LIST OF EXPERIMENT

TEXT BOOKS:

1. Database Management Systems, Raghurama Krishnan, Johannes Gehrke, Tata Mc Graw Hill,3rd Edition
2. Database System Concepts, Silberschatz, Korth, McGraw Hill, V edition.

REFERENCE BOOKS:

1. Database Systems design, Implementation, and Management, Peter Rob & Carlos Coronel 7thEdition.
2. Fundamentals of Database Systems, Elmasri Navrate, Pearson Education
3. Introduction to Database Systems, C.J. Date, Pearson Education
4. Oracle for Professionals, The X Team, S. Shah and V. Shah, SPD.
5. Database Systems Using Oracle: A Simplified guide to SQL and PL/SQL, Shah, PHI.
6. Fundamentals of Database Management Systems, M. L. Gillenson, Wiley Student Edition.

E- RESOURCES:

https://www.youtube.com/watch?v=q1kQ1vgW7D8

EXPERIMENT 1

Concept design with E-R Model

Analyze the problem with the entities which identify data persisted in the database which contains entities and attributes.

Objectives

Student will be able to learn the Entity-Relationship(ER) modeling to develop a conceptual model of data.

Outcomes

Student gains the ability

- About business rules, notations and constructs.
- To Construct E-R diagrams including:

  Entities (strong, weak, associative)

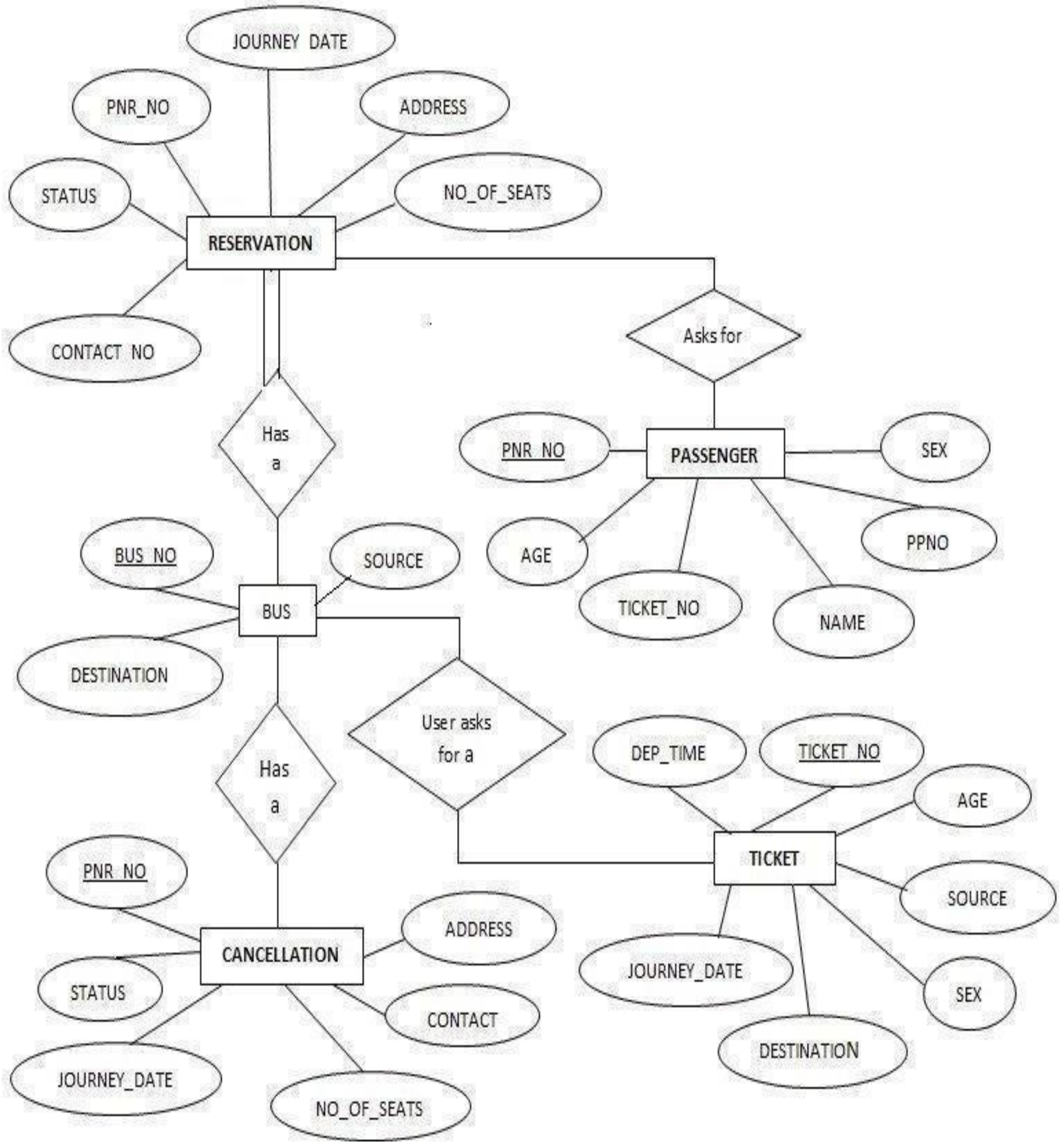
  Attributes (simple, multi-valued, derived)

  Relations (unary, binary, ternary)

E-R diagram: An entity-relationship(ER) diagram is a specified graphical representation that illustrates the interrelationships between entities in a database. We can express the overall logical structure of database graphically with an E-R diagram.
Entity: An Entity is an object or concept about which you want to store information

Relationship: A relationship is an association between several entities.

Attributes: An object is characterized by its properties or attributes. In relational database systems attributes correspond to fields.


JOURNEY DATE
PNR_NO
ADDRESS
STATUS
NO_OF_SEATS
RESERVATION
CONTACT NO
Asks for
Has a
PNR NO
PASSENGER
SEX
AGE
PPNO
TICKET_NO
NAME
BUS NO
SOURCE
BUS
DESTINATION
User asks for a
Has a
DEP_TIME
TICKET NO
AGE
TICKET
SOURCE
PNR NO
CANCELLATION
ADDRESS
STATUS
JOURNEY_DATE
SEX
CONTACT
DESTINATION
JOURNEY_DATE
NO_OF_SEATS

EXPERIMENT 2

Draw E-R diagram and convert entities and relationships to relation table for a given scenario.

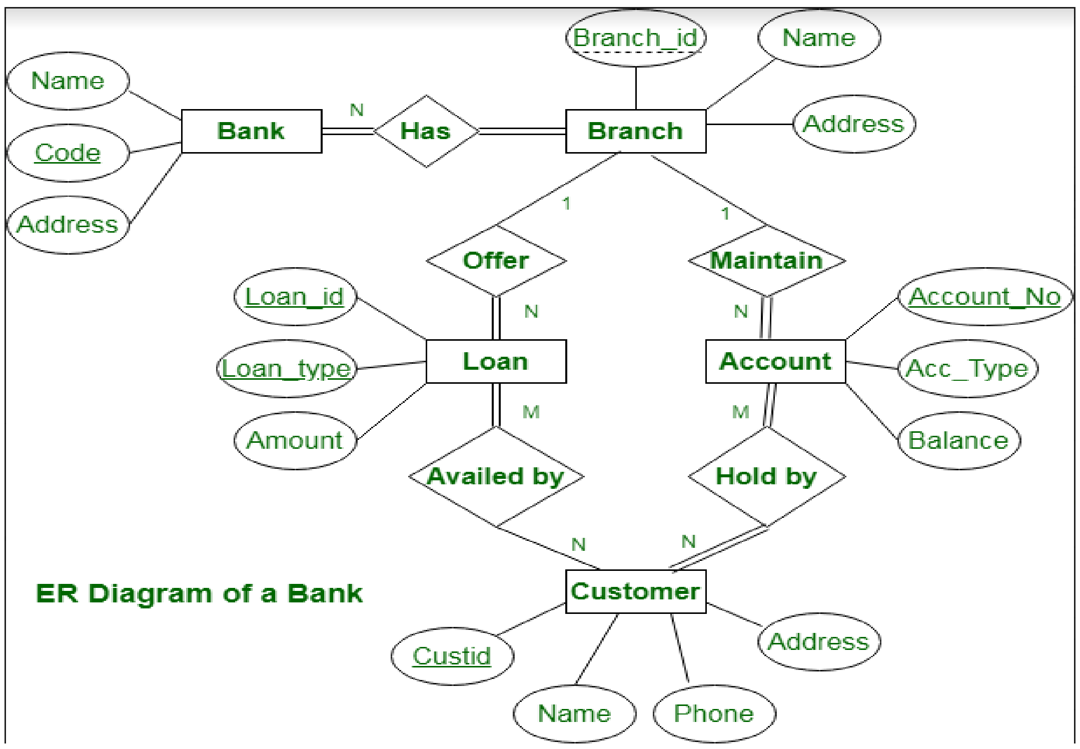
Two assignments shall be carried out i.e., consider two different scenarios (eg.bank, college)

ER diagram is known as Entity-Relationship diagram. It is used to analyze to structure of the Database. It shows relationships between entities and their attributes. An ER model provides a means of communication.

ER diagram of Bank has the following description :

- Bank have Customer.
- Banks are identified by a name, code, address of main office.
- Banks have branches.
- Branches are identified by a branch_no., branch_name, address.
- Customers are identified by name, cust-id, phone number, address.
- Customer can have one or more accounts.
- Accounts are identified by account_no., acc_type, balance.
- Customer can avail loans.
- Loans are identified by loan_id, loan_type and amount.
- Account and loans are related to bank’s branch.

ER Diagram of Bank Management System :



This bank ER diagram illustrates key information about bank, including entities such as branches, customers, accounts, and loans. It allows us to understand the relationships between entities.

Entities and their **Attributes** are :

- **Bank Entity :** Attributes of Bank Entity are Bank Name, Code and Address.
  Code is Primary Key for Bank Entity.
- **Customer Entity :** Attributes of Customer Entity are Customer_id, Name, Phone Number and Address.
  Customer_id is Primary Key for Customer Entity.
- **Branch Entity :** Attributes of Branch Entity are Branch_id, Name and Address.
  Branch_id is Primary Key for Branch Entity.
- **Account Entity :** Attributes of Account Entity are Account_number, Account_Type and Balance.
  Account_number is Primary Key for Account Entity.
- **Loan Entity :** Attributes of Loan Entity are Loan_id, Loan_Type and Amount.
  Loan_id is Primary Key for Loan Entity.

```
-- Bank Entity Table
CREATE TABLE Bank ( Code INT PRIMARY KEY,   BankName VARCHAR(255),
   Address VARCHAR(255));
```

-- Customer Entity Table

CREATE TABLE Customer (Customer_id INT PRIMARY KEY, Name VARCHAR(255), Phone_Number VARCHAR(20), Address VARCHAR(255));

-- Branch Entity Table

CREATE TABLE Branch (Branch_id INT PRIMARY KEY,

Name VARCHAR(255), Address VARCHAR(255));

-- Account Entity Table

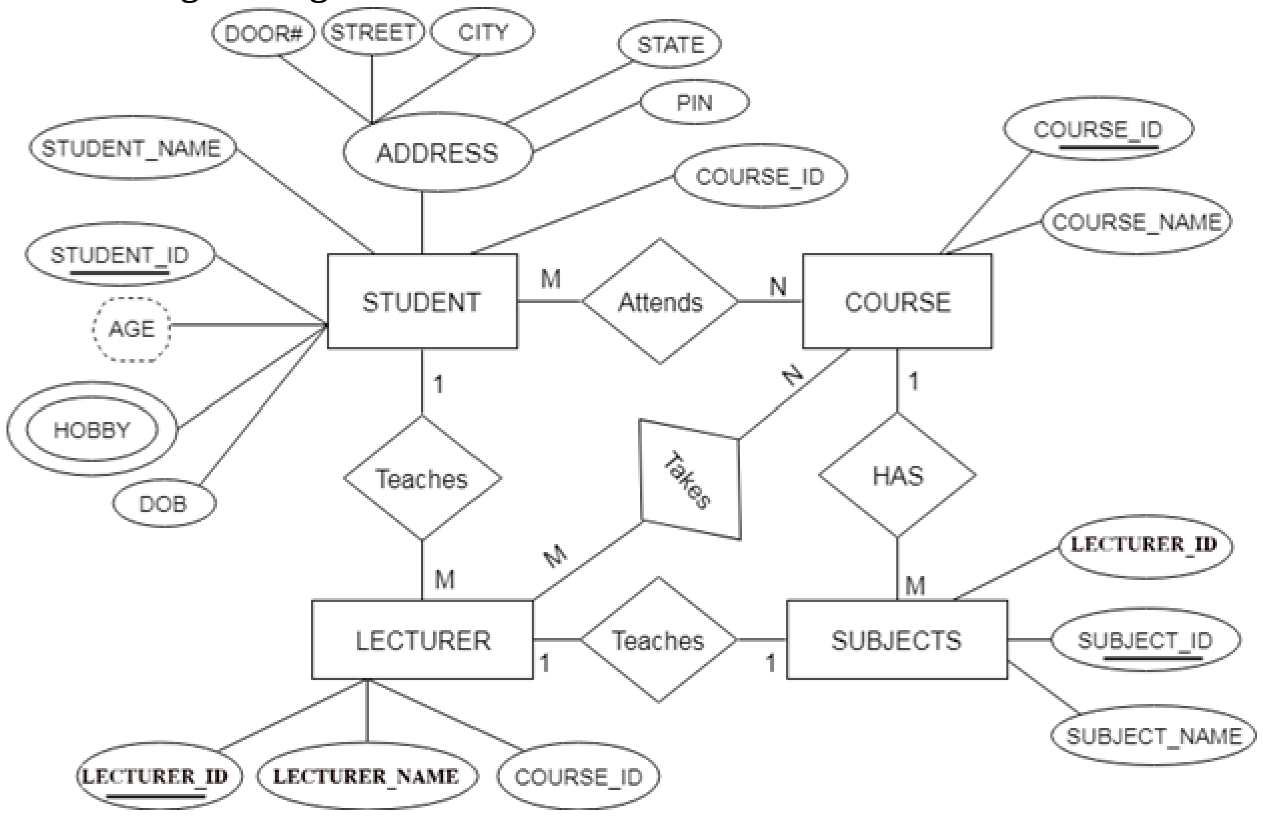
CREATE TABLE Account (Account_number INT PRIMARY KEY, Account_Type VARCHAR(50), Balance DECIMAL(10,2));

-- Loan Entity Table

CREATE TABLE Loan (Loan_id INT PRIMARY KEY, Loan_Type VARCHAR(50), Amount DECIMAL(10,2));

ER Diagram of College Management System :

The ER diagram is given below:



There are some points for converting the ER diagram to the table:

- **Entity type becomes a table.**

In the given ER diagram, LECTURE, STUDENT, SUBJECT and COURSE forms individual tables.

- **All single-valued attribute becomes a column for the table.**

  In the STUDENT entity, STUDENT_NAME and STUDENT_ID form the column of STUDENT table. Similarly, COURSE_NAME and COURSE_ID form the column of COURSE table and so on.

