

**KESHAV MEMORIAL INSTITUTE OF TECHNOLOGY
(KMIT)**

(Approved by AICTE, New Delhi and Affiliated to JNTUH)

Narayanaguda, Hyderabad – 500029

(Autonomous)



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

LAB MANUAL

DATABASE MANAGEMENT SYSTEMS LAB

B.Tech II YEAR II SEM (KR20 Regulations)

ACADEMIC YEAR 2021-22

INDEX

S.NO	TOPIC	PAGE NO
I	List of Experiments	ii
II	V/M /POs/PSOs/PEOs	iv
III	Syllabus	ix
IV	Course Objectives & Course Outcomes	xi

List of Experiments

CONTENT			
SL.NO	WEEK	TOPIC	PAGE NO
1	-	Syllabus	
2	-	Course Objectives, Outcomes	
3	Experiment: 1	<p><i>E-R Model</i></p> <p>Analyze and come up with the entities in it. Identify what data has to be persisted in the database. This contains the entities, attributes etc.</p> <p>Identify the primary keys for all the entities. Identify the other keys like Foreign Key and constraints like NULL, NOT NULL, CHECK etc.</p> <p>Example to create for products, customers, suppliers, orders, , employees, order details, categories, among others.</p>	
4	Experiment : 2	<p><i>Installation & DDL</i></p> <p>Installation of Mysql and practicing DDL commands. Creating databases, How to create tables, altering the database or tables, dropping tables and databases if not required. You will also try truncate, rename commands etc.</p> <p>Data Definition Language (DDL) : create , alter, drop.</p>	
5	Experiment : 3	<p><i>DML</i></p> <p>Data Manipulation Language Commands (DML) commands are used to for managing data within schema objects.</p> <p>Exercising the commands using DML : insert, delete, update on the following tables : products, customers, suppliers, orders, , employees, order details, categories.</p> <ul style="list-style-type: none"> ● INSERT – insert data into a table. ● UPDATE – updates existing data within a table. ● DELETE – deletes single or all records from a table. 	

6	Experiment : 4	Querying : Practice queries on Aggregate functions like count, max , min ,avg ,sum Practice queries like nested queries/co-related queries using ANY, ALL, IN,Exists, NOT EXISTS, UNION, INTERSECT, groupby ,having etc. Joins: Join , Left Outer Join, Right Outer Join, Self Join	
7	Experiment: 5	Querying(continued...) Some example to practice the queries: <ul style="list-style-type: none"> • Display all the order details of given a customer. • Display all the products. • Get the highest sold product from given supplier ID • List all products grouped by category • List the products, whose products unit price is greater then all the products of average. • List Details of order and customer of each order • List the products which were sold in year 1997 • Display the total amount for each order • Display Order Details for given an order ID Order Details: product name and unit price for given order ID Exercising Simple to complex Queries using joins, nested and co-related queries.	
8	Experiment : 6	Stored Procedures : Create a stored procedure, Alter and Drop a procedure, IN, OUT, IN & OUT parameters <ul style="list-style-type: none"> • Create a Procedure to display order details of given customer ID like ordered, order Date , RequiredDate, Shipped Date • Create a procedure to accept a customer ID and display the customer order history(product name and how much quantity ordered for that particular product) Ex: product name, Total quantity he/she ordered. • Create a procedure to display Ten Most Expensive ProductsColumns should be displayed Product name & Unit price 	

9	Experiment : 7	Views <ul style="list-style-type: none"> • Create a view to display the current product list which are available(not discontinued) • Create a view to display the products by category Display product name, quantity Per Unit, units In Stock, Discontinued • Create a view as “Invoices” to display all the information from order, customer, shipper for each “Order Details” 	
10	Experiment : 8	Triggers Demonstrate Create Trigger, Alter Trigger, Drop Trigger, Row Level , Table Level triggers, Before Insert ,AfterInsert, Before Update, After Update, Before Delete, After Delete	
11	Experiment : 9	Demonstrate the role of DBA using DCL commands	



KESHAV MEMORIAL INSTITUTE OF TECHNOLOGY

(Approved by AICTE & Govt of T.S and Affiliated to JNTUH)

3-5-1026, Narayanaguda, Hyderabad-29. Ph: 040-23261407

Department Of Computer Science & Engineering

Vision of the Institution:

To be the fountain head of latest technologies, producing highly skilled, globally competent engineers.

