename(**<columnname> <datatype>(<Size>**

  UNIQUE),<columnname> datatype(<size>) .....);

UNIQUE constraint defined at table level
Syntax :

CREATE TABLE tablename (<columnname> <datatype>(<Size>), <columnname> <datatype>(<Size>), UNIQUE(<columnname>, <columnname>));

After table creation

Alter table tablename

Add constraint constraintname unique(columnname);

- The Foreign Key (Self Reference) Constraint Foreign key represents relationships between tables.

A foreign key is a column(or group of columns) whose values are derived from primary key or unique key of some other table.

Foreign key constraint defined at column level

Syntax:

<columnName> <DataType> (<size>) REFERENCES <TableName>[(<ColumnName>)] [ON DELETE CASCADE]

- If the ON DELETE CASCADE option is set, a DELETE operation in the master table will trigger a DELETE operation for corresponding records in all detail tables.
- If the ON DELETE SET NULL option is set, a DELETE operation in the master table will set the value held by the foreign key of the detail tables to null.

Foreign key :

ALTER TABLE <child_tablename> ADD CONSTRAINT <constraint_name> FOREIGN KEY (<columnname in child_table>) REFERENCES <parent table name>;

1) FOREIGN KEY constraint at table level
2) FOREIGN KEY constraint defined with ON DELETE CASCADE
   FOREIGN KEY(<ColumnName>[,<columnname>]) REFERENCES <TableName> [(<ColumnName>, <ColumnName>) ON DELETE CASCADE

- FOREIGN KEY constraint defined with ON DELETE SET NULL

FOREIGN KEY(<ColumnName>[,<columnname>]) REFERENCES <TableName> [(<ColumnName>, <ColumnName>) ON DELETE SET NULL

- To view the constraint Syntax:

  Select constraint_name, constraint_type, search_condition from user_constraints where table_name=<tablename>;
  Select constraint_name, column_name from user_cons_columns where table_name=<tablename>;

To drop the constraints

Syntax:-

Drop constraint constraintname;

Describe commands

To view the structure of the table created use the **DESCRIBE** command.The command displays the column names and datatypes

Syntax:-

Desc[ribe]<table_name>

e.g desc student

Restrictions for creating a table:

1.Table names and column names must begin with a letter.
2.Table names and column names can be 1 to 30 characters long.
3.table names must contain only the characters A-Z,a-z,0-9,underscore_,$ and #
4.Table name should not be same as the name of another database object.
5.Table name must not be an ORACLE reserved word.
6.Column names should not be duplicate within a table definition.

Alteration of TABLE:-

Alter table command

Syntax:-

Case1:-

Alter table <table_name>

Add(colume_name 1 datatype size(),
colume_name 2 datatype size(),

colume_name n datatype size());

Case2:-

Alter table <table_name>
Modify(colume_name 1 datatype size(),
colume_name 2 datatype size(),
…..
colume_name n datatype size());

After you create a table, you may need to change the table structures because you need to have a column definition needs to be changed. Alter table statement can be used for this purpose. You can add columns to a table using the alter table statement with the ADD clause.

E.g. Suppose you want to add enroll_no in the student table then we write

Alter table student Add(enroll_no number(10));

You can modify existing column in a table by using the alter table statement with modify clause.

E.g. Suppose you want to modify or chang the size of previously defined field name in the student table then we write

Alter table student modify(name char(25));

Dropping a column from a table

Syntax :

ALTER TABLE <Tablename> DROP COLUMN <ColumnName> ;

Drop table command Syntax:-

Drop table <table_name>

Drop table command remnoves the definitions of an oracle table.When you drop a table ,the database loses all the data in the table and all the indexes associated with it.

e.g drop table student;

Truncate table command

Syntax:-

Trunc table<table_name>

The truncate table statement is used to remove all rows from a table and to release the storage space used by the table.

e.g.Trunc table student;

Rename table command

Syntax:-

Rename<oldtable_name> to<newtable_name>

Rename statement is used to rename a table,view,sequence,or synonym. e.g.

Rename student to stud;

Database objects:-

Index

An index is a schema object that can speed up retrieval of rows by using pointer. An index provides direct & fast access to rows in a table. Index can be created explicitly or automatically.

Automatically :- A unique index is created automatically when you define a primary key or unique key constraint in a table definition.

Manually :- users can create non unique indexes or columns to speed up access time to the rows.

Syntax:

Create index<index_name> On table(column[, column]…);

Eg. Create index emp_ename_idx On emp(ename);

When to create an index

a) The column is used frequently in the WHERE clause or in a join condition.

b) The column contains a wide range of values.

c) The column contains a large number of values.

To display created index of a table

eg.

```
Select ic.index_name, ic.column_name, ic.colun_position col_pos, ix.uniqueness from
user_indexes ix, user_ind_columns ic where ic.index_name=ix.index_name
and ic.table_name="emp";
```

Removing an Index

Syntax:-

Drop index <index_name>;

eg. Dropindex emp_name_idx;

Note: 1) we cannot modify indexes. 2) To change an index, we must drop it and the re-create it.

Views

View is a logical representation of subsets of data from one or more tables. A view takes the output of a query and treats it as a table therefore view can be called as stored query or a virtual table.The tables upon which a view is based are called base tables. In Oracle the SQL command to create a view (virtual table) has the form

Create [or replace] view <view-name> [(<column(s)>)] as

<select-statement> [with check option [constraint <name>]];

The optional clause or replace re-creates the view if it already exists. <column(s)> names the columns of the view. If <column(s)> is not specified in the view definition, the columns of the view get the same names as the attributes listed in the select statement (if possible).

Example: The following view contains the name, job title and the annual salary of employees working in the department 20:

Create view DEPT20 as
select ENAME, JOB, SAL 12 ANNUAL SALARY from EMP where DEPTNO = 20;

In the select statement the column alias ANNUAL SALARY is specified for the expression SAL*12 and this alias is taken by the view. An alternative formulation of the above view definition is

Create view DEPT20 (ENAME, JOB, ANNUAL SALARY) as select ENAME, JOB, SAL 12 from EMP where DEPTNO = 20;

A view can be used in the same way as a table, that is, rows can be retrieved from a view(also respective rows are not physically stored, but derived on basis of the select statement inthe view definition), or rows can even be modified. A view is evaluated again each time it is accessed. In Oracle SQL no insert, update, or delete modifications on views are allowed
that use one of the following constructs in the view definition:

- Joins
- Aggregate function such as sum, min, max etc.
- set-valued subqueries (in, any, all) or test for existence (exists)
- group by clause or distinct clause

In combination with the clause with check option any update or insertion of a row into the view is rejected if the new/modified row does not meet the view definition, i.e., these rows would not be selected based on the select statement. A with check option can be named using the constraint clause.

A view can be deleted using the command delete <view-name>. To
describe the structure of a view

e.g. **Describe stud;**

To display the contents of view e.g. Select * from stud

Removing a view:

Syntax:- **Drop view <view_name>**
e.g. Drop view stud

Sequence:

A sequence is a database object, which can generate unique, sequential integer values. It can be used to automatically generate primary key or unique key values. A sequence can be either in an ascending or descending order.

Syntax :

```
Create
sequence<sequence_name>
[increment by n]
[start with n]
[{maxvalue n |
nomaxvalue}] [{minvalue n|
nominvalue}] [{cycle |
nocycle}]
[{cache n| nocache}];
```

Increment by n	Specifies the interval betweensequence number where n is an integer. If this clause is omitted, the sequence is increment by 1.
Start with n	Specifies the first sequence number tobe generated. If this clause is omitted ,the sequence is start with 1.
Maxvalue n	Specifies the maximum value, the sequence can generate
Nomax value n	Specifies the maximum value of 10e27-1 for an ascending sequence & -1 for descending sequence. This is a default option.
Minvalue n	Specifies the minimum sequence value.
Nominvalue n	Specifies the minimum value of 1 for an ascending & 10e26-1 for descending sequence. This is a default option.
Cycle	Specifies that the sequence continues to generate values from the beginning

	after reaching either its max or minvalue.
Nocycle	Specifies that the sequence can not generate more values after reaching either its max or min value. This is a default option.
Cache / nocache	Specifies how many values the oracleserver will preallocate & keep in memory. By default, the oracle server will cache 20 values.

After creating a sequence we can access its values with the help of pseudo columns like **curval** & **nextval**.

Nextval : nextval returns initial value of the sequence when reference to for the first time. Last references to the nextval will increment the sequence using the increment by clause & returns the new value.

