Sinhgad Institutes

SINHGAD TECHNICAL EDUCATION SOCIETY'S

SINHGAD INSTITUTE OF TECHNOLOGY

Kusgaon (Bk), Lonavala 410401

DEPARTMENT OF COMPUTER ENGINEERING

LAB MANUAL

ACADEMIC YEAR (2022-23) SEMESTER-V

T.E. COMPUTER (SEM – V)

310246: DATA BASE MANAGEMENT SYSTEM

TEACHING SCHEME
PRACTICAL: 2 HRS/WEEK

EXAMINATION
TERM WORK: 25 MARKS
ORAL EXAM: 25 MARKS

Vision and Mission of Institute

VISION

उत्तमपुरुषान् उत्तमाभियंतृन् निर्मातुं कटीबध्दा: वयम्।

We are committed to produce not only good engineers but good human beings, also.

MISSION

- We believe in and work for the holistic development of students.
- We strive to achieve this by imbibing a unique value system, transparent work culture, excellent academic and physical environment conducive to learning, creativity and technology transfer.

Vision and Mission of Department

VISION

- The department of Computer Engineering in partnership with user industry will harness knowledge and Potential for application based product development in future, through world class education to empower the society around.

MISSION

- The department of Computer Engineering will be the widely recognized centers of excellence for promoting value added engineering education. We will contribute by evolving innovative technology solution to solve a wide range of complex scientific, technological and social problems.

Short Term Goals

- To establish post graduate program in different domain.
- To encourage faculty by creating opportunity of higher education.
- To initiate relevant value addition programs and certifications for improving employability.
- Build strong alliances that bring know-how of business community to complete training of students through projects.

Long Term Goals

- To establish a center of innovation in Agriculture, Tele health and ICT sector in collaboration with industry.
- To create center of excellence in network, security and computer vision.
- To establish a world class R&D institute for patent based research creating opportunity for faculty to be resource.

Program Educational Objectives (PEO's)

1. To prepare globally competent graduates having strong fundamentals and domain knowledge to provide effective solutions for engineering problems.
2. To prepare the graduates to work as a committed professional with strong professional ethics and values, sense of responsibilities, understanding of legal, safety, health, societal, cultural and environmental issues.
3. To prepare committed and motivated graduates with research attitude, lifelong learning, investigative approach, and multidisciplinary thinking.
4. To prepare the graduates with strong managerial and communication skills to work effectively as individual as well as in teams.

Program Outcomes: POs

Students are expected to know and be able –

PO1- *Engineering Knowledge*: - To apply knowledge of mathematics, science, engineering fundamentals, problem solving skills, algorithmic analysis and mathematical modelling to the solution of complex engineering problems.

PO2- *Problem Analysis*: - Identify, formulate, review research literature and analyze complex engineering problems reaching substantiated conclusion using first principals of mathematics, natural sciences and engineering sciences.

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

PO4- *Conduct Investigations of Complex Problems*: - Use research based knowledge and research methods including design of experiments, analysis and interpretation of

data, and modeling to complex engineering activities with an understanding of the limitations.

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

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

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

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

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

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

PO11- *Project management Finance*: - Demonstrate knowledge and understanding of engineering and management principles and apply these to one's own work as a member and leader in a team to manage projects and in multidisciplinary environment. **PO12**- *Life-long Learning*: - Recognize the need for, and have the preparations and ability to engage in independent and lifelong learning in the broadest context of technological change.

Program Specific Outcomes: PSOs

A graduate of the Computer Engineering Program will demonstrate-

PSO1- *Professional Skills*-The ability to understand, analyze and develop computer programs in the areas related to algorithms, system software, multimedia, web design, big data analytics, and networking for efficient design of computer-based systems of varying.

PSO2- *Problem-Solving Skills*- The ability to apply standard practices and strategies in software project development using open-ended programming environments to deliver a quality product for business success.

PSO3- *Successful Career and Entrepreneurship*- The ability to employ modern computer languages, environments, and platforms in creating innovative career paths to be an entrepreneur, and a zest for higher studies.

Sinhgad Institutes

SINHGAD INSTITUTE OF TECHNOLOGY

DEPARTMENT OF COMPUTER ENGINEERING

CERTIFICATE

This is to certify that, _____of Class____ Div. ___Roll No.___Exam Seat No._____has completed all the practical work in the subject ***Data Base Management System Laboratory (310246)***, satisfactorily, as prescribed by Savitribai Phule Pune University, Pune (SPPU) in the Academic Year 2022-23.

Prof. A K Desai

Subject In-charge

Dr. S.D.Babar

Head of Department

Assignment No. 1

Title : ER Modeling and Normalization:

Decide a case study related to real time application in group of 2-3 students and formulate a problem statement for application to be developed. Propose a Conceptual Design using ER features using tools like ERD plus, ER Win etc. (Identifying entities, relationships between entities, attributes, keys, cardinalities, generalization, specialization etc.) Convert the ER diagram into relational tables and normalize Relational data model

Problem Statement:

Project work Stage – I is an integral part of the Project work. In this, the student shall complete the partial work of the Project which will consist of problem statement, literature review, SRS, Model and Design. The student is expected to complete the project at least up to the design phase. As a part of the progress report of project work Stage-I, the candidate shall deliver a presentation on the advancement in Technology pertaining to the selected project topic. The student shall submit the duly certified progress report of Project work Stage-I in standard format for satisfactory completion of the work by the concerned guide.

The database of students who sign up for projects and are assigned mentors must be kept on file by the department of computer engineering.

1) Draw an ER diagram, which shows Student, Project, and Guide relationship.
2) Draw class diagram, Activity Diagram, Sequence diagram, Deployment Diagram.

Note:

In one Project Group, 3 students can enrol.

Use ER win, ERD plus or Staruml tool to draw all diagrams.

Outcome:

1) Students will be able to demonstrate their skills in drawing the ER diagram, and UML diagrams.

Write - up:

1) What is the main purpose of an ER diagram?
2) What are the symbols used in ER diagram?
3) What do you mean by Normalization?
4) Explain 4 types of normalization
5) How ER diagram and Normalization is related.
6) What do you mean by weak entity set?

Assignment No.2

Aim: Study of SQL DDL commands

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

Outcomes: Student should be able to

1. Create relations
2. Create view,Index,Sequence,Synonym

PEOs ,POs, PSOs and COs satisfied

PEOs : 1 **POs : 1,2,5** **PSOs : 1** **COs :1**

Software used :ORACLE

Theory : Introduction about SQL

SQL (Structured Query Language) is a nonprocedural language, you specify what you want, not how to get it. A block structured format of English key words is used in this Query language. It has the following components.

DDL (Data Definition Language)-The SQL DDL provides command for defining relation schemas, deleting relations and modifying relation schema.

DML (DATA Manipulation Language)-It includes commands to insert tuples into, delete tuples from and modify tuples in the database.

View definition-The SQL DDL includes commands for defining views.

Transaction Control- SQL includes for specifying the beginning and ending of transactions.

Integrity-The SQL DDL includes commands for specifying integrity constraints that the data stored in the database must specify. Updates that violate integrity constraints are allowed.

Authorization-The SQL DDL includes commands for specifying access rights to relations and views

.

Data Definition Language

The SQL DDL allows specification of not only a set of relations but also information about each relation, including-Schema for each relation.The domain of values associated with each attribute. The integrity constraints. The set of indices to be maintained for each relation. the security and authorization information for each relation. The physical storage structure of each relation on disk

Domain types in SQL-

The SQL standard supports a variety of built in domain types, including-
•Char (n)- A fixed length character length string with user specified length .
•Varchar (n)- A variable character length string with user specified maximum length n.
•Int- An integer.
•Small integer- A small integer.
•Numeric (p, d)-A Fixed point number with user defined precision.
•Real, double precision- Floating point and double precision floating point numbers with machine dependent precision.
•Float (n)- A floating point number, with precision of at least n digits.
•Date- A calendar date containing a (four digit) year, month and day of the month.
•Time- The time of day, in hours, minutes and seconds Eg. Time '09:30:00'.
•Number- Number is used to store numbers (fixed or floating point)