Mission of the Institution:

To provide a learning environment that inculcates problem solving skills, professional, ethical responsibilities, lifelong learning through multi modal platforms and prepare students to become successful professionals.

To establish Industry Institute Interaction to make students ready for the industry.

To provide exposure to students on latest hardware and software tools.

To promote research based projects/activities in the emerging areas of technology convergence.

To encourage and enable students to not merely seek jobs from the industry but also to create new enterprises

To induce a spirit of nationalism which will enable the student to develop, understand India's challenges and to encourage them to develop effective solutions.

To support the faculty to accelerate their learning curve to deliver excellent service to students



KESHAV MEMORIAL INSTITUTE OF TECHNOLOGY

(Approved by AICTE & Govt of T.S and Affiliated to JNTUH)

3-5-1026, Narayanaguda, Hyderabad-29. Ph: 040-23261407

Department Of Computer Science & Engineering

Vision & Mission of Department

Vision of the Department:

To be among the region's premier teaching and research Computer Science and Engineering departments producing globally competent and socially responsible graduates in the most conducive academic environment.

Mission of the Department:

To provide faculty with state of the art facilities for continuous professional development and research, both in foundational aspects and of relevance to emerging computing trends.

To impart skills that transform students to develop technical solutions for societal needs and inculcate entrepreneurial talents.

To inculcate an ability in students to pursue the advancement of knowledge in various specializations of Computer Science and Engineering and make them industry-ready.

To engage in collaborative research with academia and industry and generate adequate resources for research activities for seamless transfer of knowledge resulting in sponsored projects and consultancy.

To cultivate responsibility through sharing of knowledge and innovative computing solutions that benefit the society-at-large.

To collaborate with academia, industry and community to set high standards in academic excellence and in fulfilling societal responsibilities.



KESHAV MEMORIAL INSTITUTE OF TECHNOLOGY

(Approved by AICTE & Govt of T.S and Affiliated to JNTUH)

3-5-1026, Narayanaguda, Hyderabad-29. Ph: 040-23261407

PROGRAM OUTCOMES (POs)

1. **Engineering Knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
2. **Problem Analysis:** Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3. **Design/Development of Solutions:** Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
4. **Conduct Investigations of Complex Problems:** Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
5. **Modern Tool Usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modelling to complex engineering activities with an understanding of the limitations.
6. **The Engineer and Society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
7. **Environment and Sustainability:** Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
8. **Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9. **Individual and Team Work:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10. **Communication:** Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
11. **Project Management and Finance:** Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
12. **Life-long Learning:** Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.



KESHAV MEMORIAL INSTITUTE OF TECHNOLOGY

(Approved by AICTE & Govt of T.S and Affiliated to JNTUH)

3-5-1026, Narayanaguda, Hyderabad-29. Ph: 040-23261407

Department of Computer Science & Engineering

PROGRAM SPECIFIC OUTCOMES (PSOs)

PSO1: An ability to analyze the common business functions to design and develop appropriate Computer Science solutions for social upliftments.

PSO2: Shall have expertise on the evolving technologies like Python, Machine Learning, Deep Learning, Internet of Things (IOT), Data Science, Full stack development, Social Networks, Cyber Security, Big Data, Mobile Apps, CRM, ERP etc.



KESHAV MEMORIAL INSTITUTE OF TECHNOLOGY

(Approved by AICTE & Govt of T.S and Affiliated to JNTUH)

3-5-1026, Narayanaguda, Hyderabad-29. Ph: 040-23261407

Department of Computer Science & Engineering

PROGRAM EDUCATIONAL OBJECTIVES (PEOs)

PEO1: Graduates will have successful careers in computer related engineering fields or will be able to successfully pursue advanced higher education degrees.