- **A key attribute of the entity type represented by the primary key.**

  In the given ER diagram, COURSE_ID, STUDENT_ID, SUBJECT_ID, and LECTURE_ID are the key attribute of the entity.

- **The multivalued attribute is represented by a separate table.**

  In the student table, a hobby is a multivalued attribute. So it is not possible to represent multiple values in a single column of STUDENT table. Hence we create a table STUD_HOBBY with column name STUDENT_ID and HOBBY. Using both the column, we create a composite key.

- **Composite attribute represented by components.**

  In the given ER diagram, student address is a composite attribute. It contains CITY, PIN, DOOR#, STREET, and STATE. In the STUDENT table, these attributes can merge as an individual column.

- **Derived attributes are not considered in the table.**

  In the STUDENT table, Age is the derived attribute. It can be calculated at any point of time by calculating the difference between current date and Date of Birth.

Using these rules, you can convert the ER diagram to tables and columns and assign the mapping between the tables. Table structure for the given ER diagram is as below:

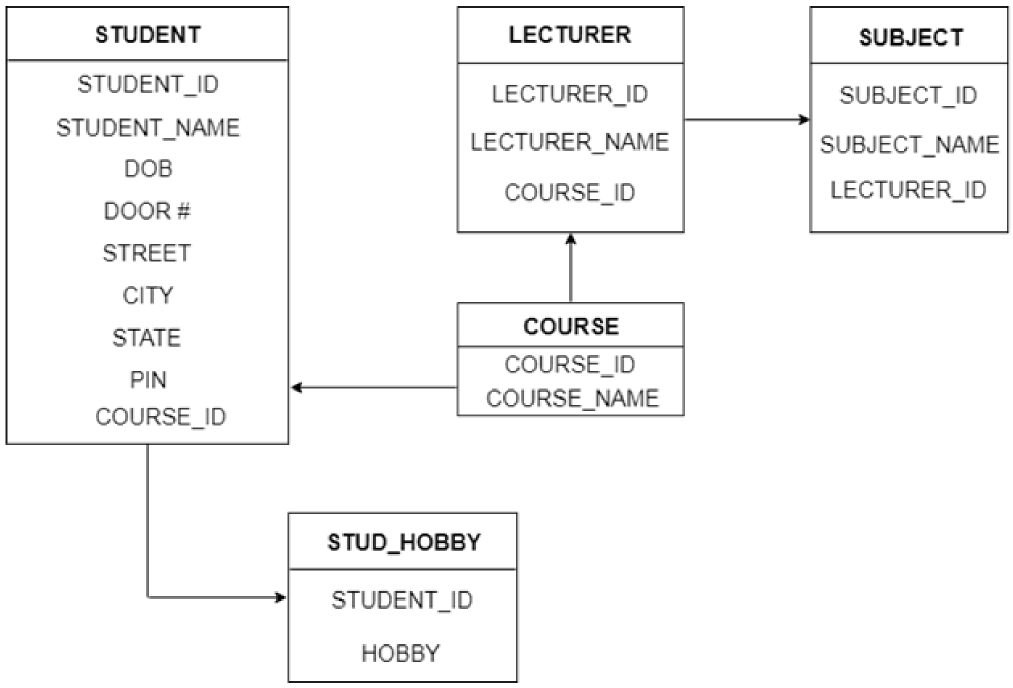


Figure: Table structure

-- Student Table
CREATE TABLE Student (Student_ID INT PRIMARY KEY, Student_Name VARCHAR(255),   Date_of_Birth DATE, City VARCHAR(100), Pin INT, Door_Number VARCHAR(50),Street VARCHAR(100), State VARCHAR(100));

-- Course Table
CREATE TABLE Course (Course_ID INT PRIMARY KEY,Course_Name VARCHAR(255));

-- Subject Table
CREATE TABLE Subject (Subject_ID INT PRIMARY KEY,Subject_Name VARCHAR(255));

-- Lecture Table
CREATE TABLE Lecture (Lecture_ID INT PRIMARY KEY, Lecture_Name VARCHAR(255));

-- Student-Hobby Table (for multivalued attribute)
CREATE TABLE Student_Hobby (Student_ID INT,Hobby VARCHAR(255),
    PRIMARY KEY (Student_ID, Hobby), FOREIGN KEY (Student_ID) REFERENCES Student(Student_ID));

EXPERIMENT 3
Relational Model

AIM

Relation Model represents attributes as columns in tables and different types of attributes like composite, Multi-valued and Derived.

Objectives

Student will be able to learn the structural components of the relational data model. Student will be able to learn **relational models(tables).**

Outcomes:

Student gains the ability

- To describe the Model Structure.
- To define Properties of Relations.
- To define Domains.
- To implement Notations to Describe the Relational Schema

Example: The passenger tables look as below. This is an example.

COLUMNNAME	DATATYPE	CONSTRAINT
BusNo	varchar2(10)	**PrimaryKey**
Source	varchar2(20)	
Destination	varchar2(20)	
CoachType	varchar2(20)	

BUS RELATION

COLUMNNAME	**DATATYPE**	**CONSTRAINT**
Ticket_No	number(9)	**PrimaryKey**
Journeydate	Date	
Age	int(4)	
Sex	Char(10)	
Source	varchar2(10)	
Destination	varchar2(10)	
Dep-time	varchar2(10)	
BusNo	Number2(10)	

TICKET RELATION

Busno	**Source**	**destination**
AP11Z 5467	Hyderabad	Vijayawada
AP29Z 1945	Hyderabad	Vizag
AP28Z 3489	Hyderabad	Warangal

Ppno	**Name**	**Age**	**Sex**	**Address**	**Phone_no**
A-1290	Raju	40	Male	Hyderabad	9966678888
B-1256	Ramu	30	Male	Secunderabad	9949116666
B-5687	Teja	20	Female	Hyderabad	9944666772

EXPERIMENT 4.

Write relational algebra queries for a given set of relations.

AIM: The goal of a relational algebra query language is to fetch data from database or to perform various operations like delete, insert, update on the data.

Relational algebra in DBMS is a procedural query language and main foundation of Relational Algebra is the Relational Database and SQL.

Relational Algebra Operations in DBMS

Different Relational algebra operations in DBMS are listed here .

- Select
- Project
- Union
- Set different
- Cartesian product

Consider two relations French, German

FRENCH

Student_Name	Roll_Number
Ram	01
Mohan	02
Vivek	13
Geeta	17

GERMAN

Student_Name	Roll_Number
Vivek	13
Geeta	17
Shyam	21
Rohan	25

1.Selection(σ): It is used to select required tuples of the relations.

For the above relation, **we** select the tuples where student name ='vivek'

Query: σ (Student_Name='vivek')FRENCH

OUTPUT:

Student_Name	Roll_Number
Vivek	13

2. Projection(π): It is used to project required column data from a relation.

Query : π(Student_Name) FRENCH

OUTPUT:

Student_Name
Ram
Mohan
Vivek
Geeta

3. Union(U): Union operation in relational algebra is the same as union operation in set theory.

Query : π(Student_Name)FRENCH U π(Student_Name)GERMAN

OUTPUT:

Student_Name
Ram
Mohan
Vivek
Geeta
Shyam
Rohan

Note: The only constraint in the union of two relations is that both relations must have the same set of Attributes.

4.Set Difference(-): Set Difference in relational algebra is the same set difference operation as in set theory.
Example: From the above table of FRENCH and GERMAN, Set Difference is used as follows
Query: π(Student_Name)FRENCH - π(Student_Name)GERMAN

OUTPUT:

Student_Name
Ram
Mohan

Note: The only constraint in the Set Difference between two relations is that both relations must have the same set of Attributes.

5. Set Intersection(∩): Set Intersection in relational algebra is the same set intersection operation in set theory.
Example: From the above table of FRENCH and GERMAN, the Set Intersection is used as follows

Query : π(Student_Name)FRENCH ∩ π(Student_Name)GERMAN

OUTPUT:

Student_Name
Vivek
Geeta

Note: The only constraint in the Set Difference between two relations is that both relations must have the same set of Attributes.

6.Cross Product(X): Cross-product between two relations. Let's say A and B, so the cross product between A X B will result in all the attributes of A followed by each attribute of B. Each record of A will pair with every record of B.
Example:

ID	**Course**

A

Name	**Age**	**Sex**
Ram	14	M
Sona	15	F
Kim	20	M

B

1	DS
2	DBMS

A X B

Name	**Age**	**Sex**	**ID**	**Course**
Ram	14	M	1	DS
Ram	14	M	2	DBMS
Sona	15	F	1	DS
Sona	15	F	2	DBMS
Kim	20	M	1	DS
Kim	20	M	2	DBMS

Note: If A has ‘n’ tuples and B has ‘m’ tuples then A X B will have ‘ n*m ‘ tuples.

EXPERIMENT 5

Perform the following :

a)creating a Database ,Viewing all databases, , Viewing all Tables in a database (with and without Constraints), Backing up/Restoring a Database.

Creating a Database:

The `CREATE DATABASE` statement is used to create a new SQL database.

Syntax

CREATE DATABASE *databasename*;

EX. `CREATE DATABASE testDB;`

a) Select database

Syntax

USE DATABASE NAME
EX. USE `testDB;`

b) Viewing all databases

Syntax

```
SHOW DATABASES;
```

c) Viewing all Tables in a database (with and without Constraints)

Syntax

```
SHOW TABLES;
```

d) Backing up/Restoring a Database.

The BACKUP DATABASE statement is used in SQL Server to create a full back up of an existing SQL database.

Syntax

BACKUP DATABASE *databasename* TO DISK = '*filepath*';

Example:

BACKUP DATABASE testDB TO DISK = 'D:\backups\testDB.bak';

EXPERIMENT 6
Practicing DDL commands

Objectives

Student will be able to learn DDL Statements to Create and Manage Tables.

Outcomes

Student gains the ability to

- Categorize the main database objects
- Review the table structure
- List the data types that are available for columns
- Create a simple table
- Describe how constraints are created at the time of table creation
- Describe how schema objects work.

Basic Data Types in SQL

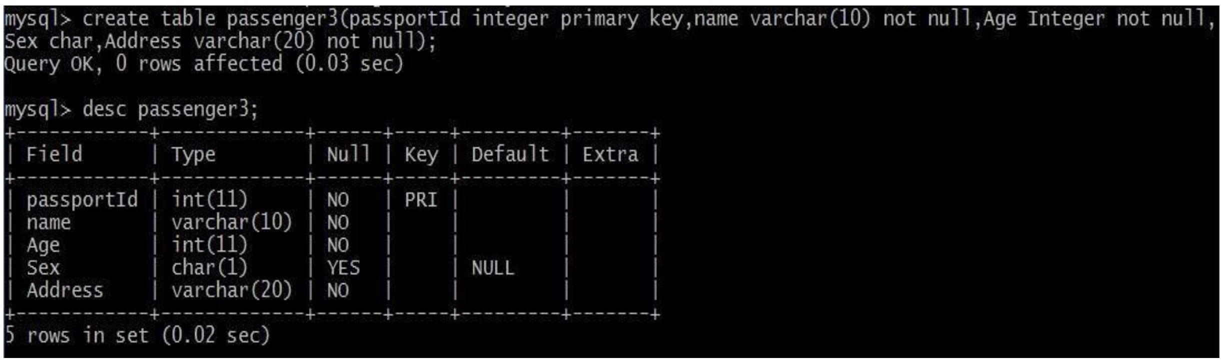
Data Type	Description
Integer	-2,147,483,648 to 2,147,483,647
real	-3.40E + 38 to 3.40E + 38
float	-1.79E + 308 to 1.79E + 308
CHAR	Fixed length with maximum length of 8,000 characters
VARCHAR	Variable length storage with maximum length of 8,000 characters
VARCHAR(max)	Variable length storage with provided max characters
DATE	Stores date in the format YYYY-MM-DD
TIME	Stores time in the format HH:MI:SS
DATETIME	Stores date and time information in the format YYYY-MM-DD HH:MI:SS

SQL-Structured Query Language

AIM : Creating Tables and altering the Tables

Mysql>Create table passenger2(passportId Integer Primary Key, Name varchar(10)NotNull, Age Integer Not Null, Sex char, Address varchar(20) Not Null);

Mysql> desc passenger2;

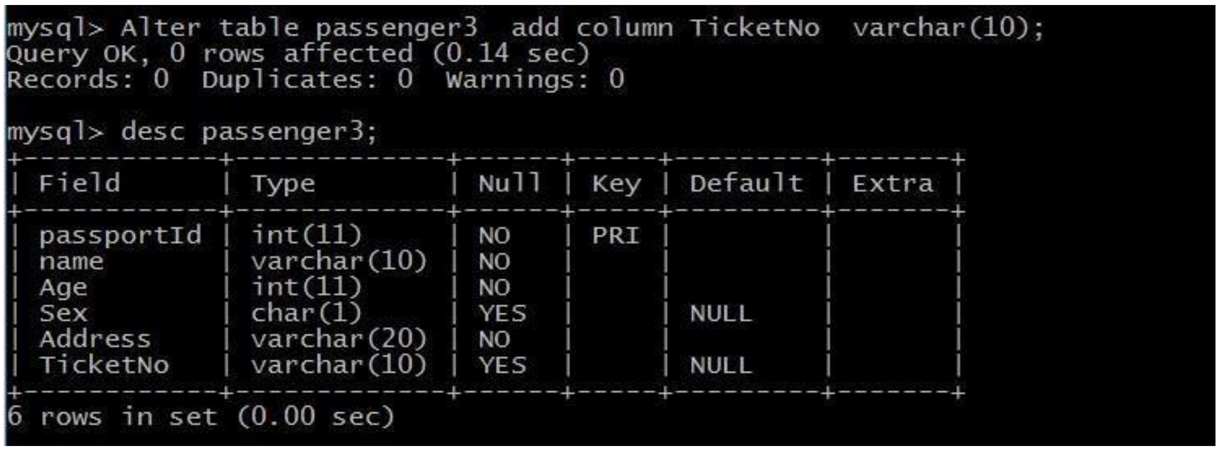


```
mysql> create table passenger3(passportId integer primary key,name varchar(10) not null,Age Integer not null,
Sex char,Address varchar(20) not null);
Query OK, 0 rows affected (0.03 sec)

mysql> desc passenger3;
+------------+-------------+------+-----+---------+-------+
| Field      | Type        | Null | Key | Default | Extra |
+------------+-------------+------+-----+---------+-------+
| passportId | int(11)     | NO   | PRI |         |       |
| name       | varchar(10) | NO   |     |         |       |
| Age        | int(11)     | NO   |     |         |       |
| Sex        | char(1)     | YES  |     | NULL    |       |
| Address    | varchar(20) | NO   |     |         |       |
+------------+-------------+------+-----+---------+-------+
5 rows in set (0.02 sec)
```