Data Definition in SQL

Creating Tables

Syntax:-

```
create table<table name>
(colume_name1 datatype size(),
colume_name 2 datatypesize(),
.
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;

Restrictions for crteating 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.

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.gdesc student

Alteration of TABLE:-

Alter table command

Syntax:-

Case1:-

Alter table <table_name>
Add(colume_name 1 datatype size(),
colume_name 2 datatypesize(),
- .
colume_name n datatype size());

Case2:-

Alter table <table_name>
Modify(colume_name 1 datatype size(),

colume_name 2 datatypesize(),

.

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 withmodify 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;

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

1**) 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);

2) 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.

1. Check constraints at column level

Syntax :

CREATE TABLE <tableName>
(<ColumnName>datatype(size)CHECK(columnName condition),<columnnamedatatype(size));

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

2. Check constraints at table level

Syntax :

CREATE TABLE <tableName>
(<ColumnName>datatype(size),
<ColumnName>datatype(size),….,
CONSTRAINT <constraint_name> CHECK (columnName condition),..
);

Syntax :

CREATE TABLE <tableName>
(<ColumnName>datatype(size),
<ColumnName>datatype(size),….,
CHECK (columnName condition));

After table creation

Alter table tablename
Add constraints constraintnameckeck(condition)

3) 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>),…..);

1.Primary key constraint defined at Table level

Syntax:

CREATE TABLE <TableName>
(<ColumnName1><DataType>(<Size>) ,…, PRIMARY KEY(<ColumnName1><ColumnName2>));

2.Primary key constraint defined at Table level

Syntax:

CREATE TABLE <TableName>
(<ColumnName1><DataType>(<Size>) <columnname2 datatype<(size)<,<columnname3 datatype<size>constraint constraintname PRIMARY KEY(<ColumnName1>));

3. After table creation

Alter table tablename
Add(constraint constraintname primary key(columnname));

Or

Alter table tablename

Add primary key(columnname));

4) 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.

1. UNIQUE constraint defined at column level

Syntax :
Create table tablename(***<columnname><datatype>(<Size> UNIQUE),<columnname>datatype(<size>)……);***

2. UNIQUE constraint defined at table level

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

3. After table creation

Alter table tablename
Add constraint constraintnameunique(columnname);

5)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.

1. 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>;

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;

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_positioncol_pos, ix.uniqueness
fromuser_indexes ix, user_ind_columnsic
whereic.index_name=ix.index_name
and ic.table_name='emp';

Removing an Index

Syntax:-

Drop index <index_name>;

eg. Drop index emp_name_idx;

Note: 1) we can not modify indexs.
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 allowedthat use one of the following constructs in the view definition:

- Joins
- Aggregate function such as sum, min, max etc.
- set-valuedsubqueries (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 between sequence 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 to be 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 min value.

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 oracle server 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;

Lab Exercises:-

Consider the following database where the primary keys are underlined. Create the following tables in oracle

1)

Person(driver_id, name, address)
Car(license, model, year)
Accident(report_no, date_acc, location)
Owns(driver_id, license)
Participated(driver_id, model, report_no, damage_amount)

2)

Employee(employee_name,street,city)
Works(employee_name,company_name,salary)
Company(company_name,city)
Manages(employee_name,manager_name)

1) Create view with the employee_name,company_name by using above tables.
2) Create index for employee & participated table.
3) Create sequence for person & insert 4 records using sequence.
4) Create the synonym for table participated & company. Display the record using this table. Update the record using the synonym tables.

FAQS:-

1) What is SQL?
2) What are the different data types used in ORACLE? Explain in short
3) List the different constraints used in oracle? Explain.
4) Explain the primary key constarints?
5) What is the difference between delete, truncate and drop command
6) What is view? Explain with ex.
7) Explain the term DDL and DML and DCL?

8) What is synonym? List the difference between view and synonym.
9) What is sequence? Write syntax of the sequence and explain the options of sequence
10) What is INDEX? List and explain the types of INDEX.

CONCLUSION:-

Design Experiments

1. Use an Entity-Relationship Diagram to depict the information needs of a small computer business:

- The employees of the company assemble a number of types of computers. For each employee a record is kept of his employee no., name, address, phone no., job title and salary.
- A record is also kept of each of the machines model, specs and name and quantity on hand.
- Each machine consists of a number of parts. An inventory must be kept of the parts in stock. For each part a record is kept of its name, price and quantity on hand.
- These parts are ordered from various suppliers. A record must be kept of the suppliers name, address and phone number.
- Once assembled these computers are sold to various customers. A record is kept of the customers name, address and phone number. Some of these customers are credit customers and for these customers a record is kept of their credit limit.

2. Use an Entity-Relationship Diagram to depict the following requirements for a restaurant:

- The restaurant employs a number of chefs. A record is kept of each chef's name, address, phone number and salary.
- Each chef can prepare a number of meals. The name of the meal and the price of the meal are recorded.
- Each meal consists of a number of ingredients. The name of the ingredient and the quantity required for that particular meal is recorded.
- These meals are ordered by customers. A record is kept of the customer's name, address and phone number. A record is kept of the time and date themeal is ordered.
- State any assumptions made in the design of the E-R diagram.

Assignment No.3

Aim: Study of SQL DML commands

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

Outcomes: Student should be able to

1. Perform different manipulation function like insert, delete and update data to and from relation
2. Use different logical and numerical functions.

PEOs ,POs, PSOs and COs satisfied

PEOs : 1 POs : 1,2,5 PSOs : 1 COs :1

Software used :ORACLE

Theory :SQL – Data Manipulation Statements.

Selection of Tuple

We can select (some) attributes of all tuples from a table. If one is interested in tuples that satisfy certain conditions, the where clause is used. In a where clause simple conditions based on comparison operators can be combined using the logical connectives and, or, and not to form complex conditions. Conditions may also include pattern matching operations and even subqueries.

Select command : It is used to retrieve the information stored in the database.

Syntax:-

Select [distinct] <column(s)>
From <table>
[where<condition>]
[order by <column(s) [asc|desc]>]

OR

Select column_name1…..,column_name n from <table_name>

OR

Select * from <table_name>

E.gTo see the contents of table student

Select name, roll, class, branch, enroll_no from student;

OR

Select * from student

Selecting distinct rows: To prevent the selection of duplicate rows,we include distinct clause in the select Command.
For e.g

select distinct name from student

Filtering table data: While viewing data from a table, it is rare that all the data from the table will be required each time. The retrieval of specific columns from a table.

Syntax:-
Select column_name1,column_name2,column_name n from <table_name>;
e.g

Select name, roll from student;
To retrieve selected rows and all columns from a table :

ORACLE provides the option of using a 'where' clause in an SQL sentence to apply a filter on the rows in the table. When a where clause is added it select only those rows that satisfy the specified condition, will be displayed.
Syntax:-

Select column_name1,column_name2,column_name n from <table_name>
Where <condition>;

Example: List the job title and the salary of those employees whose manager has the number 7698 or 7566 and who earn more than 1500:

Select JOB, SAL from EMP where (MGR = 7698 or MGR = 7566) and SAL > 1500;

For all data types, the comparison operators =, != or <>,<, >,<=, => are allowed in the conditions of a 'where' clause.

Further comparison operators are:

1**) IN and NOT IN Predicates :** In case a value needs to be compare to a list of values, then the IN and NOT IN predicate is used. We can check a single value against multiple values by using the IN & NOT IN predicate.
Syntax: IN predicate

Select column_name1, column_name2, ..,column_name n from < table name>
wherecolumn_name in ('valuel' ,'value2', ..,'valuen');
e.g.
selectempno, ename , job from emp
where job in ('analyst', 'clerk');

The above example displays employee number, name and job of all employees whose job is 'analyst' or 'clerk'.

OR

• Set Conditions: <column> [not] in (<list of values>)

Example: select ∗ from DEPT where DEPTNO in (20,30);

Syntax: NOT IN

Select column_namel, ,column_name n From < table name>
Where column name not in ('value 1 ', ..,'value n ');

e.g.
Select empno, ename , job from emp
where job in ('analyst', 'clerk');

The above example will display employee number, name and job where the JOB is not a salesman or clerk.

Guidelines:
1) The IN & NOT IN operators can be used with any datatype.
2) If characters or dates are used in the list, they must be enclosed in single quotation marks (' ').

• Null value: <column> is [not] null,
i.e., for a tuple to be selected there must (not) exist a defined value for this column.

Example: select ∗ from EMP where MGR is not null;

Note: the operations = null and ! = null are not defined!

Syntax: ¬ BETWEEN

Select column name 1 , ..,column name n from < table name>
Where column_name between <lowerbound> and <upperbound>;

You can display rows based on a range of values using BETWEEN operator The range that you specify contains a lower bound and upper bound.
OR
• Domain conditions: <column> [not] between <lower bound> and <upper bound>

Example:
select EMPNO, ENAME, SAL from EMP where SAL between 1500 and 2500;
select ENAME from EMP where HIREDATE between '02-APR-81' and '08-SEP-81'

Syntax: ¬ **NOT BETWEEN**

selectcolumn_name 1 , ..,column_name n from < table name>
wherecolumn_name not between <lowerbound> and <upperbound>;

String Operations

In order to compare an attribute with a string, it is required to surround the string by apostrophes, e.g., where LOCATION = 'DALLAS'. A powerful operator for pattern matching is the like operator. Together with this operator, two special characters are used: the percent sign % (also called wild card), and the underline , also called position marker. For example, if one is interested in all tuples of the table DEPT that contain two C in the name of the department, the condition would be where DNAME like '%C%C%'. The percent sign means that any (sub)string is allowed there, even the empty string. In contrast, the underline stands for exactly one character. Thus the condition where DNAME like '%C C%' would require that exactly one character appears between the two Cs. To test for inequality, the not clause is used.

LIKE & NOT LIKE Predicates

Oracle provides a LIKE predicate, which allows comparison of one string with another string value which is not identical. This is achieved by using wildcard characters.

Wild card characters are:
1) Percent sign (%) :- It matches any string.
2) Underscore (-) :- It matches any single character.

For char data types

Syntax: - LIKE using % (Percentage)

Select column_name 1, column _name2,. . . , column_name n from < table name>
Where column~name like 'value%'

e.g.
Select ename, job, salary from emp
whereename like 'ni%';
The above example will display all rows from table EMP where employee name begins with 'ni '.
Syntax: ¬ Like using '-' (underscore)

Select column_name 1... Column_name n from < table name>

where column name like '- value%' ;

e.g.
selectempno, ename,addrcss from emp
where address like' - une' ;
The above example will display all rows where the address starts with any character but ends with 'une'.
Further string operations are:
• upper(<string>) takes a string and converts any letters in it to uppercase, e.g.
DNAME= upper(DNAME) (The name of a department must consist only of upper case letters.)

• lower(<string>) converts any letter to lowercase,
• initcap(<string>) converts the initial letter of every word in <string> to uppercase.
• length(<string>) returns the length of the string.
• substr(<string>, n [, m]) clips out a m character piece of <string>, starting at position n. If m is not specified, the end of the string is assumed.
substr('DATABASE SYSTEMS', 10, 7) returns the string 'SYSTEMS

Aggregate Functions
Aggregate functions are statistical functions such as count, min, max etc. They are used to Compute a single value from a set of attribute values of a column: The SUM and AVG must be a collection of numbers,but the other operators can operate on collections of nonnumeric data types such as string.

1)AVG:-It is used to find out the average value.
Format:-avg(column_name)
e.g. select avg(salary) from emp;

2)SUM:-It is used to find out the total or sum.
Format:-sum(column_name)
e.g. select sum(salary) from emp;
3) MIN:-It is used to find out the minimum value.
Format:-min(column_name)
e.g. select min(salary) from emp;

4) MAX:-It is used to find out the average value.
Format:-max(column_name)
e.g. select max(salary) from emp;

e.g. Compute the difference between the minimum and maximum salary.
select max(SAL) - min(SAL) from EMP;

5) COUNT:-It is used to find out the total number of record or rows present in the relation or table.

Format:-count(column_name)
e.g. select avg(salary) from emp;
Count(Distinct column_name) : It is used to eliminate the duplicate and null values in the specified column.
e.g Select count(distinct deptno) from emp;
e.g: How many different job titles are stored in the relation EMP?
select count(distinct JOB) from EMP;

Note: avg, min and max ignore tuples that have a null value for the specified attribute, but count considers null values

Ordering operation – using ORDER BY: The order by cause is used to arrange the records in ascending or descending order

Syntax:- **Select<expr> from <table_name>**
[where condition(s)]
[order by{column,expr}[asc|desc]];

1)Asc:-orders the rows in ascending order.by default order is asc.
e.g. select * from emp order by ename
e.g. select * from emp order by ename,empno

2)Desc:-orders the rows in descending order.
e.g. select * from emp order by enamedesc

DATE FUNCTIONS:-

Date functions operate on ORACLE dates.All date functions return a value of date datatype except months_betwwen, which returns a numeric value.

Sysdate:-Sysdate is a pseudo column that contains the current date and time. It requires no arguments when selected from the table dual and returns the current date.

Dual table:-Dual is adummy table in ORACLE.It has a single column with datatypevarchar() and it contains a single row with value x.
Following are the oracle date function.

1.Months_between:-Months_between finds the number of months between d1 & d2. If date d1 is later than d2 the result is positive. If date d2 is earlier than date d2 the result is negative.

Format:- Months_between(d1,d2)
:-Where d1 and d2 are dates.

e.g select months_between('02-feb-1993','02-jan-1993')"months" from dual;

2. Add_months:-Returns the date after adding the number of months specified with the function.

Format :- Add_Months(d,n)
:- Where d-is the date
:- n is number of months to be added to the date.

e.g. selectadd_months(sysdate,4) from dual;
3. Next_day : Returns the date of the first weekday named 'char' that is after the date named by date.
Format:- next_day(date,char)
:- Where char-day of the week.
e.g select next_day('30-jul-07','monday')"next_day" from dual;
4.Last_day : Returns the last day of the month that contains date 'd'.
Format:- last_day(d)
Where d-date
e.g. select last_day(sysdate)"last" from dual;
5.Round :Returns date rounded to the unit specified by the format model"fmt". If the format model 'fmt' is omitted, date is rounded to the nearest date.
Format:- round(date,'fmt']
e.g. select round(sysdate,'month')"round" from dual;
e.g select round(sysdate,'year')"round" from dual;
e.g select round(sysdate,'day')"round" from dual;
6.Trunc: Returns date with the time portion of the day truncated to the unit specified by the format model fmt. If the format model fmt is omitted, date truncated to the nearest day.

Format:- trunc(date,'fmt']

e.g. select trunc(sysdate,'month')"round" from dual;
e.g select trunc(sysdate,'year')"round" from dual;
e.g select trunc(sysdate,'day')"round" from dual;
Date Format

Whenever a DATE value is displayed, Oracle will call TO_CHAR automatically with the default DATE format. However, you may override the default behavior by calling TO_CHAR explicitly with your own DATE format.

For example,

```
SELECT TO_CHAR(b, 'YYYY/MM/DD') AS b FROM x;
```

returns the result:

```
B
---------------------------------
1998/04/01
```

The general usage of TO_CHAR is: **TO_CHAR(<date>, '<format>')**

where the <format> string can be formed from over 40 options. Some of the more popular ones include : for example

MM	Numeric month (*e.g.*, 07)
MON	Abbreviated month name (*e.g.*, JUL)
MONTH	Full month name (*e.g.*, JULY)
DD	Day of month (*e.g.*, 24)
DY	Abbreviated name of day (*e.g.*, FRI)
YYYY	4-digit year (*e.g.*, 1998)
YY	Last 2 digits of the year (*e.g.*, 98)
RR	Like YY, but the two digits are ``rounded'' to a year in the range 1950 to 2049. Thus, 06 is considered 2006 instead of 1906, for example.
AM (or PM)	Meridian indicator
HH	Hour of day (1-12)
HH24	Hour of day (0-23)
MI	Minute (0-59)
SS	Second (0-59)

You can change the default DATE format of Oracle from "DD-MON-YY" to something you like by issuing the following command in sqlplus:

alter session set NLS_DATE_FORMAT='<my_format>';

The change is only valid for the current sqlplus session.

Data Modifications in SQL

After a table has been created using the create table command, tuples can be inserted into the table, or tuples can be deleted or modified.

INSERT command or Insertions

The most simple way to insert a tuple into a table is to use the insert statement

insert into <table> [(<column i, . . . , column j>)]
values (<value i, . . . , value j>);

For each of the listed columns, a corresponding (matching) value must be specified. Thus an insertion does not necessarily have to follow the order of the attributes as specified in the create table statement. If a column is omitted, the value null is inserted instead. If no column listis given, however, for each column as defined in the create table statement a value must be given.

Examples:

insert into PROJECT(PNO, PNAME, PERSONS, BUDGET, PSTART)
values(313, 'DBS', 4, 150000.42, '10-OCT-94');
or
insert into PROJECT
values(313, 'DBS', 7411, null, 150000.42, '10-OCT-94', null);

If there are already some data in other tables, these data can be used for insertions into a new table. For this, we write a query whose result is a set of tuples to be inserted. Such an insert statement has the form

insert into <table> [(<column i, . . . , column j>)] <query>

Example: Suppose we have defined the following table:

create table OLDEMP (ENO number(4) not null, HDATE date);

We now can use the table EMP to insert tuples into this new relation:

insert into OLDEMP (ENO, HDATE) select EMPNO, HIREDATE from EMP
where HIREDATE < '31-DEC-60';

There are three methods for inserting values into the table.

Case1:-

1)Inserting values for selected columns

Insert into student(name,roll,class,branch,enroll_no)
values('abc',12,'BE','computer',213123556);

Case2:-

1)Inserting values for all columns

Insert into student values('abc',12,'BE','computer',213123556);

Case3:-

1)Inserting more than one record

Insert into student values(‘&name’,roll,’&class’,’&branch’,enroll_no)

Enter value for name:smith
Enter value for roll:101
Enter value for class:t.e.
Enter value for branch:comp
Enter value for enroll_no:1234567
SQL>/
Note:-‘/’ is used to repeat the previous command.

Update command

The update command is used to chang or modify data values in a table
Updating of all rows:-

Syntax:-
Update<table_name>
Set column_name1=expression1,
column_name2=expression2,
.
.
column_name n=expression n
where<search_condition>;

An update statement without a where clause results in changing respective attributes of all tuples in the specified table. Typically, however, only a (small) portion of the table requires an update.
Examples:

- The employee JONES is transfered to the department 20 as a manager and his salary is increased by 1000:

update EMP set
JOB = ’MANAGER’, DEPTNO = 20, SAL = SAL +1000
where ENAME = ’JONES’;

- All employees working in the departments 10 and 30 get a 15% salary increase.

update EMP set
SAL = SAL ∗ 1.15 where DEPTNO in (10,30);

Analogous to the insert statement, other tables can be used to retrieve data that are used as new values. In such a case we have a <query> instead of an <expression>.

Example: All salesmen working in the department 20 get the same salary as the manager who has the lowest salary among all managers.

```
update EMP set
SAL = (select min(SAL) from EMP
where JOB = 'MANAGER')
where JOB = 'SALESMAN' and DEPTNO = 20;
```

Delete command

Delete command is used for deleting data(rows) from the table

Syntax:-

Delete from<table_name>

e.g. delete from student;

It will delete all rows from student table

Removal of specified rows:-It is used to remove specific rows from the table

Syntax:-

Delete from<table_name>
Where<search_condition>

To delete a student whose name is 'abc'

e.g. delete from student where name='abc';

Commit and Rollback

A sequence of database modifications, i.e., a sequence of insert, update, and delete statements, is called a transaction. Modifications of tuples are temporarily stored in the database system. They become permanent only after the statement commit; has been issued.As long as the user has not issued the commit statement, it is possible to undo all modifications since the last commit. To undo modifications, one has to issue the statement rollback;.It is advisable to complete each modification of the database with a commit (as long as the modification has the expected effect). Note that any data definition command such as create table results in an internal commit. A commit is also implicitly executed when the user terminates an Oracle session.

Table Alias

We have only focused on queries that refer to exactly one table. Furthermore,conditions in a where were restricted to simple comparisons. A major feature of relational databases,

however, is to combine (join) tuples stored in different tables in order to display more meaningful and complete information. In SQL the select statement is used for this kind of queries joining relations:

select [distinct] [<alias ak>.]<column i>, . . . , [<alias al>.]<column j>
from<table 1> [<alias a1>], . . . , <table n> [<alias an>]
[where<condition>]

The specification of table aliases in the from clause is necessary to refer to columns that have the same name in different tables. For example, the column DEPTNO occurs in both EMP and DEPT. If we want to refer to either of these columns in the where or select clause, a table alias has to be specified and put in the front of the column name. Instead of a table alias also the complete relation name can be put in front of the column such as DEPT.DEPTNO, but this sometimes can lead to rather lengthy query formulations.

SET operations

SQL supports few Set operations which can be performed on the table data. These are used to get meaningful results from data stored in the table, under different special conditions.

Following are 4 different types of SET operations

1. UNION

   UNION is used to combine the results of two or more `SELECT` statements. However it will eliminate duplicate rows from its resultset. In case of union, number of columns and datatype must be same in both the tables, on which UNION operation is being applied.

   select * from employee
   union
   select * from department;

2. UNION ALL

   This operation is similar to Union. But it also shows the duplicate rows.

   select * from employee
   union all
   select * from department;

3. INTERSECT

   Intersect operation is used to combine two `SELECT` statements, but it only retuns the records which are common from both `SELECT` statements. In case of **Intersect** the number of columns and datatype must be same.

   select * from employee

intersect
select * from department;

4. MINUS

The Minus operation combines results of two `SELECT` statements and return only those in the final result, which belongs to the first set of the result.

select * from employee
minus
select * from department;

Lab Exercises:-

Note : Use Primary key, foreign keys, unique, not null, null constraints whenever necessary.

1) Create table Department with fieldsdeptno, dname, location.
2) Insert the following records by using any one method

Deptno	Dname	Location
10	Accounting	Mumbai
20	Research	Pune
30	Sales	Nashik
40	Operations	Nagpur

3) List the department information.
4) Create table employee as shown below.

Empno	Ename	Job	Mgr	Joined_date	Salary	Commission	Deptno	Address
1001	Nileshjoshi	Clerk	1005	17-dec-95	2800	600	20	Nashik
1002	Avinashpawar	Salesman	1003	20-feb-96	5000	1200	30	Nagpur
1003	Amit kumar	Manager	1004	2-apr-86	2000	----	30	Pune
1004	Nitinkulkarni	President	--	19-apr-86	50000	----	10	Mumbai
1005	Niraj Sharma	Analyst	1003	3-dec-98	12000	----	20	Satara
1006	Pushkardeshpande	Salesman	1003	1-sep-96	6500	1500	30	Pune
1007	Sumitpatil	Manager	1004	1-may-91	25000	----	20	Mumbai
1008	Ravi sawant	Analyst	1007	17-nov-95	10000	----	---	Amaravati

5) Write a query to display employee information. Write a name of column explicitly.
6) Create a query to display unique jobs from the table.
7) Change the location of dept40 toBanglore instead of Nagpur.
8) Change the name of the employees 1003 to Nikhil Gosavi.

9) Delete Pushkardeshpande from employee table.
10) Create a table department_temp table from deptarment table, only create the structure not content.
11) Insert rows into department_temp table from deptarment table
12) Display the list of employee whose salary between 5000 and 20000.
13) Display the list of employee excluding job title as 'salesman'.
14) Display all those employees whose job title is either 'manager' or 'analyst'(write by using OR & IN operator).
15) Display the employee name & department number of all employees in dept 10,20,30& 40.
16) Display the employee number, name, job & commission of all employees who do not get any commission.
17) Display the name & salary of all employees whose salary not in the range of 5000 & 10000.
18) Find all names & joined date of employees whose names starts with 'A'.
19) Find all names of employees having 'i' as a second letter in their names.
20) Find employee number, name of employees whose commission is not null.
21) Display all employee information in the descending order of employee number.
22) Display the minimum, maximum, sum & average salary of each job type.
23) Write a query to display the number of employee with the same department.
24) Select employee number, ename according to the annual salary in ascending order.
25) Find the department number, maximum salary where the maximum salary is greater than 5000.
26) Find all distinct column values from employee & department table.
27) Find all column values with duplicate from employee & department table.
28) Find all column values which are common in both employee & department table.
29) Find all distinct column values present in employee but not in department table.
30) Display the number of employees in the department 30 who can earn a commission.

FAQS:-

1) What are the different agreegate functions? Explain
2) What are the different string operations? Explain
3) Explain group by, order by clause and having clause ?
4) What are the different date functions? Explain in short
5) Write a query to display the current date. Label the column **DATE**.
6) When you use IN & '=' operator.
7) Explain Set Operators.
8) How would you use the ORDER BY clause, without using the column names in the clause. Explain with example.
9) Explain DML commands in short.
10) Explain Wild Card Characters.

CONCLUSION:-

Assignment No.4

Aim: Study of SQL DML commands

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

Outcomes: Student should be able to

1. Perform different manipulation function like joins and subquery.
2. Use different kind of joins and subqueries.
3.Solve complex queries.

PEOs ,POs, PSOs and COs satisfied

PEOs : 1 POs : 1,2,5 PSOs : 1 COs :1

Software used : ORACLE

Theory : SQL – Data Manipulation Statements.

Subqueries

Up to now we have only concentrated on simple comparison conditions in a where clause, i.e.we have compared a column with a constant or we have compared two columns. As we have already seen for the 'select' statement, queries can be used for assignments to columns.

A Subquery or Inner query or a Nested query is a query within another SQL query and embedded within the WHERE clause.

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 command cannot be used in a subquery, although the main query can use an ORDER BY. The GROUP BY command 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 query result can also be used in a condition of a 'where' clause. In such a case the query is called a subquery and the complete select statement is called a nested query.

Types of subqueries

1.Single row subqueries

When a subquery returns a single value, or **exactly one row and exactly one column**, we call it a **scalar or single row subquery**. This type of subquery is frequently used in the WHERE clause to filter the results of the main query.

Example:

select *from sales_agents where agency_fee > (select avg(agency_fee) from sales_agents);

Here, your subquery calculates the average agency fee your sales team received last month and returns a single value Then you use this value to filter the results from your main query and return information for only those sales agents whose agency fee was higher than average.

Example: List all employees who are working in a department located in BOSTON:

```
select ∗ from EMP
where DEPTNO in
(select DEPTNO from DEPT
where LOC = 'BOSTON');
```

The subquery retrieves only one value (the number of the department located in Boston). Thus it is possible to use "=" instead of in. As long as the result of a subquery is not known in advance, i.e., whether it is a single value or a set, it is advisable to use the in operator. A subquery may use again a subquery in its where clause. Thus conditions can be nested arbitrarily. An important class of subqueries are those that refer to its surrounding (sub) query and the tables listed in the from clause, respectively. Such type of queries is called correlated subqueries.

2.Multiple row subqueries

If your subquery returns more than one row, it can be referred to as a **multiple-row subquery**. Note that this subquery type includes (1) subqueries that return one column with multiple rows (i.e. a list of values) and (2) subqueries that return multiple columns with multiple rows (i.e. tables).

Subqueries that return one column and multiple rows are often included in the WHERE clause to filter the results of the main query. In this case, they are usually used with

operators like IN, NOT IN, ANY, ALL, EXISTS, or NOT EXISTS that allow users to compare a particular value with the values in the list returned by the subquery.

A respective condition in the where clause then can have one of the following forms:

1.Set-valued subqueries

<expression> [not] in (<subquery>)
<expression> <comparison operator> [any|all] (<subquery>)

An <expression> can either be a column or a computed value.

2. Test for (non)existence

[not] exists (<subquery>)

In a where clause conditions using subqueries can be combined arbitrarily by using the logical connectives ‘and’ and ‘or’.

Example:
select avg(agency_fee)from sales_agents where id not in (select id from managers);

The inner query will return a list of all manager IDs. Then the outer query filters only those sales agents who **are not** in the managers list and calculates an average agency fee paid to these agents.

Example: List the name and salary of employees of the department 20 who are leading a project that started before December 31, 1990:

```
select ENAME, SAL from EMP
where EMPNO in
(select PMGR from PROJECT
where PSTART < ’31-DEC-90’)
and DEPTNO =20;
```

Explanation: The subquery retrieves the set of those employees who manage a project that started before December 31, 1990. If the employee working in department 20 is contained in this set (in operator), this tuple belongs to the query result set.
Example: List all those employees who are working in the same department as their manager(note that components in [ ] are optional:

```
select ∗ from EMP E1
where DEPTNO in
(select DEPTNO from EMP [E]
```

where [E.]EMPNO = E1.MGR);

Explanation: The subquery in this example is related to its surrounding query since it refers the column E1.MGR. A tuple is selected from the table EMP (E1) for the query result if the value for the column DEPTNO occurs in the set of values select in the subquery. One can think of the evaluation of this query as follows: For each tuple in the table E1, the subquery is evaluate individually. If the condition where DEPTNO in . . . evaluates to true, this tuple is selected Note that an alias for the table EMP in the subquery is not necessary since columns without preceding alias listed there always refer to the innermost query and tables. Conditions of the form <expression> <comparison operator> [any|all] <subquery> are use to compare a given <expression> with each value selected by <subquery>.

• For the clause any, the condition evaluates to true if there exists at least on row select by the subquery for which the comparison holds. If the subquery yields an empty result set, the condition is not satisfied.
• For the clause all, in contrast, the condition evaluates to true if for all rows selected the subquery the comparison holds. In this case the condition evaluates to true if the subquery does not yield any row or value.

Example: Retrieve all employees who are working in department 10 and who earn at least as much as any (i.e., at least one) employee working in department 30:

```
select * from EMP
where SAL >= any
(select SAL from EMP
where DEPTNO = 30)
and DEPTNO = 10;
```

Note: Also in this subquery no aliases are necessary since the columns refer to the innermost from clause.

Example: List all employees who are not working in department 30 and who earn more than all employees working in department 30:

```
select * from EMP
where SAL > all
(select SAL from EMP
where DEPTNO = 30)
and DEPTNO <> 30;
```

For all and any, the following equivalences hold:

in ⇔ = any
not in ⇔ <> all or != all

Often a query result depends on whether certain rows do (not) exist in (other) tables. Such type of queries is formulated using the 'exists' operator.

Example: List all departments that have no employees:

```
select * from DEPT
where not exists
(select * from EMP
where DEPTNO = DEPT.DEPTNO);
```

Explanation: For each tuple from the table DEPT, the condition is checked whether there exists a tuple in the table EMP that has the same department number (DEPT.DEPTNO). In case no such tuple exists, the condition is satisfied for the tuple under consideration and it is selected. If there exists a corresponding tuple in the table EMP, the tuple is not selected.

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 in the 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

Joins

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>;

Example: 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 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 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;**

Lab Exercises:-

Note : Use Primary key, foreign keys, unique, not null, null constraints whenever necessary.
Design the employee database with all constraints. Construct the following SQL queries for this relational database.

Employee(employee_name,street,city)
Works(employee_name,company_name,salary)
Company(company_name,city)
Manages(employee_name,manager_name)

1. Find the names of employees who work for First Bank Coorporation.

2. Find the names and cities of residence of all employees who work for First Bank Coorporation
3. Find the names, street addresses, and cities of residence of all employees who work for First Bank Coorporation and earn more than $10000.
4.Find all employees in the database who lives in the same cities as tha companies for which they work.
5.Find all employees in the database who lives in the same cities and on the same streets as do their manager.
6.Find all employees in the database who do not work for First Bank Coorporation
7.Find all employees in the database who earn more than each employee of Small Bank Coorporation
8.Assume that the company is may be located in several cities. Find all companies located in every city in which Small Bank Coorporation is located.
9.Find all employees who earn more than the average salary of all employees of their companies.
10.Find the company that has the most employees.
11.Find the company that has the smallest payroll.
12.Find those companies whose employees earn a higher salary, on average, than the average salary at First Bank Coorporation.
13.Modify the database so that "Jones" now lives in Newtown.
14.Give all employees of First Bank Coorporation" a 10% raise
15.Delete all tuples in the "Works" relation for employees of "Small Bank Coorporation".

FAQS:-
1)What is the use of Joins?
2)What are the types of join? Explain any one.
3)What is subquery?
4)List the types of subquery. Explain with ex.
5)What is an outer join and when it is required?
6)What is one way outer join and two way outer join?
7)Where can a join be efficient?
8)Explain self join with ex.
9)Write a query for find the names of employees who work for "FBC" by using join and subquery

CONCLUSION:-

Assignment No.5

Aim: Study of PL/SQL

Problem Statement:- 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)

1. Accept roll_no & name of book from user.
2. Check the number of days (from date of issue), if days are between 15 to 30 then fine amount will be Rs 5per day.
3.If no. of days>30, per day fine will be Rs 50 per day & for days less than 30, Rs. 5 per day.
4.After submitting the book, status will change from I to R.
5.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.

Outcomes: Student should be able to
1. Write simple PL/SQL blcok
2. Use of exception handling.

PEOs ,POs, PSOs and COs satisfied

PEOs : 1 POs : 1,2,5 PSOs : 1 COs :1

Software used : ORACLE

Theory : PL/SQL – Procedural Language/Structured Query Language

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 offers 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 offers 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 offers 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.
1) 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;
Exception
        when e_too_many_emp then….
        end;
```

Lab Exercises:-

1. Create table Borrower and Fine

Create table Borrower(Roll number, SName Varchar2(10), DateofIssue date, BookName Varchar2(15), Status varchar2(1));

Create table Fine(Roll number, CurrentDate date, Amt number);

2. Insert 5 records in Borrower table

Insertinto borrower values(1,'ajay','9-jun-17','book1','I')

3.Display structure and records of both tables

```
SQL> desc borrower
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 ROLL                                               NUMBER
 SNAME                                              VARCHAR2(10)
 DATEOFISSUE                                        DATE
 BOOKNAME                                           VARCHAR2(15)
 STATUS                                             VARCHAR2(1)

SQL> desc fine
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 ROLL                                               NUMBER
 CURRENTDATE                                        DATE
 AMT                                                NUMBER

SQL> select * from borrower;

     ROLL  SNAME      DATEOFISS  BOOKNAME  S
---------- ---------- --------- ---------- -
        1  ajay       09-JUN-17  book1     i
        2  vijay      23-NOV-18  book2     i
        3  vishal     30-DEC-19  book3     i

SQL> select * from fine;
no rows selected
```

3.Write PL/SQL block using exception handling

SQL> edit cur.sql

```
declare
e1 exception;
rollno Borrower.Roll%TYPE;
bname Borrower.BooKName%TYPE;
```

```
B_Issuse_Date date;
Days number(4);
fine_amt number(10);
begin
rollno:=&rollno;
bname:='&bname';
if(SQL%notfound) then
raise e1;
end if;
Select DateofIssue into B_Issuse_Date from Borrower where Roll = rollno and BookName =
bname;
Days := SYSDATE - B_Issuse_Date;
DBMS_OUTPUT.PUT_LINE ('Total Days :'||Days);
if(Days<=15)then
fine_amt:=0;
DBMS_OUTPUT.PUT_LINE ('No fine.....');
elsif (Days between 16 and 30)then
fine_amt:=(days-15)*5;
insert into Fine values(rollno, SYSDATE, fine_amt);
else
fine_amt:=(Days-30)*50+(15*5);
insert into Fine values(rollno,SYSDATE,fine_amt);
end if;
UPDATE Borrower SET Staus = 'R' WHERE Roll=rollno;
exception
when e1 then
DBMS_OUTPUT.PUT_LINE('not found');
end;
/
```

OUTPUT

```
SQL> @cur (command to run PL/SQL block)
Enter value for rollno: 1
old  10: rollno:=&rollno;
new  10: rollno:=1;
Enter value for bname: book1
old  11: bname:='&bname';
new  11: bname:='book1';

PL/SQL procedure successfully completed.

SQL> select * from fine;
```

```
     ROLL CURRENTDA       AMT
---------- --------- ----------
         1 28-SEP-20       58925
```

```
SQL> @cur
Enter value for rollno: 4
old  10: rollno:=&rollno;
new  10: rollno:=4;
Enter value for bname: book4
old  11: bname:='&bname';
new  11: bname:='book4';
declare
*
ERROR at line 1:
ORA-01403: no data found
ORA-06512: at line 15
```

FAQS:-

1)What is PL/SQL? Explain.
2)What is the difference between "SQL" and "PL/SQL"?
3)What are exceptions? What are the different types of exceptions?
4)What are the different conditional statements used in PL/SQL?
5)What are the different iterative construct used in PL/SQL? Explain in short.
6)Write a PL/SQL block to calculate factorial. Use Exception Handling.
7)Write a PL/SQL block to calculate factorial. Use Exception Handling.
8)Write a PL/SQL block to find the grade of a student. Enter marks for 5 subjects.
9)Write a PL/SQL block to update the table.

ACCT_MSTR

Table name:-ACCT_MSTR

ACCT_NO	CURBAL
SB1	500
SB5	500
SB9	500
SB13	500

CONCLUSION:-

Assignment No.6

Aim: Study of PL/SQL CURSOR

Problem Statement:- 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_RollCall with the data available in the table O_RollCall. 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

Outcomes: Student should be able to

1. Understand cursor concept
2. Write all types of cursors

PEOs ,POs, PSOs and COs satisfied

PEOs : 1 POs : 1,2,5 PSOs : 1 COs :1

Software used : ORACLE

Theory : PL/SQL – Procedural Language/Structured Query Language

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 to perform a single fetch.

This additional fetch is usually not noticeable, and you shouldn't be neurotic about using an implicit cursor 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 with an implicit query. If you have 25 users on your system all performing a similar query, your server must process 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 and go with the explicit cursor.

NOTE: In PL/SQL Release 2.3 and above, the implicit cursor has been optimized so that it may, in isolation, run faster than the corresponding explicit cursor. Generally, the differences between these two approaches from a performance standpoint are negligible. On the other hand, if you use an explicit cursor, you are more likely (or at least *able*) to reuse that cursor, which increases the chance that it will be pre-parsed in shared memory when needed -- thereby improving the performance of your application as a whole.

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 the exception section. Unless you deliberately plan to handle this scenario, use of the implicit cursor is a declaration 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 of the data ever changes, however, you may find that the SELECT statement which formerly identified a single 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 raising exceptions.

Diminished programmatic control

The implicit cursor version of a SELECT statement is a black box. You pass the SQL statement to the SQL layer in the database and it returns (you hope) a single row. You can't get inside the separate operations of the cursor, such as the open and close stages. You can't examine the attributes of the cursor -- to see whether a row was found, for example, or if the cursor has already been opened. You can't easily apply traditional programming control constructs, such as an IF statement, to your data access.

Sometimes you don't need this level of control. Sometimes you just think you don't need this level of control. I have found that if I am going to build programs in PL/SQL, I want as much control as I can possibly get.

Always Use Explicit Cursors!

My rule of thumb is always to use an explicit cursor for all SELECT statements in my applications, even if an implicit cursor might run a *little* bit faster and even if, by coding an explicit cursor, I have to write more code (declaration, open, fetch, close).

By setting and following this clear-cut rule, I give myself one less thing to think about. I do not have to determine if a particular SELECT statement will return only one row and therefore be a candidate for an implicit cursor. I do not have to wonder about the conditions under which a single-row query might suddenly return more than one row, thus requiring a TOO_MANY_ROWS exception handler. I am guaranteed to get vastly improved programmatic control over that data access and more finely-tuned exception handling for the cursor.

