EXPERIMENT 1

AIM:

Identify the case study, detail statement of problem and design ER /EER model.

THEORY:

An Entity Relationship (ER) Diagram is a type of flowchart that illustrates how "entities" such as people, objects or concepts relate to each other within a system. ER Diagrams are most often used to design or debug relational databases in the fields of software engineering, business information systems, education and research. Also known as ERDs or ER Models, they use a defined set of symbols such as rectangles, diamonds, ovals and connecting lines to depict the interconnectedness of entities, relationships and their attributes. They mirror grammatical structure, with entities as nouns and relationships as verbs.
ER Diagrams are composed of entities, relationships and attributes. They also depict cardinality, which defines relationships in terms of numbers.

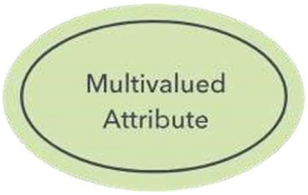
Entity: A definable thing—such as a person, object, concept or event—that can have data stored about it.

Relationship: How entities act upon each other or are associated with each other. Think of relationships as verbs. Relationships are typically shown as diamonds or labels directly on the connecting lines.

Attribute: A property or characteristic of an entity. Often shown as an oval or circle.

- Simple: Means the attribute value is atomic and can't be further divided, such as a phone number.

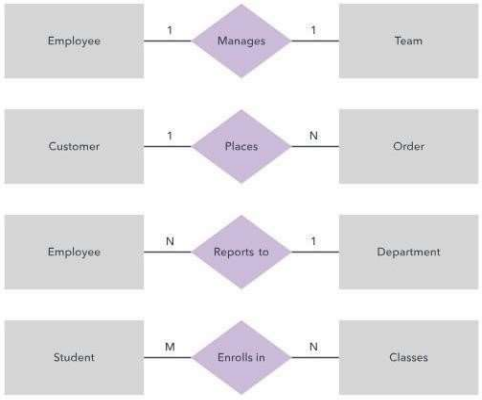
- Multivalued Attribute: More than one attribute value is denoted, such as multiple phone numbers for a person.



- Derived Attribute: Attributed is calculated or otherwise derived from another attribute, such as age from a birthdate.

- Composite Attribute: Sub-attributes spring from an attribute.

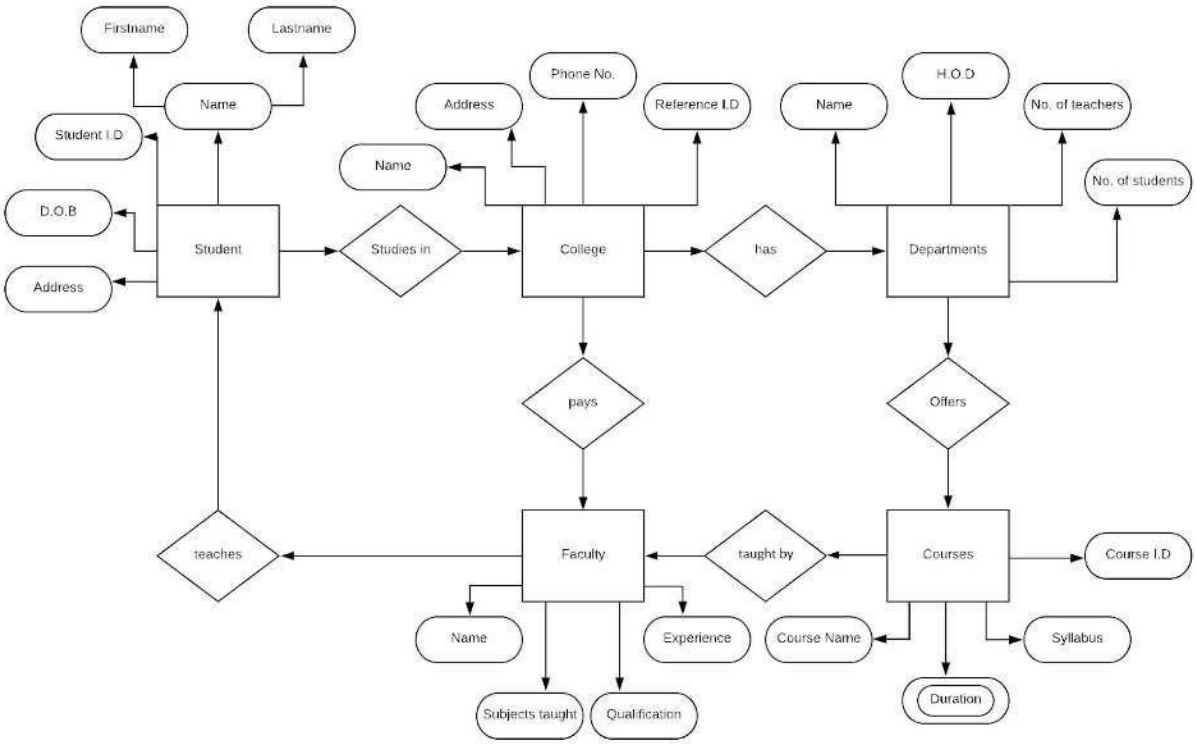
Cardinality: Defines the numerical attributes of the relationship between two entities or entity sets. The three main cardinal relationships are one-to-one, one-to-many, and many-many. A one-to-one example would be one student associated with one mailing address. A one-to-many example (or many-to-one, depending on the relationship direction): One student registers for multiple courses, but all those courses have a single line back to that one student. Many-to-many example: Students as a group are associated with multiple faculty members, and faculty members in turn are associated with multiple students.



PROBLEM STATEMENT:

Design a database for a college. Many students seek admission in the college. The college has a number of departments and students can be enrolled to these departments.The department also offers a number of courses to the students, each with a different duration from the other.
Each department has its H.O.D and many teachers under him.The syllabus of each course is also defined. Teachers are recruited by the college for teaching the said courses to the students. The teachers may have different qualifications and experience. They may also teach different subjects if required. Each student in the college has a unique ID. We need to store the names of the students studying in the college, their residential address, date of birth. We also need to store information about the college like the name, address, contact number, reference ID, departments of the college, name of the H.O.D, number of teachers and students in the department. Also, the courses offered by the departments, the syllabus, the duration and the course ID. We can also store information about the teachers like their qualification, experience, name and subjects taught.

ER DIAGRAM:



DESCRIPTION:

Shown above is the ER Diagram of a college. We have shown various entities and their relationship with the college. Each entity has a number of attributes associated with it. For example, the entity ‘College’ has various attributes such as name, address, contact number and reference ID. The reference ID can be used as a primary key for the college. Similarly, student ID, course ID can be used as primary keys for the entities Student and Course respectively. Various relationships between the different entities have also been shown in the ER diagram above. For example Student *studies in* College, Department *offers* courses, Faculty *teach*es Student are some of the relationships shown in the ER diagram.