Curval : curval returns the current value of the sequence which is the value returned by the last reference to last value.

Modifyning a sequence:

The sequence can be modified when we want to perform the following :

- Set or eliminate minvalue or maxvalue
- Change the increment value.
- Change the number of cache sequence number.

Syntax :

```
Alter  sequence <sequence_name>
[increment by n]
[start with n]
[{maxvalue n | nomaxvalue}]
[{minvalue n| nominvalue}]
[{cycle | nocycle}]
[{cache n| nocache}];
```

Synonym:

A synonym is a database object,which is used as an alias(alternative name)for a table,view or sequence.

Syntax:-

```
Create[public]synonym
<synonym_name>for<table_name>;
```

In the syntax

Public:-Creates a synonym accessible to all users.
Synonym:-Is the name of the synonym to be created.

Synonym can either be private or public.A private synonym is created by normal user,which is available to that persons.

A public synonym is created by a database administrator(DBA),which can be availed by any other database user.

Uses:-

1. Simplify SQL statements.

2. Hide the name and owner of an object.

3. Provide public access to an object.

Guidelines:-

1. User can do all DML manipulations such as insert ,delete,update on synonym.

2. User cannot perform any DDL operations on the synonym except dropping the synonym.

3. All the manipulations on it actually affect the table e.g
Create synonym stud1 for student;

Conclusion: Thus we have studied how to use SQL DDL Statements.

Assignment No. 2

Title:- Design at least 10 SQL queries for suitable database application using SQL DML statements: Insert, Select, Update, Delete with operators, functions, and set operator.

Objectives:- To study SQL DML statements

THEORY: Data Manipulation Language (DML)

A **data manipulation language** (**DML**) is a family of syntax elements similar to a computer programming language used for selecting, inserting, deleting and updating data in a database. Performing read-only queries of data is sometimes also considered a component of DML.

Data manipulation language comprises the SQL data change statements, [2] which modify stored data but not the schema or database objects.

Data manipulation languages have their functional capability organized by the initial word in a statement, which is almost always a verb. In the case of SQL, these verbs are:

- SELECT ... FROM ... WHERE ...
- INSERT INTO ... VALUES ...
- UPDATE ... SET ... WHERE ...
- DELETE FROM ... WHERE ...

The purely read-only SELECT query statement is classed with the 'SQL-data' statements and so is considered by the standard to be outside of DML. The SELECT ... INTO form is considered to be DML because it manipulates (i.e. modifies) data. In common practice though, this distinction is not made and SELECT is widely considered to be part of DML.

Most SQL database implementations extend their SQL capabilities by providing imperative, i.e. procedural languages.

Inserting Data into Table:

To insert data into table, you would need to use SQL **INSERT INTO** command. You can insert data into table by using > prompt or by using any script like PHP.

Syntax:

Here is generic SQL syntax of INSERT INTO command to insert data into table:

```
INSERT INTO table_name ( field1, field2,...fieldN )
                       VALUES  ( value1, value2,...valueN );
```

To insert string data types, it is required to keep all the values into double or single quote, for example:-**"value"**.

Inserting Data from Command Prompt:

This will use SQL INSERT INTO command to insert data into table tutorials_tbl.

Example:

Following example will create 3 records into **tutorials_tbl** table:

```
INSERT INTO tutorials_tbl (tutorial_title, tutorial_author, submission_date)
  VALUES ("Learn PHP", "John Poul", NOW());

INSERT INTO tutorials_tbl (tutorial_title, tutorial_author, submission_date)
  VALUES ("Learn  ", "Abdul S", NOW());
```

```
INSERT INTO tutorials_tbl (tutorial_title, tutorial_author, submission_date)
  VALUES ("JAVA Tutorial", "Sanjay", '2007-05-06');
```

Here, **NOW()** is a function, which returns current date and time.

☐ **Fetching Data from Table:**

The SQL **SELECT** command is used to fetch data from database. You can use this command at > prompt as well as in any script like PHP.

Syntax:

Here is generic SQL syntax of SELECT command to fetch data from table:

```
SELECT field1, field2,...fieldN table_name1, table_name2...
[WHERE Clause]
[OFFSET M ][LIMIT N]
```

- You can use one or more tables separated by comma to include various conditions using a WHERE clause, but WHERE clause is an optional part of SELECT command.
- You can fetch one or more fields in a single SELECT command.
- You can specify star (*) in place of fields. In this case, SELECT will return all the fields.
- You can specify any condition using WHERE clause.
- You can specify an offset using **OFFSET** from where SELECT will start returning records. By default offset is zero.
- You can limit the number of returns using **LIMIT** attribute.

Fetching Data from Command Prompt:

This will use SQL SELECT command to fetch data from table tutorials_tbl

Example:

Following example will return all the records from **tutorials_tbl** table:

```
> SELECT * from tutorials_tbl
+-------------+----------------+-----------------+-----------------+
| tutorial_id | tutorial_title | tutorial_author | submission_date |
+-------------+----------------+-----------------+-----------------+
|           1 | Learn PHP      | John Poul       | 2007-05-21      |
|           2 | Learn          | Abdul S         | 2007-05-21      |
|           3 | JAVA Tutorial  | Sanjay          | 2007-05-21      |
+-------------+----------------+-----------------+-----------------+
```

The SQL **SELECT** statement returns a result set of records from one or more tables. A **SELECT** statement retrieves zero or more rows from one or more database tables or database views. In most applications, SELECT is the most commonly used Data Manipulation Language (DML) command. As SQL is a declarative programming language, SELECT queries specify a result set, but do not specify how to calculate it. The database translates the query into a "query plan" which may vary between executions, database versions and database software. This functionality is called the "query optimizer" as it is responsible for finding the best possible execution plan for the query, within applicable constraints.

The SELECT statement has many optional clauses:

- WHERE specifies which rows to retrieve.

- GROUP BY groups rows sharing a property so that an aggregate function can be applied to each group.
- HAVING selects among the groups defined by the GROUP BY clause.
- ORDER BY specifies an order in which to return the rows.
- AS provides an alias which can be used to temporarily rename tables or columns.

WHERE Clause

We have seen SQL **SELECT** command to fetch data from table. We can use a conditional clause called **WHERE** clause to filter out results. Using WHERE clause, we can specify a selection criteria to select required records from a table.

Syntax:

Here is generic SQL syntax of SELECT command with WHERE clause to fetch data from table:

```
SELECT field1, field2,...fieldN table_name1, table_name2...
[WHERE condition1 [AND [OR]] condition2.....
```

- You can use one or more tables separated by comma to include various conditions using a WHERE clause, but WHERE clause is an optional part of SELECT command.
- You can specify any condition using WHERE clause.
- You can specify more than one conditions using **AND** or **OR** operators.
- A WHERE clause can be used along with DELETE or UPDATE SQL command also to specify a condition.

The **WHERE** clause works like an if condition in any programming language. This clause is used to compare given value with the field value available in table. If given value from outside is equal to the available field value in table, then it returns that row.

Here is the list of operators, which can be used with **WHERE** clause.

Assume field A holds 10 and field B holds 20, then:

Operator	Description	Example
=	Checks if the values of two operands are equal or not, if yes then condition becomes true.	(A = B) is not true.
!=	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.	(A != B) is true.
>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(A > B) is not true.
<	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(A < B) is true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(A >= B) is not true.
<=	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(A <= B) is true.

The WHERE clause is very useful when you want to fetch selected rows from a table, especially when you use **Join**.
It is a common practice to search records using **Primary Key** to make search fast.
If given condition does not match any record in the table, then query would not return any row.