Lab Exercises:-

1. Create table Cust_old and cust_new

```
Create table Cust_New(ID number, Name Varchar2(10), City
                      Varchar2(10), Salary number);

Create table Cust_Old(ID number, Name Varchar2(10), City
                      Varchar2(10), Salary number);
```

2. Insert records

```
insert into Cust_New Values ( 1,'Ajay', 'Pune', 20000);
insert into Cust_New Values ( 2,'Ramesh','Pune', 15000);
insert into Cust_New Values ( 3,'Umesh', 'Pune', 40000);
insert into Cust_New Values ( 4,'Ram', 'Pune', 25000);

insert into Cust_Old Values ( 1,'Ramesh','Pune', 15000);
insert into Cust_Old Values ( 5,'Sunil', 'Pune', 45000);
```

3.Display structure and records of both tables

```
SQL> desc cust_old;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 ID                                                 NUMBER
 NAME                                               VARCHAR2(10)
 CITY                                               VARCHAR2(10)
 SALARY                                             NUMBER
```

```
SQL> desc cust_new;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 ID                                                 NUMBER
 NAME                                               VARCHAR2(10)
 CITY                                               VARCHAR2(10)
 SALARY                                             NUMBER

SQL> select * from cust_old;

        ID NAME       CITY          SALARY
---------- ---------- ---------- ----------
         1 Ramesh     Pune           15000
         5 Sunil      Pune           45000

SQL> select * from cust_new;

        ID NAME       CITY          SALARY
---------- ---------- ---------- ----------
         1 Ajay       Pune           20000
         2 Ramesh     Pune           15000
         3 Umesh      Pune           40000
         4 Ram        Pune           25000
```

3.Write PL/SQL block

```
SQL> edit cur.sql
--implicit cursor
declare
no cust_new.id%type;
totrecord number;
begin
no:=&no;
select id into no from cust_new where id=no;
if(SQL%found) then
totrecord:=sql%rowcount;
end if;
end;
/
SQL> set serveroutput on;
SQL> edit cur.sql
```

OUTPUT

```
SQL> @cur
Enter value for no: 1
old   5: no:=&no;
new   5: no:=1;
total records=1

PL/SQL procedure successfully completed.
```

--explicit cursor

```
declare
cursor c1 is select * from cust_new;
c1_id cust_new.id%type;
c1_name cust_new.name%type;
c1_city cust_new.city%type;
c1_sal cust_new.salary%type;
begin
open c1;
loop
fetch c1 into c1_id,c1_name,c1_city,c1_sal;
exit when c1%notfound;
dbms_output.put_line(c1_id||' '||c1_name||' '||c1_city||''||c1_sal);
end loop;
close c1;
end;
/
```

OUTPUT

```
SQL> @cur
1 Ajay Pune20000
2 Ramesh Pune15000
3 Umesh Pune40000
4 Ram Pune25000

PL/SQL procedure successfully completed.
```

--cursor for loop

```
declare
cursor fc is select * from Cust_New where salary>=25000;
tmp fc%rowtype;
begin
dbms_output.put_line('ID Name City Salary');
for tmp in fc
loop
dbms_output.put_line(tmp.id || ' ' || tmp.name || ' ' || tmp.city || ' ' || tmp.salary);
end loop;
end;
/
```

OUTPUT

```
SQL> @cur
ID Name City Salary
3 Umesh Pune 40000
4 Ram Pune 25000

PL/SQL procedure successfully completed.
```

--parameterized

```
declare
cursor pc(c_id number) is SELECT * FROM Cust_New where id=c_id;
tmp pc%rowtype;
begin
dbms_output.put_line('ID Name City Salary');
for tmp in pc(2)
LOOP
dbms_output.put_line(tmp.id || ' ' || tmp.name || ' ' || tmp.city || ' ' || tmp.salary);
END LOOP;
END;
/
```

OUTPUT

```
SQL> @cur
ID Name City Salary
2 Ramesh Pune 15000

PL/SQL procedure successfully completed.
```

--Merge Table using Parameterized Cursor

```
DECLARE
   CURSOR PARAM_CURSOR(c_id number) is SELECT * FROM Cust_Old where
ID=c_id;
   cur_cust_old PARAM_CURSOR%rowtype; flag
   number;
BEGIN
  flag:=0;

  FOR v_cust_new IN (SELECT id, name, city, salary from cust_new)
    LOOP
      FOR cur_cust_old IN PARAM_CURSOR(v_cust_new.id)
        LOOP
          update cust_old set name = v_cust_new.name, city = v_cust_new.city,
salary=v_cust_new.salary where id = v_cust_new.id;
           flag:=1;
        END LOOP;

        If flag=0 Then
          insert into cust_old values (v_cust_new.id, v_cust_new.name,
                                              v_cust_new.city,v_cust_new.salary);
        end if;

        flag:=0;
     END LOOP;
END;
/
```

OUTPUT

```
SQL> @cur

PL/SQL procedure successfully completed.

SQL> select * from cust_old;

        ID NAME       CITY          SALARY
---------- ---------- ---------- ----------
         1 Ajay         Pune          20000
         5 Sunil        Pune          45000
         2 Ramesh       Pune          15000
         3 Umesh        Pune          40000
         4 Ram          Pune          25000

SQL> select * from cust_new;

        ID NAME       CITY          SALARY
---------- ---------- ---------- ----------
         1 Ajay         Pune          20000
         2 Ramesh       Pune          15000
         3 Umesh        Pune          40000
         4 Ram          Pune          25000
```

FAQS:-

1) What is cursor?
2) What are the different types of cursors?
3) What are the different attributes of explicit cursor? Explain in brief.
4) What is Implicit cursor?
5) Explain the FOR loop of Cursor.
6) What is difference between simple loop, while loop & for loop?
7) What is difference between Implicit & Explicit Cursor?
8) Explain FOR UPDATE cursor with an example.
9) What is CURRENT OF clause in cursor? Give an example.
10) List all predefined cursor.

CONCLUSION:-

Assignment No.7

Aim: PL/SQL Stored Procedure and Stored Function.

Problem Statement:- Write a Stored Procedure namely proc_Grade for the categorization of student. If marks scored by students in examination is • <=1500 and marks>=990 then student will be placed in Distinction category • if marks scored are between 989 and 900 category is First Class, • if marks 899 and 825 category is Higher Second Class. Write a PL/SQL block for using procedure created with above requirement. 1. Stud_Marks(Roll, Name, Total_marks) 2. Result(Roll, Name, Class) Frame the separate problem statement for writing PL/SQL Stored Procedure and function, inline with above statement. The problem statement should clearly state the requirements.

Outcomes: Student should be able to
1. Understand concept of Procedure and Functions
2. Write procedure and functions

PEOs ,POs, PSOs and COs satisfied
PEOs : 1 POs : 1,2,5 PSOs : 1 COs :1

Software used : ORACLE

Theory : PL/SQL – Procedural Language/Structured Query Language

PROCEDURE
A **subprogram** is a program unit/module that performs a particular task. These subprograms are combined to form larger programs. This is basically called the 'Modular design'. A subprogram can be invoked by another subprogram or program which is called the **calling program**.

PL/SQL subprograms are named PL/SQL blocks that can be invoked with a set of parameters.

PL/SQL provides two kinds of subprograms −

- **Functions** − These subprograms return a single value; mainly used to compute and return a value.
- **Procedures** − These subprograms do not return a value directly; mainly used to perform an action.

A Procedure is a subprogram unit that consists of a group of PL/SQL statements. Each procedure in Oracle has its own unique name by which it can be referred. This subprogram unit is stored as a database object. Below are the characteristics of this subprogram unit.

- Procedures are standalone blocks of a program that can be stored in the database.
- Call to these procedures can be made by referring to their name, to execute the PL/SQL statements.
- It is mainly used to execute a process in PL/SQL.
- It can have nested blocks, or it can be defined and nested inside the other blocks or packages.
- It contains declaration part (optional), execution part, exception handling part (optional).
- The values can be passed into the procedure or fetched from the procedure through parameters.

- These parameters should be included in the calling statement.
- Procedure can have a RETURN statement to return the control to the calling block, but it cannot return any values through the RETURN statement.
- Procedures cannot be called directly from SELECT statements. They can be called from another block or through EXEC keyword.

Syntax to create procedure

```
CREATE OR REPLACE PROCEDURE
<procedure_name>
        (
        <parameterl IN/OUT <datatype>
        ..
        .
        )[ IS | AS ]<declaration_part>
BEGIN
        <execution part>
EXCEPTION
        <exception handling part>
END;
```

[Parameter]

The parameter is variable or placeholder of any valid PL/SQL datatype through which the PL/SQL subprogram exchange the values with the main code. This parameter allows to give input to the subprograms and to extract from these subprograms.

- These parameters should be defined along with the subprograms at the time of creation.
- These parameters are included n the calling statement of these subprograms to interact the values with the subprograms.
- The datatype of the parameter in the subprogram and the calling statement should be same.
- The size of the datatype should not mention at the time of parameter declaration, as the size is dynamic for this type.

Based on their purpose parameters are classified as

1. IN Parameter
2. OUT Parameter
3. IN OUT Parameter

IN Parameter:

- This parameter is used for giving input to the subprograms.
- It is a read-only variable inside the subprograms. Their values cannot be changed inside the subprogram.
- In the calling statement, these parameters can be a variable or a literal value or an expression, for example, it could be the arithmetic expression like '5*8' or 'a/b' where 'a' and 'b' are variables.
- By default, the parameters are of IN type.

OUT Parameter:

- This parameter is used for getting output from the subprograms.

- It is a read-write variable inside the subprograms. Their values can be changed inside the subprograms.
- In the calling statement, these parameters should always be a variable to hold the value from the current subprograms.

IN OUT Parameter:

- This parameter is used for both giving input and for getting output from the subprograms.
- It is a read-write variable inside the subprograms. Their values can be changed inside the subprograms.
- In the calling statement, these parameters should always be a variable to hold the value from the subprograms.

These parameter type should be mentioned at the time of creating the subprograms.

1)Example:-Simple procedure to display hello world message

```
CREATE OR REPLACE PROCEDURE greetings
AS
BEGIN
  dbms_output.put_line('Hello World!');
END;
/
```

OUTPUT:-SQL> @proc (command to execute procedure)

Procedure created.

SQL> set serveroutput on;

PL/SQL block to call procedure

SQL> begin

2 greetings();

3 end;

4 /

Hello World!

PL/SQL procedure successfully completed.

2)Example:-Procedure to perform addition of two numbers

SQL> edit proc.sql

create or replace procedure a1(a in number,b in number,c out number)as

begin

```
C:=a+b;
end a1;
/
SQL> @proc
Procedure created.
SQL> edit a.sql
declare
x number;
y number;
z number;
begin
x:=&x;
y:=&y;
a1(x,y,z);
dbms_output.put_line('addition of two number='||z);
end;
/
SQL> @a
Enter value for x: 23
old   6: x:=&x;
new   6: x:=23;
Enter value for y: 34
old   7: y:=&y;
new   7: y:=34;
addition of two number=57
```

PL/SQL procedure successfully completed.

FUNCTIONS

Functions is a standalone PL/SQL subprogram. Like PL/SQL procedure, functions have a unique name by which it can be referred. These are stored as PL/SQL database objects. Below are some of the characteristics of functions.

- Functions are a standalone block that is mainly used for calculation purpose.
- Function use RETURN keyword to return the value, and the datatype of this is defined at the time of creation.
- A Function should either return a value or raise the exception, i.e. return is mandatory in functions.
- Function with no DML statements can be directly called in SELECT query whereas the function with DML operation can only be called from other PL/SQL blocks.
- It can have nested blocks, or it can be defined and nested inside the other blocks or packages.
- It contains declaration part (optional), execution part, exception handling part (optional).
- The values can be passed into the function or fetched from the procedure through the parameters.
- These parameters should be included in the calling statement.
- Function can also return the value through OUT parameters other than using RETURN.
- Since it will always return the value, in calling statement it always accompanies with assignment operator to populate the variables.