CONCLUSION:

In this experiment, we have seen the components of an ER Diagram such as Entities, Attributes, Relationships and also the types of attributes. We have drawn an ER Diagram of a college using online tools and were able to successfully show the relationships between the various entities of the ER Diagram. Thus we have successfully studied and drawn an Entity-Relationship Diagram.

Experiment 2

AIM:

Mapping of ER/EER model to Relational Schema Model.

REQUIREMENTS:

MS word, Browser - Google Chrome, ERDPlus web application.

THEORY:

The relational database model is a logical model for a database in which data are logically organized in two-dimensional tables referred to as relations. A relation is defined as a collection of data representing multiple occurrences of an object, event, or agent. Relational databases are often perceived to be a collection of tables. Consistent with a tabular representation, a *relation* consists of rows and columns. Rows are referred to as *tuples* and columns are referred to as *attributes.*

Now, we will use this five-step process to map an E-R diagram into a logical database model—in this case a well-constrained relational database implementation.

1. *Create a separate relational table for each entity.*

It is generally useful first to specify the database schema before proceeding to expansion of the relations to account for specific tuples.

2. *Determine the primary key for each of the relations. The primary key must uniquely identify any row within the table.*
3. *Determine the attributes for each of the entities.*

CLIENT

Client_No	Name	Street_Address	City	State	Zip_Code	Contact	Phone_Number

WORK_COMPLETED

Employee_No	Date	Client_No	Hours

EMPLOYEE

Employee_Number	Soc_Sec_No	Name	Supervisor_No	Billing_Rate	Pay_Rate

TRAINING_COMPLETED

Employee_No	Date	Hours	Train_Code

RELEASE_TIME

Employee_No	Date	Hours	Vacation_Sick

A complete E-R diagram includes specification of all attributes, including the key attribute.

With a single attribute specified as the key, this is a very straightforward matching between the key attribute specified in the E-R diagram and the corresponding attribute in the relation. For a composite key, we can simply break it down into its component sub-attributes. For instance, in the implementation of the WORK_COMPLETED relation, Employee_No, Date, and Client_No would be three distinct attributes in the relation, but would also be defined as the key via a combination of the three.

4. Implement the relationships among the entities. This is accomplished by ensuring that the primary key in one table also exists as an attribute in every table (entity)for which there is a relationship specified in the entity-relationship diagram.

References to the key attributes in one entity are captured through the inclusion of a corresponding attribute in the other entity participating in the relationship. Let's take a quick look at how the different categories of relationships (i.e., cardinality constraints) affect the mapping to a relational schema.

MAPPING OF BINARY 1:1 RELATIONSHIP TYPE

There are three possible approaches: A)

Foreign Key Approach

- Identify the relations that correspond to the entity types participating in relation.
- Include as foreign keys, in the relation of one entity type,the primary keys of the other entity type.

B) Merged Relation Option

- Merge the two entity types and the relationship into a single relation. ● This may be appropriate when both participation are total. C) Cross-reference or relationship relation option:
- Create a third relation for the purpose of cross-referencing the primary keys of the two relations representing the entity types.

MAPPING OF BINARY 1:N RELATIONSHIP TYPE

- Add as foreign keys, to the relation of the entity type at the N side,the primary keys of the entity type at the 1 side(don't duplicate records)
- Include also the simple attributes of the relationship type

MAPPING OF BINARY M:N RELATION TYPE

- Create a new relation.
- Add as foreign keys the primary keys of the participating entity type to the relation type ● Include the simple attributes of the relation type.

MAPPING WEAK ENTITY SETS

- Create a new relation.
- Include simple attributes.
- Add the owner's primary key attributes, as foreign key attributes.
- Declare into a primary key the partial keys of the weak entity type combined with those imported from the owner.

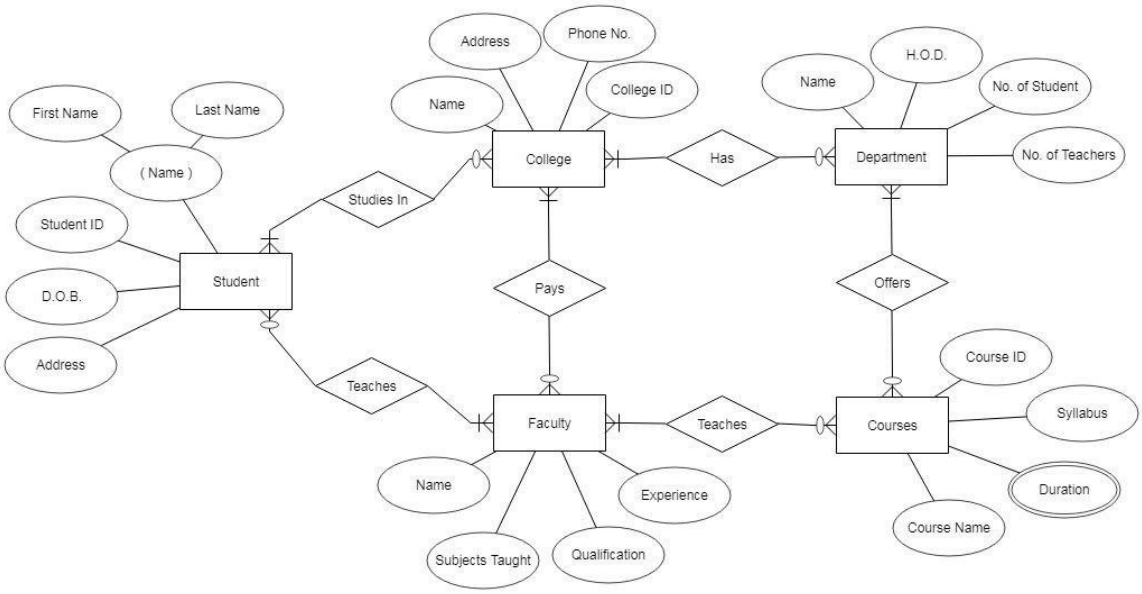
To ensure referential integrity,the attribute that is pointed to, if changed or deleted, could cause an attribute to have a nonmatching value at the source of the arrow. *5. Determine the attributes, if any, for each of the relationship tables.*

PROBLEM STATEMENT:

TECMPN1/C2 **Jatin Jawale/28**

Design a database for Library Management in a College. When students issue a book from the college there are many things to be noted for efficient communication and maintenance between the student and the library department also there are certain attributes for each book present in the library like the publisher, author, price and category type etc. So accordingly we will design an ER diagram for all the components that are required in our Library management system.