USING ALTER COMMAND

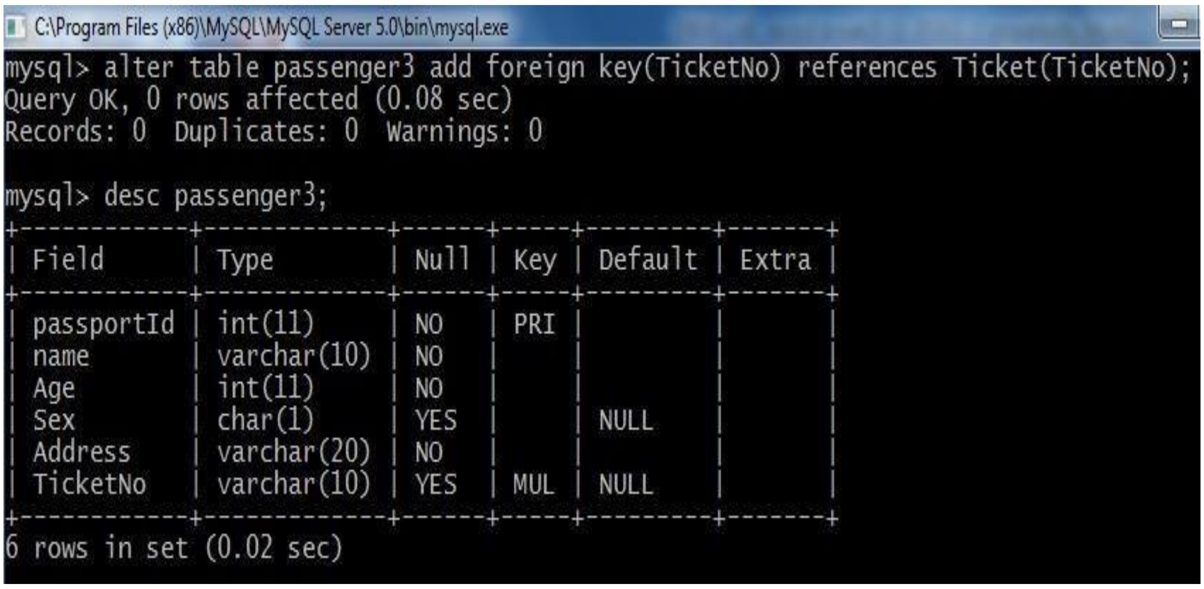
Adding Extra column to Existing Table

Mysql>Alter table passenger3 add column TicketNo varchar(10);



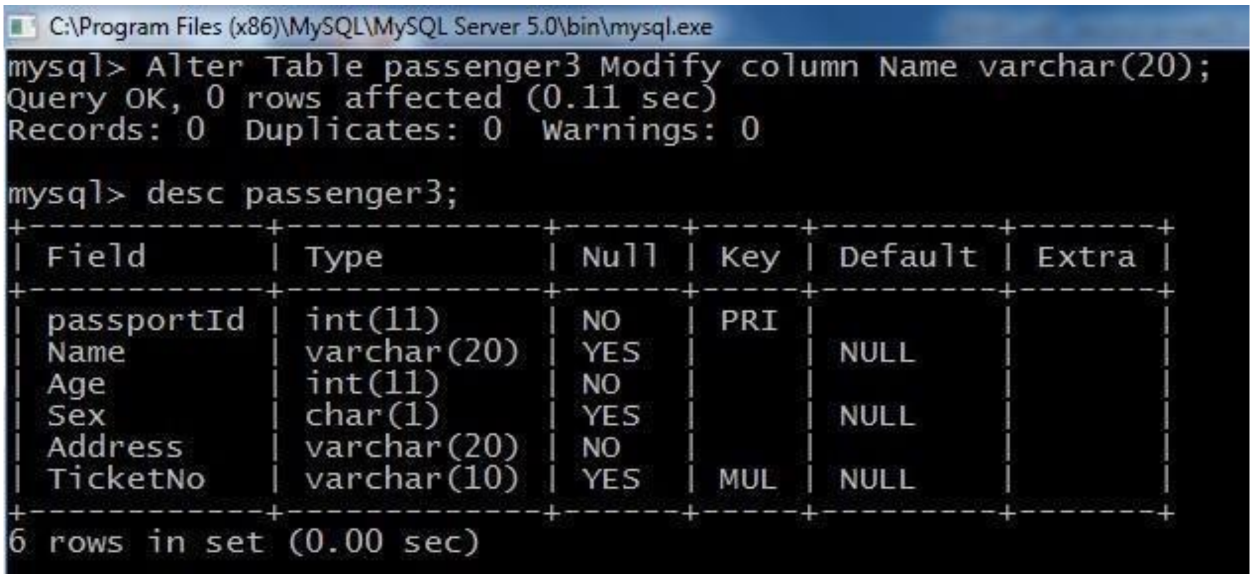
```
mysql> Alter table passenger3  add column TicketNo  varchar(10);
Query OK, 0 rows affected (0.14 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> desc passenger3;
+------------+-------------+------+-----+---------+-------+
| Field      | Type        | Null | Key | Default | Extra |
+------------+-------------+------+-----+---------+-------+
| passportId | int(11)     | NO   | PRI |         |       |
| name       | varchar(10) | NO   |     |         |       |
| Age        | int(11)     | NO   |     |         |       |
| Sex        | char(1)     | YES  |     | NULL    |       |
| Address    | varchar(20) | NO   |     |         |       |
| TicketNo   | varchar(10) | YES  |     | NULL    |       |
+------------+-------------+------+-----+---------+-------+
6 rows in set (0.00 sec)
```

Mysql>Alter Table passenger3 add Foreign key(TicketNo) references Ticket(TicketNo);



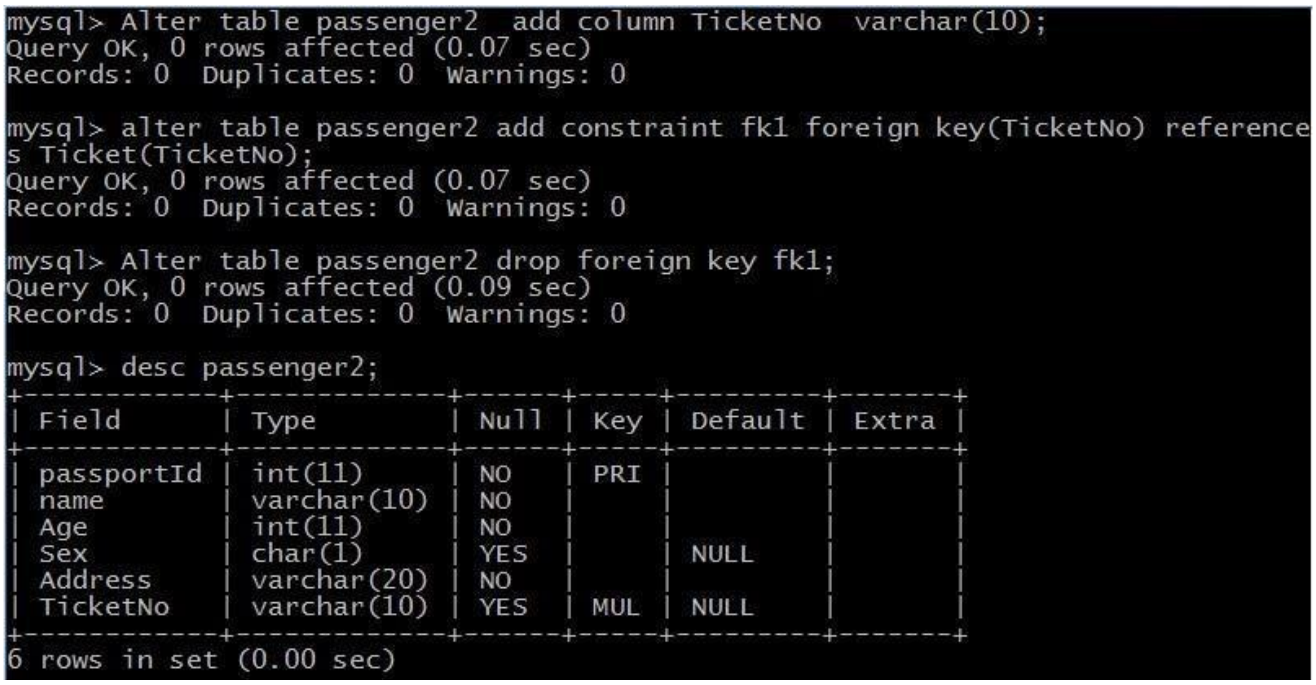
```
C:\Program Files (x86)\MySQL\MySQL Server 5.0\bin\mysql.exe
mysql> alter table passenger3 add foreign key(TicketNo) references Ticket(TicketNo);
Query OK, 0 rows affected (0.08 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> desc passenger3;
+------------+-------------+------+-----+---------+-------+
| Field      | Type        | Null | Key | Default | Extra |
+------------+-------------+------+-----+---------+-------+
| passportId | int(11)     | NO   | PRI |         |       |
| name       | varchar(10) | NO   |     |         |       |
| Age        | int(11)     | NO   |     |         |       |
| Sex        | char(1)     | YES  |     | NULL    |       |
| Address    | varchar(20) | NO   |     |         |       |
| TicketNo   | varchar(10) | YES  | MUL | NULL    |       |
+------------+-------------+------+-----+---------+-------+
6 rows in set (0.02 sec)
```

Mysql>Alter Table passenger3 Modify column Name varchar(20);



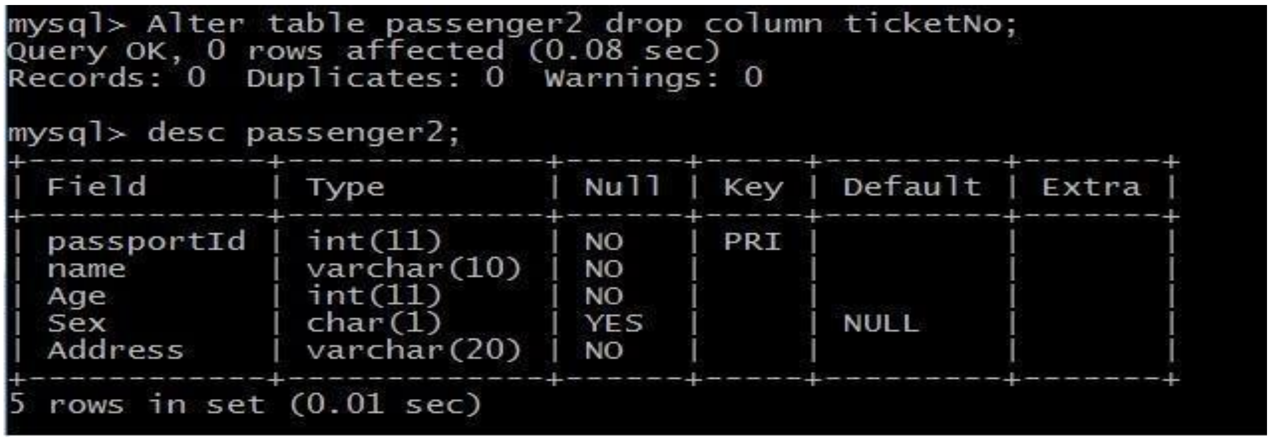
```
C:\Program Files (x86)\MySQL\MySQL Server 5.0\bin\mysql.exe
mysql> Alter Table passenger3 Modify column Name varchar(20);
Query OK, 0 rows affected (0.11 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> desc passenger3;
+------------+-------------+------+-----+---------+-------+
| Field      | Type        | Null | Key | Default | Extra |
+------------+-------------+------+-----+---------+-------+
| passportId | int(11)     | NO   | PRI |         |       |
| Name       | varchar(20) | YES  |     | NULL    |       |
| Age        | int(11)     | NO   |     |         |       |
| Sex        | char(1)     | YES  |     | NULL    |       |
| Address    | varchar(20) | NO   |     |         |       |
| TicketNo   | varchar(10) | YES  | MUL | NULL    |       |
+------------+-------------+------+-----+---------+-------+
6 rows in set (0.00 sec)
```

Mysql>Alter table passenger drop foreign key fk1;



```
mysql> Alter table passenger2  add column TicketNo  varchar(10);
Query OK, 0 rows affected (0.07 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> alter table passenger2 add constraint fk1 foreign key(TicketNo) reference
s Ticket(TicketNo);
Query OK, 0 rows affected (0.07 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> Alter table passenger2 drop foreign key fk1;
Query OK, 0 rows affected (0.09 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> desc passenger2;
+------------+-------------+------+-----+---------+-------+
| Field      | Type        | Null | Key | Default | Extra |
+------------+-------------+------+-----+---------+-------+
| passportId | int(11)     | NO   | PRI |         |       |
| name       | varchar(10) | NO   |     |         |       |
| Age        | int(11)     | NO   |     |         |       |
| Sex        | char(1)     | YES  |     | NULL    |       |
| Address    | varchar(20) | NO   |     |         |       |
| TicketNo   | varchar(10) | YES  | MUL | NULL    |       |
+------------+-------------+------+-----+---------+-------+
6 rows in set (0.00 sec)
```

Mysql> Alter table passenger2 Drop column TicketNo;



```
mysql> Alter table passenger2 drop column ticketNo;
Query OK, 0 rows affected (0.08 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> desc passenger2;
+------------+-------------+------+-----+---------+-------+
| Field      | Type        | Null | Key | Default | Extra |
+------------+-------------+------+-----+---------+-------+
| passportId | int(11)     | NO   | PRI |         |       |
| name       | varchar(10) | NO   |     |         |       |
| Age        | int(11)     | NO   |     |         |       |
| Sex        | char(1)     | YES  |     | NULL    |       |
| Address    | varchar(20) | NO   |     |         |       |
+------------+-------------+------+-----+---------+-------+
5 rows in set (0.01 sec)
```

EXPERIMENT 7
DML COMMANDS

AIM: Create a DML Commands are used to manage data within the scheme objects.

Objectives:

Student will be able to learn commands that make changes in relational database and transaction management.

