G H Raisoni College of Engineering & Management, Wagholi, Pune

**Department of Computer Engineering**

D-19

**Lab Manual (2021-22)**

**Class: TY Computer Term: V**

**BCOP303: Database Management Systems LAB**

**Faculty Name: Mrs. Sunita Nandgave**

G H Raisoni College of Engineering & Management, Wagholi, Pune

**Department of Computer Engineering**

|  |  |  |  |
| --- | --- | --- | --- |
| **Database Management Systems (BCOP19303)** | | | |
| **Course Outcome** | | | |
| **CO1** | | Design the ER diagrams as well as interpret the design of the database | |
| **CO2** | | Formulate the queries required to solve the issues in the database. | |
| **CO3** | | Design queries using SQL DML/DDL commands. | |
| **CO4** | | Perform PL/SQL programming using the concept of Cursor Management, Error  Handling,Package, and Triggers | |
| **List of Experiment** | | | |
| **Sr. No.** | **Name of Experiment** | | **CO Mapped** |
| **1** | Draw E-R diagrams for payroll database. | | **CO1, CO2** |
| **2** | Illustrate the use of constraints on employee schema: null, not null,  primary key, unique, check, default, references. | | **CO1, CO2** |
| **3** | Design at least 10 SQL queries for suitable database applications using SQL DML Statements: Insert, Select, Update, Delete with  operators, Functions, Set Operators, Clauses. | | **CO3** |
| **4** | Design and develop SQL DDL statements that demonstrate the use of SQL Objects such as Table, View, Index, Sequence,  Synonym. | | **CO3** |
| **5** | Aggregate functions in SQL (Count, Sum, Max, Min, Avg),  Commit, Rollback, and Savepoint command. | | **CO2, CO3** |
| **6** | Design SQL queries for suitable database applications using SQL  DML Statements: all types of Join, Sub-Query. | | **CO2, CO3** |
| **7** | Write a PL/SQL block to calculate the grade of a minimum 10  students. | | **CO4** |
| **8** | Write a PL/SQL block to implement all types of cursors. | | **CO4** |
| **9** | Write a PL/SQL stored procedure and function. | | **CO4** |
| **10** | Write a Row level and Statement level database Trigger. | | **CO4** |
|  | **Content Beyond Syllabus** | |  |
| **11** | NoSQL Databases: Implement CRUD operation in MongoDB | | **CO2, CO3** |
| **12** | Cassandra case study | | **CO2, CO3** |

# Experiment No-1

**AIM:** Draw E-R diagrams for payroll database.

#### Theory:

An entity-relationship diagram (ERD) shows the relationships of entity sets stored in a database. An entity in this context is an object, a component of data. An entity set is a collection of similar entities. These entities can have attributes that define their properties. By defining the entities, and their attributes, and showing the relationships between them, an ER diagram illustrates the logical structure of databases.

ER diagrams are used to sketch out the design of a database

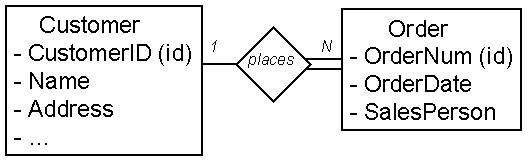
#### Chen Notation

1. **ORDER** (OrderNum (key), OrderDate, SalesPerson)

**ORDERITEMS** (OrderNum (key)(fk) , ItemNum (key), PartNum, Quantity, Cost)

1. In the above example, in the ORDERITEMS Relation: OrderNum is the *Foreign Key* and OrderNum plus ItemNum is the *Composite Key*.

#### Chen Notation



In the ORDER Relation: OrderNum is the *Key*.

#### Representing Relationships

1. **1:1** Relationships. The key of one relation is stored in the second relation. Look at example queries to determine which key is queried most often.
2. **1:N** Relationships.

**Parent** - Relation on the "1" side. **Child** - Relation on the "Many" side.

1. Represent each Entity as a relation.

Copy the key of the parent into the child relation.

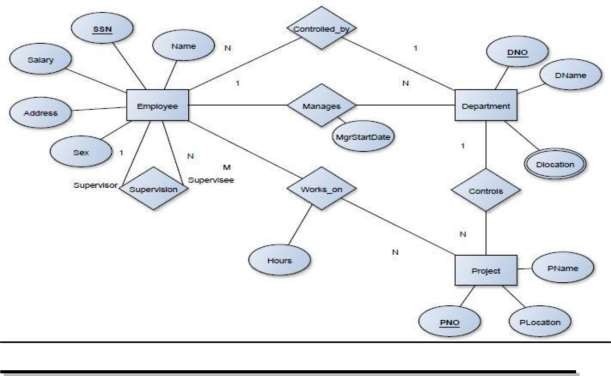
#### CUSTOMER (CustomerID (key), Name, Address, ...)

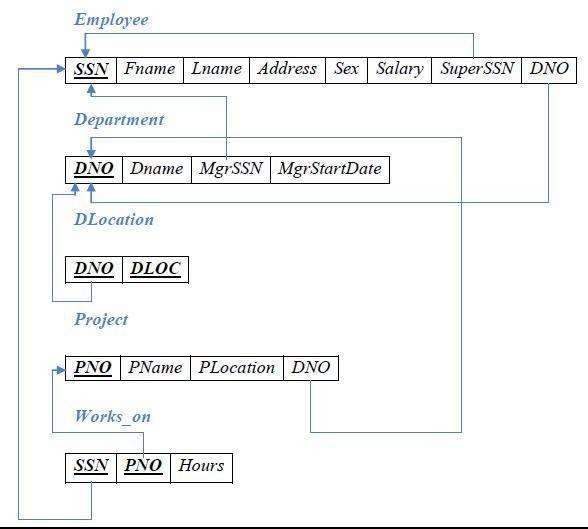
**ORDER (OrderNum (key), OrderDate, SalesPerson, CustomerID (fk)) Schema for Company Database:**

EMPLOYEE (SSN, Name, Address, Sex, Salary, SuperSSN, DNo) DEPARTMENT (DNo, DName, MgrSSN, MgrStartDate) DLOCATION (DNo, DLoc)

PROJECT (PNo, PName, PLocation, DNo) WORKS\_ON (SSN, PNo, Hours)

**ER-Diagram:**



**SCHEMA:**

**Conclusion:** In this experiment, we have studied Entity Relationships diagram notations and their use to draw the design of a database system.

# \*\*\*\*\*\*\*\*

**Experiment No-2**

## AIM: Illustrate the use of constraints on employee schema: NULL, NOT NULL, PRIMARY KEY, UNIQUE, CHECK, DEFAULT, REFERENCES.

#### Theory:

**CONSTRAINTS:**

Integrity Constraints are a mechanism to prevent invalid data entry into the table to maintain data consistency. The whole purpose of constraints is to maintain the data integrity during the various transactions like update/delete/insert on a table.

## Types of constraints:

* NOT NULL
* UNIQUE
* DEFAULT
* CHECK
* Key Constraints – PRIMARY KEY, FOREIGN KEY

#### NOT NULL:

NOT NULL constraint makes sure that a column does not hold NULL value. When we don't provide value for a particular column while inserting a record into a table, it takes NULL value by default. By specifying the NULL constraint, we can be sure that a particular column(s) cannot have NULL values.

#### UNIQUE:

#### UNIQUE Constraint enforces a column or set of columns to have unique values. If a column has a unique constraint, it means that a particular column cannot have duplicate values in a table.

#### 

#### DEFAULT:

The DEFAULT constraint provides a default value to a column when there is no value provided while inserting a record into a table.

#### CHECK:

This constraint is used for specifying a range of values for a particular column of a table. When this constraint is being set on a column, it ensures that the specified column must have the value falling in the specified range.

[The primary key](https://beginnersbook.com/2015/04/primary-key-in-dbms/) uniquely identifies each record in a table. It must have unique values and cannot contain nulls. In the below example the ROLL\_NO field is marked as the primary key, which means the ROLL\_NO field cannot have duplicate and null values.

#### FOREIGN KEY:

Foreign keys are the columns of a table that points to the primary key of another table. They act as a cross-reference between tables.

Read more about it [here](https://beginnersbook.com/2015/04/foreign-key-in-dbms/).

Create table tablename (column\_name1 data\_ type constraints, column\_name2 data\_ type constraints …)

#### Example:

Create table Emp ( EmpNo number(5), EName VarChar(15), Job Char(10) constraint un unique, DeptNo number(3) CONSTRAINT FKey2 REFERENCES DEPT(DeptNo));

Create table stud (sname varchar2(20) not null, rollno number(10) not null,dob date not null);

#### DOMAIN INTEGRITY

**Example:** Create table cust(custid number(6) not null, name char(10)); Alter table cust modify (name not null);

#### CHECK CONSTRAINT

Example: Create table student (regno number (6), mark number (3) constraint b check (mark >=0 and mark <=100)); Alter table student add constraint b2 check (length(regno<=4));

#### ENTITY INTEGRITY

1. Unique key constraint

Example: Create table cust(custid number(6) constraint unique, name char(10)); Alter table cust add(constraint c unique(custid));

1. Primary Key Constraint

Example: Create table stud(regno number(6) constraint primary key, name char(20));

#### Queries:

Q1. Create a table called EMP with the following structure. Name Type

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

EMPNO NUMBER(6) ENAME VARCHAR2(20) JOB VARCHAR2(10) DEPTNO NUMBER(3) SAL NUMBER(7,2)

Allow NULL for all columns except ename and job.

1. Understand create table syntax.
2. Use the create table syntax to create the said tables.
3. Create primary key constraints for each table as understood from logical table structure.

SQL> create table emp(empno number(6),ename varchar2(20)not null,job varchar2(10) not null, deptno number(3), sal number(7,2));

Table created.

Q2: Add a column experience to the emp table. experience numeric null allowed.

#### Solution:

1. Learn to alter table syntax.
2. Define the new column and its data type.
3. Use the altered table syntax.

Ans: SQL> alter table emp add(experience number(2)); Table altered.

Q3: Modify the column width of the job field of the emp table.

#### Solution:

1. Use the altered table syntax.
2. Modify the column width and its data type.

Ans: SQL> alter table emp modify(job varchar2(12)); Table altered.

SQL> alter table emp modify(job varchar(13)); Table altered.

Q4: Create a dept table with the following structure. Name Type

------------ --------------------- DEPT NUMBER(2) DNAME VARCHAR2(10) LOC VARCHAR2(10)

Deptno the primary key

#### Solution:

1. Understand create table syntax.
2. Decide the name of the table.
3. Decide the name of each column and its data type.
4. Use the create table syntax to create the said tables.
5. Create primary key constraint for each table as understand from logical table structure. Ans:

SQL> create table dept(deptno number(2) primary key,dname varchar2(10),loc varchar2(10));

Table created.

Q5: create the emp1 table with ename and empno, add constraints to check the empno value while entering (i.e) empno > 100.

#### Solution:

1. Learn alter table syntax.
2. Define the new constraint [columns name type]
3. Use the alter table syntax for adding constraints. Ans:

SQL> create table emp1(ename varchar2(10),empno number(6) constraint check(empno>100));

Table created.

Q6: drop a column experience to the emp table.

#### Solution:

* 1. Learn to alter table syntax. Use the alter table syntax to drop the column. Ans:

SQL> alter table emp drop column experience; Table altered.

Q7: Truncate the emp table and drop the dept table

#### Solution:

1. Learn drop, and truncate table syntax.

Ans: SQL> truncate table emp; Table truncated.

#### OUTPUT:

SQL> create table emp1(emp\_no number(6),ename varchar2(20) not null,job varchar2(10) not null,dept\_no number(3),sal number(7,2));

Table created.

SQL> desc emp1;

Name Null? Type

EMP\_NO NUMBER(6)

ENAME NOT NULL VARCHAR2(20) JOB NOT NULL VARCHAR2(10) DEPT\_NO NUMBER(3)

SAL NUMBER(7,2)

SQL> alter table emp1 add(experience number(5)); Table altered.

SQL> desc emp1;

Name Null? Type

EMP\_NO NUMBER(6)

ENAME NOT NULL VARCHAR2(20) JOB NOT NULL VARCHAR2(10) DEPT\_NO NUMBER(3)

SAL NUMBER(7,2)

EXPERIENCE NUMBER(5)

SQL> alter table emp1 modify(job varchar2(12)); Table altered.

SQL> desc emp1;

Name Null? Type

EMP\_NO NUMBER(6)

ENAME NOT NULL VARCHAR2(20) JOB NOT NULL VARCHAR2(12) DEPT\_NO NUMBER(3)

SAL NUMBER(7,2)

EXPERIENCE NUMBER(5)

SQL> create table dept1(dept\_no number(12) primary key,d\_name varchar2(10),loc varchar2(10));

Table created.

SQL> desc dept1;

Name Null? Type

DEPT\_NO NOT NULL NUMBER(12) D\_NAME VARCHAR2(10)

LOC VARCHAR2(10)

SQL> alter table emp1 modify(emp\_no number(6) check (emp\_no>100));

Table altered.

SQL> desc emp1;

Name Null? Type

EMP\_NO NUMBER(6)

ENAME NOT NULL VARCHAR2(20)

JOB NOT NULL VARCHAR2(12)

DEPT\_NO NUMBER(3)

SAL NUMBER(7,2)

EXPERIENCE NUMBER(5)

SQL> insert into emp1 values(120,'shiv','manager',13,30000,10); 1 row created.