Fetching Data from Command Prompt:
This will use SQL SELECT command with WHERE clause to fetch selected data from table tutorials_tbl.
Example:
Following example will return all the records from **tutorials_tbl** table for which author name is **Sanjay**:

```
> SELECT * from tutorials_tbl WHERE tutorial_author='Sanjay';
+-------------+----------------+-----------------+-----------------+
| tutorial_id | tutorial_title | tutorial_author | submission_date |
+-------------+----------------+-----------------+-----------------+
|           3 | JAVA Tutorial  | Sanjay          | 2007-05-21      |
+-------------+----------------+-----------------+-----------------+
```

Unless performing a **LIKE** comparison on a string, the comparison is not case sensitive. You can make your search case sensitive using **BINARY** keyword as follows:

```
SELECT * from tutorials_tbl  WHERE BINARY tutorial_author='sanjay';
```

LIKE Clause

We have seen SQL **SELECT** command to fetch data from table. We can also use a conditional clause called **WHERE** clause to select required records.
A WHERE clause with equals sign (=) works fine where we want to do an exact match. Like if "tutorial_author = 'Sanjay'". But there may be a requirement where we want to filter out all the results where tutorial_author name should contain "jay". This can be handled using SQL **LIKE** clause along with WHERE clause.
If SQL LIKE clause is used along with % characters, then it will work like a meta character (*) in UNIX while listing out all the files or directories at command prompt.
Without a % character, LIKE clause is very similar to equals sign along with WHERE clause.
Syntax:
Here is generic SQL syntax of SELECT command along with LIKE clause to fetch data from table:

```
SELECT field1, field2,...fieldN table_name1, table_name2...
WHERE field1 LIKE condition1 [AND [OR]] filed2 = 'somevalue'
```

- You can specify any condition using WHERE clause.
- You can use LIKE clause along with WHERE clause.
- You can use LIKE clause in place of equals sign.
- When LIKE is used along with % sign then it will work like a meta character search.
- You can specify more than one conditions using **AND** or **OR** operators.
- A WHERE...LIKE clause can be used along with DELETE or UPDATE SQL command also to specify a condition.

Using LIKE clause at Command Prompt:
This will use SQL SELECT command with WHERE...LIKE clause to fetch selected data from table tutorials_tbl.
Example:
Following example will return all the records from **tutorials_tbl** table for which author name ends with **jay**:

```
SELECT * from tutorials_tbl  WHERE tutorial_author LIKE '%jay';
+-------------+----------------+-----------------+-----------------+
| tutorial_id | tutorial_title | tutorial_author | submission_date |
+-------------+----------------+-----------------+-----------------+
|           3 | JAVA Tutorial  | Sanjay          | 2007-05-21      |
+-------------+----------------+-----------------+-----------------+
```

GROUP BY Clause

You can use **GROUP BY** to group values from a column, and, if you wish, perform calculations on that column. You can use COUNT, SUM, AVG, etc., functions on the grouped column.
To understand **GROUP BY** clause, consider an **employee_tbl** table, which is having the following records:

```
> SELECT * FROM employee_tbl;
+----+------+------------+--------------------+
| id | name | work_date  | daily_typing_pages |
+----+------+------------+--------------------+
|  1 | John | 2007-01-24 |                250 |
|  2 | Ram  | 2007-05-27 |                220 |
|  3 | Jack | 2007-05-06 |                170 |
|  3 | Jack | 2007-04-06 |                100 |
|  4 | Jill | 2007-04-06 |                220 |
|  5 | Zara | 2007-06-06 |                300 |
|  5 | Zara | 2007-02-06 |                350 |
+----+------+------------+--------------------+
7 rows in set (0.00 sec)
```

Now, suppose based on the above table we want to count number of days each employee did work.
If we will write a SQL query as follows, then we will get the following result:

```
SELECT COUNT(*) FROM employee_tbl;
+----------+
| COUNT(*) |
+----------+
| 7        |
```

But this is not serving our purpose, we want to display total number of pages typed by each person separately. This is done by using aggregate functions in conjunction with a **GROUP BY** clause as follows:

```
SELECT name, COUNT(*)  FROM   employee_tbl  GROUP BY name;

+------+----------+
| name | COUNT(*) |
+------+----------+
| Jack |        2 |
| Jill |        1 |
| John |        1 |
| Ram  |        1 |
| Zara |        2 |
+------+----------+
5 rows in set (0.04 sec)
```

We will see more functionality related to GROUP BY in other functions like SUM, AVG, etc.

COUNT Function

COUNT Function is the simplest function and very useful in counting the number of records, which are expected to be returned by a SELECT statement.
To understand **COUNT** function, consider an **employee_tbl** table, which is having the following records:

```
> SELECT * FROM employee_tbl;
+------+------+------------+--------------------+
| id   | name | work_date  | daily_typing_pages |
+------+------+------------+--------------------+
|    1 | John | 2007-01-24 |                250 |
|    2 | Ram  | 2007-05-27 |                220 |
|    3 | Jack | 2007-05-06 |                170 |
|    3 | Jack | 2007-04-06 |                100 |
|    4 | Jill | 2007-04-06 |                220 |
|    5 | Zara | 2007-06-06 |                300 |
|    5 | Zara | 2007-02-06 |                350 |
+------+------+------------+--------------------+
7 rows in set (0.00 sec)
```

Now, suppose based on the above table you want to count total number of rows in this table, then you can do it as follows:

```
>SELECT COUNT(*) FROM employee_tbl ;
+----------+
| COUNT(*) |
+----------+
|        7 |
+----------+
1 row in set (0.01 sec)
```

Similarly, if you want to count the number of records for Zara, then it can be done as follows:

```
SELECT COUNT(*) FROM employee_tbl  WHERE name="Zara";
+----------+
```

```
| COUNT(*) |
+----------+
|        2 |
+----------+
1 row in set (0.04 sec)
```

NOTE: All the SQL queries are case insensitive so it does not make any difference if you give ZARA or Zara in WHERE condition.

MAX Function

MAX function is used to find out the record with maximum value among a record set.
To understand **MAX** function, consider an **employee_tbl** table, which is having the following records:

```
 > SELECT * FROM employee_tbl;
+-----+------+------------+--------------------+
| id  | name | work_date  | daily_typing_pages |
+-----+------+------------+--------------------+
|   1 | John | 2007-01-24 |                250 |
|   2 | Ram  | 2007-05-27 |                220 |
|   3 | Jack | 2007-05-06 |                170 |
|   3 | Jack | 2007-04-06 |                100 |
|   4 | Jill | 2007-04-06 |                220 |
|   5 | Zara | 2007-06-06 |                300 |
|   5 | Zara | 2007-02-06 |                350 |
+-----+------+------------+--------------------+
7 rows in set (0.00 sec)
```

Now, suppose based on the above table you want to fetch maximum value of daily_typing_pages, then you can do so simply using the following command:

```
SELECT MAX(daily_typing_pages) FROM employee_tbl;
+--------------------------+
| MAX(daily_typing_pages) |
+--------------------------+
|                      350 |
+--------------------------+
1 row in set (0.00 sec)
```

You can find all the records with maximum value for each name using **GROUP BY** clause as follows:

```
 SELECT id, name, MAX(daily_typing_pages)  FROM employee_tbl GROUP BY name;
+-----+------+--------------------------+
| id  | name | MAX(daily_typing_pages) |
+-----+------+--------------------------+
|   3 | Jack |                      170 |
|   4 | Jill |                      220 |
```

```
|    1 | John |                250 |
|    2 | Ram  |                 220 |
|    5 | Zara |                350 |
+------+------+--------------------------+
5 rows in set (0.00 sec)
```

You can use **MIN** Function along with **MAX** function to find out minimum value as well. Try out the following example:

```
SELECT MIN(daily_typing_pages) least, MAX(daily_typing_pages) max FROM employee_tbl;
+--------+-------+
| least | max  |
+--------+-------+
|  100 |  350 |
+--------+-------+
1 row in set (0.01 sec)
```

MIN Function

MIN function is used to find out the record with minimum value among a record set.
To understand **MIN** function, consider an **employee_tbl** table, which is having the following records:

```
 SELECT * FROM employee_tbl;
+------+------+-------------+--------------------+
| id  | name | work_date  | daily_typing_pages |
+------+------+-------------+--------------------+
|    1 | John | 2007-01-24 |              250 |
|    2 | Ram  | 2007-05-27 |               220 |
|    3 | Jack | 2007-05-06 |              170 |
|    3 | Jack | 2007-04-06 |              100 |
|    4 | Jill | 2007-04-06 |             220 |
|    5 | Zara | 2007-06-06 |              300 |
|    5 | Zara | 2007-02-06 |              350 |
+------+------+-------------+--------------------+
7 rows in set (0.00 sec)
```

Now, suppose based on the above table you want to fetch minimum value of daily_typing_pages, then you can do so simply using the following command:

```
SELECT MIN(daily_typing_pages) FROM employee_tbl;
+--------------------------+
| MIN(daily_typing_pages) |
+--------------------------+
|                  100 |
+--------------------------+
1 row in set (0.00 sec)
```

You can find all the records with minimum value for each name using **GROUP BY** clause as follows:

```
SELECT id, name, MIN(daily_typing_pages) FROM employee_tbl GROUP BY name;
+----+------+--------------------------+
| id | name | MIN(daily_typing_pages) |
+----+------+--------------------------+
|  3 | Jack |                      100 |
|  4 | Jill |                      220 |
|  1 | John |                      250 |
|  2 | Ram  |                      220 |
|  5 | Zara |                      300 |
+----+------+--------------------------+
5 rows in set (0.00 sec)
```

You can use **MIN** Function along with **MAX** function to find out minimum value as well. Try out the following example:

```
SELECT MIN(daily_typing_pages) least, MAX(daily_typing_pages) max FROM employee_tbl;
+-------+-------+
| least | max   |
+-------+-------+
|   100 |   350 |
+-------+-------+
1 row in set (0.01 sec)
```

AVG Function

AVG function is used to find out the average of a field in various records.