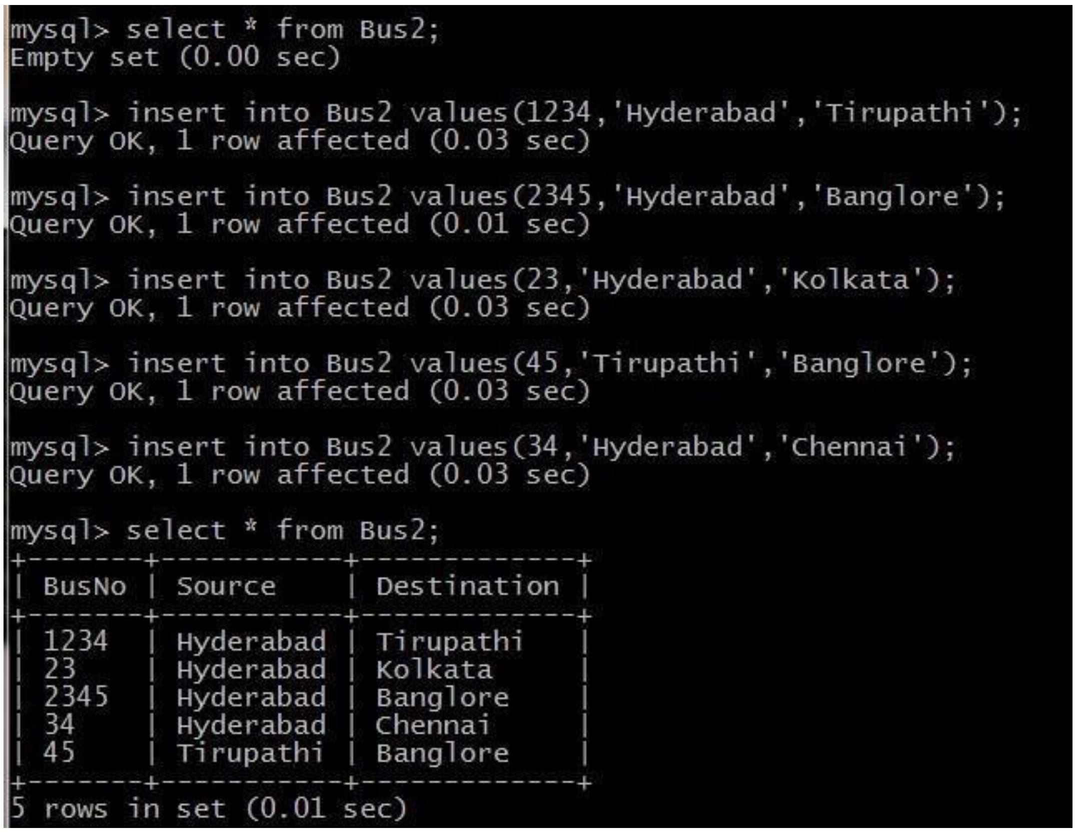
Outcomes

Student gains the knowledge to perform transactions like updating, deleting, inserting and selecting data from a data base.

DML Commands:

INSERT COMMAND ON BUS2 & PASSENGER2 RELATIONS

```
mysql> select * from Bus2; Empty set (0.00 sec)

mysql> insert into Bus2 values(1234,'Hyderabad','Tirupathi');Query OK, 1 row
affected (0.03 sec)

mysql> insert into Bus2 values(2345,'Hyderabad','Banglore');Query OK, 1 row
affected (0.01 sec)

mysql> insert into Bus2 values(23,'Hyderabad','Kolkata');Query OK, 1 row affected
(0.03 sec)

mysql> insert into Bus2 values(45,'Tirupathi','Banglore');Query OK, 1 row affected
(0.03 sec)

mysql> insert into Bus2 values(34,'Hyderabad','Chennai');Query OK, 1 row
affected (0.03 sec)

    mysql> select * from Bus2;
```



```
mysql> select * from Bus2;
Empty set (0.00 sec)

mysql> insert into Bus2 values(1234,'Hyderabad','Tirupathi');
Query OK, 1 row affected (0.03 sec)

mysql> insert into Bus2 values(2345,'Hyderabad','Banglore');
Query OK, 1 row affected (0.01 sec)

mysql> insert into Bus2 values(23,'Hyderabad','Kolkata');
Query OK, 1 row affected (0.03 sec)

mysql> insert into Bus2 values(45,'Tirupathi','Banglore');
Query OK, 1 row affected (0.03 sec)

mysql> insert into Bus2 values(34,'Hyderabad','Chennai');
Query OK, 1 row affected (0.03 sec)

mysql> select * from Bus2;
+-------+-----------+-------------+
| BusNo | Source    | Destination |
+-------+-----------+-------------+
| 1234  | Hyderabad | Tirupathi   |
| 23    | Hyderabad | Kolkata     |
| 2345  | Hyderabad | Banglore    |
| 34    | Hyderabad | Chennai     |
| 45    | Tirupathi | Banglore    |
+-------+-----------+-------------+
5 rows in set (0.01 sec)
```

mysql> select * from Passenger2;Empty set (0.00 sec)

mysql> insert into Passenger2 values(145,'Ramesh',45,'M','abc123');Query OK, 1 row affected (0.05 sec)

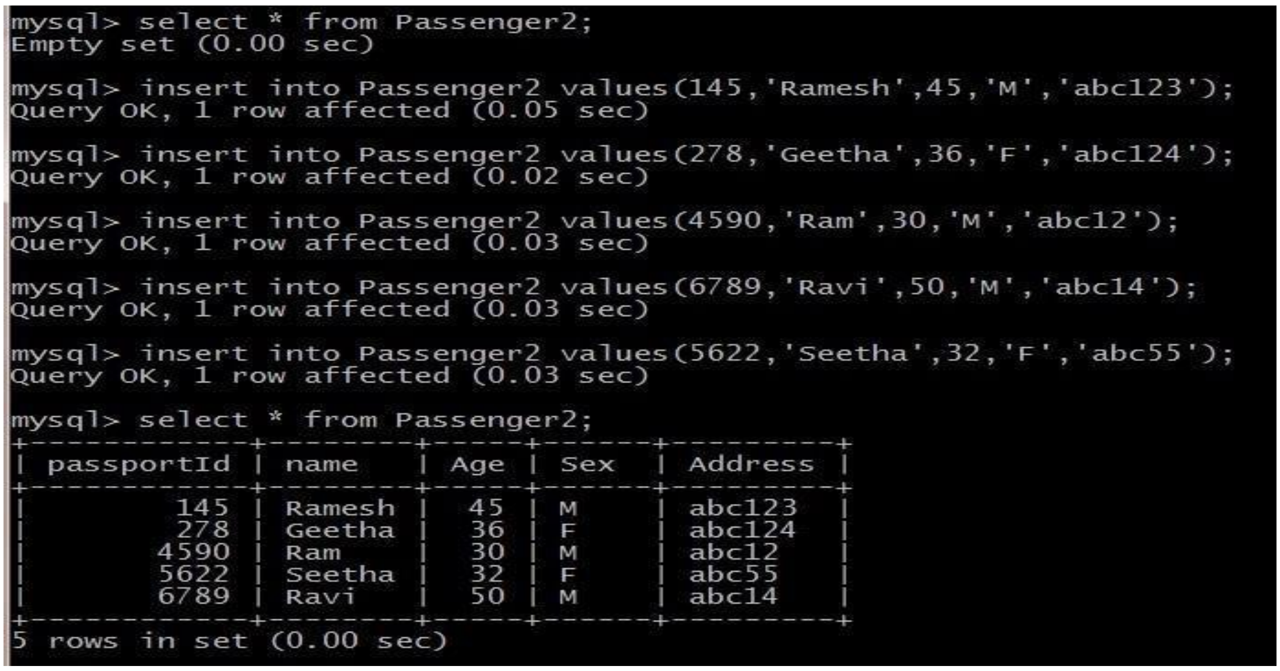
mysql> insert into Passenger2 values(278,'Geetha',36,'F','abc124');Query OK, 1 row affected (0.02 sec)

mysql> insert into Passenger2 values(4590,'Ram',30,'M','abc12');Query OK, 1 row affected (0.03 sec)

mysql> insert into Passenger2 values(6789,'Ravi',50,'M','abc14');Query OK, 1 row affected (0.03 sec)

mysql> insert into Passenger2 values(5622,'Seetha',32,'F','abc55');Query OK, 1 row affected (0.03 sec)

mysql> select * from Passenger2;



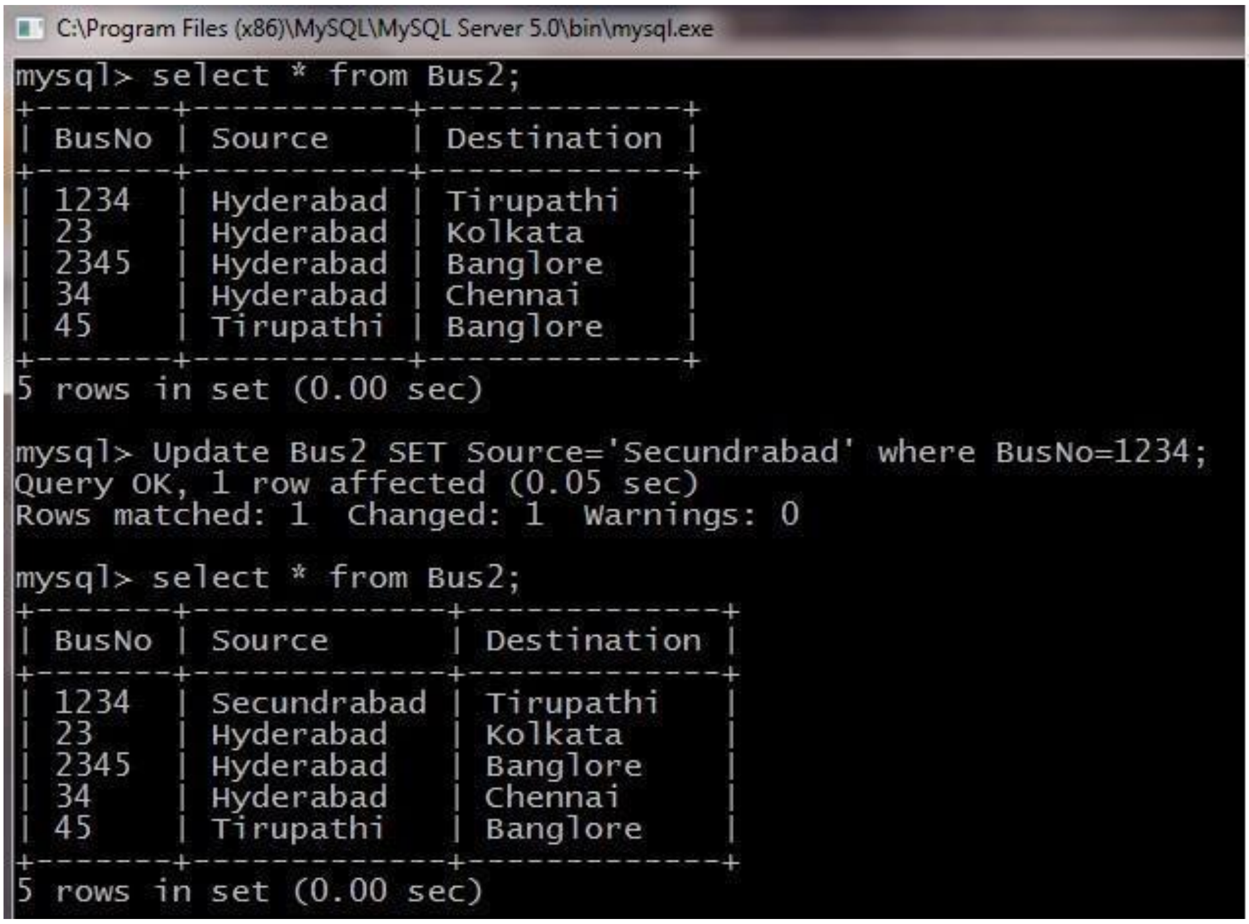
```
mysql> select * from Passenger2;
Empty set (0.00 sec)

mysql> insert into Passenger2 values(145,'Ramesh',45,'M','abc123');
Query OK, 1 row affected (0.05 sec)

mysql> insert into Passenger2 values(278,'Geetha',36,'F','abc124');
Query OK, 1 row affected (0.02 sec)

mysql> insert into Passenger2 values(4590,'Ram',30,'M','abc12');
Query OK, 1 row affected (0.03 sec)

mysql> insert into Passenger2 values(6789,'Ravi',50,'M','abc14');
Query OK, 1 row affected (0.03 sec)

mysql> insert into Passenger2 values(5622,'Seetha',32,'F','abc55');
Query OK, 1 row affected (0.03 sec)

mysql> select * from Passenger2;
+------------+--------+-----+------+---------+
| passportId | name   | Age | Sex  | Address |
+------------+--------+-----+------+---------+
|        145 | Ramesh |  45 | M    | abc123  |
|        278 | Geetha |  36 | F    | abc124  |
|       4590 | Ram    |  30 | M    | abc12   |
|       5622 | Seetha |  32 | F    | abc55   |
|       6789 | Ravi   |  50 | M    | abc14   |
+------------+--------+-----+------+---------+
5 rows in set (0.00 sec)
```

UPDATE COMMAND ON BUS2 RELATION

UPDATE Selected Rows & Multiple Rows

mysql> Update Bus2 SET Source='Secundrabad' where BusNo=1234; Query OK,

1 row affected (0.05 sec)Rows matched: 1 Changed: 1 Warnings: 0

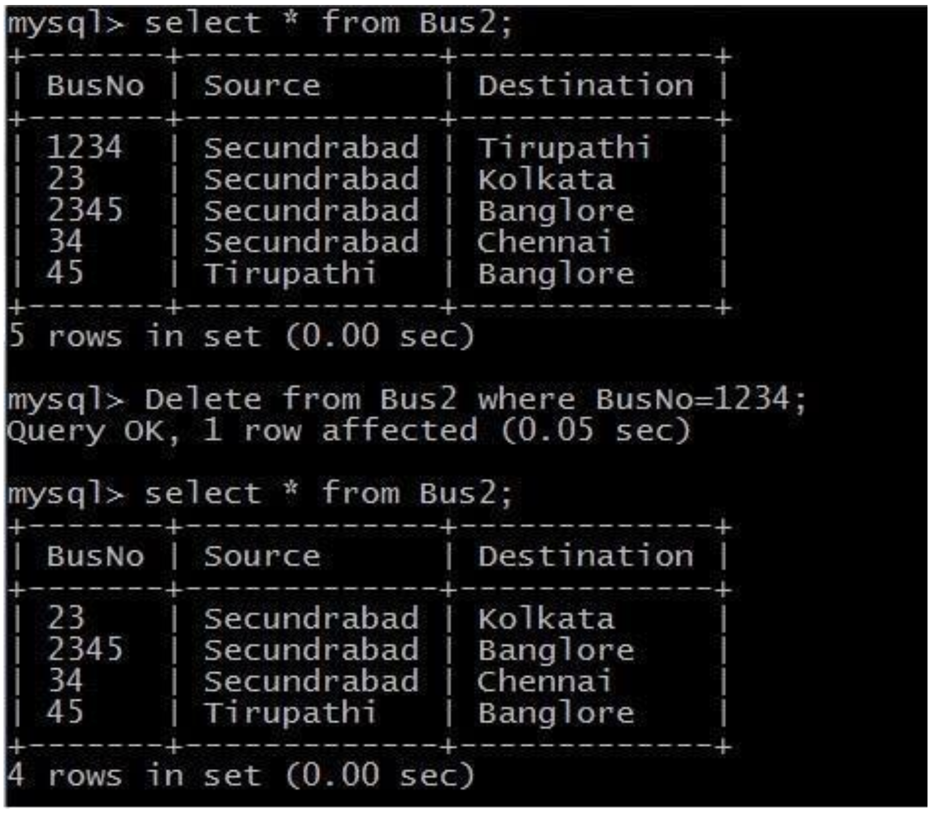


```
C:\Program Files (x86)\MySQL\MySQL Server 5.0\bin\mysql.exe
mysql> select * from Bus2;
+-------+-----------+-------------+
| BusNo | Source    | Destination |
+-------+-----------+-------------+
| 1234  | Hyderabad | Tirupathi   |
| 23    | Hyderabad | Kolkata     |
| 2345  | Hyderabad | Banglore    |
| 34    | Hyderabad | Chennai     |
| 45    | Tirupathi | Banglore    |
+-------+-----------+-------------+
5 rows in set (0.00 sec)

mysql> Update Bus2 SET Source='Secundrabad' where BusNo=1234;
Query OK, 1 row affected (0.05 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> select * from Bus2;
+-------+-------------+-------------+
| BusNo | Source      | Destination |
+-------+-------------+-------------+
| 1234  | Secundrabad | Tirupathi   |
| 23    | Hyderabad   | Kolkata     |
| 2345  | Hyderabad   | Banglore    |
| 34    | Hyderabad   | Chennai     |
| 45    | Tirupathi   | Banglore    |
+-------+-------------+-------------+
5 rows in set (0.00 sec)
```