Syntax

REATE OR REPLACE FUNCTION
<procedure_name>
(
<parameterl IN/OUT <datatype>
)
RETURN <datatype>
[IS | AS]
<declaration_part>
BEGIN
<execution part>
EXCEPTION
<exception handling part>
END;

RETURN

RETURN is the keyword that instructs the compiler to switch the control from the subprogram to the calling statement. In subprogram RETURN simply means that the control needs to exit from the subprogram. Once the controller finds RETURN keyword in the subprogram, the code after this will be skipped.

Normally, parent or main block will call the subprograms, and then the control will shift from those parent block to the called subprograms. RETURN in the subprogram will return the control back to their parent block. In the case of functions RETURN statement also returns the value. The datatype of this value is always mentioned at the time of function declaration. The datatype can be of any valid PL/SQL data type.

Example:-Function to add two numbers

```
SQL> edit fun.sql
create or replace function f1(a in number,b in number)return number is
c number;
begin
C:=a+b;
return c;
end f1;
/

SQL> @fun

Function created.
```

PL/SQL block to call function

```
SQL> edit f1.sql

declare
x number;
y number;
z number;
begin
x:=&x;
y:=&y;
z:=a1(x,y);
dbms_output.put_line('addition of two number='||z);
end;


SQL> @f1
Enter value for x: 45
old   6: x:=&x;
new   6: x:=45;
Enter value for y: 67
old   7: y:=&y;
new   7: y:=67;
addition of two number=112

PL/SQL procedure successfully completed.
```

Lab Exercises:-

1.Create table stud_marks and result

Tables : 1. Stud_Marks(Roll, Name, Total_marks) 2. Result(Roll, Name, Class)

Create table Stud_Marks (Roll number Primary Key, Name Varchar2(10), Total_marks number);

Create table Result (Roll number references Stud_Marks(Roll), Name Varchar2(10), Class Varchar2(20));

2.Insert data

```
insert into Stud_Marks Values ( 1, 'Ajay', 1100 );
insert into Stud_Marks Values ( 2, 'Ramesh', 950);
insert into Stud_Marks Values ( 3, 'Umesh', 850);
insert into Stud_Marks Values ( 4, 'Suresh', 400);
```

```
SQL> select * from stud_marks;

     ROLL NAME       TOTAL_MARKS
---------- ---------- -----------
         1 Ajay              1100
         2 Ramesh            950
         3 Umesh             850
         4 Suresh            400
```

```
SQL> select * from result;

no rows selected
```

PL/SQL procedure

```
SQL> edit proc1.sql
```

```
create or replace procedure proc_grade(rno in stud_marks.roll%type, name1 in stud_marks.name%type) is
  totmarks stud_marks.total_marks%type;
  grade varchar2(15);
  begin
  select total_marks into totmarks from stud_marks where roll=rno and name=name1;
  if(totmarks between 825 and 899) then
    grade:='secondclass';
    insert into result values(rno,name1,grade);
    elsif(totmarks between 900 and 989) then
      grade:='Firstclass';
      insert into result values(rno,name1,grade);
    elsif(totmarks between 990 and 1500)then
       grade:='Distinction';
       insert into result values(rno,name1,grade);
   else
      dbms_output.put_line('Fail');
   end if;
   end proc_grade;
/
```

```
SQL> @proc1
```

Procedure created.

PL/SQL code block to call procedure

SQL> edit p1.sql

```
declare
   result varchar2(15);
   totmarks number;
   begin
   proc_grade(1,'Ajay');
   end;
/
```

SQL> @p1

PL/SQL procedure successfully completed.

SQL> select * from result;

```
     ROLL NAME       CLASS
---------- ---------- --------------------
         1 Ajay       Distinction
```

FAQS:-

10) What is Subprogram?
11) What is procedure ?
12) What is function?.
13) Explain the different parameters used in procedure or function?
14) Write syntax to create procedure and function.

CONCLUSION:-

Assignment No.8

Aim: PL/SQL Trigger

Problem Statement:- Database Trigger (All Types: Row level and Statement level triggers, Before and After Triggers). Write a database trigger on Library table. The System should keep track of the
records that are being updated or deleted. The old value of updated or deleted records should be added in Library_ Audit table. Frame the problem statement for writing Database Triggers of all types, in-line with above statement. The problem statement should clearly state the requirements.

Outcomes: Student should be able to
1. Understand concept of Trigger
2. Write Row level and statement level Trigger.
3. Understand before and after Trigger

PEOs ,POs, PSOs and COs satisfied
PEOs : 1 POs : 3,4,5 PSOs : 1 COs :2

Software used : ORACLE

Theory : PL/SQL – Trigger

A **trigger** is a special type of stored procedure that automatically runs when an event occurs in the database server. DML **triggers** run when a user tries to modify data through a data manipulation language (DML) event. DML events are INSERT, UPDATE, or DELETE statements on a table or view.

Triggers can be defined on the table, view, schema, or database with which the event is associated.

Benefits of Triggers

Triggers can be written for the following purposes −

- Generating some derived column values automatically
- Enforcing referential integrity
- Event logging and storing information on table access
- Auditing
- Synchronous replication of tables
- Imposing security authorizations
- Preventing invalid transactions

The syntax for creating a trigger is −

```
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;
```

Where,

- CREATE [OR REPLACE] TRIGGER trigger_name − Creates or replaces an existing trigger with the *trigger_name*.
- {BEFORE | AFTER | INSTEAD OF} − This specifies when the trigger will be executed. The INSTEAD OF clause is used for creating trigger on a view.
- {INSERT [OR] | UPDATE [OR] | DELETE} − This specifies the DML operation.
- [OF col_name] − This specifies the column name that will be updated.
- [ON table_name] − This specifies the name of the table associated with the trigger.
- [REFERENCING OLD AS o NEW AS n] − This allows you to refer new and old values for various DML statements, such as INSERT, UPDATE, and DELETE.
- [FOR EACH ROW] − This specifies a row-level trigger, i.e., the trigger will 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

BEFORE and AFTER of Trigger:
BEFORE triggers run the trigger action before the triggering statement is run.
AFTER triggers run the trigger action after the triggering statement is run.

Example:
Given Student Report Database, in which student marks assessment is recorded. In such schema, create a trigger so that the total and average of specified marks is automatically inserted whenever a record is insert.

BEFORE and AFTER of Trigger:
BEFORE triggers run the trigger action before the triggering statement is run.
AFTER triggers run the trigger action after the triggering statement is run.

Lab Exercise

SQL> Create table LIBRARY (BOOK_ID number, BOOK_NAME Varchar2(10),
BOOK_PRICE number);

Table created.

SQL> desc library;

Name	Null?	Type
BOOK_ID		NUMBER
BOOK_NAME		VARCHAR2(10)
BOOK_PRICE		NUMBER

```
SQL> Create table LIBRARY_AUDIT_ROW (BOOK_ID number, BOOK_NAME
Varchar2(10), BOOK_PRICE number,AUDIT_CHANGE Varchar2(20), UPDATE_TIME Date);

Table created.

SQL> desc library_audit_row;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 BOOK_ID                                            NUMBER
 BOOK_NAME                                          VARCHAR2(10)
 BOOK_PRICE                                         NUMBER
 AUDIT_CHANGE                                       VARCHAR2(20)
 UPDATE_TIME                                        DATE


SQL> Insert into LIBRARY Values ( 1, 'BOOK1', 500 );

1 row created.

SQL> Insert into LIBRARY Values ( 2, 'BOOK2', 950 );

1 row created.

SQL> Insert into LIBRARY Values ( 3, 'BOOK3',850);

1 row created.

SQL> Insert into LIBRARY Values ( 4, 'BOOK4', 400);

1 row created.

SQL> select * from library;

   BOOK_ID BOOK_NAME  BOOK_PRICE
---------- ---------- ----------
         1 BOOK1             500
         2 BOOK2             950
         3 BOOK3             850
         4 BOOK4             400

SQL> select * from library_audit_row;

no rows selected

SQL> edit trig.sql

Create or replace trigger trig_Row after delete or update on library for each row
declare
au_ch varchar(20);
begin
if deleting then
```

```
au_ch:='DELETE';
end if;
if updating then
au_ch:='UPDATE';
end if;
insert into library_audit_row
values(:old.BOOK_ID,:old.BOOK_NAME,:old.BOOK_PRICE,au_ch,current_timestamp);
end;

/
SQL> @trig

Trigger created.

SQL> update library set book_name='SQL' where book_id=1;

1 row updated.

SQL> select * from library_audit_row;

   BOOK_ID BOOK_NAME  BOOK_PRICE AUDIT_CHANGE        UPDATE_TI
---------- ---------- ---------- -------------------- ---------
         1 BOOK1             500 UPDATE               28-SEP-20


SQL> delete from library where book_id=1;

1 row deleted.

SQL> select * from library_audit_row;

   BOOK_ID BOOK_NAME  BOOK_PRICE AUDIT_CHANGE        UPDATE_TI
---------- ---------- ---------- -------------------- ---------
         1 BOOK1             500 UPDATE               28-SEP-20
         1 SQL               500 DELETE               28-SEP-20
Statement level trigger
SQL> desc library_audit_stmt;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 AUDIT_HISTORY_ID                                   NUMBER
 AUDIT_CHANGE                                       VARCHAR2(20)
 USER_NAME                                          VARCHAR2(10)
 UPDATE_TIME                                        DATE

SQL> create sequence seq_id start with 1;

Sequence created.

create or replace trigger trigs before delete or update on library
declare
au_ch varchar(20);
begin
```

```
if deleting then
au_ch:='delete';
end if;
if updating then
au_ch:='update';
end if;
insert into LIBRARY_AUDIT_STMT VALUES (Seq_id.nextval, au_ch, USER,
CURRENT_TIMESTAMP);
end;
/
```

```
SQL> @trig2

Trigger created.

SQL> delete from library where book_id=3;

1 row deleted.

SQL> select * from library_audit_stmt;

AUDIT_HISTORY_ID AUDIT_CHANGE        USER_NAME  UPDATE_TI
---------------- -------------------- ---------- ---------
               1 update               VND        28-SEP-20
               2 delete               VND        28-SEP-20
```

FAQS:-

15) What is Trigger?
16) List Types of Trigger ?
17) Explain row level Trigger with example.
18) Explain Statement level trigger with example.
19) Explain the term before and after.

CONCLUSION:-

Assignment No.9

Aim: Large Scale Databases(MONGODB)

Problem Statement:- Study of Open Source NOSQL Database: MongoDB (Installation, Basic CRUD operations, Execution)

Outcomes: Student should be able to
1. Understand concept of Large scale database
2. Concept of unstructured data
3. Able to install MONGODB

PEOs ,POs, PSOs and COs satisfied
PEOs : 2 POs : 1,3,4,5 PSOs : 2 COs :2

Software used : MONGODB(NOSQL)

Theory : A NoSQL (originally referring to "non SQL" or "non relational") database provides a mechanism for storage and retrieval of data that is modeled in means other than the tabular relations used in relational databases. Such databases have existed since the late 1960s, but did not obtain the "NoSQL" moniker until a surge of popularity in the early twenty-first century, triggered by the needs of Web 2.0 companies such as Facebook, Google, and Amazon.com. NoSQL databases are increasingly used in big data and real-time web applications.

NoSQL systems are also sometimes called "Not only SQL" to emphasize that they may support SQL-like query languages. Motivations for this approach include: simplicity of design, simpler "horizontal" scaling to clusters of machines (which is a problem for relational databases),[2] and finer control over availability. The data structures used by NoSQL databases (e.g. key-value, wide column, graph, or document) are different from those used by default in relational databases, making some operations faster in NoSQL.

The particular suitability of a given NoSQL database depends on the problem it must solve. Sometimes the data structures used by NoSQL databases are also viewed as "more flexible" than relational database tables. Many NoSQL stores compromise consistency (in the sense of the CAP theorem) in favor of availability, partition tolerance, and speed. Barriers to the greater adoption of NoSQL stores include the use of low-level query languages (instead of SQL, for instance the lack of ability to perform ad-hoc joins across tables), lack of standardized interfaces, and huge previous investments in existing relational databases. Most NoSQL stores lack true ACID transactions, although a few databases, such as MarkLogic, Aerospike, FairCom c-treeACE, Google Spanner (though technically a NewSQL database), Symas LMDB, and OrientDB have made them central to their designs. (See ACID and join support.)