To understand **AVG** function, consider an **employee_tbl** table, which is having following records:

```
 SELECT * FROM employee_tbl;
+----+------+------------+--------------------+
| id | name | work_date  | daily_typing_pages |
+----+------+------------+--------------------+
|  1 | John | 2007-01-24 |                250 |
|  2 | Ram  | 2007-05-27 |                220 |
|  3 | Jack | 2007-05-06 |                170 |
|  3 | Jack | 2007-04-06 |                100 |
|  4 | Jill | 2007-04-06 |                220 |
|  5 | Zara | 2007-06-06 |                300 |
|  5 | Zara | 2007-02-06 |                350 |
+----+------+------------+--------------------+
7 rows in set (0.00 sec)
```

Now, suppose based on the above table you want to calculate average of all the dialy_typing_pages, then you can do so by using the following command:

```
 SELECT AVG(daily_typing_pages)  FROM employee_tbl;
+-------------------------+
| AVG(daily_typing_pages) |
+-------------------------+
|                230.0000 |
+-------------------------+
1 row in set (0.03 sec)
```

You can take average of various records set using **GROUP BY** clause. Following example will take average all the records related to a single person and you will have average typed pages by every person.

```
 SELECT name, AVG(daily_typing_pages) FROM employee_tbl GROUP BY name;
+-------+-------------------------+
| name | AVG(daily_typing_pages) |
+-------+-------------------------+
| Jack |               135.0000 |
| Jill |               220.0000 |
| John |               250.0000 |
| Ram  |               220.0000 |
| Zara |               325.0000 |
+-------+-------------------------+
5 rows in set (0.20 sec)
```

SUM Function

SUM function is used to find out the sum of a field in various records.
To understand **SUM** function, consider an **employee_tbl** table, which is having the following records:

```
SELECT * FROM employee_tbl;
+------+-------+------------+--------------------+
| id  | name | work_date  | daily_typing_pages |
+------+-------+------------+--------------------+
|    1 | John | 2007-01-24 |                250 |
|    2 | Ram  | 2007-05-27 |                220 |
|    3 | Jack | 2007-05-06 |                170 |
|    3 | Jack | 2007-04-06 |                100 |
|    4 | Jill | 2007-04-06 |                220 |
|    5 | Zara | 2007-06-06 |                300 |
|    5 | Zara | 2007-02-06 |                350 |
+------+-------+------------+--------------------+
7 rows in set (0.00 sec)
```

Now, suppose based on the above table you want to calculate total of all the dialy_typing_pages, then you can do so by using the following command:

```
SELECT SUM(daily_typing_pages)  FROM employee_tbl;
+-------------------------+
| SUM(daily_typing_pages) |
```

```
+-------------------------+
|                    1610 |
+-------------------------+
1 row in set (0.00 sec)
```

You can take sum of various records set using **GROUP BY** clause. Following example will sum up all the records related to a single person and you will have total typed pages by every person.

```
SELECT name, SUM(daily_typing_pages) FROM employee_tbl GROUP BY name;
+------+--------------------------+
| name | SUM(daily_typing_pages) |
+------+--------------------------+
| Jack |                      270 |
| Jill |                      220 |
| John |                      250 |
| Ram  |                      220 |
| Zara |                      650 |
+------+--------------------------+
5 rows in set (0.17 sec)
```

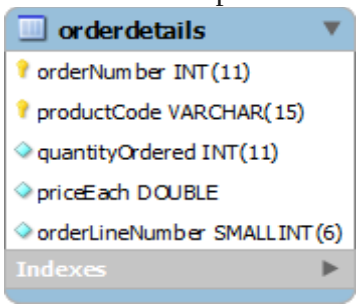
HAVING clause

The HAVING clause is used in the SELECT statement to specify filter conditions for group of rows or aggregates. The HAVING clause is often used with the GROUP BY clause. When using with the GROUP BY clause, you can apply a filter condition to the columns that appear in the GROUP BY clause. If the GROUP BY clause is omitted, the HAVING clause behaves like the WHERE clause. Notice that the HAVING clause applies the condition to each group of rows, while the WHERE clause applies the condition to each individual row.

Examples of using HAVING clause

Let"s take a look at an example of using HAVING clause.

We will use the orderdetails table in the sample database for the sake of demonstration.



We can use the GROUP BY clause to get order number, the number of items sold per order and total sales for each:

```
SELECT ordernumber,
    SUM(quantityOrdered) AS itemsCount,
    SUM(priceeach) AS total
FROM orderdetails
GROUP BY ordernumber
```

ordernumber	itemsCount	total
10100	151	301.84000000000003
10101	142	352
10102	80	138.68
10103	541	1520.3699999999997
10104	443	1251.8899999999999
10105	545	1479.71

Now, we can find which order has total sales greater than $1000. We use the HAVING clause on the aggregate as follows:

```
SELECT ordernumber,
      SUM(quantityOrdered) AS itemsCount,
      SUM(priceeach) AS total
       FROM orderdetails
      GROUP BY ordernumber
      HAVING total > 1000
```

ordernumber	itemsCount	total
10103	541	1520.3699999999997
10104	443	1251.8899999999999
10105	545	1479.71
10106	675	1427.2800000000002
10108	561	1432.86
10110	570	1338.4699999999998

We can construct a complex condition in the HAVING clause using logical operators such as OR and AND. Suppose we want to find which order has total sales greater than $1000 and contains more than 600 items, we can use the following query:

```
SELECT ordernumber,
      sum(quantityOrdered) AS itemsCount,
      sum(priceeach) AS total
       FROM orderdetails
       GROUP BY ordernumber
       HAVING total > 1000 AND itemsCount > 600
```

ordernumber	itemsCount	total
10106	675	1427.2800000000002
10126	617	1623.71
10135	607	1494.86
10165	670	1794.9399999999996
10168	642	1472.5
10204	619	1619.73
10207	615	1560.08

The HAVING clause is only useful when we use it with the GROUP BY clause to generate the output of the high-level reports. For example, we can use the HAVING clause to answer some kinds of queries like give me all the orders in this month, this quarter and this year that have total sales greater than 10K.

UPDATE Query

There may be a requirement where existing data in a table needs to be modified. You can do so by using SQL **UPDATE** command. This will modify any field value of any table.

Syntax:

Here is generic SQL syntax of UPDATE command to modify data into table:

```
UPDATE table_name SET field1=new-value1, field2=new-value2
[WHERE Clause]
```

- You can update one or more field altogether.
- You can specify any condition using WHERE clause.
- You can update values in a single table at a time.

The WHERE clause is very useful when you want to update selected rows in a table.

Updating Data from Command Prompt:

This will use SQL UPDATE command with WHERE clause to update selected data into table tutorials_tbl.

Example:

Following example will update **tutorial_title** field for a record having tutorial_id as 3.

```
UPDATE tutorials_tbl
  SET tutorial_title='Learning JAVA'
    WHERE tutorial_id=3;
```

DELETE Query

If you want to delete a record from any table, then you can use SQL command **DELETE FROM**. You can use this command at > prompt as well as in any script like PHP.

Syntax:

Here is generic SQL syntax of DELETE command to delete data from a table:

```
DELETE FROM table_name [WHERE Clause]
```

- If WHERE clause is not specified, then all the records will be deleted from the given table.
- You can specify any condition using WHERE clause.
- You can delete records in a single table at a time.

The WHERE clause is very useful when you want to delete selected rows in a table.

Deleting Data from Command Prompt:

This will use SQL DELETE command with WHERE clause to delete selected data into table tutorials_tbl.