PEO2: Graduates will try and provide solutions to challenging problems in their profession by applying computer engineering principles.

PEO3: Graduates will engage in life-long learning and professional development by rapidly adapting changing work environment.

PEO4: Graduates will communicate effectively, work collaboratively and exhibit high levels of professionalism and ethical responsibility.

SYLLABUS
DATABASE MANAGEMENT SYSTEMS LAB
B.TECH II Year II Sem

L T P C
0 0 3 1.5

List of Programs:

Scenario 1:

Product-Sales database : SouthWind

Southwind database is a sample database used by Organization. The database contains the sales data for SouthWind Traders, it is foods export-import company. Using this schema to demonstrate the how customer can choose and order products, how orders are placed and how those products get delivered to the customer.

Products: This Entity will have all the products details where suppliers will supply products based on customers demand.

Supplies: This Entity will supply the products demanded by the customers.

Shippers: This Entity will take the orders from suppliers and deliver to customers.

Employees : Employees will monitor the orders placed by customers.

Invoices: This Entity will take care of billing process based on customer order. Etc..identify some more entities and find out relationship between them.

A product-sales the above process involves many steps like

1. Analyzing the problem and identifying the Entities and Relationships,
2. E-R Model
3. Relational Model
4. Normalization
5. Creating the database
6. Querying. Students are supposed to work on these steps week wise and finally create a complete “Database System” to Product-Sales.

Experiment 1: E-R Model

Analyze and come up with the entities in it. Identify what data has to be persisted in the database. This contains the entities, attributes etc. Identify the primary keys for all the entities. Identify the other keys like Foreign Key and constraints like NULL, NOT NULL, CHECK etc. Example to create for **products, customers, suppliers, orders, , employees, order details, categories**, among others.

Students should submit E-R diagrams using the above tables.

Experiment 2: Installation & DDL

Installation of Mysql and practicing DDL commands.

Creating databases, How to create tables, altering the database or tables, dropping tables and databases if not required. You will also try truncate, rename commands etc.

Data Definition Language (DDL) : create , alter, drop.

Experiment 3: DML

Data Manipulation Language Commands (DML) commands are used to for managing data within schema objects.

Exercising the commands using **DML** : insert, delete, update on the following tables : products, customers, suppliers, orders, , employees, order details, categories.

- INSERT – insert data into a table.
- UPDATE – updates existing data within a table.
- DELETE – deletes single or all records from a table.

Data Query Language – Select

Populate all the tables designed in experiment : 2 with appropriate data.

Experiment 4: Querying

Practice queries on **Aggregate functions** like count, max , min ,avg ,sum

Practice queries like nested queries/co-related queries using ANY, ALL, IN, Exists, NOT EXISTS, UNION, INTERSECT, groupby ,having etc.

Joins: Join , Left Outer Join, Right Outer Join, Self Join

Experiment 5 : Querying(continued...)

Some example to practice the queries:

- Display all the order details of given a customer.
- Display all the products.
- Get the highest sold product from given supplier ID
- List all products grouped by category
- List the products, whose products unit price is greater then all the products of average.
- List Details of order and customer of each order
- List the products which were sold in year 1997

Display the total amount for each order

☐ Display Order Details for given an order ID

Order Details: product name and unit price for given order ID

Exercising Simple to complex Queries using joins, nested and co-related queries.

Experiment 6 : Stored Procedures :

Create a stored procedure, Alter and Drop a procedure, IN, OUT, IN & OUT parameters

☐ Create a Procedure to display order details of given customer ID like ordered, order Date , Required Date, Shipped Date

☐ Create a procedure to accept a customer ID and display the customer order history(product name and how much quantity ordered for that particular product)

Ex: product name, Total quantity he/she ordered.