DELETE COMMAND ON BUS2 RELATION

DELETES Selected Rows and Multiple Rows

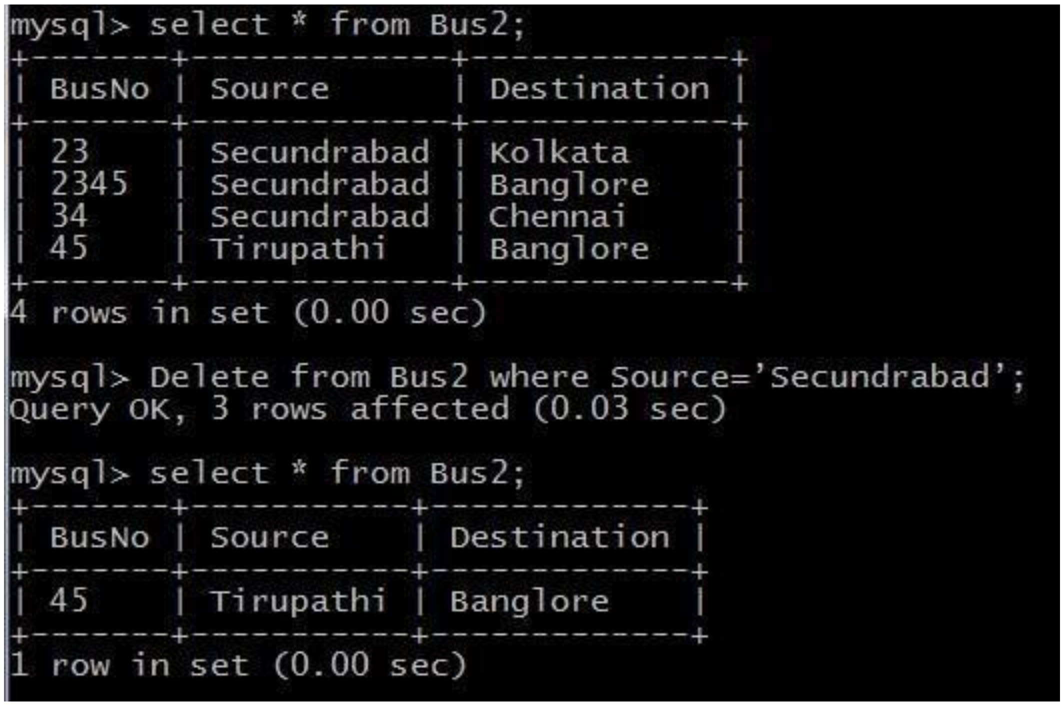
mysql> Delete from Bus2 where BusNo=1234; Query OK, 1 row affected (0.05 sec) mysql> select * from Bus2;



```
mysql> select * from Bus2;
+-------+------------+------------+
| BusNo | Source     | Destination |
+-------+------------+------------+
| 1234  | Secundrabad | Tirupathi   |
| 23    | Secundrabad | Kolkata     |
| 2345  | Secundrabad | Banglore    |
| 34    | Secundrabad | Chennai     |
| 45    | Tirupathi   | Banglore    |
+-------+------------+------------+
5 rows in set (0.00 sec)

mysql> Delete from Bus2 where BusNo=1234;
Query OK, 1 row affected (0.05 sec)

mysql> select * from Bus2;
+-------+------------+------------+
| BusNo | Source     | Destination |
+-------+------------+------------+
| 23    | Secundrabad | Kolkata     |
| 2345  | Secundrabad | Banglore    |
| 34    | Secundrabad | Chennai     |
| 45    | Tirupathi   | Banglore    |
+-------+------------+------------+
4 rows in set (0.00 sec)
```

mysql> Delete from Bus2 where Source='Secundrabad'; Query OK, 1 row

affected (0.05 sec)mysql> select * from Bus2;



```
mysql> select * from Bus2;
+-------+-------------+-------------+
| BusNo | Source      | Destination |
+-------+-------------+-------------+
| 23    | Secundrabad | Kolkata     |
| 2345  | Secundrabad | Banglore    |
| 34    | Secundrabad | Chennai     |
| 45    | Tirupathi   | Banglore    |
+-------+-------------+-------------+
4 rows in set (0.00 sec)

mysql> Delete from Bus2 where Source='Secundrabad';
Query OK, 3 rows affected (0.03 sec)

mysql> select * from Bus2;
+-------+-----------+-------------+
| BusNo | Source    | Destination |
+-------+-----------+-------------+
| 45    | Tirupathi | Banglore    |
+-------+-----------+-------------+
1 row in set (0.00 sec)
```

Queries using Aggregate Operators**,**
Queries using Group by and Having clauses ,Date Functions, String Functions, Math Functions.

EXPERIMENT 8

For a given set of relation Schemas, Create tables and perform the following Simple Queries, Simple Queries with Aggregate functions, Queries with aggregate functions (group by And having clause), Queries involving –Date Functions, String Functions, Math Functions.

AIM :-
Queriesusingaggregatefunctions(COUNT,AVG,MIN,MAX,SU M),Group by,Orderby,Having.

_id	_nam	ge	alary
)1	nu	2	)00
)2	hane	9	)00
)3	ohan	4	)00
)4	cott	4	)000
)5	iger	5	)00
)6	lex	7	)00
)7	bhi	9	)00

(i) Create Employee table containing allRecords.

SQL> create table emp(eidnumber,ename varchar2(10),age number,salary

number); Table created. SQL> desc emp;

ıme	ıll?	ype
-------------------- ID	---	----------------------- NUMBER
NAME AGE		ARCHAR2(10 NUMBER
SALARY		NUMBER

(ii) Count number of employee names from employeetable.

SQL> select count(ename) from emp; COUNT(ENAME)

```
-------------------------
7
```

(iii) Find the Maximum age from employeetable.

```
SQL> select max(age) from emp;
MAX(AGE)
----------------- 44
```

(iv) Find the Minimum age from employeetable.

```
SQL> select min(age) from emp;
MIN(AGE)
---------------- 22
```

Display the Sum of age employeetable.

```
SQL> select sum(age) from emp;
SUM(AGE)
---------------- 220
```

Display the Average of age from Employeetable.

```
SQL> select avg(age) from emp; AVG(AGE)
---------------- 31.4285714
```

Create a View for age in employeetable.

```
SQL> create or replace view A as select age from emp where age<30; View created.
```

Displayviews

```
SQL> select * from A;
AGE
____________________________________________________________
 22
29
27
29
```

(v) Find grouped salaries of employees.(group byclause)

```
SQL> select salary from emp group by salary;
 SALARY
 --------------
 9000
```

```
10000
8000
6000
7000
```

(x).Find salaries of employee in Ascending Order.(order by clause)

```
SQL> select ename,salary from emp order by salary;
    NAME                ALARY
                 ---    -------
    ------
    nan                 00
    ex                  00
    ane                 00
    hi                  00
    er                  00
    u                   00
                 sc     000
    7 rowsselected.
```

(xi) Find salaries of employee in DescendingOrder.SQL> select ename,salary from emp order by salary desc;

```
  ENAME          SALARY
--------------   ---------------
 scott           10000
 anu              9000
 Shane             8000
 Abhi              8000
 Tiger             8000
 Alex              7000
 Rohan              6000
 7 rows selected.
```

(xii)HavingClause.

SQL> select ename,salary from emp where age<29 group by ename,salary having salary<10000;

```
ENAME    SALARY
```

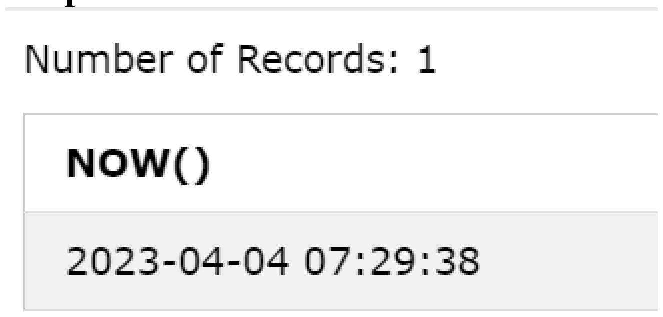
```
            -----------   --------------
alex        7000
anu         9000
```

Date functions:

NOW()
Returns the current date and time.

Query:
SELECT NOW();

Output:

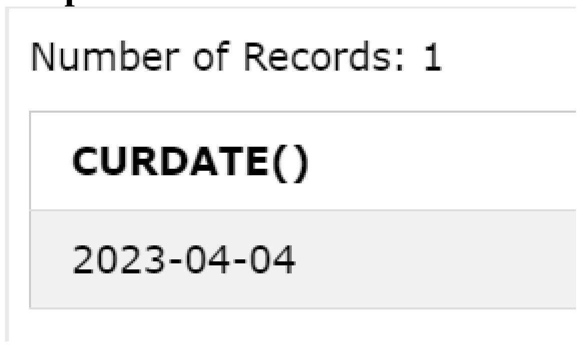


```
Number of Records: 1

NOW()
2023-04-04 07:29:38
```

CURDATE()
Returns the current date.

Query:
SELECT CURDATE();

Output:

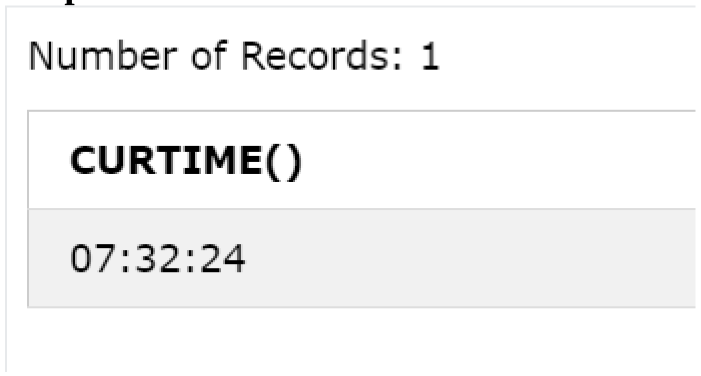


```
Number of Records: 1

CURDATE()
2023-04-04
```

CURTIME()
Returns the current time.

Query:
SELECT CURTIME();

Output:



```
Number of Records: 1

CURTIME()
07:32:24
```

DATE()
Extracts the date part of a date or date/time expression. Example: For the below table named ‘Test’

I	ame	irthTime
120	ratik	996-09-26 16:44:15.581

Query:
SELECT Name, DATE(BirthTime)

AS BirthDate FROM Test;

Output:

ame	irthDate
ratik	996-09-26

EXTRACT()
Returns a single part of a date/time.

Syntax
EXTRACT(unit FROM date);

Several units can be considered but only some are used such as **MICROSECOND, SECOND, MINUTE, HOUR, DAY, WEEK, MONTH, QUARTER, YEAR, etc.** And ‘date’ is a valid date expression. Example: For the below table named ‘Test’

l	Iame	3irthTime
120	'ratik	996-09-26 16:44:15.581

Query:
SELECT Name, Extract(DAY FROM

BirthTime) AS BirthDay FROM Test;

Output:

ame	3irthday
'ratik	6

Query:
SELECT Name, Extract(YEAR FROM BirthTime)

AS BirthYear FROM Test;

Output:

ame	3irthYear
'ratik	996

Query:
SELECT Name, Extract(SECOND FROM

BirthTime) AS BirthSecond FROM Test;

Output:

ame	3irthSecond
'ratik	81

DATE_ADD()
 Adds a specified time interval to a date.

Syntax:
DATE_ADD(date, INTERVAL expr type);

Where, date – valid date expression, and expr is the number of intervals we want to add. and type can be one of the following: MICROSECOND, SECOND, MINUTE, HOUR, DAY, WEEK, MONTH, QUARTER, YEAR, etc. Example: For the below table named ‘Test’

l	Name	BirthTime
120	ratik	996-09-26 16:44:15.581

DATE_SUB()
Subtracts a specified time interval from a date. The syntax for DATE_SUB is the same as DATE_ADD just the difference is that DATE_SUB is used to subtract a given interval of date.

String functions

CONCAT(): Concatenates two or more strings together.

Example:

```
SELECT CONCAT('Hello ', 'World') AS concatenated_string;
Output:

+-------------------+
| concatenated_string |
+-------------------+
| Hello World       |
+-------------------+
```

UPPER(): Converts a string to uppercase.

Example:

```
SELECT UPPER('hello') AS uppercase_string;
Output:

+------------------+
| uppercase_string |
+------------------+
| HELLO            |
+------------------+
```

LOWER(): Converts a string to lowercase.

Example:

```
SELECT LOWER('WORLD') AS lowercase_string;
```

Output:

```
+------------------+
| lowercase_string |
+------------------+
| world            |
+------------------+
```

LENGTH(): Returns the length of a string.

Example:

```
SELECT LENGTH('Hello World') AS string_length;
```

Output:

```
+--------------+
| string_length |
+--------------+
| 11           |
+--------------+
```

SUBSTRING(): Extracts a substring from a string.

Example:

```
SELECT SUBSTRING('Hello World', 7, 5) AS extracted_substring;
```

Output:

```
+-------------------+
| extracted_substring |
+-------------------+
| World             |
+-------------------+
```

TRIM(): Removes leading and trailing spaces from a string.

Example:

```
SELECT TRIM('   Hello World   ') AS trimmed_string;
Output:

+---------------+
| trimmed_string |
+---------------+
| Hello World   |
+---------------+
```

REPLACE(): Replaces occurrences of a substring within a string with another substring.

Example:

```
SELECT REPLACE('Hello World', 'Hello', 'Hi') AS replaced_string;
Output:

+-----------------+
| replaced_string  |
+-----------------+
| Hi World        |
    +-----------------+
```

EXPERIMENT 9

For a given set of Relation tables perform the following

a. Creating Views (with and without check option), Dropping views, Selecting from a view

a).Creating Views

Database views are created using the **CREATE VIEW** statement. Views can be created from a single table, multiple tables or another view.

To create a view, a user must have the appropriate system privilege according to the specific implementation.