Example:

Following example will delete a record into tutorial_tbl whose tutorial_id is 3.

```
DELETE FROM tutorials_tbl WHERE tutorial_id=3;
```

Create table **location**(location_id numeric(3) primary key,regional_group varchar(15));

Create table **department**(Department_ID numeric(2) primary key,name varchar(20),location_id int, foreign key(location_id) references location(location_id));

Create table **job**(job_ID numeric(3) primary key,function varchar(20));

Create table **employee**(employee_ID numeric(4) primary key,last_name varchar(20),first_name varchar(20),middle_name varchar(20),job_id numeric(3),manager_id varchar(20), hired_date date, salary numeric(6), comm numeric(4), department_id numeric(2) not null,FOREIGN KEY (job_id) REFERENCES job(job_id),FOREIGN KEY (department_id) REFERENCES department(department_id));

1. List the details about "SMITH"

   *Select * from employee where last_name='SMITH';*

2. List out the employees who are working in department 20

   *Select * from employee where department_id=20*

3. List out the employees who are earning salary between 3000 and 4500

   *Select * from employee where salary between 3000 and 4500*

4. List out the employees who are working in department 10 or 20

   *Select * from employee where department_id in (10,20)*

5. Find out the employees who are not working in department 10 or 30

   Select last_name, salary, comm, department_id from employee where department_id not in (10,30)

6. List out the employees whose name starts with "S"

   *Select * from employee where last_name like 'S%';*

7. List out the employees whose name start with "S" and end with "H"

   *Select * from employee where last_name Like 'S%H';*

8. List out the employees whose name length is 5 and start with "S"

   *Select * from employee where last_name like 'S____';*

9. List out the employees who are working in department 10 and draw the salaries more than 3500

   *Select * from employee where department_id=10 and salary>3500*

10. List out the employees who are not receiving commission.

    *Select * from employee where commission is Null*

11. List out the employee id, last name in ascending order based on the employee id.

    Select employee_id, last_name from employee order by employee_id

12. List out the employee id, name in descending order based on salary column

    Select employee_id, last_name, salary from employee order by salary desc

13. List out the employee details according to their last_name in ascending order and salaries in descending order

Conclusion: Thus we have studied to use & implement various DML queries.

Assignment No: 3

Title:- Design at least 10 SQL queries for suitable database application using SQL DML statements: all types of Join, Sub-Query and View.

Objectives:- To study all types of Join, Sub-Query and View SQL statements.

THEORY: SQL – Join

The ability of relational „join" operator is an important feature of relational systems. A join makes it possible to select data from more than table by means of a single statement. This joining of tables may be done in a many ways.

Types of JOIN

1) Inner
2) Outer(left, right,full)
3) Cross

1) Inner join :

- Also known as equi join.
- Statements generally compares two columns from two columns with the equivalence operator =.
- This type of join can be used in situations where selecting only those rows that have values in common in the columns specified in the ON clause, is required.

• Syntax :

(ANSI style)

SELECT<columnname1>, <columnname2> <columnNameN> FROM <tablename1> INNER JOIN <tablename2>ON <tablename1>.<columnname> = <tablename2>.<columnname> WHERE <condition> ORDER BY <columnname1>;

(theta style)

SELECT<columnname1>, <columnname2> <columnNameN> FROM <tablename1>, <tablename2> WHERE <tablename1>.<columnname> = <tablename2>.<columnname> AND <condition> ORDER BY <columnname1>;

- List the employee details along with branch names to which they belong.

Emp(empno,fname,lname,dept,desig,branchno)

Branch(bname,branchno)

Select e.empno,e.fname,e.lname,e.dept, b.bname, e.desig from emp e inner join branch b on b.branchno=e.branchno;

Select e.empno, e.fname, e.lname, e.dept, b.bname, e.desig from emp e, branch b on where b.branchno=e.branchno;

Eg. List the customers along with the account details associated with them.

Customer(custno,fname,lname)

Acc_cust_dtls(fdno,custno)

Acc_mstr(accno,branchno,curbal)

Branch_mstr(name,branchno)

- Select c.custno, c.fname, c.lname, a.accno,a.curbal,b.branchno,b.name from customer c inner join acc_cust_dtls k on c.custno=k.custno inner join acc_mstr a on k.fdno=a.accno inner join branch b on b.branchno=a.branchno where c.custno like „C%" order by c.custno;
- Select c.custno, c.fname, c.lname, a.accno,a.curbal,b.branchno,b.name from customer c, acc_cust_dtls k, acc_mstr a, branch b where c.custno=k.custno and k.fdno=a.accno and b.branchno=a.branchno and c.custno like „C%" order by c.custno;

Outer Join

Outer joins are similar to inner joins, but give a little bit more flexibility when selecting data from related tables. This type of joins can be used in situations where it is desired, to select all rows from the table on left(or right, or both) regardless of whether the other table has values in common & (usually) enter NULL where data is missing.

- Tables

Emp_mstr(empno,fname,lname,dept)

Cntc_dtls(codeno,cntc_type,cntc_data)

Left Outer Join

List the employee details along with the contact details(if any) using left outer join.

- Select e.empno, e.fname, e.lname, e.dept, c.cntc_type, c.cntc_data from emp_mstr e left join cntc_dtls c on e.empno=c.codeno;
- Select e.empno, e.fname, e.lname, e.dept, c.cntc_type, c.cntc_data from emp_mstr e cntc_dtls c where e.empno=c.codeno(+);

All the employee details have to be listed even though their corresponding contact information is not present. This indicates all the rows from the first table will be displayed even though there exists no matching rows in the second table.

Right outer join

List the employee details with contact details(if any using right outer join.

- Tables

Emp_mstr(empno,fname,lname,dept)

Cntc_dtls(codeno,cntc_type,cntc_data)

- Select e.empno, e.fname, e.lname, e.dept, c.cntc_type, c.cntc_data from emp_mstr e right join cntc_dtls c on e.empno=c.codeno;
- Select e.empno, e.fname, e.lname, e.dept, c.cntc_type, c.cntc_data from emp_mstr e cntc_dtls c where e.empno(+)=c.codeno;

Since the RIGHT JOIN returns all the rows from the second table even if there are no matches in the first table.

Cross join

A cross join returns what known as a Cartesian Product. This means that the join combines every row from the left table with every row in the right table. As can be imagined, sometimes this join produces a mess, but under the right circumstances, it can be very useful. This type of join can be used in situation where it is desired, to select all possible combinations of rows & columns from both tables. The kind of join is usually not preferred as it may run for a very long time & produce a huge result set that may not be useful.

- Create a report using cross join that will display the maturity amounts for predefined deposits, based on min & max period fixed/ time deposit.
- Tables

Tem_fd_amt(fd_amt)

Fd_mstr(minprd,maxprd,intrate)

- Select fd_amt, s.minprd, s.maxprd, s.intrate,round (t.fd_amt+(s.intrate/100) * (s.minprd/365))) “amount_min_period”,round(t.fd_amt+(s.intrate/100)*(s.maxprd/365))) “amount_max_period” from fd_mstr s cross join tem_fd_amt t;
- Select t.fd_amt, s.minprd, s.maxprd, s.intrate, round(t.fd_amt+(s.intrate/100) * (s.minprd/365))) “amount_min_period”, round(t.fd_amt+(s.intrate/100)*(s.maxprd/365))) “amount_max_period” from fd_mstr s, tem_fd_amt t;

Self join

- In some situation, it is necessary to join to itself, as though joining 2 separate tables.
- This is referred to as self join

Example

- Emp_mgr(empno,fname, lname,mgrno)
- Select e.empno,e.fname,e.lname, m.fname "manager" from emp_mgr e, emp_mgr m where e.mgrno=m.empno;

Three tire Architecture:

A three-tier architecture is a client-server architecture in which the functional process logic, data access, computer data storage and user interface are developed and maintained as independent modules on separate platforms. Three-tier architecture is a software design pattern and a well-established software architecture.

In a Three-tier architecture, the client machine acts as merely a front end and does not contain any direct database calls. Instead, the client end communicates with an application server, usually through a forms interface. The application server in turn communicates with a

database system to access data. The business logic of the application, which says what actions to carry out under what conditions, is embedded in the application server, instead of being distributed across multiple clients. Three-tier applications are more appropriate for large applications, and for applications that run on the World Wide Web.