☐ Create a procedure to display Ten Most Expensive Products

Columns should be displayed Product name & Unit price

Experiment 7: Views

☐ Create a view to display the current product list which are available(not discontinued)

☐ Create a view to display the products by category

Display product name, quantity Per Unit, units In Stock, Discontinued

☐ Create a view as “Invoices” to display all the information from order, customer, shipper for each “Order Details”

Experiment 8: Triggers

Demonstrate Create Trigger, Alter Trigger, Drop Trigger, Row Level , Table Level triggers, Before Insert ,After Insert, Before Update, After Update, Before Delete, After Delete

Experiment 9 :

Demonstrate the role of DBA using DCL commands

TEXT BOOKS:

1. Database Management Systems, Raghurama Krishnan, Johannes Gehrke, Tata McGraw Hill, 3rd Edition

2. Database System Concepts, Silberschatz, Korth, McGraw Hill, V edition.

REFERENCES BOOKS:

1. Introduction to SQL, Rick F. Vander Lans, Pearson education.

2. Oracle PL/SQL, B. Rosenzweig and E. Silvestrova, Pearson education.

3. SQL & PL/SQL for Oracle 10 g, Black Book, Dr. P. S. Deshpande, Dream Tech.

4. Oracle Database 11 g PL/SQL Programming, M. Mc Laughlin, TMH

Course Objectives:

1. Introduce ER data model, database design and normalization
2. Learn SQL basics for data definition and data manipulation

Course Outcomes:

After learning the contents of this course the student is able to

CS407.CO1: Design database schema for a given application and apply normalization

CS407.CO2: Gather skills in using SQL commands for data definition and data manipulation.

CS407.CO3: Develop solutions for database applications using procedures, cursors and triggers

CO VS PO MAPPING:

	CO	PO1	PO2	PO3	PO4	PO 5	PO 6	PO 7	PO 8	PO 9	PO 10	PO 11	PO12	PSO 1	PSO 2
DBMS Lab	CO1	M	H	M										M	S
	CO2	S	M	H										S	M
	CO3		S	H										M	H

CO VS PSO MAPPING:

	PSO-1	PSO-2
CO1	1	3
CO2	2	3
CO3	1	2

Experiment 1

E-R Model

Analyze and come up with the entities in it. Identify what data has to be persisted in the database. This contains the entities, attributes etc.

Identify the primary keys for all the entities. Identify the other keys like Foreign Key and constraints like NULL, NOT NULL, CHECK etc.

Example to create for **products, customers, suppliers, orders, , employees, order details, categories**, among others.

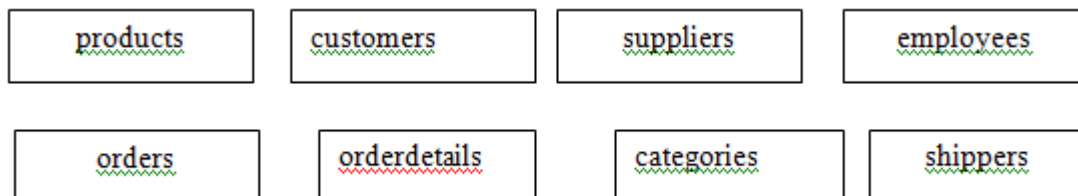
Students should submit E-R diagrams using the above tables.

MODEL INPUT

DEFINITIONS

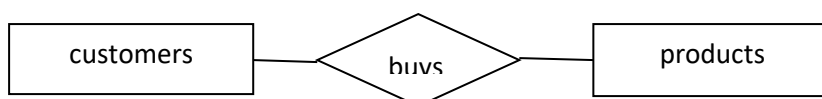
Entity: the object in the ER Model represents is an entity which is thing in the real world with an independent existence.