ER DIAGRAM:



RELATIONAL SCHEMA MODEL:

Student

Student ID	Name	D.O.B	Address
1001	Reeya Nand	01-08-1998	Mumbai
1002	Jatin Jawale	02-10-1998	Mumbai
1003	Sanika Jog	04-04-1998	Mumbai
1004	Shreya Bhat	14-04-1998	Pune
1005	Sanket Birwa	10-11-1998	Pune

College

College ID	Name	Address	Ph. No.
532	Atharva	Mumbai	22887416
789	Cummins	Pune	27709278
246	DGET	Thane	28369584
698	Garware	Pune	25282513

Department

Name	H.O.D.	No. of Students	No. of Teachers
Engineering	M. Patil	520	25
Medical	V. Shah	360	15
Arts	S. Singh	280	10

Faculty

Name	Qualification	Subj. taught	Experience
K. Soni	M.E.	Mechanics	10yrs
P. Dagde	M.B.B.S.	Anatomy	8yrs
J. Joe	M.P.Th	Zoology	6yrs
L. Dutta	L.L.B.	Law	7yrs
R. Wilson	M.A.	Mathematics	12yrs

Courses

Course ID	Name	Syllabus	Duration
1100	Electrical	MU	4yrs
1101	Computer	MU	4yrs
1102	Mechatronics	DU	4yrs
1301	Automobile	JNU	5yrs

Key	
Multivalue	
Derived	

CONCLUSION:

From the experiment we learnt about how to obtain relational schema from an ER Diagram. The model serves as a tool for creation of databases and design. We mapped the components of the ER diagram, such as Entities, Attributes, Relationships and also the various types of attributes into a table form. Thus we created a relational schema on College Database Management and understood various relationships

---

EXPERIMENT 3

AIM:

Create Database using DDL and DML commands.

REQUIREMENTS:

Relational Schema, Web Browser- Google Chrome, SQL Tryit Editor v1.6 - W3Schools

THEORY:

SQL is a standard language for storing, manipulating and retrieving data in databases.

- SQL stands for Structured Query Language
- SQL lets you access and manipulate databases
- SQL became a standard of the American National Standards Institute (ANSI) in 1986, and of the International Organization for Standardization (ISO) in 1987 What Can SQL do?

- SQL can execute queries against a database
- SQL can retrieve data from a database
- SQL can insert records in a database
- SQL can update records in a database
- SQL can delete records from a database
- SQL can create new databases
- SQL can create new tables in a database
- SQL can create stored procedures in a database
- SQL can create views in a database
- SQL can set permissions on tables, procedures, and views

SQL Commands:

The standard SQL commands to interact with relational databases are CREATE, SELECT, INSERT, UPDATE, DELETE and DROP. These commands can be classified into the following groups based on their nature −

Data Definition Language	
Command	**Description**
CREATE	Creates a new table, a view of a table, or other object in the database.
ALTER	Modifies an existing database object, such as a table.
DROP	Deletes an entire table, a view of a table or other objects in the database.

Data Manipulation Language	
Command	**Description**
SELECT	Retrieves certain records from one or more tables.
UPDATE	Modifies records.
INSERT	Creates a record.
DELETE	Deletes records.

Creating Table:

SQL Statement:

```
CREATE TABLE Student
(student_ID INTEGER,
student_name CHAR(50),
student_dob CHAR(8),
student_address VARCHAR(50),
PRIMARY KEY (student_ID));
```

Inserting Values:

SQL Statement:

```
INSERT INTO Student(student_ID, student_name,student_dob,student_address)
VALUES(1002,'Jatin Jawale','01-10-1998','Andheri West, Mumbai');
```

Result:

Number of Records: 1

student_ID	student_name	student_dob	student_address
1002	Jatin Jawale	01-10-1998	Andheri West, Mumbai

Result:

Number of Records: 5

student_ID	student_name	student_dob	student_address
1001	Reeya Nand	04-09-1998	Malad West, Mumbai
1002	Jatin Jawale	01-10-1998	Andheri West, Mumbai
1003	Sanika Jog	04-04-1998	Goregaon West, Mumbai
1004	Shreya Bhat	14-04-1998	Kalyani Nagar, Pune
1005	Sanket Birwatkar	10-11-1998	Warje, Pune

Alter Command:

SQL Statement:

```
ALTER TABLE Student
ADD COLUMN student_branch CHAR(5)
```

Result:

Number of Records: 5

student_ID	student_name	student_dob	student_address	student_branch
1001	Reeya Nand	04-09-1998	Malad West, Mumbai	*null*
1002	Jatin Jawale	01-10-1998	Andheri West, Mumbai	*null*
1003	Sanika Jog	04-04-1998	Goregaon West, Mumbai	*null*
1004	Shreya Bhat	14-04-1998	Kalyani Nagar, Pune	*null*
1005	Sanket Birwatkar	10-11-1998	Warje, Pune	*null*

Delete Command:

SQL Statement:

```
DELETE FROM Student
WHERE student_ID=1002;
```

Result:

Number of Records: 4

student_ID	student_name	student_dob	student_address	student_branch
1001	Reeya Nand	04-09-1998	Malad West, Mumbai	*null*
1003	Sanika Jog	04-04-1998	Goregaon West, Mumbai	*null*
1004	Shreya Bhat	14-04-1998	Kalyani Nagar, Pune	*null*
1005	Sanket Birwatkar	10-11-1998	Warje, Pune	*null*

SQL Statement:

```
UPDATE Student
SET student_branch='CMPN'
WHERE student_ID=1002;
```

Update Command:

Result:

Number of Records: 5

student_ID	student_name	student_dob	student_address	student_branch
1001	Reeya Nand	04-09-1998	Malad West, Mumbai	*null*
1002	Jatin Jawale	01-10-1998	Andheri West, Mumbai	CMPN
1003	Sanika Jog	04-04-1998	Goregaon West, Mumbai	*null*
1004	Shreya Bhat	14-04-1998	Kalyani Nagar, Pune	*null*
1005	Sanket Birwatkar	10-11-1998	Warje, Pune	*null*

Drop Command:

SQL Statement:

```
DROP TABLE Student;
```

Your Database:

Tablename	Records
Student	5

Your Database:

Restore Database

CONCLUSION:

In this experiment, we learnt about SQL and the various commands available. We made use of DDL and DML commands in order to create and make changes in the database according to the ER model created previously. We saw how we can create, update, alter, drop, insert values into and delete certain values from the table using various SQL commands.

EXPERIMENT 4

AIM

Perform SQL commands to enforce Integrity Constraints for specified system.