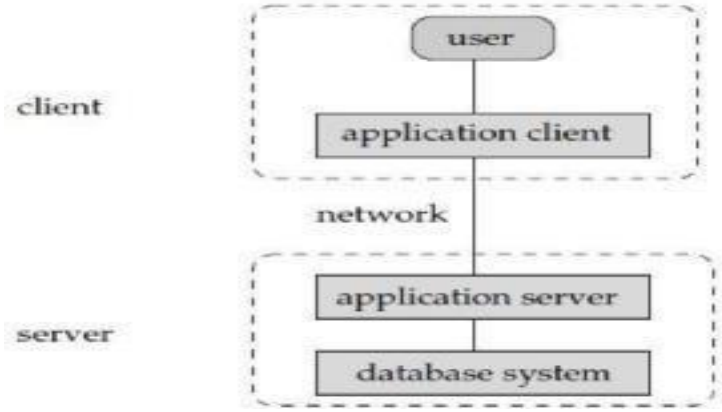


Fig. 1 Three Tire Architecture

Three-tier architecture allows any one of the three tiers to be upgraded or replaced independently. The user interface is implemented on a desktop PC and uses a standard graphical user interface with different modules running on the application server. The relational database management system on the database server contains the computer data storage logic. The middle tiers are usually multitiered.
The three tiers in a three-tier architecture are:

1. Presentation Tier: Occupies the top level and displays information related to services available on a website. This tier communicates with other tiers by sending results to the browser and other tiers in the network.
2. Application Tier: Also called the middle tier, logic tier, business logic or logic tier, this tier is pulled from the presentation tier. It controls application functionality by performing detailed processing.
3. Data Tier: Houses database servers where information is stored and retrieved. Data in this tier is kept independent of application servers or business logic.

Employee (Eno, Ename, Deptno, Salary) Eno=pk, Deptno=fk

Department (Deptno, Dname) Deptno=pk

Implement all join operation –cross join, natural join ,equi join, left outer ,right outer join etc & Write SQL Queries for following questions

i) List of employee names of 'Computer' department.

ii) Find the Employee who"s Salary above 50000 of each department.

iii) Find department name of employee name 'Amit'.

Conclusion: Thus we have studied to use & implement various join operation with nested queries.

Assignment No: 4

Title:- Unnamed PL/SQL code block: Use of Control structure and Exception handling is mandatory.

Write a PL/SQL block of code for the following requirements:-
Schema:
1. Borrower(Rollin, Name, DateofIssue, NameofBook, Status)
2. Fine(Roll_no,Date,Amt)

- Accept roll_no & name of book from user.
- Check the number of days (from date of issue), if days are between 15 to 30 then fine amount will be Rs 5per day.
- If no. of days>30, per day fine will be Rs 50 per day & for days less than 30, Rs. 5 per day.
- After submitting the book, status will change from I to R.
- If condition of fine is true, then details will be stored into fine table.

Frame the problem statement for writing PL/SQL block inline with above statement.

Objective:- Learn the concept of PL/SQL

Theory:
Introduction :-PL/SQL

The development of database applications typically requires language constructs similar to those that can be found in programming languages such as C, C++, or Pascal. These constructs are necessary in order to implement complex data structures and algorithms. A major restriction of the database language SQL, however, is that many tasks cannot be accomplished by using only the provided language elements.

PL/SQL (Procedural Language/SQL) is a procedural extension of Oracle-SQL that o ffers language constructs similar to those in imperative programming languages.

Or

A PL/SQL is a procedural language extension to the SQL in which you can declare and use the variables, constants, do exception handling and you can also write the program modules in the form of PL/SQL subprograms.PL/SQL combines the features of a procedural language with structured query language

PL/SQL allows users and designers to develop complex database applications that require the usage of control structures and procedural elements such as procedures, functions, and modules.

The basic construct in PL/SQL is a block. Blocks allow designers to combine logically related (SQL-) statements into units. In a block, constants and variables can be declared, and variables can be used to store query results. Statements in a PL/SQL block include SQL statements, control structures (loops), condition statements (if-then-else), exception handling, and calls of other PL/SQL blocks.

PL/SQL blocks that specify procedures and functions can be grouped into packages. A package is similar to a module and has an interface and an implementation part. Oracle o ffers several predefined packages, for example, input/output routines, file handling, job scheduling etc. (see directory $ORACLE HOME/rdbms/admin).

Another important feature of PL/SQL is that it o ffers a mechanism to process query results in a tuple-oriented way, that is, one tuple at a time. For this, cursors are used. A cursor basically is a pointer to a query result and is used to read attribute values of selected tuples into variables. A cursor typically is

used in combination with a loop construct such that each tuple read by the cursor can be processed individually.

In summary, the major goals of PL/SQL are to

- Increase the expressiveness of SQL,
- Process query results in a tuple-oriented way,
- Optimize combined SQL statements,
- Develop modular database application programs,
- Reuse program code, and
- Reduce the cost for maintaining and changing applications

Advantages of PL/SQL:-

Following are some advantages of Pl/SQL

1) Support for SQL :-PL/SQL is the procedural language extension to SQL supports all the functionalities of SQL.

2) Improved performance:- In SQL every statement individually goes to the ORACLE server, get processed and then execute. But in PL/SQL an entire block of statements can be sent to ORACLE server at one time, where SQL statements are processed one at atime.PL/SQL block statements drastically reduce communication between the application and ORACLE. This helps in improving the performance.

3) Higher Productivity:- Users use procedural features to build applications.PL/SQL code is written in the form of PL/SQL block.PL/SQL blocks can also used in other ORACLE Forms, ORACLE reports. This code reusability increases the programmers productivity.

4) Portability :- Applications written in PL/SQL are portable. We can port them from one environment to any computer hardware and operating system environment running ORACLE.

5) Integration with ORACLE :-Both PL/SQL and ORACLE are SQL based.PL/SQL variables have datatypes native to the oracle RDBMS dictionary. This gives tight integration with ORACLE.

Features of PL/SQL:-

1) We can define and use variables and constants in PL/SQL.

2) PL/SQL provides control structures to control the flow of a program. The control structures supported by PL/SQL are if..Then, loop, for..loop and others.

3) We can do row by row processing of data in PL/SQL.PL/SQL supports row by row processing using the mechanism called cursor.

4) We can handle pre-defined and user-defined error situations. Errors are warnings and called as exceptions in PL/SQL.

5) We can write modular application by using sub programs.

The structure of PL/SQL program:-

The basic unit of code in any PL/SQL program is a block. All PL/SQL programs are composed of blocks. These blocks can be written sequentially.

The structure of PL/SQL block:-

DECLARE

Declaration section
BEGIN
Executable section
EXCEPTION
Exception handling section
END;

Where

1) Declaration section
   PL/SQL variables, types, cursors, and local subprograms are defined here.
2) Executable section
   Procedural and SQL statements are written here. This is the main section of the block. This section is required.
3) Exception handling section
   Error handling code is written here
   This section is optional whether it is defined within body or outside body of program.

Conditional statements and Loops used in PL/SQL

Conditional statements check the validity of a condition and accordingly execute a set of statements. The conditional statements supported by Pl/SQL is

1) IF..THEN
2) IF..THEN..ELSE
3) IF..THEN..ELSIF

1) IF..THEN

Syntax1:-

```
If condition THEN
Statement list
END IF;
```

2) IF..THEN..ELSE

Syntax 2:-

```
IF condition THEN
        Statement list
ELSE
        Statements
END IF;
```

3) IF..THEN..ELSIF

Syntax 3:-

```
If condition THEN
        Statement list
ELSIF condition THEN
        Statement list
ELSE
        Statement list
END IF;

END IF;
```

2) CASE Expression :CASE expression can also be used to control the branching logic within PL/SQL blocks. The general syntax is

```
CASE
WHEN <expression> THEN <statements>;
WHEN <expression> THEN <statements>;
.
.
ELSE
<statements>;
END CASE;
```

Here expression in WHEN clause is evaluated sequentially. When result of expression is TRUE, then corresponding set of statements are executed and program flow goes to END CASE.

ITERATIVE Constructs : Iterative constructs are used to execute a set of statements respectively. The iterative constructs supported by PL/SQL are follows:

1) SIMPLE LOOP
2) WHILE LOOP
3) FOR LOOP

1) The Simple LOOP : It is the simplest iterative construct and has syntax like:

```
LOOP
   Statements
END LOOP;
```

The LOOP does not facilitate a checking for a condition and so it is an endless loop. To end the iterations, the EXIT statement can be used.

```
LOOP
        <statement list>
                IF condition THEN
                EXIT;
                END IF;
END LOOP;
```

The statements here is executable statements,which will be executed repeatedly until the condition given if IF..THEN evaluates TRUE.

2) THE WHILE LOOP

The WHILE…LOOP is a condition driven construct i.e the condition is a part of the loop construct and not to be checked separately. The loop is executed as long as the condition evaluates to TRUE.