Eg:



ER-MODEL

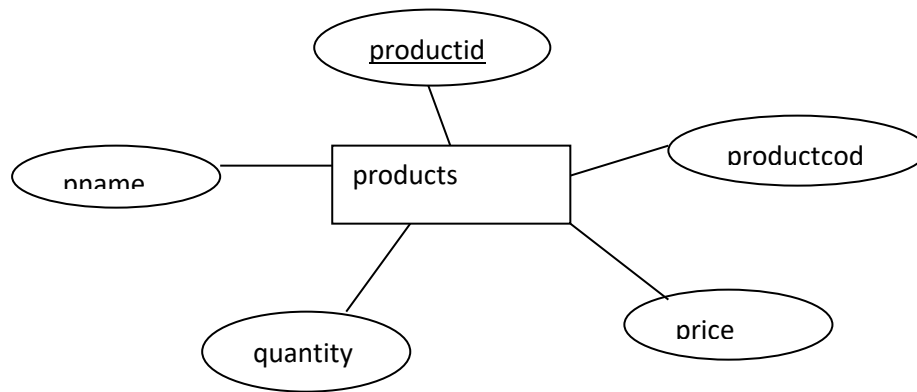
Describes data as entities, relationships and attributes .The ER-Model is important preliminary for its role in database design. ER Model is usually shown pictorially using entity relationship diagrams.



ATTRIBUTES

The properties that characterize an entity set are called its attributes. An attribute is referred to by the terms data items, data element, data field item.

Ex: attributes for products entity.



Candidate key:

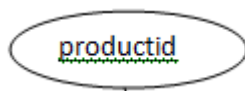
It can be defined as minimal super key or irreducible super key. In other words an attribute or combination of attributes that identifies the record uniquely but none of its proper subsets can identify the record uniquely.

<u>productid</u>	<u>Productcode</u>	price	quantity	pname
------------------	--------------------	-------	----------	-------

productid,productcode----->candidate key

Primary key:

A candidate key that is used by the database designer for unique identification of each row in a table is known as primary key. A primary key consists of one or more attributes of the table.



Partial key:

A weak entity type normally has a partial key which is the set of attributes that can uniquely identify weak entity that are related to the same owner entity.

EXPECTED OUTPUT

The entities in the “product sales” is

- 1) Products 2)customers 3)suppliers 4)orders 5) employees 6)order details 7)categories 8)shippers

Products entity:

Attributes for the products entity are

Productid, productName,SupplierID,CustomerID, quantityperunit,unitPrice,ReorderLevel and discontinued.

products schema:

<u>productid</u>	<u>SupplierID</u>	CustomerID	productName	Quantityperunit	unitPrice	UnitsIn Stock	UnitsOn Order	Reorder Level	Discontinued
------------------	-------------------	------------	-------------	-----------------	-----------	---------------	---------------	---------------	--------------

Productid, productcode----->candidate key

Productid -----> primary key

Suppliers entity:

Attributes for the supplier entity are

Supplierid ,CompanyName,ContactName,ContactJobTitle,Address,phone

Suppliers schema:

<u>Supplierid</u>	CompanyName	ContactName	ContactJobTitle	Address	phone
-------------------	-------------	-------------	-----------------	---------	-------

Supplierid,productid----- >Super key

productid-----> candidate key

supplierid----- >primary key

Shippers entity:

Attributes for the shippers entity are

Shipperid, CompanyName,phone

Shippers schema:

Shipperid	CompanyName	Phone
------------------	-------------	-------

Supplierid,productid----- >Super key

productid-----> candidate key

supplierid----- >primary key

Employees entity:

Attributes for the employees entity are

EmployeeId,LastName,FirstName,JobTitle,BirthDate,HireDate,Address,City,PostalCode,Country,Phone,ReportsTo,Salary

Employees schema:

EmployeeId	LastName	FirstName	JobTitle	BirthDate	HireDate	Address	City	PostalCode	Country	Phone	ReportsTo	Salary
-------------------	----------	-----------	----------	-----------	----------	---------	------	------------	---------	-------	-----------	--------

empno,name----- super key

empno ----- candidate key

empno -----primary key

Customers Entity:

Attributes for the Customers entity are

customerId,CompanyName,ContactName,ContactTitle,Address,City,PostalCode,Country,Phone,fax