SQL> select \* from emp1;

EMP\_NO ENAME JOB DEPT\_NO SAL EXPERIENCE

120 shiv manager 13 30000 10

SQL> alter table emp1 drop(experience); Table altered.

SQL> desc emp1;

Name Null? Type

EMP\_NO NUMBER(6)

ENAME NOT NULL VARCHAR2(20) JOB NOT NULL VARCHAR2(12) DEPT\_NO NUMBER(3)

SAL NUMBER(7,2)

SQL> truncate table emp1; Table truncated.

SQL> desc emp1;

Name Null? Type

EMP\_NO NUMBER(6)

ENAME NOT NULL VARCHAR2(20) JOB NOT NULL VARCHAR2(12) DEPT\_NO NUMBER(3)

SAL NUMBER(7,2)

SQL> select \* from emp1; no rows selected

SQL> drop table emp1;

Table dropped.

SQL> create table dept(deptno number(5) constraint dept\_deptno\_pk primary key, dname varchar2(20),loc varchar2(20));

Table created.

SQL> desc dept;

Name Null? Type

DEPTNO NOT NULL NUMBER(5) DNAME VARCHAR2(20)

LOC VARCHAR2(20)

SQL> create table emp(empno number(5),ename varchar2(25) not null,job varchar2(15),mgr number(5),joindate date,salary number(7,2),comm number(7,2),deptno number(7) constraint emp\_deptno\_fk references dept(deptno));

Table created.

SQL> desc emp;

Name Null? Type

EMPNO NUMBER(5)

ENAME NOT NULL VARCHAR2(25) JOB VARCHAR2(15)

MGR NUMBER(5)

JOINDATE DATE

SALARY NUMBER(7,2)

COMM NUMBER(7,2)

DEPTNO NUMBER(7)

SQL> insert into dept values(12,'computer','pune'); 1 row created.

SQL> select \* from dept; DEPTNO DNAME LOC

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

12 computer pune

SQL> insert into emp values(23,'shiv','manager',55,'12sep2019',30000,55,12); 1 row created.

SQL> select \* from emp;

EMPNO ENAME JOB MGR JOINDATE SALARY COMM DEPTNO

23 shiv manager 55 12-SEP-19 30000 55 12

SQL> create table emp1(emp\_no number(6),ename varchar2(20) not null,job varchar2(10) not null,dept\_no number(3),sal number(7,2));

Table created.

SQL> desc emp1;

Name Null? Type

EMP\_NO NUMBER(6)

ENAME NOT NULL VARCHAR2(20) JOB NOT NULL VARCHAR2(10) DEPT\_NO NUMBER(3)

SAL NUMBER(7,2)

SQL> alter table emp1 add(experience number(5)); Table altered.

SQL> desc emp1;

Name Null? Type

EMP\_NO NUMBER(6)

ENAME NOT NULL VARCHAR2(20) JOB NOT NULL VARCHAR2(10) DEPT\_NO NUMBER(3)

SAL NUMBER(7,2)

EXPERIENCE NUMBER(5)

SQL> alter table emp1 modify(job varchar2(12)); Table altered.

SQL> desc emp1;

Name Null? Type

EMP\_NO NUMBER(6)

ENAME NOT NULL VARCHAR2(20) JOB NOT NULL VARCHAR2(12) DEPT\_NO NUMBER(3)

SAL NUMBER(7,2)

EXPERIENCE NUMBER(5)

SQL> create table dept1(dept\_no number(12) primary key,d\_name varchar2(10),loc varchar2(10));

Table created.

SQL> desc dept1;

Name Null? Type

DEPT\_NO NOT NULL NUMBER(12) D\_NAME VARCHAR2(10)

LOC VARCHAR2(10)

SQL> alter table emp1 modify(emp\_no number(6) check (emp\_no>100)); Table altered.

SQL> desc emp1;

Name Null? Type

EMP\_NO NUMBER(6)

ENAME NOT NULL VARCHAR2(20) JOB NOT NULL VARCHAR2(12) DEPT\_NO NUMBER(3)

SAL NUMBER(7,2)

EXPERIENCE NUMBER(5)

SQL> insert into emp1 values(120,'shiv','manager',13,30000,10); 1 row created.

SQL> select \* from emp1;

EMP\_NO ENAME JOB DEPT\_NO SAL EXPERIENCE

120 shiv manager 13 30000 10

SQL> alter table emp1 drop(experience); Table altered.

SQL> desc emp1;

Name Null? Type

EMP\_NO NUMBER(6)

ENAME NOT NULL VARCHAR2(20) JOB NOT NULL VARCHAR2(12) DEPT\_NO NUMBER(3)

SAL NUMBER(7,2)

SQL> truncate table emp1; Table truncated.

SQL> desc emp1;

Name Null? Type

EMP\_NO NUMBER(6)

ENAME NOT NULL VARCHAR2(20)

JOB NOT NULL VARCHAR2(12) DEPT\_NO NUMBER(3)

SAL NUMBER(7,2)

SQL> select \* from emp1; no rows selected

SQL> drop table emp1;

Table dropped.

SQL> desc emp1;

ERROR:

ORA-04043: object emp1 does not exist

SQL> create table dept(deptno number(5) constraint dept\_deptno\_pk primary key,dname varchar2(20),loc varchar2(20));

Table created.

SQL> desc dept;

Name Null? Type

DEPTNO NOT NULL NUMBER(5) DNAME VARCHAR2(20)

LOC VARCHAR2(20)

SQL> create table emp(empno number(5),ename varchar2(25) not null,job varchar2(15),mgr number(5),joindate date,salary number(7,2),comm number(7,2),deptno number(7) constraint emp\_deptno\_fk references dept(deptno));

Table created.

SQL> desc emp;

Name Null? Type

EMPNO NUMBER(5)

ENAME NOT NULL VARCHAR2(25) JOB VARCHAR2(15)

MGR NUMBER(5)

JOINDATE DATE

SALARY NUMBER(7,2)

COMM NUMBER(7,2)

DEPTNO NUMBER(7)

SQL> insert into dept values(12,'computer','pune'); 1 row created.

SQL> select \* from dept;

DEPTNO DNAME LOC

12 computer pune

SQL> insert into emp values(23,'shiv','manager',55,'12sep2019',30000,55,12); 1 row created.

SQL> select \* from emp;

EMPNO ENAME JOB MGR JOINDATE SALARY COMM DEPTNO

23 shiv manager 55 12-SEP-19 30000 55 12

**Conclusion:** In this experiment, we have studied the concept of constraint and executed all constraints such as NULL, NOT NULL, PRIMARY KEY, UNIQUE, CHECK, DEFAULT, and REFERENCES.

**\*\*\*\*\*\*\*\***

# Experiment No-3

## Aim: Design SQL queries for suitable database applications using SQL DML Statements: Insert, Select, Update, Delete with operators, Functions, Set Operators, Clauses.

**Theory:**

**DML COMMANDS:** DML commands are the most frequently used SQL commands and are used to query and manipulate the existing database objects. Some of the commands are Insert, Select, Update, and Delete.

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

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

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

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

Q1: Insert a single record into the dept table.

Ans: SQL> insert into dept values (1,'IT','Tholudur'); 1 row created.

Q2: Insert more than a record into emp table using a single insert command.

Ans: SQL> insert into emp values(&empno,'&ename','&job',&deptno,&sal); Enter value for empno: 1

Enter value for ename: Mathi Enter value for job: AP

Enter value for deptno: 1 Enter value for sal: 10000

old 1: insert into emp values(&empno,'&ename','&job',&deptno,&sal)

new 1: insert into emp values(1,'Mathi','AP',1,10000) 1 row created.

SQL> / Enter value for empno: 2 Enter value for ename: Arjun Enter value for job: ASP

Enter value for deptno: 2 Enter value for sal: 12000

old 1: insert into emp values(&empno,'&ename','&job',&deptno,&sal) new 1: insert into emp values(2,'Arjun','ASP',2,12000)

1 row created.

SQL> / Enter value for empno: 3 Enter value for ename: Gugan Enter value for job: ASP

Enter value for deptno: 1 Enter value for sal: 12000

old 1: insert into emp values(&empno,'&ename','&job',&deptno,&sal) new 1: insert into emp values(3,'Gugan','ASP',1,12000)

1 row created.

Q3: Update the emp table to set the salary of all employees to Rs15000/- who are working as ASP

Ans: SQL> select \* from emp; EMPNO ENAME JOB DEPTNO SAL

1. Mathi AP 1 10000
2. Arjun ASP 2 12000
3. Gugan ASP 1 12000

SQL> update emp set sal=15000 where job='ASP'; 2 rows updated. SQL> select \* from emp;

EMPNO ENAME JOB DEPTNO SAL

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

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | Mathi | AP | 1 | 10000 |
| 2 | Arjun | ASP | 2 | 15000 |
| 3 | Gugan | ASP | 1 | 15000 |

Q4: Create a pseudo table employee with the same structure as the table emp and insert rows into the table using select clauses.

Ans: SQL> create table employee as select \* from emp; Table created.

SQL> desc employee; Name Null? Type

----------------------------------------- -------- ---------------------------- EMPNO NUMBER(6)

ENAME NOT NULL VARCHAR2(20) JOB NOT NULL VARCHAR2(13) DEPTNO NUMBER(3)

SAL NUMBER(7,2)

Q5: select employee name, job from the emp table Ans: SQL> select ename, job from emp;

ENAME JOB

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

Mathi AP

Arjun ASP

Gugan ASP

Karthik Prof

Akalya AP

Suresh lect

6 rows were selected.

Q6: Delete only those who are working as lecturers Ans: SQL> select \* from emp;

EMPNO ENAME JOB DEPTNO SAL

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

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | Mathi | AP | 1 | 10000 |
| 2 | Arjun | ASP | 2 | 15000 |
| 3 | Gugan | ASP | 1 | 15000 |
| 4 | Karthik | Prof | 2 | 30000 |
| 5 | Akalya | AP | 1 | 10000 |
| 6 | Suresh | lect | 1 | 8000 |

6 rows were selected.

SQL> delete from emp where job='lect'; 1 row deleted.

SQL> select \* from emp;

EMPNO ENAME JOB DEPTNO SAL

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

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | Mathi | AP | 1 | 10000 |
| 2 | Arjun | ASP | 2 | 15000 |
| 3 | Gugan | ASP | 1 | 15000 |
| 4 | Karthik | Prof | 2 | 30000 |
| 5 | Akalya | AP | 1 | 10000 |

Q7: List the records in the emp table order by salary in ascending order. Ans: SQL> select \* from emp order by sal;

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| EMPNO | ENAME | JOB | DEPTNO | | SAL |
| 1 | Mathi | AP | 1 | | 10000 |
| 5 | Akalya | AP | 1 | 10000 | |
| 2 | Arjun | ASP | 2 | 15000 | |
| 3 | Gugan | ASP | 1 | 15000 | |
| 4 | Karthik | Prof | 2 | 30000 | |

Q8: List the records in the emp table order by salary in descending order. Ans: SQL> select \* from emp order by sal desc;

EMPNO ENAME JOB DEPTNO SAL

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

4 Karthik Prof 2 30000

1. Arjun ASP 2 15000
2. Gugan ASP 1 15000

1 Mathi AP 1 10000

5 Akalya AP 1 10000

Q9: Display only those employees whose deptno is 30. Solution: Use SELECT FROM WHERE syntax.

Ans: SQL> select \* from emp where deptno=1; EMPNO ENAME JOB DEPTNO SAL

---------- -------------------- ------------- ---------- ---------- 1 Mathi AP 1 10000

3 Gugan ASP 1 15000

5 Akalya AP 1 10000

Q10: Display deptno from the table employee avoiding the duplicated values. Solution:

1. Use SELECT FROM syntax.
2. Select should include a distinct clause for the deptno. Ans: SQL> select distinct deptno from emp;

DEPTNO

---------- 1

2

#### IMPLEMENTATION OF DATA AND BUILT IN FUNCTIONS IN SQL CHARACTER/STRING FUNCTION:

SQL> select upper('welcome') from dual;

----------- WELCOME

SQL> select upper('hai') from dual;

--- HAI

SQL> select lower('HAI') from dual; LOW

---

hai

SQL> select initcap(‘hello world') from dual;

INITCAP('Hello world’)

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

Hello World

SQL> select ltrim(' hai') from dual; LTR

---

hai

SQL> select rtrim('hai ')from dual; RTR

---

hai

SQL> select rtrim(' hai ')from dual; RTRIM('hai)

-------

hai

SQL> select concat('GHRCEM',' Pune')from dual;

------------------------ GHRCEM Pune

SQL> select length('SRM’)from dual; LENGTH('SRM')

---------------------- 12

SQL> select replace('SRM university', 'SRM','Anna')from dual;

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

Anna university

SQL> select substr('SRM', 7,6)from dual; SUBSTR

------

lingam

SQL> select rpad('hai',3,'\*')from dual; RPAD('

------

hai\*\*\*

SQL> select lpad('hai',3,'\*')from dual; LPAD

------

\*\*\*hai

SQL> select replace('Dany','y','ie')from dual; REPLACE

-------

Danie

SQL> select translate('cold','ld','ol')from dual; TRANSL

------

cool

SET OPERATORS

#### Set operations:

Union/ Intersect/ Except operations – These operations operate on relations, which must be

compatible i.e. they must have the same no. of attributes with the same domain types. Syntax:

(select query1) Union/ Intersect/Except (select query2)

Here set operations are applied to tuples in the results of multiple select queries. All these operations eliminate duplicate tuples from the result.