The basic **CREATE VIEW** syntax is as follows −

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

You can include multiple tables in your SELECT statement in a similar way as you use them in a normal SQL SELECT query

Example

Consider the CUSTOMERS table having the following records −

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
6	Komal	**22**	Mp	4500.00
7	Muffy	**24**	Indore	10000.00

Following is an example to create a view from the CUSTOMERS table. This view would be used to have customer name and age from the CUSTOMERS table.

```
SQL > CREATE VIEW CUSTOMERS_VIEW AS
SELECT name, age
```

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

OUTPUT ;

NAME	AGE
Ramesh	**32**
Khilan	**25**
kaushik	**23**
Chaitali	**25**
Hardik	**27**
Komal	**22**
Muffy	**24**

The WITH CHECK OPTION

The following code block has an example of creating same view CUSTOMERS_VIEW with the WITH CHECK OPTION.

```
CREATE VIEW CUSTOMERS_VIEW AS
SELECT name, age
FROM  CUSTOMERS
WHERE age IS NOT NULL
WITH CHECK OPTION;
```

Updating a View

```
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

DELETING ROWS INTO A VIEW

Rows of data can be deleted from a view. The same rules that apply to the UPDATE and INSERT commands apply to the DELETE command.
Following is an example to delete a record having AGE = 22.

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

OUTPUT

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

DROPPING VIEWS

```
DROP VIEW CUSTOMERS_VIEW;
```

EXPERIMENT 10
Normalization of tables

Objectives:

Student will be able to learn to avoid problems that are associated with updating redundant data.

Outcomes:

Student gains the knowledge to build the database that does not have redundant data.

Normalization

Database normalization is a technique for designing relational database tables to minimize duplication of information and, in so doing, to safeguard the database against certain types of logical or structural problems, namely data anomalies.

In relational databases, normalization is a process that eliminates redundancy, organizes data efficiently; Normalization is the process of efficiently organizing data in a database. There are two goals of the normalization process: eliminating redundant data(for example, storing the same data in more than one table) and ensuring data dependencies make sense(only storing related data in a table). Both of these are worthy goals as they reduce the amount of space a database consumes and ensure that data is logically stored.

A basic objective of the first normal form defined by Edgar Frank "Ted" Codd in 1970 was to permit data to be queried and manipulated using a "universal data sub-language" grounded in first-order logic. (SQL is an example of such a data sub-language, albeit one that Codd regarded as seriously flawed.)

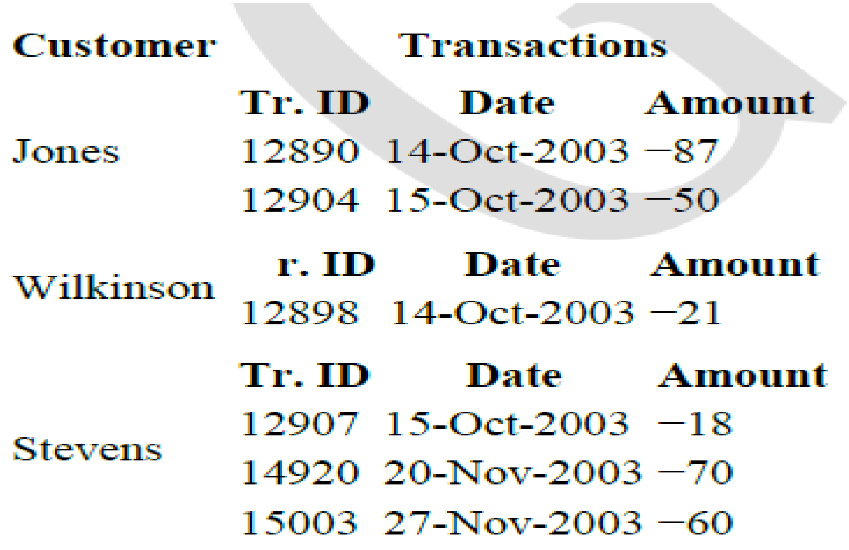
The objectives of normalization beyond 1NF (First Normal Form) were stated as follows by Codd:

1. To free the collection of relations from undesirable insertion, update and deletion dependencies;

2. To reduce the need for restructuring the collection of relations, as new types of data are introduced, and thus increase the life span of application programs;

3. To make the relational model more informative to users;

4. To make the collection of relations neutral to the query statistics, where these statistics are liable to change as time goes by.



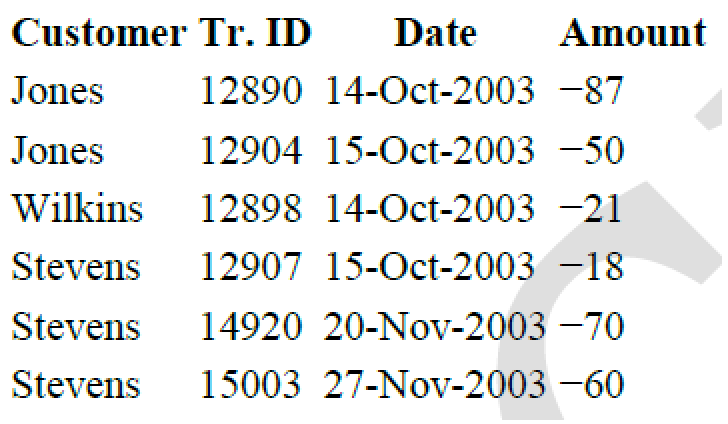
Customer	Transactions		
	Tr. ID	Date	Amount
Jones	12890	14-Oct-2003	−87
	12904	15-Oct-2003	−50
Wilkinson	r. ID	Date	Amount
	12898	14-Oct-2003	−21
	Tr. ID	Date	Amount
Stevens	12907	15-Oct-2003	−18
	14920	20-Nov-2003	−70
	15003	27-Nov-2003	−60

To each customer corresponds a repeating group of transactions. The automated evaluation of any query relating to customers' transactions therefore would broadly involve two stages:

1. Unpacking one or more customers' groups of transactions allowing the individual transactions in a group to be examined, and
2. Deriving a query result based on the results of the first stage

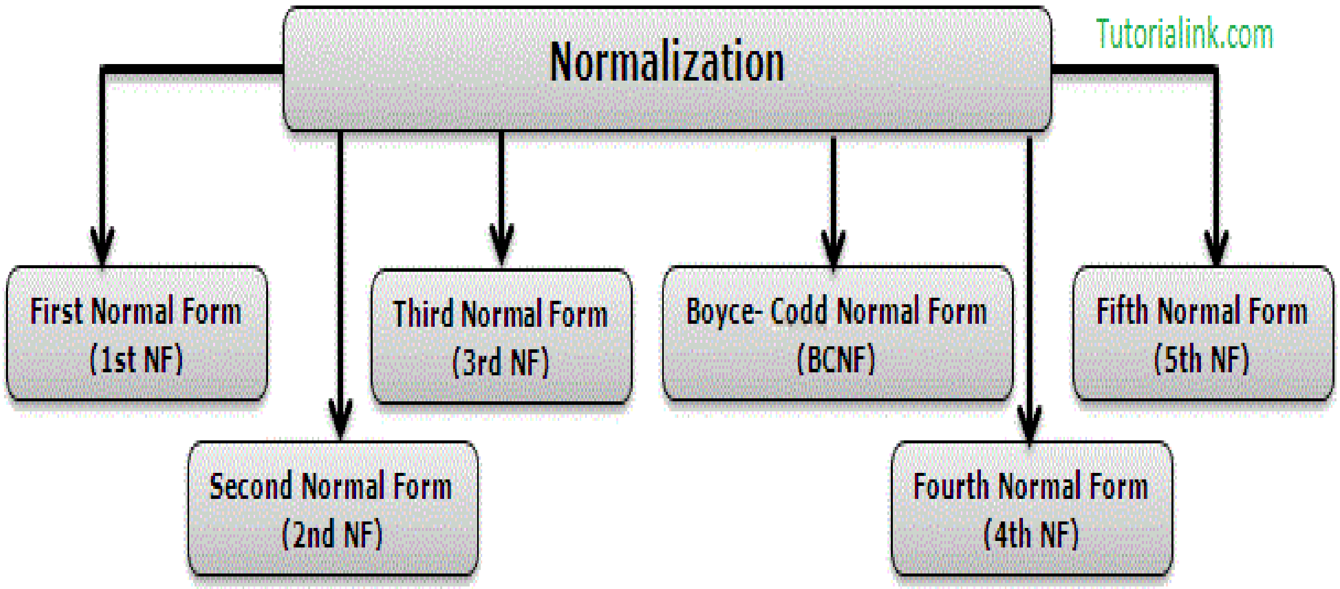
For example, in order to find out the monetary sum of all transactions that occurred in October 2003 for all customers, the system would have to know that it must first unpack the *Transactions* group of each customer, then sum the *Amounts* of all transactions thus obtained where the *Date* of the transaction falls in October 2003.

One of Codd's important insights was that this structural complexity could always be removed completely, leading to much greater power and flexibility in the way queries could be formulated (by users and applications) and evaluated (by the DBMS). The normalized equivalent of the structure above would look like this:



Customer	Tr. ID	Date	Amount
Jones	12890	14-Oct-2003	−87
Jones	12904	15-Oct-2003	−50
Wilkins	12898	14-Oct-2003	−21
Stevens	12907	15-Oct-2003	−18
Stevens	14920	20-Nov-2003	−70
Stevens	15003	27-Nov-2003	−60

Now each row represents an individual credit card transaction, and the DBMS can obtain the answer of interest, simply by finding all rows with a Date falling in October, and summing their Amounts. The data structure places all of the values on an equal footing, exposing each to the DBMS directly, so each can potentially participate directly in queries; whereas in the previous situation some values were embedded in lower-level structures that had to be handled specially. Accordingly, the normalized design lends itself to general-purpose query processing, whereas the unnormalized design does not.


Normalization
Tutorialink.com
First Normal Form (1st NF)
Second Normal Form (2nd NF)
Third Normal Form (3rd NF)
Boyce- Codd Normal Form (BCNF)
Fourth Normal Form (4th NF)
Fifth Normal Form (5th NF)

A relation is in **1 NF** if and only if all the domains(fields or columns) contain only atomic values(it should not contain list of values or set of values)

A relation R is said to be in **2 NF** if and only if it is in 1 NF(first normal form) and all non-prime attributes are fully functional dependent on the primary key

"A relation R is in **3NF**, if it is in Second normal form, and not any non-prime attribute of the relation is transitively dependent on the primary key."

A relation R is said to be in **BCNF** if and only if it is in 3 NF and all the determinants are candidate keys

For a table to satisfy the **Fourth Normal Form**, it should satisfy the following two conditions

It should be in the **Boyce-Codd Normal Form**.

And, the table should not have any **Multi-valued Dependency**

<u>Fifth normal form (5NF)</u>
A relation R is in 5NF if and only if it satisfies the following conditions:

R should be already in 4NF.

It cannot be further decomposed

5NF is satisfied when all the tables are broken into as many tables as possible in order to avoid redundancy.

The broken smaller tables must satisfy join dependency. Means after joining the smaller tables we must get the original table without losing any data.

5NF is also known as Project-join normal form (PJ/NF).
If there is any decomposition of the Relational Schema R there will be lossless decomposition in join dependency. So, the 5NF is called as project-join normal form (PJNF).

EXPERIMENT 11

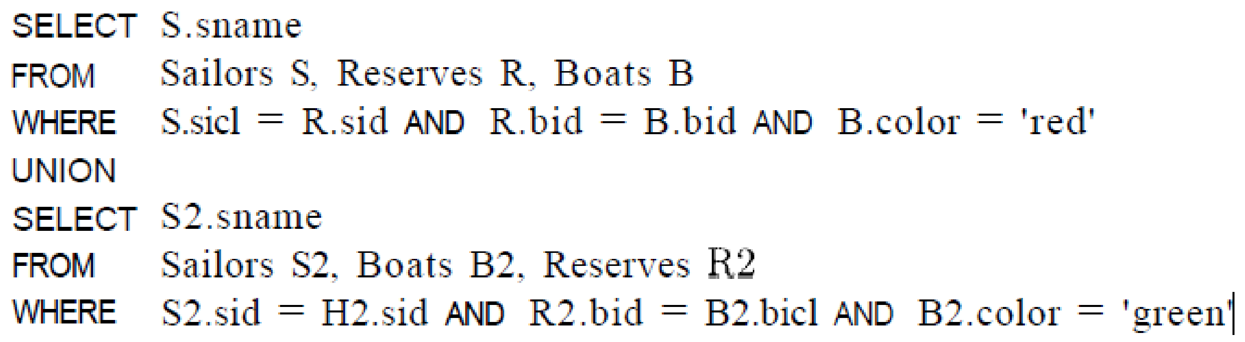
QUERIES USING UNION, INTERSECT, EXCEPT, IN, ANY, ALL, EXISTS

Objectives:

Student will be able to learn to operate on multiple result sets to return a single result set. Student will be able to learn to perform nested Queries, Correlated Queries

UNIOR, INTERSECT AND EXCEPT

Q: Find the names of sailors who have reserved a red or a green boat.



```
SELECT  S.sname
FROM    Sailors S, Reserves R, Boats B
WHERE   S.sicl = R.sid AND R.bid = B.bid AND B.color = 'red'
UNION
SELECT  S2.sname
FROM    Sailors S2, Boats B2, Reserves R2
WHERE   S2.sid = H2.sid AND R2.bid = B2.bicl AND B2.color = 'green'
```

Q: Find the names of sailors who have reserved both a red and a green boat.

```
SELECT    S.sname
From     Sailors   S,  Reserves  R, Boats   B
Where    S.sid=R.sid  AND  R.bid=B.bid   AND  B.color=’red’
INTERSECT
SELECT    S2.sname
From     Sailors   S2,  Reserves  R2, Boats   B2
Where    S2.sid=R2.sid  AND  R2.bid=B2.bid   AND  B2.color=’green’;
```

Q: Find the sids of all sailors who have reserved red boats but not green boats

```
SELECT  S.sid
FROM    Sailors S, Reserves R, Boats B
WHERE   S.sid = R.sid AND  R.bid = B.bid AND  B.color = 'red'
EXCEPT
SELECT  S2.sid
FROM    Sailors S2, Reserves R2, Boats B2
WHERE   S2.sid = R2.sid AND  R2.bid = B2.bid AND  B2.color = ‘green’
```

Set-Comparison Operators

Q: Find sailors whose rating is better than some sailor called Horatio.

```
SELECT  S.sid
FROM    Sailors S
WHERE   S.rating > ANY  ( SELECT  S2.rating
                          FROM    Sailors S2
                          WHERE   S2.sname =  'Horatio' )
```

Q: Find the Sailors with the highest rating.