The syntax is:-

```
WHILE condition LOOP
     Statements
END LOOP;
```

The condition is evaluated before each iteration of loop. If it evaluates to TRUE, sequence of statements are executed. If the condition is evaluated to FALSE or NULL, the loop is finished and the control resumes after the END LOOP statement.

3) THE FOR LOOP :The number of iterations for LOOP and WHILE LOOP is not known in advance. THE number of iterations depends on the loop condition. The FOR LOOP can be used to have a definite numbers of iterations.

The syntax is:-

```
For loop counter IN  [REVERSE] Low bound..High bound LOOP
Statements;
End loop;
```

Where

- loop counter –is the implicitly declared index variable as BINARY_INTEGER.
- Low bound and high bound specify the number of iteration .
- Statements:-Are the contents of the loop

EXCEPTIONS:- Exceptions are errors or warnings in a PL/SQL program.PL/SQL implements error handling using exceptions and exception handler.
Exceptions are the run time error that a PL/SQL program may encounter.
There are two types of exceptions

1) Predefined exceptions
2) User defined exceptions

1) Predefined exceptions:- Predefined exceptions are the error condition that are defined by ORACLE. Predefined exceptions cannot be changed. Predefined exceptions correspond to common SQL errors. The predefined exceptions are raised automatically whenever a PL/SQL program violates an ORACLE rule.

2)User defined Exceptions:- A user defined exceptions is an error or a warning that is defined by the program.User defined exceptions can be define in the declaration section of PL/SQL block.
User defined exceptions are declared in the declarative section of a PL/SQL block. Exceptions have a type Exception and scope**.**

Syntax :

```
DECLARE
        <Exception Name> EXCEPTION;
BEGIN
        ….
        RAISE <Exception Name>
        …
EXCEPTION
        WHEN <Exception name> THEN
        <Action>
END;
```

Exception Handling

A PL/SQL block may contain statements that specify exception handling routines. Each error or warning during the execution of a PL/SQL block raises an exception. One can distinguish between two types of exceptions:

• **System defined exceptions**
• **User defined exceptions** (which must be declared by the user in the declaration part of a block where the exception is used/implemented)

System defined exceptions are always automatically raised whenever corresponding errors or warnings occur. User defined exceptions, in contrast, must be raised explicitly in a sequence of statements using raise <exception name>. After the keyword exception at the end of a block, user defined exception handling routines are implemented. An implementation has the pattern

when <exception name> then <sequence of statements>;

The most common errors that can occur during the execution of PL/SQL programs are handled by system defined exceptions. The table below lists some of these exceptions with their names and a short description.

Oracle Error	**Equivalent Exception**	**Description**
ORA-0001	DUP_VAL_ON_INDEX	Unique constraint violated.
ORA-0051	TIMEOUT_ON_RESOURSE	Time-out occurred while waiting for recourse
ORA-0061	TRANSACTION_BACKED_OUT	The transaction was rolled back to due to deadlock.
ORA-1001	INVALID_CURSOR	Illegal cursor operation.
ORA-1012	NOT_LOGGED_ON	Not connected to Oracle.
ORA-1017	LOGIN_DENIED	Invalid username/passward
ORA-1403	NO_DATA_FOUND	No data found.
ORA-1410	SYS_INVALID_CURSOR	Conversion to a universal rowed failed.
ORA-1422	TOO_MANY_ROWS	A SELECT…INTO statement matches more than one row.
ORA-1476	ZERO_DIVIDE	Division by zero.
ORA-1722	INVALID_NUMBER	Conversion to a number failed.
ORA-6500	STORAGE_ERROR	Internal PL/SQL error raised if PL/SQL runs out of memory.
ORA-6501	PROGRAM_ERROR	Internal PL/SQL error.
ORA-6502	VALUE_ERROR	Truncation, arithmetic or conversion error.
ORA-6504	ROWTYPE_MISMATCH	Host cursor variable and PL/SQL cursor variable have incompatible row type
ORA-6511	CURSOR_ALREADY_OPEN	Attempt to open a cursor that is already open.
ORA-6530	ACCESS_INTO_NULL	Attempt to assign values to the attributes of a NULL object.
ORA-6531	COLLECTION_IS_NULL	Attempt to apply collection methods other than EXISTS to a NULL PL/SQL table or varray.
ORA-6532	SUBSCRIPT_OUTSIDE_LIMIT	Reference to a nested table or varray index outside the declared range.
ORA-6533	SUBSCRIPT_BEYOND_COUNT	Reference to a nested table or varray index higher than the number of

		elements in the collection
ORA-6592	CASE_NOT_FOUND	No matching WHEN clause in a CASE statement is found
ORA-30625	SELF_IS_NULL	Attempt to call a method on a NULL object instance

Syntax:-

<Exception_name>Exception;

Handling Exceptions:- Exceptions handlers for all the exceptions are written in the exception handling section of a PL/SQL block.
Syntax:-

```
Exception
        When exception_name then
                Sequence_of_statements1;
        When exception_name then
                Sequence_of_statements2;
        When exception_name then
                Sequence_of_statements3;
End;
```

Example:

```
Declare
        emp sal EMP.SAL%TYPE;
        emp no EMP.EMPNO%TYPE;
        too_high_sal exception;
begin
        select EMPNO, SAL into emp no, emp sal
        from EMP where ENAME = "KING";
        if emp sal *1.05 > 4000 then raise too high sal
        else update EMP set SQL . . .
        end if ;
exception
        when NO DATA FOUND -- no tuple selected
        then rollback;
        when too_high_sal then insert into high sal emps values(emp no);
        commit;
end;
```

After the keyword when a list of exception names connected with or can be specified. The last when clause in the exception part may contain the exception name others. This introduces the default exception handling routine, for example, a rollback.

If a PL/SQL program is executed from the SQL*Plus shell, exception handling routines may contain statements that display error or warning messages on the screen. For this, the procedure raise application error can be used. This procedure has two parameters <error number> and <message text>. <error number> is a negative integer defined by the user and must range between -20000 and -20999. <error message> is a string with a length up to 2048 characters.

The concatenation operator "||" can be used to concatenate single strings to one string. In order to display numeric variables, these variables must be converted to strings using the function to char. If the procedure raise application error is called from a PL/SQL block, processing the PL/SQL block terminates and all database modifications are undone, that is, an implicit rollback is performed in addition to displaying the error message.

Example:

```
if emp sal *1.05 > 4000
then raise application error(-20010, "Salary increase for employee with Id "|| to char (EMP no) ||" is too high");
```

E.g.

```
Declare
    V_maxno number (2):=20;
    V_curno number (2);
    E_too_many_emp exception;
Begin
    Select count (empno)into v_curno from emp
    Where deptno=10;
    If v_curno>25 then
    Raise e_too_many_Emp;
    End if;
```

Conclusion:

Thus we have studied to use & implement various control structures and loop with nested queries.

Assignment No: 5

Title :- Cursors: (All types: Implicit, Explicit, Cursor FOR Loop, Parameterized Cursor) Write a PL/SQL block of code using parameterized Cursor, that will merge the data available in the newly created table Cust_New with the data available in the table Cust_Old. If the data in the first table already exist in the second table then that data should be skipped. Frame the separate problem statement for writing PL/SQL block to implement all types

Objective :- Learning the concept of cursor in PL/SQL

Theory :- CURSOR:-

For the processing of any SQL statement, database needs to allocate memory. This memory is called context area. The context area is a part of PGA (Process global area) and is allocated on the oracle server.

A cursor is associated with this work area used by ORACLE, for multi row queries. A cursor is a handle or pointer to the context area .The cursor allows to process contents in the context area row by row.

There are two types of cursors.

1) Implicit cursor:-Implicit cursors are defined by ORACLE implicitly. ORACLE defines implicit cursor for every DML statements.
2) Explicit cursor:-These are user-defined cursors which are defined in the declaration section of the PL/SQL block. There are four steps in which the explicit cursor is processed.
    1) Declaring a cursor
    2) Opening a cursor
    3) Fetching rows from an opened cursor
    4) Closing cursor

General syntax for CURSOR:-

```
DECLARE
    Cursor cursor_name IS select_statement or query;
BEGIN
    Open cursor_name;
Fetch cursor_name into list_of_variables;
    Close cursor_name;
END;
```

Where

1) Cursor_name:-is the name of the cursor.
2) Select_statement:-is the query that defines the set of rows to be processed by the cursor.
3) Open cursor_name:-open the cursor that has been previously declared.

When cursor is opened following things happen

i) The active set pointer is set to the first row.

ii) The value of the binding variables are examined.