REQUIREMENTS

Web Browser- Google Chrome, SQL Tryit Editor v1.6 - W3Schools

THEORY-

- NOT NULL - Ensures that a column cannot have a NULL value
- UNIQUE - Ensures that all values in a column are different
- PRIMARY KEY - A combination of a NOT NULL and UNIQUE. Uniquely identifies each row in a table
- FOREIGN KEY - Uniquely identifies a row/record in another table
- CHECK - Ensures that all values in a column satisfies a specific condition

PRIMARY KEY CONSTRAINT

```
CREATE TABLE Student
(student_ID INTEGER NOT NULL PRIMARY KEY,
student_name CHAR(50),
student_dob CHAR(8),
student_address VARCHAR(50)
);
```

UNIQUE KEY CONSTRAINT

```
CREATE TABLE Department
(dept_name CHAR(20),
dept_hod CHAR(50) NOT NULL UNIQUE,
dept_no_students INTEGER,
dept_no_teachers INTEGER NOT NULL,
);
```

NOT NULL CONSTRAINT

```
CREATE TABLE College
(college_id INTEGER NOT NULL,
college_name VARCHAR(20) NOT NULL,
college_address VARCHAR(20) NOT NULL,
college_ph_no INTEGER NOT NULL
);
```

FOREIGN KEY CONSTRAINT

```
CREATE TABLE Faculty
(faculty_name VARCHAR(20) NOT NULL,
faculty_qualification CHAR(20) NOT NULL,
faculty_subj_taught VARCHAR(20) NOT NULL,
faculty_experience INTEGER NOT NULL,
college_id INTEGER, FOREIGN KEY(college_id) REFERENCES College);
```

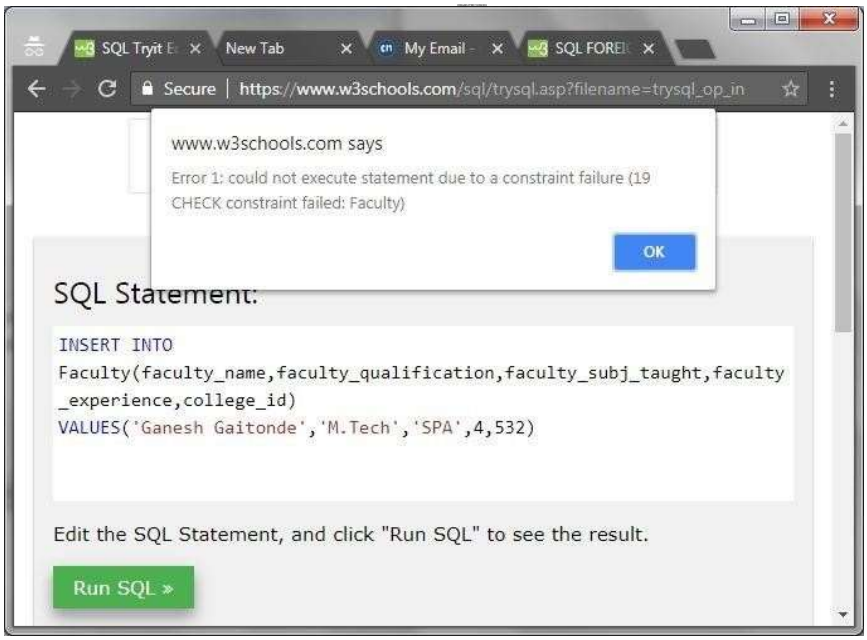
Result:

Number of Records: 2

faculty_name	faculty_qualification	faculty_subj_taught	faculty_experience	college_id
Kumar Soni	M.E	Mechanics	10	532
Piyush Dagde	M.B.B.S	Anatomy	8	246

CHECK CONSTRAINT

```
CREATE TABLE Faculty
(faculty_name VARCHAR(20) NOT NULL,
faculty_qualification CHAR(20) NOT NULL,
faculty_subj_taught VARCHAR(20) NOT NULL,
faculty_experience INTEGER CHECK(faculty_experience>=5),
college_id INTEGER, FOREIGN KEY(college_id) REFERENCES College);
```



CONCLUSION

In this experiment, we learnt how SQL constraints can be used to impose any kind of conditions on the data to be inserted into the table. We learnt about the various constraints and imposed them on our data in order to ensure accuracy and reliability. Constraints ensure that there is no violation while entering the data into the database. If there is any, the action is aborted. We also learnt how constraints can be column or table level.

Thus, in this experiment we learnt about and implemented constraints successfully.

EXPERIMENT 5

AIM:
Perform SQL commands for nested queries and complex queries.

REQUIREMENTS:
Relational Schema, Web Browser- Google Chrome, SQL Tryit Editor v1.6 - W3Schools

THEORY:
A Subquery or Inner query or a Nested query is a query within another SQL query and embedded within the WHERE clause.
A subquery is used to return data that will be used in the main query as a condition to further restrict the data to be retrieved.
Subqueries can be used with the SELECT, INSERT, UPDATE, and DELETE statements along with the operators like =, <, >, >=, <=, IN, BETWEEN, etc.

There are a few rules that subqueries must follow :

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

- The BETWEEN operator cannot be used with a subquery. However, the BETWEEN operator can be used within the subquery.

- **DATABASE:**

EMPLOYEE

Employee_ID	First_Name	Last_Name	Salary	Joining_Date	Department
1	John	Abraham	1000000	01-01-13 00:00:00	Banking
2	Michael	Clarke	800000	01-01-13 00:00:00	Insurance
3	Roy	Thomas	700000	01-02-13 00:00:00	Banking
4	Tom	Jose	600000	01-02-13 00:00:00	Insurance
5	Jerry	Pinto	650000	01-02-13 00:00:00	Insurance
6	Philip	Matthew	750000	01-01-13 00:00:00	Services
7	TestName1	123	650000	01-01-13 00:00:00	Services
8	TestName2	LName%	600000	01-02-13 00:00:00	Insurance

INCENTIVES

Employee_ref_ID	Incentive_date	Incentive_amount
1	01-02-13	5000
2	01-02-13	3000
3	01-02-13	4000
1	01-01-13	4500
2	01-01-13	3500

1. Get all employee details from the employee table: Ans.

SELECT * FROM EMPLOYEE

Result:

Number of Records: 8

Employee_ID	First_Name	Last_Name	Salary	Joining_Date	Department
1	John	Abraham	1000000	01-01-13 00:00:00	Banking
2	Michael	Clarke	800000	01-01-13 00:00:00	Insurance
3	Roy	Thomas	700000	01-02-13 00:00:00	Banking
4	Tom	Jose	600000	01-02-13 00:00:00	Insurance
5	Jerry	Pinto	650000	01-02-13 00:00:00	Insurance
6	Philip	Matthew	750000	01-01-13 00:00:00	Services
7	TestName1	123	650000	01-01-13 00:00:00	Services
8	TestName2	LName%	600000	01-02-13 00:00:00	Insurance

2. Get First_Name,Last_Name from employee table:
Ans.

SELECT First_Name, Last_Name FROM EMPLOYEE

Result:

Number of Records: 8

First_Name	Last_Name
John	Abraham
Michael	Clarke
Roy	Thomas
Tom	Jose
Jerry	Pinto
Philip	Matthew
TestName1	123
TestName2	LName%

3. Get First_Name from employee table using alias name "Employee Name" Ans.

SELECT First_Name AS
Employee_Name FROM
Employee

Result:

Number of Records: 8

Employee_Name
John
Michael
Roy
Tom
Jerry
Philip
TestName1
TestName2

4. Get First_Name from employee table in upper case Ans.

SELECT UPPER(First_Name) AS
UpperFirstName FROM Employee

Result:

Number of Records: 8

UpperFirstName
JOHN
MICHAEL
ROY
TOM
JERRY
PHILIP
TESTNAME1
TESTNAME2

5. Get First_Name from employee table in lower case Ans.

SELECT LOWER(First_Name) AS
LowerFirstName FROM Employee

Result:

Number of Records: 8

LowerFirstName
john
michael
roy
tom
jerry
philip
testname1
testname2

6. Get unique DEPARTMENT from employee table Ans.

SELECT
DISTINCT
Department
From
Employee

Result:

Number of Records: 3

Department
Banking
Insurance
Services

7. Select first 3 characters of FIRST_NAME from EMPLOYEE Ans.

SELECT SUBSTR(First_Name,1,3)
AS First_3_Chars FROM Employee;

Result:

Number of Records: 8

First_3_Chars
Joh
Mic
Roy
Tom
Jer
Phi
Tes
Tes

8. Get position of 'o' in name "John" from employee table Ans.

SELECT CHARINDEX('o',(SELECT First_Name FROM Employee WHERE First_Name="John")) AS MatchIndex

Result:

Number of Records: 1

MatchPosition
2

9. Get FIRST_NAME from employee table after removing white spaces from left side Ans.

SELECT LTRIM(First_Name) AS
LeftWhitespaceRemoved FROM Employee

Result:

Number of Records: 8

LeftWhitespaceRemoved
John
Michael
Roy
Tom
Jerry
Philip
TestName1
TestName2

10. Get FIRST_NAME from employee table after removing white spaces from right side Ans.

SELECT RTRIM(First_Name) AS
RightWhitespaceRemoved FROM Employee

Result:

Number of Records: 8

RightWhitespaceRemoved
John
Michael
Roy
Tom
Jerry
Philip
TestName1
TestName2

11. Get length of FIRST_NAME from employee table Ans.

SELECT LENGTH(First_Name) AS
LengthOfFirstName FROM
Employee;

Result:

Number of Records: 8

LengthOfFirstName
4
7
3
3
5
6
9
9

12. Get First_Name from employee table after replacing “o” with “$”: Ans.

```
SELECT REPLACE(First_Name, "o",
"$") AS ReplacingFROM Employee
```

Result:

Number of Records: 8

Replacing
J$hn
Michael
R$y
T$m
Jerry
Phillip
TestName1
TestName2

13. Get First_Name and Last_Name as single column from employee table separated by a “_” Ans.

```
SELECT
First_Name||’_’||Last_Name
FROM Employee
```

FIRST_NAME|| '_' ||LAST_NAME
John_ Abraham
Michael_Clarke
Roy_ Thomas
Tom_Jose
Jerry_Pinto
Philip_ Mathew
TestName1_123
TestName2_Lname%

14. Get FIRST_NAME , Joining year, Joining Month and Joining Date from employee table: Ans.

Select FIRST_NAME, to_char(joining_date,'YYYY') JoinYear ,
to_char(joining_date,'Mon'), to_char(joining_date,'dd') from EMPLOYEE

CONCLUSION:

In this experiment, we learnt about complex and nested queries in SQL and implemented various examples using a Database. We saw how to perform various operations such as string concatenation, getting position of specific characters in a string, getting upper or lowercase equivalent of strings, etc.

Thus, in this experiment we successfully demonstrated the use of nested/complex queries on Database.

EXPERIMENT 6

AIM :- SQL commands to perform Join operations.

REQUIREMENTS

paiza.io (Online SQL), Database.

THEORY

A JOIN clause is used to combine rows from two or more tables, based on a related column between them.

- (INNER) JOIN: Returns records that have matching values in both tables.
- LEFT (OUTER) JOIN: Return all records from the left table, and the matched records from the right table. ○ RIGHT (OUTER) JOIN: Return all records from the right table, and the matched records from the left table.
- FULL (OUTER) JOIN: Return all records when there is a match in either left or right table.

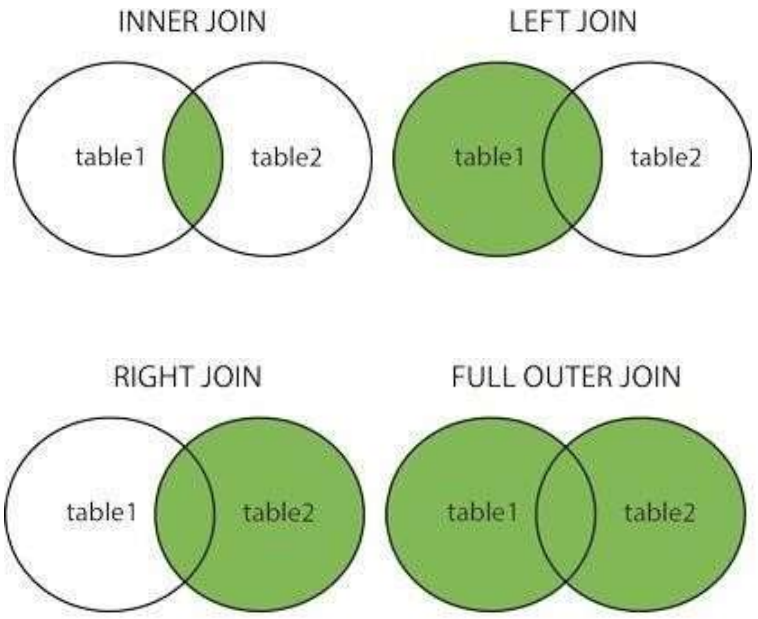


Diagram- ● Inner Join:

```
create table Orders(Order_id int, CustomerID int, Order_date varchar(255));
insert into Orders(Order_id, CustomerID, Order_date) values(10301, 12345, "01-08-2018");
insert into Orders(Order_id, CustomerID, Order_date) values(10302, 12346, "17-08-2018");
insert into Orders(Order_id, CustomerID, Order_date) values(10303, 12347, "28-08-2018");
select * from Orders;
create table Customers(CustomerID int, CustomerName varchar(250), ContactName varchar(250), Country varchar(250));
insert into Customers(CustomerID, CustomerName, ContactName, Country) values(12345, "Reeya", "Nand", "India");
insert into Customers(CustomerID, CustomerName, ContactName, Country) values(12346, "Sanket", "Fugga", "India");
insert into Customers(CustomerID, CustomerName, ContactName, Country) values(12340, "Jatin", "Juju", "India");
select * from Customers;

SELECT Orders.Order_id, Customers.CustomerName, Orders.Order_date
FROM Orders
INNER JOIN Customers ON Orders.CustomerID=Customers.CustomerID;
```