SELECT customerNumber id, contactLastname name FROM customers UNION

SELECT employeeNumber id, firstname name FROM employees Output:

id name

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

103 Schmitt

112 King

114 Ferguson

119 Labrune

121 Bergulfsen

1. Nelson
2. Piestrzeniewicz
3. Keitel
4. Murphy

131 Lee

## Conclusion: In this experiment, we have studied the concept of Data Manipulation Language and implemented Insert, Select, Update, Delete with operators, Functions, Set Operators, and Clauses.

**\*\*\*\*\*\*\*\*\*\***

# Experiment No-4

### **AIM:** Design and develop SQL DDL statements that demonstrate the use of SQL Objects such as Table, View, Index, Sequence, and Synonym.

#### Theory:

**Data Definition Language (DDL)**: This language allows the users to define data and its relationships to other types of data. It is used to create data tables, and files within databases.

1. The Create Table Command: - Defines each column of the table uniquely. Each column has a minimum of three attributes, a name, data type, and size.

#### Syntax:

Create table <table name> (<col1> <datatype>(<size>),<col2>

<datatype><size>)); Ex: create table emp(empno number(4) primary key, ename char(10));

1. Modifying the structure of tables.

a) Add new columns

#### Syntax:

Alter table <tablename> add(<new col><datatype(size),<new col>datatype(size)); Ex:alter table emp add(sal number(7,2));

1. Dropping a column from a table.

#### Syntax:

Alter table <tablename> drop column <col>; Ex: alter table emp drop column sal;

1. Modifying existing columns.

#### Syntax:

Alter table <tablename> modify(<col><newdatatype>(<newsize>)); Ex: alter table emp modify(ename varchar2(15));

1. Renaming the tables

#### Syntax:

Rename <oldtable> to <new table>; Ex:rename emp to emp1;

1. truncating the tables.

#### Syntax:

Truncate table <tablename>; Ex:trunc table emp1;

1. Destroying tables.

#### Syntax:

Drop table <tablename>; Ex:drop table emp; **CREATION OF TABLE:**

#### SYNTAX:

create table<tablename>(column1 datatype,column2 datatype...);

#### EXAMPLE:

SQL>create table std(sno number(5),sname varchar(20),age number(5),sdob date,sm1 number(4,2),sm2 number(4,2),sm3 number(4,4)); Table created.

SQL>insert into std values(101,‟AAA‟,16,‟03-jul-88‟,80,90,98); 1 row created.

SQL>insert into std values(102,‟BBB‟,18,‟04-aug-89‟,88,98,90); 1 row created.

#### OUTPUT:

Select \* from std;

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| SNO | SNAME | AGE | SDOBSM1 | SM2 | SM3 |
| 101 | AAA | 16 | 03-jul-88 80 | 90 | 98 |
| 102 | BBB | 18 | 04-aug-89 88 | 98 | 90 |

#### ALTER TABLE WITH ADD:

SQL>create table student(id number(5),name varchar(10),game varchar(20)); Table created.

SQL>insert into student values(1,‟mercy‟,‟cricket‟); 1 row created.

#### SYNTAX:

alter table<tablename>add(col1 datatype,col2 datatype..); EXAMPLE:

SQL>alter table student add(age number(4));

SQL>insert into student values(2,‟sharmi‟,‟tennis‟,19);

#### OUTPUT:

ALTER: select \* from student; ID NAME GAME

1 Mercy Cricket

ADD: select \* from student; ID NAME GAME AGE

1. Mercy cricket
2. Sharmi Tennis 19

#### ALTER TABLE WITH MODIFY:

**SYNTAX:**

Alter table<tablename>modify(col1 datatype,col2 datatype..);

#### EXAMPLE:

SQL>alter table student modify(id number(6),game varchar(25));

#### OUTPUT:

MODIFY

desc student;

NAME NULL? TYPE

Id Number(6) Name Varchar(20)

Game Varchar(25) Age Number(4)

#### DROP:

**SYNTAX:** drop table<tablename>;

EXAMPLE:

SQL>drop table student; SQL>Table dropped.

#### TRUNCATE TABLE

**SYNTAX:** TRUNCATE TABLE <TABLE NAME>;

Example: Truncate table stud;

#### DESC

Example: desc emp;

Name Null? Type

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

EmpNo NOT NULL number(5) EName VarChar(15)

Job NOT NULL Char(10)

DeptNo NOT NULL number(3) PHONE\_NO number (10)

**View:**

* A database view is a virtual table or logical table which is defined as a SQL SELECT query with joins. Because a database view is similar to a database table, which consists of rows and columns, so you can query data against it. Most database management systems, including MySQL, allows you to update data in the underlying tables through the database view with some prerequisites.
* A database view is dynamic because it is not related to the physical schema. The database system stores database views as a SQL SELECT statement with joins. When the data of the tables changes, the view reflects that changes as well.

Advantages of database view:

* + A database view allows you to simplify complex queries: a database view is defined by an SQL statement that associates with many underlying tables. You can use a database view to hide the complexity of underlying tables to the end-users and external applications. Through a database view, you only have to use simple SQL statements instead of complex ones with many joins.
  + A database view helps limit data access to specific users: You may not want a subset of sensitive data that can be queryable by all users. You can use database views to expose only non-sensitive data to a specific group of users.
  + A database view provides an extra security layer. Security is a vital part of any relational database management system. Database views provide extra security for a database management system. A database view allows you to create only a read-only view to expose read-only data to specific users. Users can only retrieve data in the read-only view but cannot update it.
  + A database view enables computed columns. A database table should not have calculated columns however a database view should.
  + Database view enables backward compatibility. Suppose you have a central database, which many applications are using it. One day you decided to redesign the database to adapt to the new business requirements. You remove some tables and create several new tables, and you don’t want the changes to affect other

applications. In this scenario, you can create database views with the same schema as the legacy tables that you have removed.

Disadvantages of database view:

Besides the advantages above, there are several disadvantages of using database views:

* + Performance: querying data from a database view can be slow especially if the view is created based on other views.
  + Tables dependency: you create a view based on underlying tables of the database. Whenever you change the structure of those tables that view associates with, you have to change the view as well.

#### DDL COMMAND ON VIEW:

* 1. CREATE : Syntax :CREATE VIEW view\_name AS SELECT column\_name(s) FROM table\_name WHERE condition;

Example: CREATE VIEW Emp\_View AS Select \* From emp;

mysql> select \* from emp;

+-------+----------+--------+----------+

| empid | emp\_name | salary | Dept |

+-------+----------+--------+----------+

| 1 | XYZ | 10000 | COMPUTER |

| 2 | PQR | 12000 | CIVIL |

| 3 | LMN | 15000 | E&TC |

| 4 | ABC | 25000 | COMPUTER |

+-------+----------+--------+----------+

4 rows in set (0.00 sec)

mysql> CREATE VIEW VEMP AS select \* from emp; Query OK, 0 rows affected (0.05 sec)

mysql> select \* from VEMP;

+-------+----------+--------+----------+

| empid | emp\_name | salary | Dept |

+-------+----------+--------+----------+

| 1 | XYZ | 10000 | COMPUTER |

| 2 | PQR | 12000 | CIVIL |

| 3 | LMN | 15000 | E&TC |

| 4 | ABC | 25000 | COMPUTER |

+-------+----------+--------+----------+

4 rows in set (0.00 sec)

* 1. ALTER: Once a view is defined, you can modify it by using the ALTER VIEW statement. The syntax of the ALTER VIEW statement is similar to the CREATE VIEW statement except the CREATE keyword is replaced by the ALTER keyword

Syntax : ALTER VIEW view\_name AS SELECT column\_name(s) FROM table\_name WHERE condition;

Example: ALTER VIEW Emp\_View AS Select \* From emp;

* 1. DROP: Syntax: DROP VIEW [IF EXISTS] [database\_name].[view\_name];

Example: DROP VIEW emp\_view; or DROP VIEW EMP.emp\_view; // EMP is Database name.

* 1. TRUNCATE: We can’t do truncate on view.

#### INDEX:

* A database index is a data structure that improves the speed of operations in a table. Indexes can be created using one or more columns, providing the basis for both rapid random lookups and efficient ordering of access to records.
* While creating an index, should be considered that what are the columns which will be used to make SQL queries and create one or more indexes on those columns.
* Practically, indexes are also types of tables, which keep the primary key or index field and a pointer to each record in the actual table.
* The users cannot see the indexes, they are just used to speed up queries and will be used by Database Search Engine to locate records very fast.
* INSERT and UPDATE statements take more time on tables having indexes whereas SELECT statements become fast on those tables. The reason is that while doing insert or update, the database needs to insert or update index values as well.

#### DDL Commands on Index:

1. **CREATE:**

Syntax : CREATE INDEX index\_name ON table\_name(field\_name1,filed\_name2..);

Example: CREATE INDEX EINEDX ON emp(empid);

mysql> CREATE INDEX EINEDX ON emp(empid); Query OK, 0 rows affected (0.20 sec)

Records: 0 Duplicates: 0 Warnings: 0

mysql> show index from emp;

+-------+------------+----------+--------------+-------------+-----------+-----------

--+----------+--------+------+------------+---------+---------------+

| Table | Non\_unique | Key\_name | Seq\_in\_index | Column\_name | Collation | Cardinality | Sub\_part | Packed | Null | Index\_type | Comment | Index\_comment |

1. **ALTER** :

+-------+------------+----------+--------------+-------------+-----------+-----------

--+----------+--------+------+------------+---------+---------------+

| emp | 1 | EINEDX | 1 | empid | A | 4 | NULL

| NULL | YES | BTREE | | |

+-------+------------+----------+--------------+-------------+-----------+-----------

--+----------+--------+------+------------+---------+---------------+

1 row in set (0.00 sec)

Syntax : ALTER TABLE tbl\_name ADD INDEX index\_name (column\_list) Example : ALTER TABLE emp ADD INDEX(emp\_name);

mysql> SHOW INDEX FROM emp;

+-------+------------+----------+--------------+-------------+-----------+-------------+----------+--------+--

----+------------+---------+---------------+

| Table | Non\_unique | Key\_name | Seq\_in\_index | Column\_name | Collation | Cardinality | Sub\_part | Packed | Null | Index\_type | Comment | Index\_comment |

+-------+------------+----------+--------------+-------------+-----------+-------------+----------+--------+--

----+------------+---------+---------------+

| emp | 1 | empid | 1 | empid | A | 4 | NULL | NULL | YES | BTREE | | |

+-------+------------+----------+--------------+-------------+-----------+-------------+----------+--------+--

----+------------+---------+---------------+

1 row in set (0.00 sec)

Syntax : DROP INDEX index\_name ON table\_name; Example :

mysql> DROP INDEX empid on emp; Query OK, 0 rows affected (0.12 sec) Records: 0 Duplicates: 0 Warnings: 0

**TRUNCATE**: We can’t use the truncate command on INDEX.

**Conclusion:** In this experiment, we have studied SQL Data Definition Language statements and we have implemented SQL Objects such as Table, View, Index, Sequence, and Synonym.

**\*\*\*\*\*\*\*\*\***

# Experiment No-5

### **AIM:** Aggregate functions in SQL (Count, Sum, Max, Min, Avg), Commit, Rollback, and Savepoint commands

#### Theory:

**Aggregate functions** perform a calculation on a set of values and return a single value. There are different types of aggregate functions such as min, max, sum, avg, count, etc.

### Why use aggregate functions: From a business perspective, different organization levels have different information requirements. Top levels managers are usually interested in knowing whole figures and not necessary the individual details.

Aggregate functions allow us to easily produce summarized data from our database.

Aggregate Functions are all about

* Performing calculations on multiple rows
* Of a single column of a table
* And returning a single value.

**TCL COMMAND:** Transaction Language Commands allows us to control and manage transactions to maintain the integrity of data within SQL statements

COMMIT: command is used to save the Records. ROLLBACK: command is used to undo the Records.

SAVEPOINT command is used to undo the Records in a particular transaction.

#### Queries:

Tables Used: Consider the following tables namely “DEPARTMENTS” and “EMPLOYEES”

Their schemas are as follows , Departments ( dept \_no , dept\_ name , dept\_location ); Employees ( emp\_id , emp\_name , emp\_salary );

Q1: Develop a query to grant all privileges of employees table into departments table Ans: SQL> Grant all on employees to departments;

Grant succeeded.

Q2: Develop a query to grant some privileges of employees table into departments table

Ans: SQL> Grant select, update, insert on departments to departments with grant option; Grant succeeded.

Q3: Develop a query to revoke all privileges of employees table from departments table Ans: SQL> Revoke all on employees from departments; Revoke succeeded.

Q4: Develop a query to revoke some privileges of employees table from departments table Ans: SQL> Revoke select, update, insert on departments from departments;

Revoke succeeded.