```
SELECT  S.sid
FROM    Sailors S
WHERE   S.rating >= ALL  ( SELECT  S2.rating
                           FROM    Sailors S2 )
```

SQL Join statement is used to combine data or rows from two or more tables based on a common field between them.

Consider the two tables below as follows:
Student

ROLL_NO	NAME	ADDRESS	PHONE	Age
1	HARSH	DELHI	XXXXXXXXXX	18
2	PRATIK	BIHAR	XXXXXXXXXX	19
3	RIYANKA	SILIGURI	XXXXXXXXXX	20
4	DEEP	RAMNAGAR	XXXXXXXXXX	18
5	SAPTARHI	KOLKATA	XXXXXXXXXX	19
6	DHANRAJ	BARABAJAR	XXXXXXXXXX	20
7	ROHIT	BALURGHAT	XXXXXXXXXX	18
8	NIRAJ	ALIPUR	XXXXXXXXXX	19

StudentCourse

COURSE_ID	ROLL_NO
1	1
2	2
2	3
3	4
1	5
4	9
5	10
4	11

The simplest Join is INNER JOIN.

A. INNER JOIN
The INNER JOIN keyword selects all rows from both the tables as long as the condition is satisfied. This keyword will create the result-set by combining all rows from both the tables where the condition satisfies i.e value of the common field will be the same.
Example Queries(INNER JOIN)
This query will show the names and age of students enrolled in different courses.

SELECT StudentCourse.COURSE_ID, Student.NAME, Student.AGE FROM
Student
INNER JOIN StudentCourse
ON Student.ROLL_NO = StudentCourse.ROLL_NO;

Output:

COURSE_ID	NAME	Age
1	HARSH	18
2	PRATIK	19
2	RIYANKA	20
3	DEEP	18
1	SAPTARHI	19

Example Queries(LEFT JOIN):
SELECT Student.NAME,StudentCourse.COURSE_ID
FROM Student
LEFT JOIN StudentCourse
ON StudentCourse.ROLL_NO = Student.ROLL_NO;

Output:

NAME	COURSE_ID
HARSH	1
PRATIK	2
RIYANKA	2
DEEP	3
SAPTARHI	1
DHANRAJ	*NULL*
ROHIT	*NULL*
NIRAJ	*NULL*

C. RIGHT JOIN
RIGHT JOIN is similar to LEFT JOIN. This join returns all the rows of the table on the right side of the join and matching rows for the table on the left side of the join.

For the rows for which there is no matching row on the left side, the result-set will contain *null*. RIGHT JOIN is also known as RIGHT OUTER JOIN.

Syntax:

```
SELECT table1.column1,table1.column2,table2.column1,....
FROM table1
RIGHT JOIN table2
ON table1.matching_column = table2.matching_column;
```

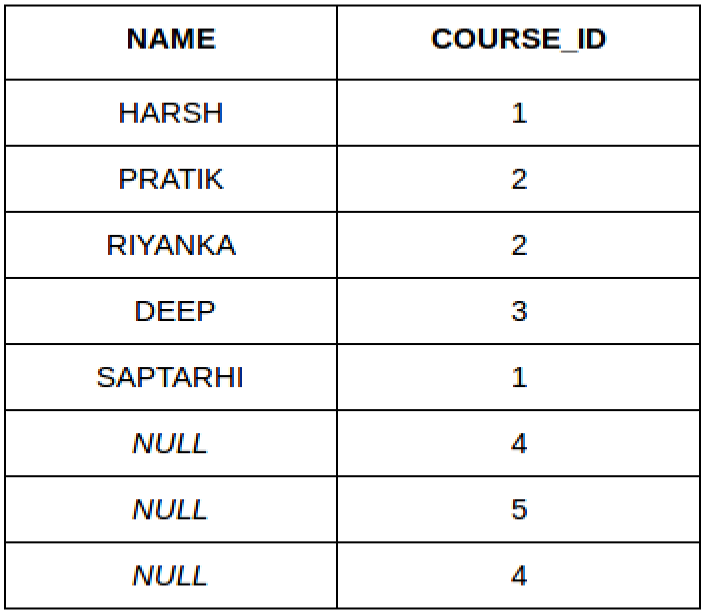
table1: First table.
table2: Second table
matching_column: Column common to both the tables.

Note*: We can also use RIGHT OUTER JOIN instead of RIGHT JOIN, both are the same.*

Example Queries(RIGHT JOIN):

```
SELECT Student.NAME,StudentCourse.COURSE_ID
FROM Student
RIGHT JOIN StudentCourse
ON StudentCourse.ROLL_NO = Student.ROLL_NO;
```

Output:



NAME	COURSE_ID
HARSH	1
PRATIK	2
RIYANKA	2
DEEP	3
SAPTARHI	1
NULL	4
NULL	5
NULL	4

D. FULL JOIN

FULL JOIN creates the result-set by combining results of both LEFT JOIN and RIGHT JOIN. The result-set will contain all the rows from both tables. For the rows for which there is no matching, the result-set will contain *NULL* values.

Syntax:

```
SELECT table1.column1,table1.column2,table2.column1,....
FROM table1
FULL JOIN table2
ON table1.matching_column = table2.matching_column;
```

table1: First table.
table2: Second table
matching_column: Column common to both the tables.

Example Queries(FULL JOIN):

```
SELECT Student.NAME,StudentCourse.COURSE_ID
FROM Student
FULL JOIN StudentCourse
ON StudentCourse.ROLL_NO = Student.ROLL_NO;
```

Output:

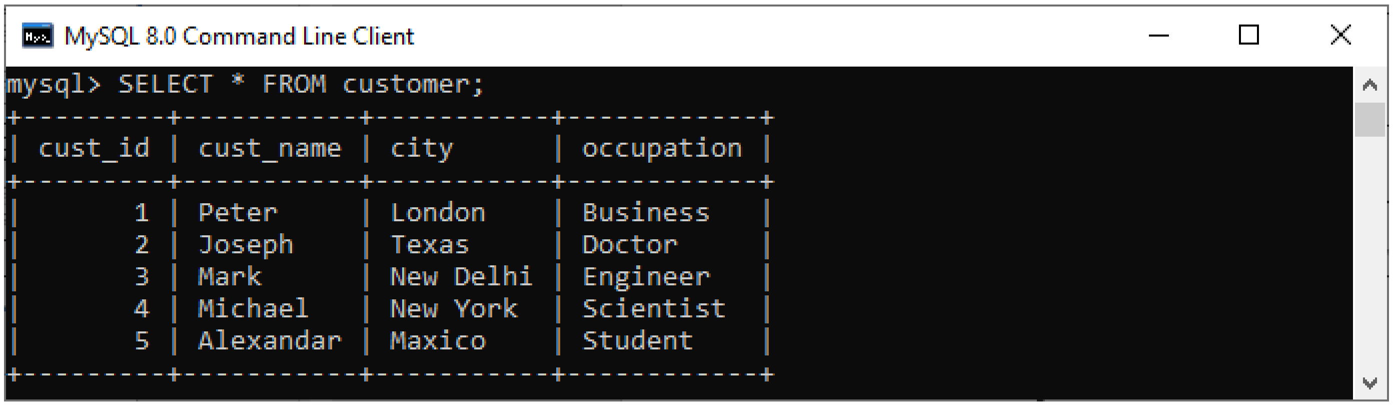
AME	**OURSE_ID**
IARSH	
RATIK	
IYANKA	
EEP	
APTARHI	
HANRAJ	ULL
OHIT	ULL

AME	COURSE_ID
IIRAJ	IULL
IULL	
IULL	
IULL	

IN vs. EXISTS

IN Operator

The IN operator is used to **retrieves results when the specified value matches any value in a set of values or is returned by a subquery**.

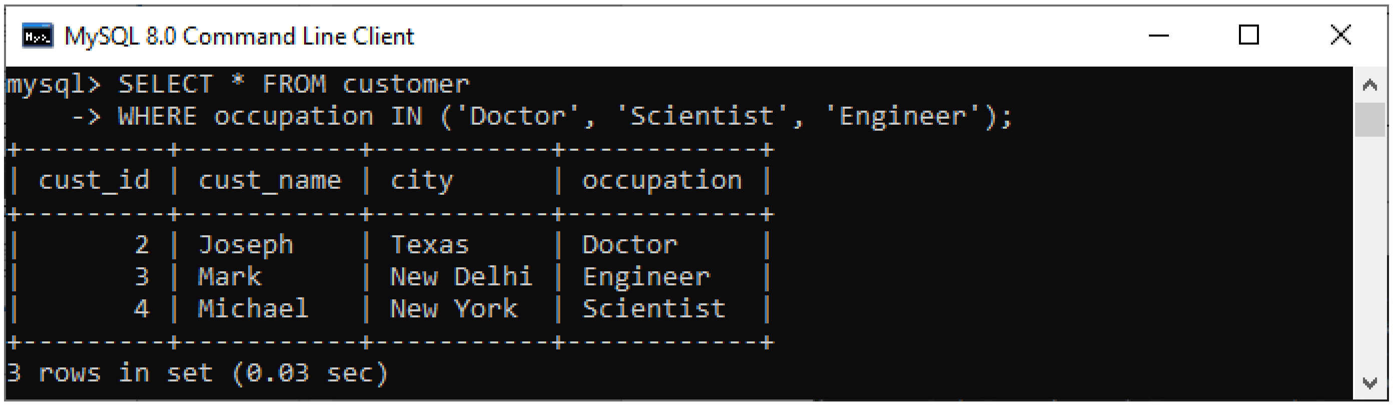

MySQL 8.0 Command Line Client

```
mysql> SELECT * FROM customer;
+---------+-----------+-----------+------------+
| cust_id | cust_name | city      | occupation |
+---------+-----------+-----------+------------+
|       1 | Peter     | London    | Business   |
|       2 | Joseph    | Texas     | Doctor     |
|       3 | Mark      | New Delhi | Engineer   |
|       4 | Michael   | New York  | Scientist  |
|       5 | Alexandar | Maxico    | Student    |
+---------+-----------+-----------+------------+
```

If we want **to get all customer details whose occupation is either doctor, engineer, or scientist**, then we can use the statement as follows:

1. mysql> **SELECT** * **FROM** customer
2. **WHERE** occupation IN ('Doctor', 'Scientist', 'Engineer');

Here is the output:

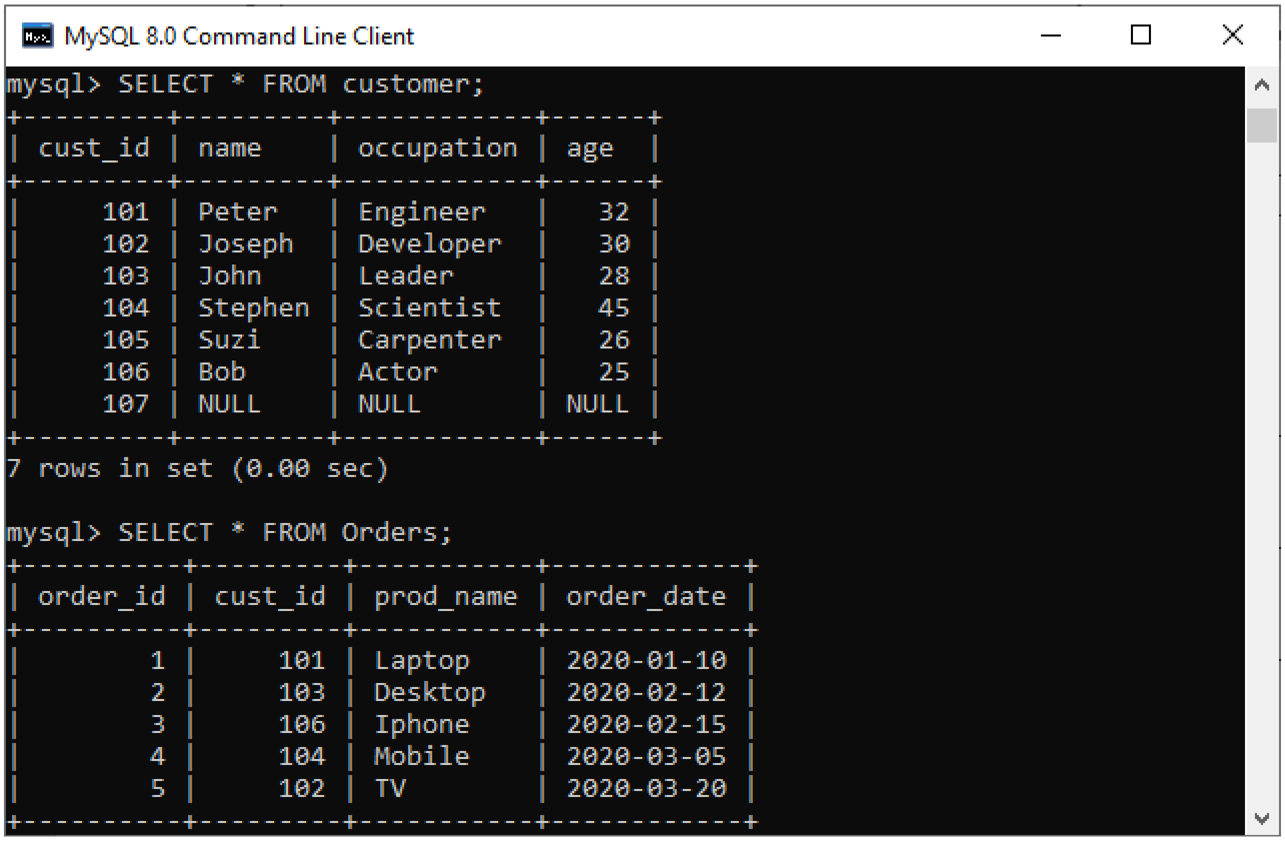


MySQL 8.0 Command Line Client

```
mysql> SELECT * FROM customer
    -> WHERE occupation IN ('Doctor', 'Scientist', 'Engineer');
+---------+-----------+-----------+------------+
| cust_id | cust_name | city      | occupation |
+---------+-----------+-----------+------------+
|       2 | Joseph    | Texas     | Doctor     |
|       3 | Mark      | New Delhi | Engineer   |
|       4 | Michael   | New York  | Scientist  |
+---------+-----------+-----------+------------+
3 rows in set (0.03 sec)
```

EXISTS Operator

EXISTS is a Boolean operator which **checks the subquery result and returns an either TRUE or FALSE value**. It is used in combination with subquery and checks whether a row is returned through this subquery or not. This operator returns **TRUE** if the subquery returns single or multiple records. Otherwise, it gives a **FALSE** result when no records are returned.



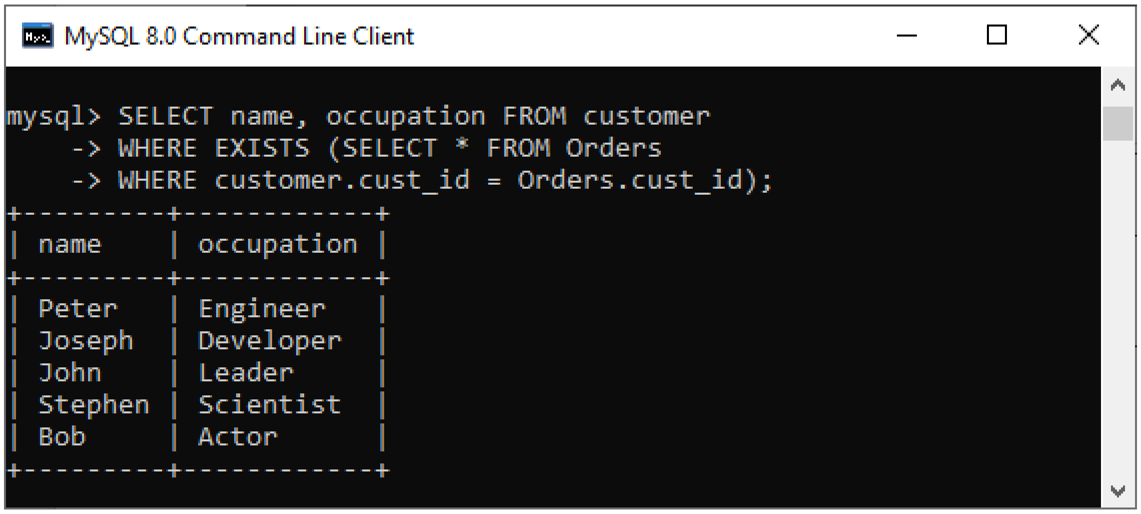
MySQL 8.0 Command Line Client