MongoDB is a NoSQL database that offers a high performance, high availability, and automatic scaling enterprise database. MongoDB is a NoSQL database, so you can't use SQL (Structured Query Language) to insert and retrieve data, and it does not store data in tables like MySQL or Postgres. Data is stored in a "document" structure in JSON format (in MongoDB called BSON). MongoDB was first introduced in 2009 and is currently developed by the company MongoDB Inc.

MongoDB MongoDB is a cross-platform, document oriented database that provides, high performance, high availability, and easy scalability. MongoDB works on concept of collection and document.

Database

Database is a physical container for collections. Each database gets its own set of files on the file system. A single MongoDB server typically has multiple databases.

Collection

Collection is a group of MongoDB documents. It is the equivalent of an RDBMS table. A collection exists within a single database. Collections do not enforce a schema. Documents within a collection can have different fields. Typically, all documents in a collection are of similar or related purpose.

Document

A document is a set of key-value pairs. Documents have dynamic schema. Dynamic schema means that documents in the same collection do not need to have the same set of fields or structure, and common fields in a collection's documents may hold different types of data.

Installation of MONGODB using WINDOWs

Download and installation of latest Mongodb 3.6 release
1. https://mongodb.com
2. Click on download
3. Select community server
4. Select Operating system
5. Download MSI file
6. Start installation
7. Select complete
8. Click on finish
9.Open C drive
10.Select program files
11.Select mongodb folder
12.Select bin
13.Open command prompt and copy that path
14.on command prompt type
Cd c:\programfiles\mongodb\server\3.6\bin
15.Type mongodb –dbpath d:/mongodb\data1db1
16.Open new terminal of command prompt and type mongo

For other Operating system
1. Copy to Download

[root@localhost Downloads]# tar -xvzf mongodb-linux-x86_64-3.4.9.tgz
[root@localhost Downloads]# cd mongodb-linux-x86_64-3.4.9/
[root@localhost mongodb-linux-x86_64-3.4.9]# cd bin
[root@localhost bin]# ./mongod -dbpath /home/admin/
[root@localhost bin]# ./mongo

CRUD Operations in MongoDB

CRUD operations refer to the basic Insert, Read, Update and Delete operations. Now, let us learn how to perform CRUD (Create/Read/Update/Delete) operations in MongoDB.

Create Operations Create or insert operations add new documents to a collection. If the collection does not currently exist, insert operations will create the collection.

MongoDB provides the following methods to insert documents into a collection:

- db.collection.insertOne() New in version 3.2
- db.collection.insertMany() New in version 3.2

In MongoDB, insert operations target a single collection. All write operations in MongoDB are atomic on the level of a single document.

The use Command MongoDB use DATABASE_NAME is used to create database.

The command will create a new database if it doesn't exist, otherwise it will return the existing database.

Basic syntax of use DATABASE statement is as follows:

use DATABASE_NAME

Example If you want to create a database with name , then use DATABASE statement would be as follows:

>use mydb

switched to db mydb

To check your currently selected database, use the command

db >db mydb

If you want to check your databases list, use the command show dbs.

>show dbs

local 0.78125GB

test 0.23012GB

Your created database (mydb) is not present in list.

To display database, you need to insert at least one document into it. >db.movie.insert({"name":"tutorials point"})

In MongoDB default database is test. If you didn't create any database, then collections will be stored in test database.

The dropDatabase() Method

MongoDB db.dropDatabase() command is used to drop a existing database.

Syntax db.dropDatabase()

The createCollection() Method

MongoDB db.createCollection(name, options) is used to create collection.

Syntax

db.createCollection(name, options)

In the command, name is name of collection to be created. Options is a document and is used to specify configuration of collection.

Read Operations

Read operations retrieve documents from a collection; i.e. queries a collection for documents. MongoDB provides the following methods to read documents from a collection:

- db.collection.find()

You can specify query filters or criteria that identify the documents to return.

For examples:

- Query Documents
- Query on Embedded/Nested Documents
- Query an Array
- Query an Array of Embedded Documents

Update Operations

Update operations modify existing documents in a collection. MongoDB provides the following methods to update documents of a collection:

- db.collection.updateOne() New in version 3.2
- db.collection.updateMany() New in version 3.2
- db.collection.replaceOne() New in version 3.2

In MongoDB, update operations target a single collection. All write operations in MongoDB are atomic on the level of a single document. You can specify criteria, or filters, that identify the documents to update. These filters use the same syntax as read operations.

Delete Operations

Delete operations remove documents from a collection. MongoDB provides the following methods to delete documents of a collection:

- db.collection.deleteOne() New in version 3.2
- db.collection.deleteMany() New in version 3.2

In MongoDB, delete operations target a single collection. All write operations in MongoDB are atomic on the level of a single document. You can specify criteria, or filters, that identify the documents to remove. These filters use the same syntax as read operations.

FAQS:-

1. What makes MongoDB the best?
2. If you remove an object attribute, is it deleted from the database? Explain with example.
3. How does MongoDB provide consistency?
4. Define MongoDB.
5. What are the key features of mongodb?
6. Which command is use to create database? Explain with example
7. Which command is use to drop database? Explain with example
8. What is the use of pretty() method? Explain with example
9. Which method is used to remove the document form the collection? Explain with example

CONCLUSION:-

Assignment No.10

Aim: Mongodb Queries using CURD Operation

Problem Statement:- Design and Develop MongoDB Queries using CRUD operations. (Use CRUD operations, SAVE method, logical operators)

Outcomes: Student should be able to
1. Understand concept of CURD operation
2. Concept of logical operators
3. Able to understand SAVE method

PEOs ,POs, PSOs and COs satisfied
PEOs : 2 POs : 1,3,4,5 PSOs : 2 COs :2

Software used : Mongodb

Theory : CURD operation

Advantages of MongoDB over RDBMS
- Schema less : MongoDB is document database in which one collection holds different
- different documents.
- Number of fields, content and size of the document can be differ from one document to
- another.
- Structure of a single object is clear
- No complex joins
- Deep query-ability. MongoDB supports dynamic queries on documents using a documentbased query language that's nearly as powerful as SQL
- Tuning
- Ease of scale-out: MongoDB is easy to scale
- Conversion / mapping of application objects to database objects not needed
- Uses internal memory for storing the (windowed) working set, enabling faster access of data.

Why should use MongoDB
- Document Oriented Storage : Data is stored in the form of JSON style documents
- Index on any attribute
- Replication & High Availability
- Auto-Sharding
- Rich Queries
- Fast In-Place Updates
- Professional Support By MongoDB

Where should use MongoDB?
- Big Data
- Content Management and Delivery
- Mobile and Social Infrastructure
- User Data Management
- Data Hub

MongoDB Save() Method
The save() method replaces the existing document with the new document passed in save() method
Syntax
>db.COLLECTION_NAME.save({_id:ObjectId(),NEW_DATA})

MongoDB Delete Document
The remove() Method

- MongoDB's remove() method is used to remove document from the collection.
- remove() method accepts two parameters. One is deletion criteria and second is justOne flag

1. deletion criteria : (Optional) deletion criteria according to documents will be removed.
2. justOne : (Optional) if set to true or 1, then remove only one document.

Syntax:
>db.COLLECTION_NAME.remove(DELLETION_CRITTERIA)

Following example will remove all the documents whose title is 'MongoDB Overview'
>db.mycol.remove({'title':'MongoDB Overview'})

Remove only one
If there are multiple records and you want to delete only first record, then set justOne parameter in remove() method
>db.COLLECTION_NAME.remove(DELETION_CRITERIA,1)

Remove All documents
If you don't specify deletion criteria, then mongodb will delete whole documents from the collection. This is equivalent of SQL's truncate command.

>db.mycol.remove()

Create Database : db; // Shows By Default Database Name " Test "
use College; // Create "College" Database
db; // Shows Current Database Name

Drop Database
db.dropDatabase( ) ;

Create Collection :
db.createCollection("stud");

Drop Collection :
db.stud.drop();

Insert Data Operations
db.stud.insert({roll:1, name:"Raj", age:22,});
db.stud.insert({roll:2, name:"Sagar", age:20});
db.stud.insert({roll:3, name:"Ajay", age:19});
db.stud.insert({roll:4, name:"Manoj", age:18});
db.stud.insert({roll:5, name:"Pavan", age:25});
db.stud.insert({roll:6, name:"Nilesh", age:20});

```
db.stud.insert({roll:7, name:"Kiran", age:23});
db.stud.insert({roll:8, name:"Abhi", age:24});
db.stud.insert({roll:9, name:"Madhav",age:18});
db.stud.insert({roll:10, name:"Sunil", age:17});
```

Display Data : find();

```
db.stud.find();
db.stud.find().pretty();
```

RDBMS Where Clause Equivalents in MongoDB
($gt-Greater than
$gte-Greater than equal
$lt-Less than
$lte-Less than equal
$ne-Not equal
$in-in
$nin-Not in)

```
db.stud.find({"roll" : 2});

db.stud.find({"roll" : 2}).pretty( );

db.stud.find({"age" : {"$lt":20}});

db.stud.find({"age" : {"$lt":20}}).pretty( );

db.stud.find({"age" : {"$gte" : 20, "$lte" :25}},{name:1,age:1}).pretty( );

db.stud.find({"age" : {"$ne" : 25}}).pretty( );

db.stud.find({"age" : {"$gt" : 20, "$lt" :25}}).pretty( );

db.stud.find({"age" : {"$in" : [17,22,25]}}).pretty( );
```

Like Operator

```
db.stud.find({name:/^A/}).pretty():; // name field Start with “A” Character

db.stud.find({name:/a/}).pretty(); // name contains with “a” Character
```

or in Mongodb for pattern matching we can use $regex operator

Search document with name start with S

```
db.stud.find({“name”:{$regex:/^s/}}.pretty()
```

Search document ending with s

```
db.stud.find({“name”:{$regex:/s$/}}.pretty()
```

Logical Operator ($or, $and, $not, $nor)

AND Operator ($and) :

```
db.stud.find ({roll:3, name: "Ajay"});
db.stud.find ({roll:3, name: "Ajay"}).pretty( );
db.stud.find ({"$and": [{roll:3}, {name: "Ajay"}] }).pretty( );
```

OR Operator ($or):

```
db.stud.find ({"$or": [{roll:3}, {name: "Abhi"}]} ).pretty( );
```

NOT Operator ($not) :

```
db.stud.find ({ name: { $not: /^A/}}); // Student name not like with “A” char
```

Using AND and OR Together

```
db.stud.find ({{age:25} ,{"$or": [{roll:3}, {name: "Abhi"}] }}).pretty();
```

Update Data : update(); (and Insert Data using :- save();)

```
db.stud.save({roll:11, name:"Rajesh", age:21});
db.stud.update({roll:11},{$set:{name:"Anil"}});
```

Delete Data : remove();

```
db.stud.remove({"roll":11});
db.stud.find().pretty();
```

MongoDB Sort Documents

The sort() Method

- To sort documents in MongoDB, you need to use sort() method.
- sort() method accepts a document containing list of fields along with their sorting order.
- To specify sorting order 1 and -1 are used. 1 is used for ascending order while -1 is used for descending order.

Syntax:

```
>db.COLLECTION_NAME.find().sort({KEY:1})
```

FAQS:-Write command for the following

1. Create database Employee.
2.Create collection emp1 using createCollection method.
3.Insert 4 to 5 documents in emp1 collection.(eno,ename,address,sal)
4.display all documents.
5.Insert document by creating own object id.
6.Display all employess having salary greater than 5000
7. Display all employess having salary less than 15000
8. Display all employess having salary greater than 10000 and less than 20000.
9.update sal of all employee with 10%
10.Delete employee having salary less than 5000.
11.Rename collection emp1 with employee1.
12.Display employee whose name start with n.
13.sort name in ascending order.