Q5: Write a query to implement the save point Ans: SQL> SAVEPOINT S1;

Savepoint created. SQL> select \* from emp;

EMPNO ENAME JOB DEPTNO SAL

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

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | Mathi | AP | 1 | 10000 |
| 2 | Arjun | ASP | 2 | 15000 |
| 3 | Gugan | ASP | 1 | 15000 |
| 4 | Karthik | Prof | 2 | 30000 |

SQL> INSERT INTO EMP VALUES(5,'Akalya','AP',1,10000); 1 row created.

SQL> select \* from emp;

EMPNO ENAME JOB DEPTNO SAL

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

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | Mathi | AP | 1 | 10000 |
| 2 | Arjun | ASP | 2 | 15000 |
| 3 | Gugan | ASP | 1 | 15000 |
| 4 | Karthik | Prof | 2 | 30000 |
| 5 | Akalya | AP | 1 | 10000 |

Q6: Write a query to implement the rollback Ans: SQL> rollback s1; SQL> select \* from emp;

EMPNO ENAME JOB DEPTNO SAL

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

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | | Mathi | AP | | 1 | | 10000 | |
| 2 | | Arjun | ASP | | 2 | | 15000 | |
| 3 | Gugan | | | ASP | | 1 | 15000 |
| 4 | Karthik | | | Prof | | 2 | 30000 |

Q6: Write a query to implement the commit

Ans: SQL> COMMIT;

Commit complete.

Example:

#### For each department that has more than five employees, retrieve the department number and the number of its employees who are making more than Rs.

**6,00,000**.

SELECT DNO, COUNT(SSN) FROM THE EMPLOYEE

WHERE SALARY>600000 AND DNO IN(SELECT DNO

FROM THE EMPLOYEE GROUP BY DNO

HAVING COUNT(SSN)>5)

GROUP BY DNO ;

DNO COUNT(SSN)

--------- ---------- 3

ubuntu@ubuntu-OptiPlex-380:~$ MySQL -u root -p Enter password:

Welcome to the MySQL monitor. Commands end with; or \g. Your MySQL connection id is 36

Server version: 5.5.62-0ubuntu0.14.04.1 (Ubuntu)

Copyright (c) 2000, 2018, Oracle and/or its affiliates. All rights reserved. Oracle is a registered trademark of Oracle Corporation and/or its

affiliates. Other names may be trademarks of their respective owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> show databases;

+--------------------+

| Database |

+--------------------+

| information\_schema |

| anya |

| book |

| first |

| info |

| jn |

| job |

| last |

| MySQL |

| performance\_schema |

| se |

| sec |

| sece |

| second |

| suraj |

| third |

| universalbank |

| viju |

+--------------------+

18 rows in set (0.13 sec)

**mysql> use mysql;**

Reading table information for completion of table and column names You can turn off this feature to get a quicker startup with -A

**Database changed**

mysql> create table product(id int(4),pname varchar(9) NOT NULL,pprice int(8) NOT NULL,discount int(5));Query OK, 0 rows affected (0.08 sec)

mysql> desc product;

+----------+------------+------+-----+---------+-------+

| Field | Type | Null | Key | Default | Extra |

+----------+------------+------+-----+---------+-------+

| id | int(4) | YES | | NULL | |

| pname | varchar(9) | NO | | NULL | |

| pprice | int(8) | NO | | NULL | |

| discount | int(5) | YES | | NULL | |

+----------+------------+------+-----+---------+-------+

4 rows in set (0.00 sec)

mysql> insert into product VALUES(1,'AMIT',2000,100); Query OK, 1 row affected (0.04 sec)

mysql> insert into product VALUES(2,'CORNFLAKES',800,50); Query OK, 1 row affected, 1 warning (0.04 sec)

mysql> insert into product VALUES(3,'PARLE',90,5); Query OK, 1 row affected (0.04 sec)

mysql> insert into product VALUES(4,'NESTLE',70,2); Query OK, 1 row affected (0.04 sec)

mysql> insert into product VALUES(5,'CADBOURY',100,10); Query OK, 1 row affected (0.04 sec)

mysql> insert into product VALUES(6,'AMUL',200,20); Query OK, 1 row affected (0.03 sec)

mysql> insert into product VALUES(7,'JAM',70,8); Query OK, 1 row affected (0.04 sec)

mysql> select \* from product;

+------+-----------+--------+----------+

| id | pname | pprice | discount |

+------+-----------+--------+----------+

| 1 | AMIT | 2000 | 100 |

| 2 | CORNFLAKE | 800 | 50 |

| 3 | PARLE | 90 | 5 |

| 4 | NESTLE | 70 | 2 |

| 5 | CADBOURY | 100 | 10 |

| 6 | AMUL | 200 | 20 |

| 7 | JAM | 70 | 8 |

+------+-----------+--------+----------+

7 rows in set (0.00 sec)

mysql> select count(id) AS Numberofproducts FROM product;

+------------------+

| Numberofproducts |

+------------------+

| 7 |

+------------------+

1 row in set (0.00 sec)

mysql> select Sum(pprice) AS Totalprice FROM product;

+------------+

| Totalprice |

+------------+

| 3330 |

+------------+

1 row in set (0.00 sec)

mysql> select Max(discount) AS Largestdiscount FROM product;

+-----------------+

| Largestdiscount |

+-----------------+

| 100 |

+-----------------+

1 row in set (0.00 sec)

mysql> select Min(pprice) AS Smallestprice FROM product;

+---------------+

| Smallestprice |

+---------------+

| 70 |

+---------------+

1 row in set (0.00 sec)

mysql> select Avg(pprice) AS AveragePrice FROM product;

+--------------+

| AveragePrice |

+--------------+

| 475.7143 |

+--------------+

1 row in set (0.00 sec)

**Conclusion:** In this experiment, we have studied aggregate functions and Transection Control Language and implemented Count, Sum, Max, Min, Avg, Commit, Rollback, and Savepoint commands.

**\*\*\*\*\*\*\*\*\***

# Experiment No-6

### **AIM:** Design SQL queries for suitable database applications using SQL DML Statements: all types of Join, Sub-Query.

#### Theory:

1. Equi-join/Inner join
2. Non-equi-join
3. Self-join
4. Outer join

#### Equi-join:

A join that is based on equalities is called equi-join. '=' operator is used in equi- join comparison. It retrieves rows from tables having a common column. It is also called simple join.

#### Non-equi-join:

A join that specifies the relationship between columns belonging to different tables by making use of the relational operators (<, >, <=, >=, !=) other than the '=' operator is called as non-equi-join

#### Self-join:

Joining a table to itself is known as self-join i.e. it joins one row in a table to another. It can compare each row of the table to itself and also with other rows of the same table.

#### Outer join:

An outer join returns all the rows returned by simple join or equi join as well as those rows from one table that do not match any row from the other table.

The symbol (+) represents outer join.

#### Implementation:

1. **Equi join:**

select e.empno, e.ename, e.dept, d.deptno, d.loc, d.dname from emp e, dept d where e.dept=d.deptno;

#### Output:

EMPNO ENAME DEPT DEPTNO LOC DNAME

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1001 | Nilesh Joshi | 10 | 10 | Fourth | Computer |
| 1002 | Avinash Pawar | 30 | 30 | Second | Electrical |
| 1003 | Amit Kumar | 30 | 30 | Second | Electrical |
| 1005 | Niraj Sharma |  | 20 | 20 | First Mechanical |
| 1006 | Pushkar Deshpande | 30 | 30 | Second | Electrical |
| 1007 | Sumit Patil | 20 | 20 | First | Mechanical |
| 1008 | ravi sawant | 20 | 20 | First | Mechanical |

#### Non-equi-join:

Select e.ename, e.salary, s.grade from emp e, salgrade s where e.salary>=s.losal and e.salary<=s.hisal;

|  |  |  |
| --- | --- | --- |
| **Output:** |  | |
| ENAME | SALARY | GRADE |
| Amit Kumar | 2000 | 3 |
| Nilesh Joshi | 2800 | 3 |
| Avinash Pawar | 5000 | 4 |
| Pushkar Deshpande  **3. Self Join:** | 6500 | 4 |

select worker.ename “employee”, manager.ename “manager” from emp worker, emp manager

where worker.mgr=manager.empno;

#### Output:

Employee Manager Avinash Pawar Amit Kumar Pushkar Deshpande Amit Kumar Niraj Sharma Amit Kumar

Sumit Patil nitin kulkarni

Amit kumar nitin kulkarni

**DEPARTMENT**

INSERT INTO DEPARTMENT VALUES(&DNO,'&DNAME',&MGRSSN,'&MGRSTARTDATE'); SELECT \* FROM DEPARTMENT;

DNO DNAME MGRSSN

MGRSTAR TD

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

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

10-AUG-

|  |  |  |
| --- | --- | --- |
| 1 RESEARCH | 111111 | 12  10-AUG- |
| 2 ACCOUNTS | 222222 | 10 |
|  |  | 15-APR- |
| 3 AI | 333333 | 12 |
|  |  | 18-MAY- |
| 4 NETWORKS | 111111 | 14 |
| 5 BIGDATA | 666666 | 21-JAN-10 |

5 rows were selected.

**EMPLOYEE**

INSERT INTO EMPLOYEE VALUES('&SSN','&NAME','&ADDRESS','&SEX',&SALARY,'&SUPERSSN',& DNO);

SELECT \* FROM EMPLOYEE;

SSN NAME ADDRESS SEX SALARY SUPERSSN DNO

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

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |
| 111111 | RAJ |  | BENGALURU | M | 700000 |  | 1 |
| 222222 | RASHMI |  | MYSORE | F | 400000 | 111111 | 2 |
| 333333 | RAGAVI |  | TUMKUR | F | 800000 |  | 3 |
| 444444 | RAJESH |  | TUMKUR | M | 650000 | 333333 | 3 |
| 555555 | RAVEESH |  | BENGALURU | M | 500000 | 333333 | 3 |
| 666666 | SCOTT |  | ENGLAND | M | 700000 | 444444 | 5 |
| 777777 | NIGANTH |  | GUBBI | M | 200000 | 222222 | 2 |
| 888888 | RAMYA |  | GUBBI | F | 400000 | 222222 | 3 |
| 999999 | VIDYA |  | TUMKUR | F | 650000 | 333333 | 3 |
| 100000 | GEETHA |  | TUMKUR | F | 800000 |  | 3 |

10 rows selected.

**DLOCATION**

INSERT INTO DLOCATION VALUES(&DNO,'&DLOC');

SELECT \* FROM DLOCATION;

DNO DLOC

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

1 MYSORE

1 TUMKUR 2 BENGALURU

* GUBBI
* DELHI
* BENGALURU

6 rows were selected.

**PROJECT**

INSERT INTO PROJECT VALUES(&PNO,'&PNAME','&PLOCATION',’&DNO’); SELECT \* FROM PROJECT;

|  |  |  |
| --- | --- | --- |
| PNO PNAME | PLOCATION | DNO |
| --------- ---------- | ---------- | --------- |
| 111 IOT | GUBBI | 3 |
| 222 TEXTSPEECH | GUBBI | 3 |
| 333 IPSECURITY | DELHI | 4 |
| 444 TRAFICANAL | BENGALURU | 5 |
| 555 CLOUDSEC | DELHI | 1 |

5 rows were selected.

**WORKS\_ON**

INSERT INTO WORKS\_ON VALUES('&SSN',&PNO,&HOURS);

SELECT \* FROM WORKS\_ON;

|  |  |  |  |
| --- | --- | --- | --- |
| SSN | PNO | HOURS | |
| ---------- 666666 | --------- --------- 333 | | 4 |
| 666666 | 111 | | 2 |
| 111111 | 222 | | 3 |
| 555555 | 222 | | 2 |
| 333333 | 111 | | 4 |
| 444444 | 111 | | 6 |
| 222222 | 111 | | 2 |

8 rows were selected.

* Make a list of all project numbers for projects that involve an employee whose last name is ‘Scott’, either as a worker or as a manager of the department that controls the project.

(SELECT DISTINCT PNO

FROM PROJECT P, DEPARTMENT D, EMPLOYEE E WHERE P.DNO=D.DNO AND

SSN=MGRSSN AND NAME='SCOTT')

UNION

(SELECT DISTINCT P.PNO

FROM PROJECT P, WORKS\_ON W, EMPLOYEE E WHERE P.PNO=W.PNO AND

W.SSN=E.SSN AND NAME='SCOTT');

PNO

--------- 111

333

444

* + Show the resulting salaries if every employee working on the ‘IoT’ project is given a 10 percent raise.

SELECT FNAME, LNAME, 1.1\*SALARY AS INCR\_SAL FROM EMPLOYEE E, WORKS\_ON W, PROJECT P

WHERE E.SSN=W.SSN AND W.PNO=P.PNO AND

P.PNAME=’IOT’;

SSN NAME ADDRESS SEX

SALARY

SUPERSSN DNO

----------

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

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

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 111111 | RAJ | | BENGALURU | | M | | 700000 | 1 |
| 222222 | RASHMI | | MYSORE | | F | | 440000 111111 | 2 |
| 333333 | RAGAVI | | TUMKUR | | F | | 880000 | 3 |
| 444444 | RAJESH | | TUMKUR | | M | | 715000 333333 | 3 |
| 555555 | RAVEESH | | BENGALURU | | M | | 500000 333333 | 3 |
| 666666 | SCOTT | | ENGLAND | | M | | 770000 444444 | 5 |
| 777777 | NIGANTH | | GUBBI | | M | | 200000 222222 | 2 |
|  | RAMYA | | GUBBI | | F | | 400000 222222 | 3 |
| 888888 |  | | | | | | | |
| 999999 | VIDYA | TUMKUR | | F | | 650000 333333 | | 3 |
| 100000 | GEETHA | TUMKUR | | F | | 800000 | | 3 |

1. rows selected.