```
mysql> SELECT * FROM customer;
+---------+---------+------------+------+
| cust_id | name    | occupation | age  |
+---------+---------+------------+------+
|     101 | Peter   | Engineer   |   32 |
|     102 | Joseph  | Developer  |   30 |
|     103 | John    | Leader     |   28 |
|     104 | Stephen | Scientist  |   45 |
|     105 | Suzi    | Carpenter  |   26 |
|     106 | Bob     | Actor      |   25 |
|     107 | NULL    | NULL       | NULL |
+---------+---------+------------+------+
7 rows in set (0.00 sec)

mysql> SELECT * FROM Orders;
+----------+---------+-----------+------------+
| order_id | cust_id | prod_name | order_date |
+----------+---------+-----------+------------+
|        1 |     101 | Laptop    | 2020-01-10 |
|        2 |     103 | Desktop   | 2020-02-12 |
|        3 |     106 | Iphone    | 2020-02-15 |
|        4 |     104 | Mobile    | 2020-03-05 |
|        5 |     102 | TV        | 2020-03-20 |
+----------+---------+-----------+------------+
```

If we want **to get all customer names and occupation who has placed at least one order**, then we can use the statement as follows:

1. mysql> **SELECT name**, occupation **FROM** customer
2. **WHERE** EXISTS (**SELECT** * **FROM** Orders
3. **WHERE** customer.cust_id = Orders.cust_id);

Here is the output:



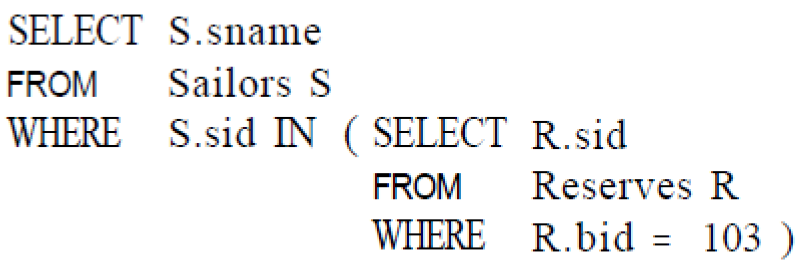
```
mysql> SELECT name, occupation FROM customer
    -> WHERE EXISTS (SELECT * FROM Orders
    -> WHERE customer.cust_id = Orders.cust_id);
+---------+------------+
| name    | occupation |
+---------+------------+
| Peter   | Engineer   |
| Joseph  | Developer  |
| John    | Leader     |
| Stephen | Scientist  |
| Bob     | Actor      |
+---------+------------+
```

EXPERIMENT 12

NESTED QUERIES

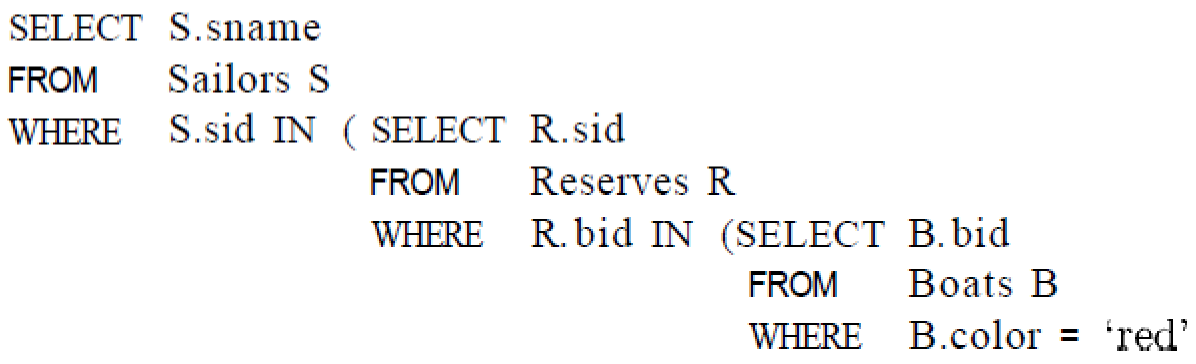
A nested query is a query that has another query embedded within it. The embedded query is called a subquery. Outer and inner queries are completely independent of each other

Q: Find the names of sailors who have reserved boat 103.



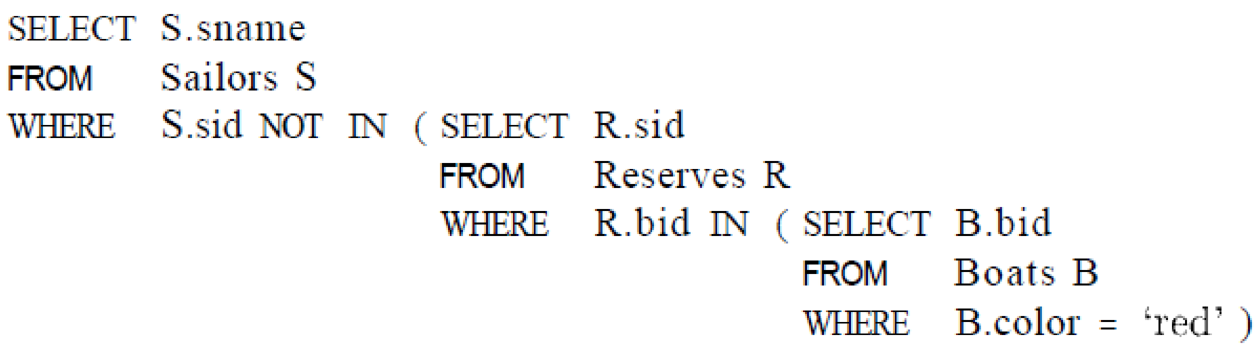
```
SELECT S.sname
FROM   Sailors S
WHERE  S.sid IN ( SELECT R.sid
                  FROM   Reserves R
                  WHERE  R.bid =  103 )
```

Q: Find the names of sailors who have reserved a red boat.



```
SELECT S.sname
FROM   Sailors S
WHERE  S.sid IN ( SELECT R.sid
                  FROM   Reserves R
                  WHERE  R.bid IN (SELECT B.bid
                                   FROM   Boats B
                                   WHERE  B.color = 'red'
```

Q: Find the names of sailors who have not reserved a red boat.



```
SELECT S.sname
FROM   Sailors S
WHERE  S.sid NOT  IN ( SELECT R.sid
                       FROM   Reserves R
                       WHERE  R.bid IN ( SELECT B.bid
                                         FROM   Boats B
                                         WHERE  B.color = 'red' )
```

Correlated Nested Queries

Outer and inner queries are dependent on each other

Q: Find the names of sailors who have reserved boat number 103.

```
SELECT  S.sname
FROM    Sailors S
WHERE   EXISTS  ( SELECT  *
                  FROM    Reserves R
                  WHERE   R.bid =  103
                          AND  R.sid =  S.sid )
```

EXPERIMENT 13

Write a PL/SQL Program using FOR loop to insert ten rows into a database table.

```
-- Create a table to store the data
CREATE TABLE example_table (
   id NUMBER,
   name VARCHAR2(50)
);

-- PL/SQL program to insert ten rows into the table
DECLARE
   v_counter NUMBER := 1;
BEGIN
   FOR i IN 1..10 LOOP
      INSERT INTO example_table (id, name) VALUES (v_counter, 'Name ' ||
v_counter);
      v_counter := v_counter + 1;
   END LOOP;
   COMMIT; -- Committing the transaction
   DBMS_OUTPUT.PUT_LINE('Ten rows inserted successfully.');
END;
/
```

Creates a table called example_table with columns id and name.
Declares a PL/SQL block that initializes a counter variable v_counter to 1.
Executes a FOR loop from 1 to 10, where each iteration inserts a row into the example_table with the id incremented by 1 and the name set to 'Name 1', 'Name 2', and so on.
Commits the transaction to save the changes to the database.
Prints a message confirming the successful insertion of ten rows.

Output:

```
Ten rows inserted successfully.
```

EXPERIMENT 14

Given the table EMPLOYEE (Emp No, Name, Salary, Designation, Dept Id) write

cursor to select the five highest paid employees from the table.

Declare Variables: Declare variables to hold employee information and loop counter.

sql:

DECLARE @EmpNo INT, @Name VARCHAR(50), @Salary DECIMAL(10,2), @Designation VARCHAR(50), @DeptId INT, @Rank INT

Declare Cursor: Declare a cursor to fetch employee records ordered by salary in descending order.

DECLARE employee_cursor CURSOR FOR SELECT EmpNo, Name, Salary, Designation, DeptId, ROW_NUMBER() OVER (ORDER BY Salary DESC) AS Rank FROM EMPLOYEE

Open Cursor: Open the cursor to start fetching records.

sql:

OPEN employee_cursor

Fetch Records: Fetch records one by one until the desired number of records are fetched or until there are no more records.

sql:

FETCH NEXT FROM employee_cursor INTO @EmpNo, @Name, @Salary, @Designation, @DeptId, @Rank

Process Records: Check if the fetched record's rank is within the top five. If it is, print the employee information.

sql:

WHILE @@FETCH_STATUS = 0 AND @Rank <= 5 BEGIN PRINT 'EmpNo: ' + CAST(@EmpNo AS VARCHAR(10)) + ', Name: ' + @Name + ', Salary: ' + CAST(@Salary AS VARCHAR(10)) + ', Designation: ' + @Designation + ', DeptId: ' + CAST(@DeptId AS VARCHAR(10)) FETCH NEXT FROM employee_cursor

INTO @EmpNo, @Name, @Salary, @Designation, @DeptId, @Rank END

Close Cursor: Close the cursor after fetching all required records.

sql:

CLOSE employee_cursor DEALLOCATE employee_cursor

Output of the program:

```
EmpNo: 101, Name: John Doe, Salary: 10000.00, Designation: Manager, DeptId: 1
EmpNo: 102, Name: Jane Smith, Salary: 9500.00, Designation: Engineer, DeptId: 2
EmpNo: 103, Name: Mike Johnson, Salary: 9000.00, Designation: Analyst, DeptId: 1
EmpNo: 104, Name: Emily Davis, Salary: 8500.00, Designation: Developer, DeptId: 3
EmpNo: 105, Name: Chris Wilson, Salary: 8000.00, Designation: Designer, DeptId: 2
```

EXPERIMENT 15

a) Given an integer, Write a PL/SQL procedure to insert the tuple(1,'xx') into a given relation.

```
CREATE OR REPLACE PROCEDURE InsertTuple(
   p_table_name    IN VARCHAR2,
   p_integer_value IN NUMBER
)
IS
BEGIN
   EXECUTE IMMEDIATE 'INSERT INTO ' || p_table_name || ' VALUES (' ||
p_integer_value || ', ''xx'')';
   COMMIT;
   DBMS_OUTPUT.PUT_LINE('Tuple inserted successfully.');
EXCEPTION
   WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE('Error: ' || SQLERRM);
END;
/
```

This procedure InsertTuple accepts two parameters:

p_table_name: The name of the table where the tuple will be inserted.
p_integer_value: The integer value to be inserted into the table along with the string value 'xx'.
Here's we can call this procedure:

```
sql
BEGIN
   InsertTuple('your_table_name', 1);
END;
/
```

Output

```
Tuple inserted successfully.
```

EXPERIMENT 16
Queries for Creating Triggers

Trigger: A trigger is a stored procedure(function) in a database which is automatically invoked(called or activated) and executed whenever a special event in the database occurs.

For example, a trigger can be invoked when a row is inserted into a specified table or when certain table columns are being updated.

A trigger is specified by the DBA(Database Administrator)

A **Database Administrator (DBA)** is a person who is responsible for controlling, maintaining, coordinating, and operating the database server. Managing, securing, allowing users to access the DBMS server, and taking care of database system are the prime responsibilities of a DBA.

A database that has a set of associated triggers is called an **active database.**

Syntax to create a trigger

```
create   trigger   [trigger_name]
[before | after]
{insert | update | delete}
on   [table_name]
[for each row]
[trigger_body]
```

Explanation of syntax:

create trigger [trigger_name]:	Creates a trigger with the trigger_name.
[before | after]:	This specifies when the trigger will be executed.
{insert | update | delete}:	This specifies the DML operation.
on [table_name]:	This specifies the name of the table associated with the trigger.
[for each row]:	The trigger will be executed for each row being affected.
[trigger_body]:	This provides the operation to be performed when a trigger is activated

Example:

Given Student marks Database, in which student marks are recorded.

In such schema, create a trigger so that the total and percentage of specified marks are automatically calculated and inserted whenever a student row is inserted.

```
Mysql>  Create table student (rollno      integer NOT NULL,
                              name     varchar(20),
                              sub1     integer,
                              sub2     integer,
                              sub3     integer,
                              total    integer,
                              percentage       real,
                              PRIMARY KEY (rollno) );
```

The database schema

```
Mysql>  desc student;
```

Field	Type	Null	Key	Default	extra
Rollno	Integer	NO	PRI	NULL	
Name	Varchar(20)	YES		NULL	
Sub1	Integer	YES		NULL	
Sub2	Integer	YES		NULL	
Sub3	Integer	YES		NULL	
Total	Integer	YES		NULL	
percentage	real	YES		NULL	

```
Mys
            on   student
          for   each   row
      set    student.total = student.subj1 + student.subj2 + student.subj3,
                         student.percentage = student.total/300*100;
```

Above SQL statement will create a trigger in the student table in which whenever subjects marks are entered,
before inserting the marks into the database, the trigger will compute the total marks & percentage and insert with the marks.

```
mysql> insert into student(rollno,name,sub1,sub2,sub3,total,percentage)
              values(501, 'Dennis', 90, 90, 80, 0, 0);

mysql> insert into student(rollno,name,sub1,sub2,sub3,total,percentage)
              values(502, 'Ritche', 100, 95, 90, 0, 0);

 mysql> select   *   from   student;
```

Rollno	Name	Sub1	Sub2	Sub3	Total	percentage
501	Dennis	90	90	80	260	86.66
502	Ritche	100	95	90	285	95.00

In this way triggers can be created and executed in a database.