CONCLUSION:-

Assignment No.11

Aim: Study of Aggregation framework and Indexing

Problem Statement:- Implement aggregation and indexing with suitable example using MongoDB.

Outcomes: Student should be able to
1. Understand concept Aggregation framework
2. Concept of Indexing

PEOs ,POs, PSOs and COs satisfied
PEOs : 2 POs : 1,3,4,5 PSOs : 2 COs :2

Software used : MONGODB(NOSQL)

Theory :

INDEXING
Indexes support the efficient resolution of queries. Without indexes, MongoDB must scan every document of a collection to select those documents that match the query statement.
This scan is highly inefficient and require the **mongod** to process a large volume of data.
"Indexes are special data structures that store a small portion of the data set in an easy to traverse form."
The index stores the value of a specific field or set of fields, ordered by the value of the field as specified in index.

The createIndex() Method
To create an index you need to use createIndex() method of mongodb.

SYNTAX:
Basic syntax of create**Index()** method is as follows()

```
>db.COLLECTION_NAME.createIndex({KEY:1})
```

Here key is the name of filed on which you want to create index and 1 is for ascending order. To create index in descending order you need to use -1.

EXAMPLE

```
db.mycol.createIndex({"title":1})
>
```

In **createIndex()** method you can pass multiple fields, to create index on multiple fields.

```
>db.mycol.createIndex({"title":1,"description":-1})
>
```

AGGREGATION
Aggregations operations process data records and return computed results. Aggregation operations group values from multiple documents together, and can perform a variety of operations on the

grouped data to return a single result. In sql count(*) and with group by is an equivalent of mongodb aggregation.

The aggregate() Method
For the aggregation in mongodb you should use **aggregate()** method.
SYNTAX:
Basic syntax of **aggregate()** method is as follows

```
>db.COLLECTION_NAME.aggregate(AGGREGATE_OPERATION)
```

EXAMPLE:
In the collection you have the following data:

```
{
  _id: ObjectId(7df78ad8902c)
  title: 'MongoDB Overview',
  description: 'MongoDB is no sql database',
  by_user: 'tutorials point',
  url: 'http://www.tutorialspoint.com',
  tags: ['mongodb', 'database', 'NoSQL'],
  likes: 100
},
{
  _id: ObjectId(7df78ad8902d)
  title: 'NoSQL Overview',
  description: 'No sql database is very fast',
  by_user: 'tutorials point',
  url: 'http://www.tutorialspoint.com',
  tags: ['mongodb', 'database', 'NoSQL'],
  likes: 10
},
{
  _id: ObjectId(7df78ad8902e)
  title: 'Neo4j Overview',
  description: 'Neo4j is no sql database',
  by_user: 'Neo4j',
  url: 'http://www.neo4j.com',
  tags: ['neo4j', 'database', 'NoSQL'],
  likes: 750
},
```

Now from the above collection if you want to display a list that how many tutorials are written by each user then you will use **aggregate()** method as shown below:

```
> db.mycol.aggregate([{$group : {_id : "$by_user", num_tutorial : {$sum : 1}}}])
{
  "result" : [
    {
      "_id" : "tutorials point",
```

```
      "num_tutorial" : 2
    },
    {
      "_id" : "tutorials point",
      "num_tutorial" : 1
    }
  ],
  "ok" : 1
}
>
```

SQL equivalent query for the above use case will be **select by_user, count(*) from mycol group by by_user**

FAQS:-Write command for following

1. Create a hypothetical sports club database that contains a users collection that tracks the user's join dates, sport preferences, and stores these data in documents and perform the following operation on it using aggregation
    - Returns user names in upper case and in alphabetical order.
    - Returns user names sorted by the month they joined.
    - Show how many people joined each month of the year.

CONCLUSION:-

Assignment No.12

Aim: Study of MAP-REDUCE in Mongodb

Problem Statement:- Implement Map reduces operation with suitable example using MongoDB.

Outcomes: Student should be able to
1. Understand concept of Map Reduce
2. Implement Map Reduce

PEOs ,POs, PSOs and COs satisfied
PEOs : 2 POs : 1,3,4,5 PSOs : 2 COs :2

Software used : MONGODB(NOSQL)

Theory :

Map-reduce is a data processing paradigm for condensing large volumes of data into useful aggregated results. MongoDB uses mapReduce command for map-reduce operations. MapReduce is generally used for processing large data sets.

MapReduce Command

Following is the syntax of the basic mapReduce command −

```
>db.collection.mapReduce(
   function() {emit(key,value);},  //map function
   function(key,values) {return reduceFunction}, {    //reduce function
      out: collection,
      query: document,
      sort: document,
      limit: number
   }
)
```

The map-reduce function first queries the collection, then maps the result documents to emit key-value pairs, which is then reduced based on the keys that have multiple values.

In the above syntax −

- **map** is a javascript function that maps a value with a key and emits a key-value pair
- **reduce** is a javascript function that reduces or groups all the documents having the same key
- **out** specifies the location of the map-reduce query result
- **query** specifies the optional selection criteria for selecting documents
- **sort** specifies the optional sort criteria
- **limit** specifies the optional maximum number of documents to be returned

In very simple terms, the mapReduce command takes 2 primary inputs, the mapper function and the reducer function.

AMapper will start off by readinga collection of data and building a Map with only the required fields we wish to process and group them into one array based on the key.
And then this key value pair is fed into a Reducer, which will process the values.

Example1 MapReduce function
•Consider the following document structure that stores book details author wise.
The document stores author_name of the book author and the status of book
•> db.author.save({"book_title" : "MongoDB Tutorial","author_name" : "aparajita","status" : "active", "publish_year": "2016" })
•> db.author.save({"book_title" : "Software Testing Tutorial","author_name" : "aparajita","status" : "active","publish_year": "2015" })
•> db.author.save({"book_title" : "Node.js Tutorial","author_name" : "Kritika","status" : "active","publish_year": "2016" })
•> db.author.save({"book_title" : "PHP7 Tutorial","author_name" : "aparajita","status" : "passive","publish_year": "2016" })

db.author.find()
{ "_id" : ObjectId("59333022523476d644344db9"), "book_title" : "MongoDB Tutorial", "author_name" : "aparajita", "status" : "active", "publish_year" : "2016" }

{ "_id" : ObjectId("59333031523476d644344dba"), "book_title" : "Software Testing Tutorial", "author_name" : "aparajita", "status" : "active", "publish_year" : "2015" }

{ "_id" : ObjectId("5933303e523476d644344dbb"), "book_title" : "Node.js Tutorial", "author_name" : "aparajita", "status" : "active", "publish_year" : "2016" }

{ "_id" : ObjectId("5933304b523476d644344dbc"), "book_title" : "PHP7 Tutorial", "author_name" : "aparajita", "status" : "active", "publish_year" : "2016" }

Now, use the mapReduce function
•To select all the active books, •Group them together on the basis of author_nameand
•Then count the number of books by each author by using the following code in MongoDB.

Code:

db.author.mapReduce(function() { emit(this.author_name,1) },function(key, values) {return Array.sum(values)}, { query:{status:"active"},out:"author_total"}).find()

Out-Put{ "_id" : "aparajita", "value" : 2 }{ "_id" : "Kritika", "value" : 1 }

Code:
db.author.mapReduce(function() { emit(this.author_name,1); },function(key, values) {return Array.sum(values)}, {query: { status : "active" },out: "author_total" })

CONCLUSION:-

Assignment No.13

Aim: Queries using MONGODB

Problem Statement:- Design and Implement any 5 query using MongoDB

Outcomes: Student should be able to
1. Concept of unstructured data.
2. Solve queries using MONGODB.

PEOs ,POs, PSOs and COs satisfied
PEOs : 2 POs : 1,3,4,5 PSOs : 2 COs :2

Software used : MONGODB(NOSQL)

Theory: What is MongoDB?

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

The best way we learn anything is by practice and exercise questions. We have started this section for those (beginner to intermediate) who are familiar with NoSQL and MongoDB. Hope, these exercises help you to improve your MongoDB query skills. Currently, following exercises are available based on '***restaurants'*** collection, we are working hard to add more exercises. Happy Coding!

Lab Exercise:-
1.Create Database Restaurants.
2.Create Collection REST1
3.Insert documents
Sample Documents

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

"name": "Morris Park Bake Shop",
"restaurant_id": "30075445"
}

4. Write a MongoDB query to display all the documents in the collection restaurants
5. Write a MongoDB query to display the fields restaurant_id, name, borough and cuisine for all the documents in the collection restaurant.
6. Write a MongoDB query to display all the restaurant which is in the borough Bronx
7. Write a MongoDB query to display the fields restaurant_id, name, borough and zip code, but exclude the field _id for all the documents in the collection restaurant.
8. Write a MongoDB query to find the restaurants who achieved a score more than 90
9. Write a MongoDB query to find the restaurants that achieved a score, more than 80 but less than 100

CONCLUSION:-

Assignment No.14

Aim: JAVA connectivity with MONGODB

Problem Statement:- Write a program to implement MogoDB database connectivity with PHP/ python/Java Implement Database navigation operations (add, delete, edit etc.) using ODBC/JDBC.

Outcomes: Student should be able to

1. JAVA and Mongodb connectivity
2. Write a program for connectivity

PEOs ,POs, PSOs and COs satisfied

PEOs : 2 POs : 1,3,4,5 PSOs : 2 COs :2

Software used :

- Eclipse
- JDK 1.6
- MongoDB
- MongoDB-Java-Driver

Theory:

Steps

In Eclipse perform following steps:

1. File - New – Java Project –Give Project Name – ok
2. In project Explorer window- right click on project namenew- class- give Class name- ok
3. In project Explorer window- right click on project nameBuild path- Configure build path- Libraries- Add External Jar - MongoDB-Java-Driver
4. Start Mongo server before running the program

Import packages, Create connection,database and collection

• Import packages

```
import com.mongodb.*;
```

• Create connection

```
MongoClient mongo = new MongoClient( "localhost" , 27017 );
```

• Create Database

```
DB db = mongo.getDB("database name");
```

• Create Collection

```
DBCollection coll = db.getCollection("Collection Name");
```

Insert Document

```
BasicDBObject d1 = new BasicDBObject("rno","1").append("name", "Monika").append("age", "17")
BasicDBObject d2 = new BasicDBObject("rno","2").append("name","Roshan").append("age", "18")

coll.insert(d1);
coll.insert(d2);
```

Display document

```
DBCursor cursor = coll.find();
while (cursor.hasNext())
 {
 System.out.println(cursor.next());
 }
```

Update Document

• BasicDBObject query = new BasicDBObject();
• query.put("name", "Monika");
• BasicDBObject newDocument = new BasicDBObject();
• newDocument.put("name", "Ragini");

• BasicDBObject updateObj = new BasicDBObject();

• updateObj.put("$set", newDocument);

• Coll.update(query, updateObj);

Remove document

BasicDBObject searchQuery = new
BasicDBObject();
searchQuery.put("name", “Monika");
Coll.remove(searchQuery);

Program

```
import com.mongodb.*;
public class conmongo {
public static void main(String[] args) {
try {
MongoClient mongoClient = new MongoClient( "localhost" , 27017 );
DB db = mongoClient.getDB( "mydb" );
DBCollection coll = db.createCollection(“Stud",null);
BasicDBObject doc1 = new BasicDBObject("rno","1").append("name",“Mona");
BasicDBObject doc2 = new BasicDBObject("rno","2").append("name","swati");
coll.insert(doc1);
coll.insert(doc2);
DBCursor cursor = coll.find(searchQuery);
while (cursor.hasNext())
 {
System.out.println(cursor.next());
 }
BasicDBObject query = new BasicDBObject();
query.put("name", “Monika");
BasicDBObject N1 = new BasicDBObject();
N1.put("name", “Ragini");
```

```
BasicDBObject S1= new BasicDBObject();
S1.put("$set", newDocument);
coll.update(query, S1);
BasicDBObject R1 = new BasicDBObject();
R1.put("name", “Monika");
coll.remove(R1);
}
catch(Exception e)
{
e.printStackTrace();
}
}
}
```

CONCLUSION:-