3. Find the sum of the salaries of all employees of the ‘Accounts’ department, as well as the maximum salary, the minimum salary, and the average salary in this department.

SELECT SUM(SALARY), MAX(SALARY), MIN(SALARY), AVG(SALARY) FROM EMPLOYEE E, DEPARTMENT D

WHERE DNAME='ACCOUNTS' AND D.DNO=E.DNO;

SUM(SALARY) MAX(SALARY) MIN(SALARY) AVG(SALARY)

----------

40000 200000 320000

mysql> create database jn;

Query OK, 1 row affected (0.00 sec)

mysql> use jn;

Database changed

mysql> create table orders(orderID int NOT NULL,custID int NOT NULL,empID VARCHAR(10) NOT NULL,PRIMARY KEY(orderID));

Query OK, 0 rows affected (0.08 sec)

mysql> DESC orders;

+ + + + + + +

| Field | Type | Null | Key | Default | Extra |

+ + + + + + +

| orderID | int(11) | NO | PRI | NULL | |

| custID | int(11) | NO | | NULL | |

| empID | varchar(10) | NO | | NULL | |

+ + + + + + + 3 rows in set (0.00 sec)

mysql> create table customers(custID int NOT NULL,cust\_name VARCHAR(30) NOT NULL,city varchar(20) NOT NULL);

Query OK, 0 rows affected (0.07 sec)

mysql> desc customers;

+ + + + + + +

| Field | Type | Null | Key | Default | Extra |

+ + + + + + +

| custID | int(11) | NO | | NULL | |

| cust\_name | varchar(30) | NO | | NULL | |

| city | varchar(20) | NO | | NULL | |

+ + + + + + + 3 rows in set (0.00 sec)

mysql> INSERT INTO orders VALUES(101,1,'emp1'); Query OK, 1 row affected (0.04 sec)

mysql> INSERT INTO orders VALUES(103,2,'emp2'); Query OK, 1 row affected (0.03 sec)

mysql> INSERT INTO orders VALUES(136,3,'emp3'); Query OK, 1 row affected (0.02 sec)

mysql> INSERT INTO customers VALUES(5,'riya','delhi'); Query OK, 1 row affected (0.05 sec)

mysql> INSERT INTO customers VALUES(7,'raj','goa'); Query OK, 1 row affected (0.03 sec)

mysql> INSERT INTO customers VALUES(8,'priya','pune'); Query OK, 1 row affected (0.04 sec)

mysql> SELECT orders.orderID,customers.cust\_name,customers.city FROM orders INNER JOIN customers ON orders.custID=customers.custID;

Empty set (0.00 sec)

mysql> INSERT INTO customers VALUES(2,'puja','pune'); Query OK, 1 row affected (0.03 sec)

//INNER JOIN

mysql> SELECT orders.orderID,customers.cust\_name,customers.city FROM orders INNER JOIN customers ON orders.custID=customers.custID;

+ + + +

| orderID | cust\_name | city |

+ + + +

| 103 | puja | pune |

+ + + + 1 row in set (0.00 sec)

mysql> INSERT INTO customers VALUES(3,'divya','mumbai'); Query OK, 1 row affected (0.05 sec)

mysql> INSERT INTO customers VALUES(1,'jay','mumbai'); Query OK, 1 row affected (0.04 sec)

mysql> SELECT orders.orderID,customers.cust\_name,customers.city FROM orders INNER JOIN customers ON orders.custID=customers.custID;

+ + + +

| orderID | cust\_name | city |

+ + + +

| 103 | puja | pune |

| 136 | divya | mumbai |

| 101 | jay | mumbai |

+ + + + 3 rows in set (0.00 sec)

mysql> INSERT INTO orders VALUES(171,7,'emp7'); Query OK, 1 row affected (0.04 sec)

mysql> INSERT INTO orders VALUES(172,6,'emp8'); Query OK, 1 row affected (0.05 sec)

mysql> SELECT orders.orderID,customers.cust\_name,customers.city FROM orders INNER JOIN customers ON orders.custID=customers.custID;

+ + + +

| orderID | cust\_name | city |

+ + + +

| 171 | raj | goa |

| 103 | puja | pune |

| 136 | divya | mumbai |

| 101 | jay | mumbai |

+ + + + 4 rows in set (0.00 sec)

// LEFT JOIN

mysql> SELECT orders.orderID, customers.cust\_name FROM orders LEFT JOIN customers ON orders.custID=customers.custID;

+ + +

| orderID | cust\_name |

+ + +

| 101 | jay |

| 103 | puja |

| 136 | divya |

| 171 | raj |

| 172 | NULL |

+ + +

5 rows in set (0.00 sec) mysql> select \* from orders;

| orderID | custID | empID |

+ + + +

|  |  |  |  |
| --- | --- | --- | --- |
| | | 101 | | 1 | emp1 | | |
| | | 103 | | 2 | emp2 | | |
| | | 136 | | 3 | emp3 | | |
| | | 171 | | 7 | emp7 | | |
| | | 172 | | 6 | emp8 | | |

+ + + + 5 rows in set (0.00 sec)

mysql> SELECT orders.orderID,customers.cust\_name,customers.city FROM orders LEFT JOIN customers ON orders.custID=customers.custID;

+ + + +

| orderID | cust\_name | city |

+ + + +

| 101 | jay | mumbai |

| 103 | puja | pune |

| 136 | divya | mumbai |

| 171 | raj | goa |

| 172 | NULL | NULL |

+ + + + 5 rows in set (0.00 sec)

# RIGHT JOIN

mysql> SELECT orders.orderID,customers.cust\_name,customers.city FROM orders RIGHT JOIN customers ON orders.custID=customers.custID;

+ + + +

| orderID | cust\_name | city |

+ + + +

| NULL | riya | delhi |

| 171 | raj | goa |

|  |  |  |
| --- | --- | --- |
| | | NULL | priya | | pune | |
| | | 103 | puja | | pune | |
| | | 136 | divya | | mumbai | |
| | | 101 | jay | | mumbai | |

+ + + + 6 rows in set (0.00 sec)

mysql> select \* from customers;

+ + + +

| custID | cust\_name | city |

+ + + +

| 5 | riya | delhi |

| 7 | raj | goa |

| 8 | priya | pune |

| 2 | puja | pune |

| 3 | divya | mumbai |

| 1 | jay | mumbai |

+ + + + 6 rows in set (0.00 sec)

mysql> SELECT \* FROM orders JOIN customers ON orders.custID=customers.custID;

+ + + + + + +

| orderID | custID | empID | custID | cust\_name | city |

+ + + + + + +

| 171 | 7 | emp7 | 7 | raj | goa |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| | | 103 | | 2 | emp2 | | | 2 | puja | | pune | |
| | | 136 | | 3 | emp3 | | | 3 | divya | | mumbai | |
| | | 101 | | 1 | emp1 | | | 1 | jay | | mumbai | |

+ + + + + + + 4 rows in set (0.00 sec)

//EQUI JOIN

mysql> SELECT orders.orderID,customers.cust\_name,customers.city FROM orders JOIN customers ON orders.custID=customers.custID;

+ + + +

| orderID | cust\_name | city |

+ + + +

| 171 | raj | goa |

| 103 | puja | pune |

| 136 | divya | mumbai |

| 101 | jay | mumbai |

+ + + + 4 rows in set (0.00 sec)

//NON-EQUI JOIN

mysql> SELECT orders.orderID,customers.cust\_name,customers.city FROM orders JOIN customers WHERE orders.custID BETWEEN 3 AND 10;

+ + + +

| orderID | cust\_name | city |

+ + + +

|

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| |  |  |  | | 136 | riya   1. | riya 2. | riya   136 | raj | | delhi  | delhi  | delhi  | goa |  | | |
| |  |  | | 1. | raj 2. | raj   136 | priya | | goa  | goa  | pune |  |  | | |
| |  | | 1. | priya 2. | priya | | pune  | pune |  |  | |  | |
| |  |  | | 136 | puja   1. | puja 2. | puja | | pune  | pune  | pune |  |  | |  |  | |
| | | 136 | divya | | mumbai | | | | |
| | | 171 | divya | | mumbai | | | | |
| | | 172 | divya | | mumbai | | | | |
| | | 136 | jay | | mumbai | | | | |
| | | 171 | jay | | mumbai | | | | |
| | | 172 | jay | | mumbai | | | | |

|

|

|

|

|

+ + + + 18 rows in set (0.00 sec)

**Conclusion:** In this experiment, we have studied SQL queries for suitable database applications using SQL DML Statements and implemented all types of Join, and Sub-Query.

# \*\*\*\*\*\*\*\*\*\*

**Experiment No-7**

# AIM: Write a PL/SQL block to calculate the grade of minimum of 10 students

#### Theory:

#### PL/SQL is a combination of SQL along with the procedural features of programming languages. It was developed by Oracle Corporation in the early 90s to enhance the capabilities of SQL. PL/SQL is one of three key programming languages embedded in the Oracle Database, along with SQL itself and Java. This tutorial will give you a great understanding of PL/SQL to proceed with Oracle database and other advanced RDBMS concepts. PL/SQL programming language provides the following types of decision-making statements.

**Decision-making statements:** PL/SQL provides IF statement to execute a statement or sequence of statements conditionally. There are 3 forms of IF statement –IF-THEN, IF-THEN-ELSE, and IF-THEN-ELSIF.

1. IF-THEN Statement:

Syntax:

IF<condition>THEN Statement(s)

END IF;

* 1. IF-THEN-ELSE Statement

IF condition THEN Statement(s); ELSE

Statement(s);

END IF;

* 1. IF-THEN-ELSIF

This form of IF statement is used when we have to implement logic that has many alternatives.

Syntax: IF condition 1 THEN Statement 1;

ELSIF condition 2 THEN Statement 2;

ELSE

Statement 3; END IF;

#### Program:

The following program calculates the grade of students when the percentage of marks obtained is entered.

Percentage>=80 -Grade A 80>Percentage>=60 -Grade B 60>Percentage>=45 -Grade C 45>Percentage -Fail

DECLARE v\_student Students%rowtype; v\_result Dev\_2000\_result%rowtype; grade varchar2(10);

CURSOR c1 IS SELECT \*FROM Students;

BEGIN

FOR v\_student IN c1 LOOP IF v\_student.class='Computer' THEN IF v\_student.Dev\_2000 <50 THEN grade:='FAIL';

ELSE grade:='PASS'; END IF;

elsif v\_student.class='Student' THEN IF v\_student.Dev\_2000 >=80 THEN grade:='HONOURS';

elsif v\_student.Dev\_2000 >=60 THEN grade:='A'; elsif v\_student.Dev\_2000 >=50 THEN grade:='B'; elsif v\_student.Dev\_2000 >=40 THEN grade:='C'; ELSE grade:='B';

END IF; END IF;

INSERT INTO Dev\_2000\_result VALUES(v\_student.Roll\_no,

v\_student.Oracle, v\_student.Dev\_2000, grade);

END LOOP;

END;

/

Output:

create table Students(roll\_no varchar(20), name varchar(20), section varchar(20),class varchar(20), ORACLE varchar(20),DEV\_2000 varchar(20));

Table created.

SQL> insert into Students values('1','Naveen','A','Computer','89','87');

1 row created.

SQL> insert into Students values('2','Veena','A','Computer','90','56');

1 row was created.

SQL> insert into Students values('3','Kiran','A','Computer','34','43');

1 row was created.

SQL> insert into Students values('4','Ganesh','A','E&TC','88','100');

Enter value for tc: 7

old 1: insert into Students values('4','Ganesh','A','E&TC','88','100')

new 1: insert into Students values('4','Ganesh','A','E7','88','100')

1 row was created.

SQL> insert into Students values('5','Madhuri','A','Civil','56','90');

1 row was created.

SQL> insert into Students values('6','Sanket','A','Computer','78','23');

1 row created.

SQL> insert into Students values('7','Mohit','A','Computer','46','67');

1 row was created.

SQL> insert into Students values('8','Ashutosh','A','IT','50','28');

1 row was created.

SQL> insert into Students values('9','Pragati','A','IT','67','50');

1 row was created.

SQL> insert into Students values('10','Rushikesh','A','Mechanical','89','65');

1 row was created.

SQL> @grade1.sql

PL/SQL procedure successfully completed. SQL> select \*from Dev\_2000\_result;

ROLL\_NO TOTAL PERCENT GRADE

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 1 |  | 89 |  | 87 PASS |  | |
| 2 |  | 90 |  | 56 PASS |  |  |
| 3 |  | 34 |  | 43 FAIL |  |  |
| 4 |  | 8 |  | 10 FAIL |  |  |
| 5 |  | 56 |  | 45 FAIL |  |  |
| 6 |  | 78 |  | 23 FAIL |  |  |
| 7 |  | 46 |  | 67 PASS |  |  |
| 8 |  | 50 |  | 28 PASS |  |  |
| 9 |  | 67 |  | 50 PASS |  |  |

10 89 65 PASS

10 rows selected.