Success Tweet Share 0

Run (Ctrl-Enter)

Output Input Comments 0

```
10301   Reeya   01-08-2018
10302   Sanket  17-08-2018
```

- **Left Join:**

Success Tweet Share 0

```
create table Orders(Order_id int, CustomerID int, Order_date varchar(255));
insert into Orders(Order_id, CustomerID, Order_date) values(10301, 12345, "01-08-2018");
insert into Orders(Order_id, CustomerID, Order_date) values(10302, 12346, "17-08-2018");
insert into Orders(Order_id, CustomerID, Order_date) values(10303, 12347, "28-08-2018");
select * from Orders;
create table Customers(CustomerID int, CustomerName varchar(250), ContactName varchar(250), Country varchar(250));
insert into Customers(CustomerID, CustomerName, ContactName, Country) values(12345, "Reeya", "Nand", "India");
insert into Customers(CustomerID, CustomerName, ContactName, Country) values(12346, "Sanket", "Fugga", "India");
insert into Customers(CustomerID, CustomerName, ContactName, Country) values(12347, "Jatin", "Juju", "India");
select * from Customers;

SELECT Orders.Order_id, Customers.CustomerName, Orders.Order_date
FROM Orders
LEFT JOIN Customers ON Orders.CustomerID=Customers.CustomerID;
```

Run (Ctrl-Enter)

Output Input Comments 0

```
10301   Reeya   01-08-2018
10302   Sanket  17-08-2018
10303   Jatin   28-08-2018
```

- **Right Join:**

```
create table Orders(Order_id int, CustomerID int, Order_date varchar(255));
insert into Orders(Order_id, CustomerID, Order_date) values(10301, 12345, "01-08-2018");
insert into Orders(Order_id, CustomerID, Order_date) values(10302, 12346, "17-08-2018");
insert into Orders(Order_id, CustomerID, Order_date) values(10303, 12347, "28-08-2018");
select * from Orders;
create table Customers(CustomerID int, CustomerName varchar(250), ContactName varchar(250), Country varchar(250));
insert into Customers(CustomerID, CustomerName, ContactName, Country) values(12345, "Reeya", "Nand", "India");
insert into Customers(CustomerID, CustomerName, ContactName, Country) values(12340, "Sanket", "Fugga", "India");
insert into Customers(CustomerID, CustomerName, ContactName, Country) values(12347, "Jatin", "Juju", "India");
select * from Customers;

SELECT Orders.Order_id, Customers.CustomerName, Orders.Order_date
FROM Orders
RIGHT JOIN Customers ON Orders.CustomerID=Customers.CustomerID;
```

Run (Ctrl-Enter)

Output Input Comments 0

```
10301   Reeya   01-08-2018
10303   Jatin   28-08-2018
NULL    Sanket  NULL
```

CONCLUSION

In this experiment we tried out various types of joins that are possible in SQL. We performed joins like Inner Join, Left Join, Right Join and Outer Join and saw the different outcomes that are obtained in each case. These joins themselves serve similar, yet different purposes in order to make the database more usable. Joins basically combines tables based on the type of join in use.

The LEFT JOIN keyword returns all records from the left table (table1), and the matched records from the right table (table2). The result is NULL from the right side, if there is no match. The RIGHT JOIN keyword returns all records from the right table (table2), and the matched records from the left table (table1). The result is NULL from the left side, when there is no match. The FULL OUTER JOIN keyword return all records when there is a match in either left (table1) or right (table2) table records. A self JOIN is a regular join, but the table is joined with itself.

EXPERIMENT NO : 07

AIM: SQL commands to perform views and triggers.

THEORY:

A view can contain all rows of a table or select rows from a table. A view can be created from one or many tables which depends on the written SQL query to create a view.
Views, which are a type of virtual tables allow users to do the following −

1) Structure data in a way that users or classes of users find natural or intuitive.
2) Restrict access to the data in such a way that a user can see and (sometimes) modify exactly what they need and no more.
3) Summarize data from various tables which can be used to generate reports.

SQL CREATE VIEW Statement:

In SQL, a view is a virtual table based on the result-set of an SQL statement.
A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database.
You can add SQL functions, WHERE, and JOIN statements to a view and present the data as if the data were coming from one single table. CREATE VIEW Syntax:

CREATE VIEW view_name AS SELECT column1,
column2, ... FROM table_name
WHERE condition;

Then, we can query the view as follows:

SELECT * FROM [Current Product List];

OUTPUT:

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00

```
SQL > CREATE VIEW CUSTOMERS_VIEW AS
SELECT name, age
FROM  CUSTOMERS;
```

```
SQL > SELECT * FROM CUSTOMERS_VIEW;
```

```
+----------+-----+
| name     | age |
+----------+-----+
| Ramesh   |  32 |
| Khilan   |  25 |
| kaushik  |  23 |
| Chaitali |  25 |
| Hardik   |  27 |
| Komal    |  22 |
| Muffy    |  24 |
```

SQL Updating a View

You can update a view by using the following syntax:

CREATE OR REPLACE VIEW view_name AS SELECT column1, column2, ... FROM table_name WHERE condition;

OUTPUT:

```
SQL > UPDATE CUSTOMERS_VIEW
   SET AGE = 35
   WHERE name = 'Ramesh';
```

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  35 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

```
SQL > DELETE FROM CUSTOMERS_VIEW
   WHERE age = 22;
```

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  35 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

SQL Dropping a View
You can delete a view with the DROP VIEW command.
SQL DROP VIEW Syntax:

```
DROP VIEW view_name;
```

OUTPUT:
The table will be deleted.

CONCLUSION:

Hence SQL commands to perform views and triggers implemented successfully. We learnt about the concept of views and triggers and created various views for a table in the database and successfully executed queries on them.

EXPERIMENT NO 8

AIM:

Study of Functions, cursor and procedure Using PL/SQL and write a program to perform following:

1) To check whether a given number is EVEN/ODD

2) To print table of given number.

3)To reverse a given number.