4) Fetch statement is used to retrieve a row from the selected rows, one at a time, into PL/SQL variables.

5) Close cursor_name:-When all of cursor rows have been retrieved, the cursor should be closed.

Explicit cursor attributes:-

Following are the cursor attributes

1. %FOUND: - This is Boolean attribute. It returns TRUE if the previous fetch returns a row and false if it doesn‟t.

2. %NOTFOUND:-If fetch returns a row it returns FALSE and TRUE if it doesn‟t. This is often used as the exit condition for the fetch loop;

3. %ISOPEN:-This attribute is used to determine whether or not the associated cursor is open. If so it returns TRUE otherwise FALSE.

4. %ROWCOUNT:-This numeric attribute returns a number of rows fetched by the cursor.

Cursor Fetch Loops

1) Simple Loop

Syntax:-

```
LOOP
    Fetch cursorname into list of variables;
EXIT WHEN cursorname%NOTFOUND
    Sequence_of_statements;
END LOOP;
```

2) WHILE Loop

Syntax:-

```
FETCH cursorname INTO list of variables;
WHILE cursorname%FOUND LOOP
    Sequence_of_statements;
    FETCH cursorname INTO list of variables;
END LOOP;
```

3) Cursor FOR Loop

Syntax:

```
FOR variable_name IN cursorname LOOP
    -- an implicit fetch is done here.
    -- cursorname%NOTFOUND is also implicitly checked.
```

```
        -- process the fetch records.
        Sequence_of_statements;
    END LOOP;
```

There are two important things to note about :-

i) Variable_name is not declared in the DECLARE section. This variable is implicitly declared by the PL/SQL compiler.

ii) Type of this variable is cursorname%ROWTYPE.

Implicit Cursors

PL/SQL issues an implicit cursor whenever you execute a SQL statement directly in your code, as long as that code does not employ an explicit cursor. It is called an "implicit" cursor because you, the developer, do not explicitly declare a cursor for the SQL statement.

If you use an implicit cursor, Oracle performs the open, fetches, and close for you automatically; these actions are outside of your programmatic control. You can, however, obtain information about the most recently executed SQL statement by examining the values in the implicit SQL cursor attributes.

PL/SQL employs an implicit cursor for each UPDATE, DELETE, or INSERT statement you execute in a program. You cannot, in other words, execute these statements within an explicit

cursor, even if you want to. You have a choice between using an implicit or explicit cursor only when you execute a single-row SELECT statement (a SELECT that returns only one row).

In the following UPDATE statement, which gives everyone in the company a 10% raise, PL/SQL creates an implicit cursor to identify the set of rows in the table which would be affected by the update:

```
UPDATE employee
   SET salary = salary * 1.1;
```

The following single-row query calculates and returns the total salary for a department. Once again, PL/SQL creates an implicit cursor for this statement:

```
SELECT SUM (salary) INTO department_total
  FROM employee
 WHERE department_number = 10;
```

If you have a SELECT statement that returns more than one row, you must use an explicit cursor for that query and then process the rows returned one at a time. PL/SQL does not yet support any kind of array interface between a database table and a composite PL/SQL datatype such as a PL/SQL table.

Drawbacks of Implicit Cursors

Even if your query returns only a single row, you might still decide to use an explicit cursor. The implicit cursor has the following drawbacks:

- It is less efficient than an explicit cursor
- It is more vulnerable to data errors
- It gives you less programmatic control

The following sections explore each of these limitations to the implicit cursor.

Inefficiencies of implicit cursors

An explicit cursor is, at least theoretically, more efficient than an implicit cursor. An implicit cursor executes as a SQL statement and Oracle's SQL is ANSI-standard. ANSI dictates that a single-row query must not only fetch the first record, but must also perform a second fetch to determine if too many rows will be returned by that query (such a situation will RAISE the TOO_MANY_ROWS PL/SQL exception).Thus, an implicit query always performs a minimum of two fetches, while an explicit cursor only needs toperform a single fetch.

This additional fetch is usually not noticeable, and you shouldn't be neurotic about using an implicitcursor for a single-row query (it takes less coding, so the temptation is always there). Look out for indiscriminate use of the implicit cursor in the parts of your application where that cursor will be executed repeatedly. A good example is the Post-Query trigger in the Oracle Forms.

Post-Query fires once for each record retrieved by the query (created from the base table block and the criteria entered by the user). If a query retrieves ten rows, then an additional ten fetches are needed withan implicit query. If you have 25 users on your system all performing a similar query, your server mustprocess 250 additional (unnecessary) fetches against the database. So, while it might be easier to write an implicit query, there are some places in your code where you will want to make that extra effort andgo with the explicit cursor.

Vulnerability to data errors

If an implicit SELECT statement returns more than one row, it raises the TOO_MANY_ROWS exception. When this happens, execution in the current block terminates and control is passed to theexception section. Unless you deliberately plan to handle this scenario, use of the implicit cursor is adeclaration of faith. You are saying, "I trust that query to always return a single row!"

It may well be that today, with the current data, the query will only return a single row. If the nature ofthe data ever changes, however, you may find that the SELECT statement which formerly identified asingle row now returns several. Your program will raise an exception. Perhaps this is what you will want. On the other hand, perhaps the presence of additional records is inconsequential and should be ignored.

With the implicit query, you cannot easily handle these different possibilities. With an explicit query,your program will be protected against changes in data and will continue to fetch rows without raisingexceptions.

Conclusion: We have learned the concept of cursor and types of cursor.

Assignment no.06

Title of Assignment: MYSQL database connectivity

Assignment Name: -. Implement MYSQL/Oracle database connectivity with front end language(Java) to implement Database navigation operations (add, delete, edit, etc.).

Theory: -

Software Required and Steps

- Eclipse
- JDK 1.6
- MySQL
- Java-MySQL Connector

Connection to database with Java

The interface for accessing relational databases from Java is *Java Database Connectivity (JDBC)*. Via JDBC you create a connection to the database, issue database queries and updateas well as receive the results.

JDBC provides an interface which allows you to perform SQL operations independently ofthe instance of the used database. To use JDBC, you require the database specific implementation of the JDBC driver.

MySQL JDBC driver(Connector)

To connect to MySQL from Java, you have to use the JDBC driver from MySQL. The MySQL JDBC driver is called *MySQL Connector/J*. You find the latest MySQL JDBC driver under the following URL: http://dev.mysql.com/downloads/connector/j.

The download contains a JAR file which we require later.

To connect Java application with the MySQL database, we need to follow followingsteps.

1. **Driver class:** The driver class for the mysql database is **com.mysql.jdbc.Driver**.
2. **Connection URL:** The connection URL for the mysql database is **jdbc:mysql://localhost:3306/db1** where jdbc is the API, mysql is the database, localhost is the server name on which mysql is running, we may also use IP address,

3306 is the port number and sonoo is the database name. We may use any database, in such case, we need to replace the db1 with our database name.

3. **Username:** The default username for the mysql database is **root**.
4. **Password:** It is the password given by the user at the time of installing the mysql database.

In Eclipse perform following steps:

1. File - New – Java Project –Give Project Name – ok
2. In project Explorer window- right click on project name-newclass- give Class name-ok
3. In project Explorer window- right click on project name- Build path- Configure build path- Libraries- Add External Jar - JavaMySQL Connector
4. In MySQL first Create one Database with name db1 and one table with name stud(name,age)

Steps to be perform in Java to write Code:

1. Import packages

import java.sql.*;

2. Load Driver

Class.forName("oracle.jdbc.driver.OracleDriver");

3. Create connection

Connection con = DriverManager.getConnection(

"jdbc:mysql://localhost:3306/DatabaseName","username","passwd ")

4. Creates a Statement object for sending SQL statements to the database

Statement stmt = con.createStatement() ;

5. Executing SQL Statements

executeUpdate – This Method is used for insert, update and delete queries

- Example

String sql = “INSERT INTO stud VALUES

(‘Ankita’,21)";stmt.executeUpdate(sql);

6. Get ResultSet (SELECT Query)

```
String sql1 = "SELECT name, age FROM stud";
 //Write a select query in any string variable
 ResultSet rs = stmt.executeQuery(sql1);
 //executeQuery used to run the select query and store the result in ResultSetwhile
 (rs.next()) //Iterate through ResultSet till data is found
 {
 String name1 = rs.getString(“name");
 //Store name data into name1 variableint
 age1 = rs.getInt(“age");
 //Store age data into age1 variable
 }
```

7. Close connection

- stmt.close();

con.close();

Conclusion: We have learned the database connectivity.