**Conclusion**: In this experiment, we have studied Procedural Language/Structured Query Language decision-making statements and implemented a PL/SQL block to calculate the grade of a minimum of 10 students.

# Experiment No-8

**AIM:** Write a PL/SQL block to implement all types of cursors.

#### Theory:

Cursor:

A cursor is a type of pointer built into PL/SQL for querying the database, retrieving a set of records, and allowing a developer to access the active data set, a row at a time. This allows the programmers to accomplish tasks that require procedural code to be performed on each record in a result set individually.

When a cursor is loaded with multiple rows using a query, the oracle engine opens and maintains a row pointer into the active data set.

The row pointer will be moved within the active data set depending on the user's request. Types of Cursors

1. Implicit cursor
2. Explicit cursor

Implicit cursors are cursors that are opened by the oracle engine for its internal processing. This kind of cursor is called implicit because oracle automatically handles many of the cursor-related operations such as open, fetch, close, etc.

Explicit cursors are the cursors open by the user for processing data as required. It is also known as a user-defined cursor. The user explicitly performs operations such as open, fetch, control, etc. against the cursor.

General Cursor Attributes:

|  |  |
| --- | --- |
| **Name** | **Description** |
| %FOUND | Returns true if the record was fetched successfully,falses otherwise. |
| %NOTFOUND | Returns true if the record was not fetched successfully, false otherwise |
| %ROWCOUNT | Returns numbers of records fetched from cursors at that point  in time |
| %ISOPEN | Return True if the cursor is open, false otherwise |

#### Implicit Cursor:

PL/SQL declared and manages an implicit cursor every time you execute a SQL DML statement such as Insert, Update, or a SELECT INFO statement that returns a single row

from the database. This Kind of cursor is called implicit because Oracle automatically or implicitly handles many of the cursor related operations such as

* Reserving area in memory
* Populating this area with appropriate data
* Processing the data in the memory area
* Related to the memory area when processing is complete

Oracle allows you to access information about the most recently executed implicit cursor by referencing special implicit cursor attributes. These attributes can be used to access information about the status of the most recently executed SQL statement (insert, update, delete, etc.) regardless of the block or from which the SQL statement was executed. All the implicit cursor attributes return NULL if no implicit cursor has been executed in the session. Because the cursor are implicit, they have no name and therefore the keyword ‘SQL’ is used to denote the implicit cursor.

* SQL%FOUND - It is used to determine if any rows were retrieved. This attribute will return TRUE if an INSERT, UPDATE, DELETE statement affected one or more rows or a SELECTED INTO statement returns one or more rows, otherwise it returns FALSE. SQL%FOUND

This attribute is opposite to the SQL %FOUND. It returns true if no rules were found otherwise it returns false.

SQL% ROW COUNT This attribute is used to determine the no of rows affected by INSERT, UPDATE OR DELETE OR SELECT INTO statement.

SQL% is open this attribute always return false because implicit cursor is open and close implicitly before you can reference SQL% is open to check their status.

#### 2. Explicit cursor

An explicit cursor is a SELECT statement that is explicitly defined in the declaration section of code and is also assigned a name. We cannot use an explicit cursor for UPDATE, DELETE AND INSERT statements.

Explicit cursor handling involves steps. Declaring the Cursor

Opening the Cursor

Fetching rows from the cursor one at a time Closing the Cursor

Declaration of the explicit cursor,

Declaring a cursor means giving it a name and specifying the select statement with which the cursor is associated. There is no memory allocation at this point. Declaration of PL/SQL block may declare more than one cursor at a time but of different names.

Syntax:

CURSOR<cursorname> IS SQL select statement;

Opening an explicit cursor: After declaration, the cursor is open with an open statement for processing rows in the cursor. While opening the cursor, actual memory allocation to the cursor takes place. When you open the cursor, then select statement associated with the cursor is executed. It also identifies active data sets that are rows from all involved tables that meet the criteria in the where clause and the join condition.

This syntax OPEN<cursorname>; Fetching records

Syntax of fetch statement FETCH<cursorname>INTO<var\_list>;

#### CLOSING THE CURSOR

After processing the rows in the cursor it is released with a closing statement. The syntax for closed command is

CLOSE<cursorname>;

#### CURSOR FOR LOOP:

The cursor for loop should be preferred when you need to fetch and process each and every record from the cursor.

The cursor for the loop reduces the volume if the code needs to write to fetch data from a cursor.

Syntax:

FOR loop\_index IN <cursorname> LOOP

Executable\_staement;

END LOOP;

The cursor for the loop automatically does the following;

* Implicitly declares its loop index as a %ROWTYPE record.
* Open a cursor
* Fetches a row from the cursor for each loop iteration. After each execution of loops body, PL/SQL performs another fetch. If the % is not found attribute of the cursor evaluates a true, then the loop terminates. The loop never executes its body if the cursor returns no rows.

Close the cursor when all rows have been fetched. we can also terminate the loops with an exit statement but this is not recommended.

PASSING PARAMETERS TO CURSORS

Syntax for declaring parameterized cursor is CURSOR<cursorname>(var\_name datatype)IS

{

SELECT statement

};

You can pass multiple parameters in the cursor OPEN<cursorname>(variable/value/expression);

#### Program:

1. **Implicit Cursor:**

#### Output:

SQL\*Plus: Release 11.2.0.2.0 Production on Thu Jan 1 01:20:36 2009 Copyright (c) 1982, 2010, Oracle. All rights reserved.

SQL> connect

Enter user-name: system Enter password:

Connected.

SQL> edit shubham

SQL> SET SERVEROUTPUT ON; SQL> BEGIN

1. UPDATE N\_ROLLCALL SET ATTENDANCE='a' WHERE NAME='shrikant';
2. IF SQL%FOUND THEN
3. dbms\_output.put\_line('Updated - If Found');
4. END IF;
5. IF SQL%NOTFOUND THEN
6. dbms\_output.put\_line('NOT Updated - If NOT Found');
7. END IF;
8. IF SQL%ROWCOUNT>0 THEN
9. dbms\_output.put\_line(SQL%ROWCOUNT||' Rows Updated');
10. ELSE
11. dbms\_output.put\_line('NO Rows Updated Found');
12. END IF; 14 END;

15 /

Updated - If Found 1 Rows Updated

PL/SQL procedure successfully completed.

SQL> select \* from n\_rollcall;

ROLL\_NO NAME ATTDATE ATTENDANCE

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

1 shrikant 02-JAN-09 a

2 shubham 02-JAN-09 p 3 pragati 02-JAN-09 a

#### Explicit Cursor

SQL> edit sham

SQL> DECLARE

CURSOR EXPLICIT\_CUR is select ROLL\_NO, NAME, ATTDATE from N\_ROLLCALL where ATTENDANCE='a';

tmp EXPLICIT\_CUR %rowtype; BEGIN

dbms\_output.put\_line('HI THIS IS TEST'); OPEN EXPLICIT\_CUR;

Loop exit when EXPLICIT\_CUR%NOTFOUND; FETCH EXPLICIT\_CUR into tmp;

dbms\_output.put\_line('ROLL NO :' || tmp.ROLL\_NO || 'NAME : ' || tmp.NAME || 'DATE :'

|| tmp.ATTDATE); END Loop;

IF EXPLICIT\_CUR%ROWCOUNT>0 THEN

dbms\_output.put\_line(SQL%ROWCOUNT||' ROWS FOUND'); ELSE

dbms\_output.put\_line('NO ROWS FOUND'); END IF;

CLOSE EXPLICIT\_CUR;

END;

/

HI THIS IS A TEST

ROLL NO :1NAME: shrikantDATE:02-JAN-09 ROLL NO :3NAME: pragatiDATE :02-JAN-09 ROLL NO :3NAME: pragatiDATE :02-JAN-09 ROWS FOUND

PL/SQL procedure successfully completed.

#### Cursor For Loop

SQL> edit new SQL> DECLARE

cursor FOR\_CUR is select ROLL\_NO,NAME, ATTDATE from N\_ROLLCALL where ATTENDANCE='ABSENT';

tmp FOR\_CUR %rowtype; BEGIN

FOR tmp IN FOR\_CUR LOOP

dbms\_output.put\_line('ROLL NO :' || tmp.ROLL\_NO ||' '|| 'NAME : ' || tmp.NAME ||' '|| 'DATE :' || tmp.ATTDATE);

END Loop;

END;

/

PL/SQL procedure successfully completed. SQL> select \* from n\_rollcall;

ROLL\_NO NAME ATTDATE ATTENDANCE

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

|  |  |  |  |
| --- | --- | --- | --- |
| 1 shrikant | 02-JAN-09 | A | |
| 1. shubham 2. pragati | 02-JAN-09  02-JAN-09 | a | P |

#### Parametrized cursor

SQL> edit shree SQL> DECLARE

CURSOR PARAM\_CURSOR (ROLL NUMBER) IS SELECT \* FROM N\_ROLLCALL WHERE ROLL\_NO = ROLL;

TMP PARAM\_CURSOR %ROWTYPE; BEGIN

FOR TMP IN PARAM\_CURSOR (101) LOOP

dbms\_output.put\_line('NAME '||TMP.NAME); dbms\_output.put\_line('DATE '||TMP.ATTDATE); dbms\_output.put\_line('ATTENDANCE: '||TMP.ATTENDANCE); END Loop;

END;

/

PL/SQL procedure successfully completed. SQL> select \* from n\_rollcall;

ROLL\_NO NAME ATTDATE ATTENDANCE

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

|  |  |  |
| --- | --- | --- |
| 1 shrikant | 02-JAN-09 | a |
| 2 shubham | 02-JAN-09 | p |
| 3 pragati | 02-JAN-09 | a |
| ROLL\_NO NAME | ATTDATE | ATTENDANCE ------------------------------ |
| 5 rushikesh | 02-JAN-09 | p |
| 4 krishna | 02-JAN-09 | p |

#### CURSOR PROGRAM FOR ELECTRICITY BILL CALCULATION:

SQL> create table bill(name varchar2(10), address varchar2(20), city varchar2(20), unit number(10));

Table created.

SQL> insert into bill values('&name','&addess','&city','&unit'); Enter value for name: yuva

Enter value for addess: srivi Enter value for city: srivilliputur Enter value for unit: 100

old 1: insert into bill values('&name','&addess','&city','&unit') new 1: insert into bill values('yuva','srivi','srivilliputur','100') 1 row created.

SQL> /

Enter value for name: nithya

Enter value for addess: Lakshmi nagar Enter value for city: sivakasi

Enter value for unit: 200

old 1: insert into bill values('&name','&addess','&city','&unit')

new 1: insert into bill values('nithya','Lakshmi nagar','sivakasi','200') 1 row created.

SQL> /

Enter value for name: maya

Enter value for addess: housing board Enter value for city: sivakasi

Enter value for unit: 300

old 1: insert into bill values('&name','&addess','&city','&unit')

new 1: insert into bill values('maya','housing board','sivakasi','300')

1. row created. SQL> /

Enter value for name: jeeva

Enter value for addess: RRR nagar Enter value for city: sivaganagai Enter value for unit: 400

old 1: insert into bill values('&name','&addess','&city','&unit')

new 1: insert into bill values('jeeva','RRR nagar','sivaganagai','400') 1 row created.

SQL> select \* from bill;

NAME ADDRESS CITY UNIT

yuva srivi srivilliputur 100

nithya Lakshmi nagar sivakasi 200

maya housing board Sivakasi 300

jeeva RRR nagar sivaganagai 400

SQL> declare

cursor c is select \* from bill; b bill %ROWTYPE; begin

open c;

dbms\_output.put\_line('Name Address city Unit Amount'); loop

fetch c into b;

if(c%notfound) then 10 exit;

else

if(b.unit<=100) then

dbms\_output.put\_line(b.name||' '||b.address||' '||b.city||' '||b.unit||' '||b.unit\*1); elsif(b.unit>100 and b.unit<=200) then

dbms\_output.put\_line(b.name||' '||b.address||' '||b.city||' '||b.unit||' '||b. unit\*2);

elsif(b.unit>200 and b.unit<=300) then

dbms\_output.put\_line(b.name||' '||b.address||' '||b.city||' '||b.unit||' '||b. unit\*3);

elsif(b.unit>300 and b.unit<=400) then

dbms\_output.put\_line(b.name|

|' dbms\_output.put\_line(b.name|

'||b.address|

|' '||b.address|

'||b.city|| '

'||b.city||

'||b.unit|

|' '||b.unit|

'||b.unit\*

|'

Else dbms\_output.put\_line(b.name|

|'

end if; end if;

end loop;

close c;

|'

'||b.address|

|'

'

'||b.city|| '

|' '||b.unit\*4);

'||b.unit|

|' '||b.unit\*5);

Lakshmi

Amount

100

|  |  |  |  |
| --- | --- | --- | --- |
| end;  / |  | | |
| Name | Address | city | Unit |
| yuva | srivi | srivilliputur | 100 |

nithya nagar Sivakasi 200 400

maya housing board Sivakasi 300 900

jeeva RRR nagar sivaganagai 400 1600

**Conclusion:** In this experiment, we have studied the cursor concept and implemented implicit, explicit, cursor for loop, and parameterized types of cursors.