1. PL/SQL Program to Check Number is Odd or Even

Here you will get pl/sql program to check number is odd or even.
A number is even if it is completely divisible by 2 otherwise it is odd

```
declare n
        number:=&n;
```

```
begin if mod(n,2)=0
        then
dbms_output.put_line('number      is      even');      else
        dbms_output.put_line('number is odd');
end if; end;
/
```

Output

```
Enter value for n: 7
old   2: n number:=&n; new
2:  n number:=7; number is
odd
```

2. PL/SQL Program to Print Table of a Number

Here you will get pl/sql program to print table of a given number.
You can ask your queries in comment section.

```
Declare n
        number; i
        number;

begin
        n:=&n
        ;
for i in 1..10 loop
                dbms_output.put_line(n||' x '||i||' = '||n*i); end loop;

end;
/
```

Output

```
Enter value for n:
5 old 6: n:=&n;
new 6: n:=5; 5 x 1
= 5
5 x 2 = 10
5 x 3 = 15
5 x 4 = 20
5 x 5 = 25
5 x 6 = 30
5 x 7 = 35
5 x 8 = 40
5 x 9 = 45
```

5 x 10 = 50

3. PL/SQL Program to Reverse a String

Here you will get pl/sql program to reverse a string.
The **substr()** function returns a part of string. It has following syntax.
substr(string, position, length);
We will use this function to extract character by character from string in reverse order and concatenate it previous extracted character. || is used to concatenate string. In this way string is reversed.

```
declare str1 varchar2(50):='&str';
        str2
        varchar2(50); len
        number; i
        number;

begin
        len:=length(str1)
        ;

        for i in reverse 1..len loop str2:=str2 ||
                substr(str1,i,1); end loop;


        dbms_output.put_line('Reverse of String is:'||str2);
end;
/
```

Output
Enter value for str: hello world old
2: str1 varchar2(50):=’&str’;
new 2: str1 varchar2(50):=’hello world’;
Reverse of String is:dlrow olleh

CONCLUSION

Thus we successfully executed PL/SQL statements for:

- Checking if the number is even or odd
- Print the multiplication table of a number
- Reversing a string And we got the desired results.

EXPERIMENT 9

AIM

Write a program in Java using JDBC ODBC connectivity to access data from database

REQUIREMENTS

Database, Java

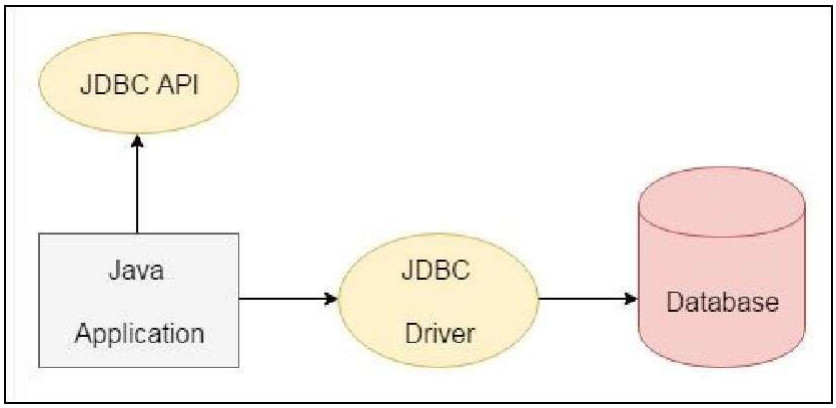
THEORY

JDBC stands for Java Database Connectivity. JDBC is a Java API to connect and execute the query with the database. It is a part of JavaSE (Java Standard Edition). JDBC API uses JDBC drivers to connect with the database. There are four types of JDBC drivers:

- JDBC-ODBC Bridge Driver,
- Native Driver,
- Network Protocol Driver, and
- Thin Driver

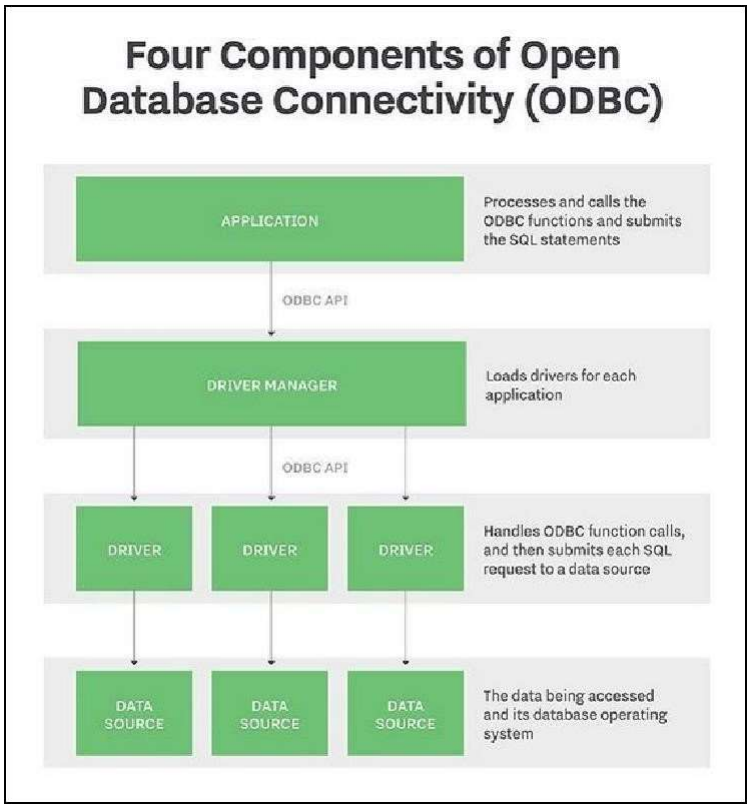
We have discussed the above four drivers in the next chapter.

We can use JDBC API to access tabular data stored in any relational database. By the help of JDBC API, we can save, update, delete and fetch data from the database. It is like Open Database Connectivity (ODBC) provided by Microsoft.



Open Database Connectivity (ODBC) is an open standard application programming interface (API) that allows application programmers to access any database.
The main proponent and supplier of ODBC programming support is Microsoft, but ODBC is based on and closely aligned with The Open Group standard Structured Query Language (SQL) Call-Level Interface (CLI). The Open Group is sponsored by many major vendors, including Oracle, IBM and Hewlett Packard Enterprise, and this consortium develops and manufactures The Open Group Architecture Framework (TOGAF).

ODBC consists of four components, working together to enable functions. ODBC allows programs to use SQL requests that access databases without knowing the proprietary interfaces to the databases. ODBC handles the SQL request and converts it into a request each database system understands.