\*\*\*\*\*\*\*

# Experiment No-9

**AIM:** Write PL/SQL stored procedure and function.

**Theory:** The stored procedure is a group of SQL statements that can be executed repeatedly. Create **a stored procedure** to debit a bank account. The user must enter the account number and amount to be debited. The minimum balance in the account should be 500. Suppose that a table BANK contains the account number (Accno) and balance(balance) fields and has 10 records in it.

**Creating a Procedure:**

A procedure is created with the CREATE OR REPLACE PROCEDURE statement. The simplified syntax for the CREATE OR REPLACE PROCEDURE statement is as follows −

CREATE [OR REPLACE] PROCEDURE procedure\_name

[(parameter\_name [IN | OUT | IN OUT] type [, ...])]

{IS | AS}

BEGIN

< procedure\_body >

END procedure\_name;

Where,

* procedure-name specifies the name of the procedure.
* [OR REPLACE] option allows the modification of an existing procedure.
* The optional parameter list contains the name, mode, and types of the parameters. IN represents the value that will be passed from outside and OUT represents the parameter that will be used to return a value outside of the procedure.
* procedure-body contains the executable part.
* The AS keyword is used instead of the IS keyword for creating a standalone procedure.

**Creating a function:**

A standalone function is created using the CREATE FUNCTION statement.

The simplified syntax for the CREATE OR REPLACE PROCEDURE statement is as follows −

CREATE [OR REPLACE] FUNCTION function\_name

[(parameter\_name [IN | OUT | IN OUT] type [, ...])]

RETURN return\_datatype

{IS | AS}

BEGIN

< function\_body >

END [function\_name];

Where,

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

Ed bal.sql

create or replace procedure chk\_balance (ACC number, Amount number) As LESS\_balance EXCEPTION;

curr\_amount number(8,2); Begin

Select balance INTO curr\_amount FROM Bank Where Accno=Acc;

IF curr\_amount>500 THEN

UPDATE Bank set balance=balance-amount where accno=acc; dbms\_output.put\_line (‘updation successful’);

ELSE

dbms\_output.put\_line (‘Minium balance should be 500’); END IF;

EXCEPTION

When No\_data\_found THEN

Dbms\_output.put\_line(‘Acc number ‘||Acc||’not exists’); END chk\_balance;

/

SQL> @ BAL

Procedure created.

#### Stored function:

Create a stored function that computes the greatest of the three numbers. SQL> ED GREAT

Create or Replace Function Great (A number, B number, C number) Return number AS

Begin

IF (A>B AND A>C) THEN

Return A;

ELSEIF (A<B AND B>C) THEN

Return B;

ELSE

Return C;

END IF;

END Great;

/

On execution, it will create a function ‘Great’.

**Step 1:** SQL> @ GREAT

**Step 2(optional):** SQL>SHOW ERRORS;

**Step 3:** SQL>Select GREAT (7,10,12) FROM DUAL;

On execution of the above statement, it will run the function ‘Great’ and return the correct value.

**Conclusion:** In this experiment, we have studied the concept of procedure and functions and implemented PL/SQL stored procedure and function

# \*\*\*\*\*\*\*\*

**Experiment No-10**

# AIM: Write a database Trigger (Row-level and Statement level)

**Theory:**

The trigger is a set-off action that gets executed automatically when a specified change operation (SQL INSERT, UPDATE or DELETE statement) is performed on a particular table.

Triggers are stored programs, which are automatically executed or fired when some events occur. Triggers are, in fact, written to be executed in response to any of the following events

The syntax for Creating a Trigger

**CREATE** TRIGGER name BEFORE|AFTER **INSERT**|**UPDATE**|**DELETE** ON tablename FO R

EACH ROW code

c. after delete trigger mysql> delimiter $$

mysql> **CREATE** TRIGGER myTrigger

-> BEFORE **DELETE** ON employee

-> FOR EACH ROW

-> **BEGIN**

-> **INSERT** into transaction\_log

-> (user\_id, description)

-> VALUES (user(), 'Employee deleted ');

-> END$$

Query OK, 0 rows affected (0.00 sec) mysql> delimiter ;

mysql> **delete from** employee **where** id = 1; Query OK, 1 row affected (0.00 sec)

**f.** AFTER UPDATE trigger

mysql> delimiter $$ mysql>

mysql> **CREATE** TRIGGER myTrigger

-> AFTER **UPDATE** ON employee FOR EACH ROW ->

#### BEGIN

-> **INSERT** into transaction\_log

-> (user\_id, description)

-

* VALUES (user(), CONCAT('Adjusted account ',NEW.id,' from ',OLD.salar y, ' to ', NEW.salary));

-> END$$

Query OK, 0 rows affected (0.00 sec)

mysql> delimiter ;

mysql> **update** Employee set salary = salary + 1000; Query OK, 8 rows affected (0.00 sec) Rows

matched: 8 Changed: 8 Warnings: 0 mysql> drop trigger myTrigger; Query OK, 0 rows affected (0.01 sec)

1. Before insert trigger mysql> delimiter $$ mysql>

mysql> CREATE TRIGGER myTrigger

-> BEFORE INSERT ON employee

-> FOR EACH ROW

-> BEGIN

-> IF NEW.salary > 500 THEN

-> SET NEW.first\_name='Y';

-> ELSE

-> SET NEW.first\_name='N';

-> END IF;

-> END$$

Query OK, 0 rows affected (0.00 sec) mysql>

mysql> delimiter ; mysql>

mysql> update Employee set salary = 400, first\_name=''; Query OK, 8 rows affected (0.00 sec)

Rows matched: 8 Changed: 8 Warnings: 0

1. Before update trigger mysql> delimiter $$

mysql> **CREATE** TRIGGER myTrigger

-> BEFORE **UPDATE** ON employee

-> FOR EACH ROW

-> **BEGIN**

-> IF NEW.salary <5000 THEN

-> SET NEW.first\_name=CONCAT('NEW.first\_name','\*'); -> ELSE

-> SET NEW.last\_name=CONCAT('NEW.last\_name','\*');

-> **END** IF;

-> END$$

Query OK, 0 rows affected (0.01 sec) mysql> delimiter ;

mysql>

mysql> **update** employee set salary = 1000, first\_name='new first Name'; Query OK, 8 rows affected (0.00 sec) Rows matched: 8 Changed: 8 Warnings: 0

1. Change the NEW value based on the input in a BEFORE INSERT trigger mysql> DELIMITER //

mysql>

mysql> CREATE TRIGGER myTrigger BEFORE INSERT ON employee

-> FOR EACH ROW

-> BEGIN

-> IF NEW.salary IS NULL OR NEW.salary = 0 THEN

-> SET NEW.salary = 100;

-> ELSE

-> SET NEW.salary = NEW.salary + 100;

-> END IF;

->

-> END

-> //

Query OK, 0 rows affected (0.02 sec)

mysql> DELIMITER ; mysql>

mysql> delimiter ; mysql>

mysql> update Employee set salary = 400, first\_name=''; Query OK, 8 rows affected (0.00 sec)

Rows matched: 8 Changed: 8 Warnings: 0

e. Before update trigger mysql> delimiter $$

mysql> **CREATE** TRIGGER myTrigger

-> BEFORE **UPDATE** ON employee

-> FOR EACH ROW

-> **BEGIN**

-> IF NEW.salary <5000 THEN

-> SET NEW.first\_name=CONCAT('NEW.first\_name','\*'); -> ELSE

-> SET NEW.last\_name=CONCAT('NEW.last\_name','\*');

-> **END** IF;

-> END$$

Query OK, 0 rows affected (0.01 sec) mysql> delimiter ;

mysql>

mysql> **update** employee set salary = 1000, first\_name='new first Name'; Query OK, 8 rows affected (0.00 sec) Rows matched: 8 Changed: 8 Warnings: 0

1. Change the NEW value based on the input in a BEFORE INSERT trigger mysql> DELIMITER //

mysql>

mysql> CREATE TRIGGER myTrigger BEFORE INSERT ON employee

-> FOR EACH ROW

-> BEGIN

-> IF NEW.salary IS NULL OR NEW.salary = 0 THEN

-> SET NEW.salary = 100;

-> ELSE

-> SET NEW.salary = NEW.salary + 100;

-> END IF;

->

-> END

-> //

Query OK, 0 rows affected (0.02 sec) mysql> DELIMITER ;

1. Row level trigger

**create** table test( percent decimal

);

delimiter $$

**CREATE** TRIGGER test\_before\_insert BEFORE **INSERT** ON test FOR EACH ROW

#### BEGIN

IF NEW.percent < 0.0 OR NEW.percent > 1.0 THEN

SET NEW.percent = NULL;

**END** IF;

END$$ delimiter ;

1. Viewing Triggers

mysql> DELIMITER // mysql>

mysql> **CREATE** TRIGGER mytrigger BEFORE **UPDATE** ON employee

-> FOR EACH ROW

-> **BEGIN**

->

-> IF NEW.id > 5 THEN

-> SET NEW.first\_name = NEW.first\_name+ " \*";

-> **END** IF;

->

-> IF NEW.salary IS NULL OR NEW.salary = 0 THEN

-> SET NEW.salary = 100;

-> ELSE

-> SET NEW.salary = NEW.salary + 100;

-> **END** IF;

-> **END**

-> //

Query OK, 0 rows affected (0.00 sec) mysql> DELIMITER ;

mysql>

mysql> SELECT \* FROM INFORMATION\_SCHEMA.TRIGGERS;

**Conclusion:** In this experiment, we have studied the PL/SQL concept of the trigger and implemented row-level and statement-level triggers.

# Experiment No-11

**AIM:** Design and Develop MongoDB Queries using CRUD operations.

**Theory:** MongoDB CRUD Operations

* [Create Operations](https://docs.mongodb.com/manual/crud/#create-operations)
* [Read Operations](https://docs.mongodb.com/manual/crud/#read-operations)
* [Update Operations](https://docs.mongodb.com/manual/crud/#update-operations)
* [Delete Operations](https://docs.mongodb.com/manual/crud/#delete-operations)

##### Create Operations

Create or insert operations and add new [documents](https://docs.mongodb.com/manual/core/document/#bson-document-format) to a [collection.](https://docs.mongodb.com/manual/core/databases-and-collections/#collections) 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()](https://docs.mongodb.com/manual/reference/method/db.collection.insertOne/#db.collection.insertOne) *New in version 3.2*
* [db.collection.insertMany()](https://docs.mongodb.com/manual/reference/method/db.collection.insertMany/#db.collection.insertMany) *New in version 3.2*

In MongoDB, insert operations target a single [collection](https://docs.mongodb.com/manual/reference/glossary/#term-collection). All write operations in MongoDB are [atomic](https://docs.mongodb.com/manual/core/write-operations-atomicity/) on the level of a single [document.](https://docs.mongodb.com/manual/core/document/)

##### Read Operations

Read operations retrieves [documents](https://docs.mongodb.com/manual/core/document/#bson-document-format) from a [collection](https://docs.mongodb.com/manual/core/databases-and-collections/#collections); i.e. queries a collection for documents. MongoDB provides the following methods to read documents from a collection:

* [db.collection.find()](https://docs.mongodb.com/manual/reference/method/db.collection.find/#db.collection.find)

You can specify [query filters or criteria](https://docs.mongodb.com/manual/tutorial/query-documents/#read-operations-query-argument) that identify the documents to return.

##### Update Operations

Update operations modify existing [documents](https://docs.mongodb.com/manual/core/document/#bson-document-format) in a [collection.](https://docs.mongodb.com/manual/core/databases-and-collections/#collections) MongoDB provides the following methods to update documents of a collection:

* [db.collection.updateOne()](https://docs.mongodb.com/manual/reference/method/db.collection.updateOne/#db.collection.updateOne) *New in version 3.2*
* [db.collection.updateMany()](https://docs.mongodb.com/manual/reference/method/db.collection.updateMany/#db.collection.updateMany) *New in version 3.2*
* [db.collection.replaceOne()](https://docs.mongodb.com/manual/reference/method/db.collection.replaceOne/#db.collection.replaceOne) *New in version 3.2*

In MongoDB, update operations target a single collection. All write operations in MongoDB are [atomic](https://docs.mongodb.com/manual/core/write-operations-atomicity/) on the level of a single document.

You can specify criteria, or filters, that identify the documents to update. These [filters](https://docs.mongodb.com/manual/core/document/#document-query-filter) use the same syntax as reading operations.

For examples, see [Update Documents.](https://docs.mongodb.com/manual/tutorial/update-documents/)

##### Delete Operations

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

* [db.collection.deleteOne()](https://docs.mongodb.com/manual/reference/method/db.collection.deleteOne/#db.collection.deleteOne) *New in version 3.2*
* [db.collection.deleteMany()](https://docs.mongodb.com/manual/reference/method/db.collection.deleteMany/#db.collection.deleteMany) *New in version 3.2*

In MongoDB, delete operations target a single [collection](https://docs.mongodb.com/manual/reference/glossary/#term-collection). All write operations in MongoDB are [atomic](https://docs.mongodb.com/manual/core/write-operations-atomicity/) on the level of a single document.

You can specify criteria, or filters, that identify the documents to remove. These [filters](https://docs.mongodb.com/manual/core/document/#document-query-filter) use the same syntax as reading operations.

* 1. **The Use Command:** MongoDB **uses DATABASE\_NAME** is used to create a database. The command will create a new database; if it doesn't exist otherwise it will return the existing database.

Syntax: Use DATABASE\_NAME;

Example: If you want to create a database with the name **<mydb>**, then **use the database**

the 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, then 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"})

>show dbs

local 0.78125GB mydb 0.23012GB

test 0.23012GB

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

1. **MongoDB Drop Database:**
   1. **The Drop Database:** MongoDB **db.dropDatabase ()** command is used to drop an existing database.

Syntax:

Basic syntax of **dropDatabase ()** command is as follows: db.dropDatabase()

This will delete the selected database. If you have not selected any database, then it will delete the default 'test' database

Example:

First, check the list of available databases by using the command **show dbs**

>show dbs

local 0.78125GB mydb 0.23012GB test 0.23012GB

If you want to delete new database **<mydb>**, then **dropDatabase()** command would be as follows:

>use mydb

switched to db mydb

>db.dropDatabase()

>{ "dropped" : "mydb", "ok" : 1 }

Now check list of databases

>show dbs

local 0.78125GB test 0.23012GB

1. **Basic Operations with the Shell:** We can use the four basic operations, create, read, update, and delete (CRUD) to manipulate and view data in the shell.
   1. **Create:** The insert function adds a document to a collection. For example, suppose we want to store a blog post. First, we’ll create a local variable called post which is a JavaScript object representing our document. It will have the keys "title", "content", and "date". (the date that it was published):
      * post = {"title" : "My Blog Post",

... "content": "Here's my blog post.",

... "date" : new Date()}

{

"title" : "My Blog Post",

"content" : "Here's my blog post.",

"date" : ISODate("2012-08-24T21:12:09.982Z")

}

This object is a valid MongoDB document, so we can save it to the blog collection using the insert method:

* db.blog.insert(post)
  1. **Read:** Find and findOne can be used to query a collection. If we just want to see one document from a collection, we can use findOne.
     + db.blog.findOne()

{

"\_id" : ObjectId("5037ee4a1084eb3ffeef7228"), "title" : "My Blog Post",

"content" : "Here's my blog post.",

"date" : ISODate("2012-08-24T21:12:09.982Z")

}

Find and findOne can also be passed criteria in the form of a query document. This will restrict the documents matched by the query. The shell will automatically display up to 20 documents matching a find, but more can be fetched. See Chapter 4 for more information on querying.

* 1. **Update:** If we would like to modify our post, we can use an update. The update takes (at least) two parameters: the first is the criteria to find which document to update, and the second is the new document. Suppose we decide to enable comments on the blog post we created earlier. We’ll need to add an array of comments as the value for a new key in our document.

The first step is to modify the variable post and add a "comments" key:

* + - post.comments = [] [ ]

Then we perform the update, replacing the post titled “My Blog Post” with our new version of the document:

* + - db.blog.update({title : "My Blog Post"}, post)

Now the document has a "comments" key. If we call find again, we can see the new key:

* + - db.blog.find()

{

"\_id" : ObjectId("5037ee4a1084eb3ffeef7228"), "title" : "My Blog Post",

"content" : "Here's my blog post.",

"date" : ISODate("2012-08-24T21:12:09.982Z"),

"comments" : [ ]

}

* 1. **Delete:** Remove permanently deletes documents from the database. Called with no parameters, it removes all documents from a collection. It can also take a document specifying criteria for removal. For example, this would remove the post we just created:
     + db.blog.remove({title : "My Blog Post"})

[Sunita@localhost bin]$ ./mongo MongoDB shell version: 2.6.1 connecting to: test

* + show databases;

VBK 0.078GB

admin (empty)

local 0.078GB newsletter 0.078GB

* use COEM // Created a new Database named "COEM" switched to db COEM
* db.createCollection("TE") // Created new Collection named as "TE"

{ "ok" : 1 }

* db.TE.insert({Roll:1,Name:"ABC",Address:"Pune",Per:76}) WriteResult({ "nInserted" : 1 })
* db.TE.insert({Roll:2,Name:"PQR",Address:"Pune",Per:75}) WriteResult({ "nInserted" : 1 })
* db.TE.insert({Roll:3,Name:"LMN",Address:"Hadapsar",Per:70}) WriteResult({ "nInserted" : 1 })
* db.TE.find({})

{ "\_id" : ObjectId("541963be2741c7552caef0a9"), "Roll" : 1, "Name" : "ABC", "Address" : "Pune", "Per" : 76 }

{ "\_id" : ObjectId("541963cb2741c7552caef0aa"), "Roll" : 2, "Name" : "PQR", "Address" : "Pune", "Per" : 75 }

{ "\_id" : ObjectId("541963dc2741c7552caef0ab"), "Roll" : 3, "Name" : "LMN", "Address" : "Hadapsar", "Per" : 70 }

* db.TE.find({})

{ "\_id" : ObjectId("541963be2741c7552caef0a9"), "Roll" : 1, "Name" : "ABC", "Address" : "Pune", "Per" : 76 }

{ "\_id" : ObjectId("541963cb2741c7552caef0aa"), "Roll" : 2, "Name" : "PQR", "Address" : "Pune", "Per" : 75 }

{ "\_id" : ObjectId("541963dc2741c7552caef0ab"), "Roll" : 3, "Name" : "LMN", "Address" : "Hadapsar", "Per" : 70 }

* db.TE.update({Roll:2},{$set:{Name:"Wagholi"}}) WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
* db.TE.find({})

{ "\_id" : ObjectId("541963be2741c7552caef0a9"), "Roll" : 1, "Name" : "ABC", "Address" : "Pune", "Per" : 76 }

{ "\_id" : ObjectId("541963cb2741c7552caef0aa"), "Roll" : 2, "Name" : "Wagholi", "Address" : "Pune", "Per" : 75 }

{ "\_id" : ObjectId("541963dc2741c7552caef0ab"), "Roll" : 3, "Name" : "LMN", "Address" : "Hadapsar", "Per" : 70 }

* db.TE.remove({Roll:3}) WriteResult({ "nRemoved" : 1 })
* db.TE.find({})

{ "\_id" : ObjectId("541963be2741c7552caef0a9"), "Roll" : 1, "Name" : "ABC", "Address" : "Pune", "Per" : 76 }

{ "\_id" : ObjectId("541963cb2741c7552caef0aa"), "Roll" : 2, "Name" : "Wagholi", "Address" : "Pune", "Per" : 75 }

* db.TE.drop() true
* db.TE.find({})
* show collections system.indexes

#### Output:

* show databases;

COEM 0.078GB

VBK 0.078GB

admin (empty)

local 0.078GB newsletter 0.078GB

* db.dropDatabase("COEM")

2014-09-17T16:17:18.278+0530 dropDatabase doesn't take arguments at src/mongo/shell/db.js:141

* db.dropDatabase()

{ "dropped" : "COEM", "ok" : 1 }

* show databases;

VBK 0.078GB

admin (empty)

local 0.078GB newsletter 0.078GB

**Conclusion:** In this experiment, we have studied NoSQL and implemented CRUD operations for MongoDB

\*\*\*\*\*\*\*\*

# Content Beyond Syllabus Experiment

**12. Cassendra Query Execution**

**Aim:** Cassandra case study

**Theory:** Apache Cassandra is a highly scalable, high-performance distributed database designed to handle large amounts of data across many commodity servers, providing high availability with no single point of failure. Apache Cassandra is an open-source NoSQL distributed database trusted by thousands of companies for scalability and high availability without compromising performance It is a type of NoSQL database.

A NoSQL database (sometimes called as Not Only SQL) is a database that provides a mechanism to store and retrieve data other than the tabular relations used in relational databases. These databases are schema-free, support easy replication, have simple API, are eventually consistent, and can handle huge amounts of data.

## Features of Cassandra

Cassandra has become so popular because of its outstanding technical features. Given below are some of the features of Cassandra:

* Elastic scalability − Cassandra is highly scalable; it allows to add of more hardware to accommodate more customers and more data as per requirement.
* Always on architecture − Cassandra has no single point of failure and it is continuously available for business-critical applications that cannot afford a failure.
* Fast linear-scale performance − Cassandra is linearly scalable, i.e., it increases your throughput as you increase the number of nodes in the cluster. Therefore it maintains a quick response time.
* Flexible data storage − Cassandra accommodates all possible data formats including structured, semi-structured, and unstructured. It can dynamically accommodate changes to your data structures according to your need.
* Easy data distribution − Cassandra provides the flexibility to distribute data where you need it by replicating data across multiple data centers.
* Transaction support − Cassandra supports properties like Atomicity, Consistency, Isolation, and Durability (ACID).
* Fast writes − Cassandra was designed to run on cheap commodity hardware. It performs blazingly fast writes and can store hundreds of terabytes of data, without sacrificing the read efficiency.

### C:\apache-cassandra-3.11.6\bin>cqlsh Connected to Test Cluster at 127.0.0.1:9042.

[cqlsh 5.0.1 | Cassandra 3.11.6 | CQL spec 3.4.4 | Native protocol v4] Use HELP for help.

(6 rows)

cqlsh> CREATE KEYSPACE test

... ... WITH REPLICATION = { 'class' : 'NetworkTopologyStrategy', 'datacenter1' : 3 }

... ... AND DURABLE\_WRITES = false;

SyntaxException: line 2:0 mismatched input '.' expecting K\_WITH (CREATE KEYSPACE test[.]...)

cqlsh> SELECT \* FROM system\_schema.keyspaces;

keyspace\_name | durable\_writes | replication

+ +

system\_auth | True | {'class': 'org.apache.cassandra.locator.SimpleStrategy', 'replication\_factor': '1'}

system\_schema | True | {'class': 'org.apache.cassandra.locator.LocalStrategy'}

tutorialspoint | True | {'class': 'org.apache.cassandra.locator.SimpleStrategy', 'replication\_factor': '3'}

system\_distributed | True | {'class': 'org.apache.cassandra.locator.SimpleStrategy', 'replication\_factor': '3'}

system | True | {'class': 'org.apache.cassandra.locator.LocalStrategy'}

system\_traces | True | {'class': 'org.apache.cassandra.locator.SimpleStrategy', 'replication\_factor': '2'}

cqlsh> USE tutorialspoint;

cqlsh:tutorialspoint> INSERT INTO emp (emp\_id, emp\_name, emp\_city,

... emp\_phone, emp\_sal) VALUES(1,'ram', 'Hyderabad', 9848022338, 50000);

InvalidRequest: Error from server: code=2200 [Invalid query] message="unconfigured table emp"

cqlsh:tutorialspoint> CREATE TABLE emp(

... emp\_id int PRIMARY KEY,

... emp\_name text,

... emp\_city text,

... emp\_sal varint,

... emp\_phone varint

... );

cqlsh:tutorialspoint> INSERT INTO emp (emp\_id, emp\_name, emp\_city,

... emp\_phone, emp\_sal) VALUES(1,'ram', 'Hyderabad', 9848022338, 50000); cqlsh:tutorialspoint> INSERT INTO emp (emp\_id, emp\_name, emp\_city,

... emp\_phone, emp\_sal) VALUES(2,'robin', 'Hyderabad', 9848022339, 40000); cqlsh:tutorialspoint> INSERT INTO emp (emp\_id, emp\_name, emp\_city,

... emp\_phone, emp\_sal) VALUES(3,'rahman', 'Chennai', 9848022330, 45000); cqlsh:tutorialspoint> SELECT \* FROM emp;

emp\_id | emp\_city | emp\_name | emp\_phone | emp\_sal

+ + + +

1 | Hyderabad | ram | 9848022338 | 50000

2 | Hyderabad | robin | 9848022339 | 40000

3 | Chennai | rahman | 9848022330 | 45000

(3 rows)

cqlsh:tutorialspoint> UPDATE emp SET emp\_city='Delhi',emp\_sal=50000

... WHERE emp\_id=2; cqlsh:tutorialspoint> select \* from emp;

emp\_id | emp\_city | emp\_name | emp\_phone | emp\_sal

+ + + +

1 | Hyderabad | ram | 9848022338 | 50000

2 | Delhi | robin | 9848022339 | 50000

3 | Chennai | rahman | 9848022330 | 45000

(3 rows)

cqlsh:tutorialspoint> select \* from emp;

emp\_id | emp\_city | emp\_name | emp\_phone | emp\_sal

+ + + +

1 | Hyderabad | ram | 9848022338 | 50000

2 | Delhi | robin | 9848022339 | 50000

3 | Chennai | rahman | 9848022330 | 45000

(3 rows)

cqlsh:tutorialspoint> SELECT emp\_name, emp\_sal from emp;

emp\_name | emp\_sal

+

ram | 50000

robin | 50000

rahman | 45000

(3 rows)

cqlsh> DELETE emp\_sal FROM emp WHERE emp\_id=3;

InvalidRequest: Error from server: code=2200 [Invalid query] message="No keyspace has been specified. USE a keyspace, or explicitly specify keyspace.tablename"

cqlsh> USE tutorialspoint;

cqlsh:tutorialspoint> DELETE FROM emp WHERE emp\_id=3; cqlsh:tutorialspoint> select \* from emp;

emp\_id | emp\_city | emp\_name | emp\_phone | emp\_sal

+ + + +

1 | Hyderabad | ram | 9848022338 | 50000

2 | Delhi | robin | 9848022339 | 50000

(2 rows)

**Conclusion:** In this experiment, we have studied Cassandra and implemented Cassandra queries creating keyspace, insert, select, and delete operations.