CODE

EXAMPLE 1:

```
import java.sql.*;
class Test
{ public static void main(String ar[])
   { try
          {
             String url="jdbc:odbc:mydsn";
             Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
             Connection c=DriverManager.getConnection(url);
             Statement st=c.createStatement();
             ResultSet rs=st.executeQuery("select * from login");
while(rs.next())
                    {
              System.out.println(rs.getString(1));
                    }
```

```
            }
            catch(Exception ee)
            {
                    System.out.println(ee);
            }

            }
}
```

EXAMPLE 2:

```
import java.sql.*; public
class
MysqlDemo

{ public static void main(String[] args)
   { try
      {
         Class.forName("com.mysql.jdbc.Driver");
         System.out.println("Driver is loaded");

         Connection c =
DriverManager.getConnection("jdbc:mysql://localhost:3306/java2all","root","root");
         System.out.println("Connection created"); Statement
         s = c.createStatement();
         ResultSet rs = s.executeQuery("select * from
data"); System.out.println("ID\tName\tCity\tAge");
while(rs.next()) // Retrieve data from ResultSet {
System.out.println(rs.getString(1)+"\t"+rs.getString(2)+"\t"+rs.getString(3)+"\t"+rs.getString(4));
         }
      }
      catch(Exception e)
      {
         System.out.println("Exception : " +e);
      }
   }
}
```

OUTPUT

```
Output:

Driver is loaded

Connection created

ID   Name  City    Age

1     java      abc    300

2     JDBC  xyz   200

3     JSP     mno  100
```

CONCLUSION

Hence, in this experiment we studied about Java Database Connectivity and Open Database Connectivity.

We saw how we can access an existing Database on a computer using JDBC and ODBC and wrote Java Programs to accomplish the task. We also learnt about the ways in which the two differ.

Thus, we performed this experiment on JDBC and ODBC successfully.

EXPERIMENT NO:10

AIM :- Case study on transaction and Concurrency Control in DBMS.

THEORY

TRANSACTION:

- A sequence of logical steps that will accomplish a single task (or what seems like a single task)

  Example:

  - add an employee
  - enter an order
  - enroll a student in a course

- A single task may require MANY changes to the database.
- If all changes are NOT made database integrity will be lost

Following are the transaction properties also known as ACID properties :

Atomicity
Transactions are often composed of multiple statements. Atomicity guarantees that each transaction is treated as a single "unit", which either succeeds completely, or fails completely: if any of the statements constituting a transaction fails to complete, the entire transaction fails and the database is left unchanged. An atomic system must guarantee atomicity in each and every situation, including power failures, errors and crashes.

Consistency
Consistency ensures that a transaction can only bring the database from one valid state to another, maintaining database invariants: any data written to the database must be valid according to all defined rules, including constraints, cascades, triggers, and any combination thereof. This prevents database corruption by an illegal transaction, but does not guarantee that a transaction is *correct*.

Isolation
Transactions are often executed concurrently (e.g., reading and writing to multiple tables at the same time). Isolation ensures that concurrent execution of transactions leaves the database in the same state that would have been obtained if the transactions were executed sequentially. Isolation

is the main goal of concurrency control; depending on the method used, the effects of an incomplete transaction might not even be visible to other transactions.

Durability

Durability guarantees that once a transaction has been committed, it will remain committed even in the case of a system failure (e.g., power outage or crash). This usually means that completed transactions (or their effects) are recorded in non-volatile memory.

SQL provides transaction support through :

- **COMMIT** (permanently saves changes to disk)
- **ROLLBACK** (restores the previous database state) **Transaction Logs :**

Keeps track of all transactions that update the DB.

Info kept in a typical log:

- Trans. ID – time of trans. – type of trans.
- object of action
- BEFORE image
- AFTER image

This allows for FORWARD & BACKWARD recovery.

Scheduler :

Multi-user DBMSs use a process known as a *scheduler* to enforce concurrency control.

It serializes the transactions to ensure “isolation”.

Concurrency Control :

In a multiprogramming environment where multiple transactions can be executed simultaneously, it is highly important to control the concurrency of transactions. We have concurrency control protocols to ensure atomicity, isolation, and serializability of concurrent transactions. Concurrency control protocols can be broadly divided into two categories − • Lock based protocols

- Timestamp based protocols

Lock-based Protocols :

Database systems equipped with lock-based protocols use a mechanism by which any transaction cannot read or write data until it acquires an appropriate lock on it. Locks are of two kinds –

Binary Locks
A lock on a data item can be in two states; it is either locked or unlocked.

Shared/exclusive
This type of locking mechanism differentiates the locks based on their uses. If a lock is acquired on a data item to perform a write operation, it is an exclusive lock.
Allowing more than one transaction to write on the same data item would lead the database into an inconsistent state. Read locks are shared because no data value is being changed.

There are four types of lock protocols available −

- **Simplistic Lock Protocol**
  Simplistic lock-based protocols allow transactions to obtain a lock on every object before a 'write' operation is performed. Transactions may unlock the data item after completing the ‘write’ operation.

- **Pre-claiming Lock Protocol**
  Pre-claiming protocols evaluate their operations and create a list of data items on which they need locks. Before initiating an execution, the transaction requests the system for all the locks it needs beforehand. If all the locks are granted, the transaction executes and releases all the locks when all its operations are over. If all the locks are not granted, the transaction rolls back and waits until all the locks are granted.

- **Two-Phase Locking 2PL**
  This locking protocol divides the execution phase of a transaction into three parts. In the first part, when the transaction starts executing, it seeks permission for the locks it requires. The second part is where the transaction acquires all the locks. As soon as the transaction releases its first lock, the third phase starts. In this phase, the transaction cannot demand any new locks; it only releases the acquired locks.
  Two-phase locking has two phases, one is growing, where all the locks are being acquired by the transaction; and the second phase is shrinking, where the locks held by the transaction are being released. To claim an exclusive write lock, a transaction must first acquire a shared read lock and then upgrade it to an exclusive lock.

- **Strict Two-Phase Locking**
  The first phase of Strict-2PL is same as 2PL. After acquiring all the locks in the first phase, the transaction continues to execute normally. But in contrast to 2PL, Strict-2PL does not release a lock after using it. Strict-2PL holds all the locks until the commit point and releases all the locks at a time. Strict-2PL does not have cascading abort as 2PL does.

Timestamp-based Protocols

The most commonly used concurrency protocol is the timestamp based protocol. This protocol uses either system time or logical counter as a timestamp.

Lock-based protocols manage the order between the conflicting pairs among transactions at the time of execution, whereas timestamp-based protocols start working as soon as a transaction is created.

Every transaction has a timestamp associated with it, and the ordering is determined by the age of the transaction. A transaction created at 0002 clock time would be older than all other transactions that come after it. For example, any transaction 'y' entering the system at 0004 is two seconds younger and the priority would be given to the older one.

In addition, every data item is given the latest read and write-timestamp. This lets the system know when the last ‘read and write’ operation was performed on the data item.

Timestamp Ordering Protocol

The timestamp-ordering protocol ensures serializability among transactions in their conflicting read and write operations. This is the responsibility of the protocol system that the conflicting pair of tasks should be executed according to the timestamp values of the transactions.

CONCLUSION

Hence, we have successfully conducted a Case-Study on Transaction and Concurrency Control in DBMS. We learnt about what a transaction is, what its properties are, the need of concurrency control, how DBMS provides protocols to maintain concurrency in data and makes the database more reliable