Customers schema:

custo merId	Company Name	Contact Name	Contact Title	Addr ess	Ci ty	Postal Code	Cou ntry	Pho ne	f a x	custo merId
----------------	-----------------	-----------------	------------------	-------------	----------	----------------	-------------	-----------	-------------	----------------

Orders Entity:

Attributes for the Orders entity are

OrderID,CustomerID,EmployeeID,OrderDate,RequiredDate,ShippedDate,ShipID,ShipName,Ship Address

Orders Schema:

Order ID	Custo merID	EmployeeID	OrderDate	Required Date	ShippedDate	ShipID	ShipName	ShipAddress
-------------	----------------	------------	-----------	------------------	-------------	--------	----------	-------------

Categories Entity:

Attributes for the Categories entity are

CategoryID,CategoryName

Categories Schema:

<u>CategoryID</u>	CategoryName
-------------------	--------------

OrderDetails Entity:

Attributes for the OrderDetails entity are

OrderID,ProductID,UnitPrice,Quantity

OrderDetails Schema:

<u>OrderID</u>	ProductID	UnitPrice	Quantity
----------------	-----------	-----------	----------

Sample data for Customers entity:

CustomerID	CompanyName	ContactName	ContactTitle	Address	City
Region	PostalCode	Country	Phone	Fax	
+-----+-----+-----+-----+-----+-----+					
1001	tcs	maria	sales representative	Obere Str. 57	Berlin
NULL	12209	Germany	030-0074321	030-0076545	
1002	wipro	Ana	Owner	Avda. de la Constitucin 2222	Mxico D.F.
NULL	05021	Mexico	(5) 555-4729	(5) 555-3745	
1003	polaris	Antonio	Owner	Mataderos 2312	Mxico D.F.
NULL	05023	Mexico	(5) 555-3932	(5) 565-3532	
1004	virtuasa	Thomas Hardy	Sales Representative	120 Hanover Sq.	
London	NULL	WA1 1DP	UK	(171) 555-7788	(171) 555-6750
1005	genpact	Frdrique Citeaux	Marketing Manager	24, place Klber	
Strasbourg	NULL	67000	France	88.60.15.31	88.60.15.32

Sample data for employees entity:

EmployeeID	LastName	FirstName	JobTitle	BirthDate	HireDate	Address	City	PostalCode	Country	Phone	ReportsTo	Salary
5001	rao	krishna	Sales Representative	1948-12-08 00:00:00	1992-05-01 00:00:00	507 - 20th Ave. E.Apt. 2A	Seattle	98122	USA	22255556	1234	30000
5002	krishna	ram	Manager	1950-10-06 00:00:00	1995-06-02 00:00:00	307 - 15th Ave. E.Apt	Hyderabad	500031	India	9849789648	1315	35000
5003	hare	ram	Accountant	1975-10-05 00:00:00	1997-09-01 00:00:00	107 Apt	Secunderabad	500031	India	9849755648	1415	45000
5004	hare	krishna	Assistant Manager	1980-08-06 00:00:00	2002-10-05 00:00:00	407 Apt	Vijayawada	520031	India	9888755648	2415	40000
5005	murali	mohan	Manager	1982-08-07 00:00:00	2007-06-01 00:00:00	237 Apt	Hyderabad	500031	India	9888695648	2695	45000

Sample data for products entity:

ProductID	ProductName	SupplierID	CategoryID	QuantityPerUnit	UnitPrice	UnitsInStock	UnitsOnOrder	ReorderLevel	Discontinued
6001	computer	2001	7001	25 computers	25000.0000	20	10	5	☺
6002	laptop	2002	7002	20 laptops	45000.0000	10	5	3	
6003	pendrive	2003	7003	35 pendrives	1000.0000	30	10	5	
6004	memory card	2004	7004	45 memory cards	500.0000	50	20	10	☺
6005	headphones	2005	7005	20 headphones	1000.0000	30	15	10	

Sample data for orderdetails entity:

OrderID	ProductID	UnitPrice	Quantity
9001	6001	20000.0000	10
9002	6002	25000.0000	5
9003	6003	30000.0000	10
9004	6004	23000.0000	5
9005	6005	25000.0000	10

Sample data for shippers entity:

ShipperID	CompanyName	Phone
1	samsung	90000001
2	lg	6258215545
3	redme	6454545454
4	nokia	895633552
5	oppo	789658258

5 rows in set (0.00 sec)

Sample data for categories entity:

CategoryID	CategoryName
7001	electronics
7002	groceries
7003	clothing
7004	utensils
7005	bags

Sample data for suppliers entity:

|--|

SupplierID	CompanyName	ContactName	ContactJobTitle	Address	Phone
------------	-------------	-------------	-----------------	---------	-------

2001	samsung	abc	Salesman	hyderabad	987456321
------	---------	-----	----------	-----------	-----------

2002	lg	xyz	manager	delhi	879654123
------	----	-----	---------	-------	-----------

2003	redme	pqr	accountant	agra	658749234
------	-------	-----	------------	------	-----------

2004	nokia	uvw	assistant manager	vijayawada	896547213
------	-------	-----	-------------------	------------	-----------

2005	oppo	abcd	executive	secunderabad	985462317
------	------	------	-----------	--------------	-----------

2005	oppo	abcd	executive	secunderabad	985462317
------	------	------	-----------	--------------	-----------

Sample data for orders entity:

OrderID	CustomerID	EmployeeID	OrderDate	RequiredDate	ShippedDate	ShipId	ShipName	ShipAddress
---------	------------	------------	-----------	--------------	-------------	--------	----------	-------------

9001	1001	5001	1996-07-04 00:00:00	1996-08-01 00:00:00	1996-07-16 00:00:00	3	Reims	France
------	------	------	---------------------	---------------------	---------------------	---	-------	--------

9002	1002	5002	1996-07-05 00:00:00	1996-08-16 00:00:00	1996-07-10 00:00:00	1	Mnster	Germany
------	------	------	---------------------	---------------------	---------------------	---	--------	---------

9003	1003	5003	1998-11-06 00:00:00	1998-12-04 00:00:00	1998-11-08 00:00:00	3	SP	Brazil
------	------	------	---------------------	---------------------	---------------------	---	----	--------

9004	1004	5004	2002-11-12 00:00:00	2002-11-26 00:00:00	2002-11-18 00:00:00	2	Lisboa	Portugal
------	------	------	---------------------	---------------------	---------------------	---	--------	----------

9005	1005	5005	2008-11-25 00:00:00	2008-12-23 00:00:00	2008-11-28 00:00:00	4	Marseille	France
------	------	------	---------------------	---------------------	---------------------	---	-----------	--------

9005	1005	5005	2008-11-25 00:00:00	2008-12-23 00:00:00	2008-11-28 00:00:00	4	Marseille	France
------	------	------	---------------------	---------------------	---------------------	---	-----------	--------

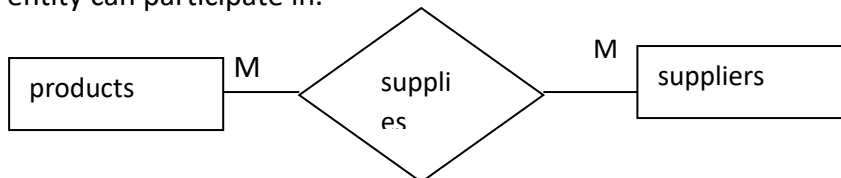
9005	1005	5005	2008-11-25 00:00:00	2008-12-23 00:00:00	2008-11-28 00:00:00	4	Marseille	France
------	------	------	---------------------	---------------------	---------------------	---	-----------	--------

Relate the entities appropriately. Apply cardinalities for each relationship. Identify strong entities and weak entities (if any). Indicate the type of relationship (total/partial). Try to incorporate generalization, aggregation, specialization etc wherever required.

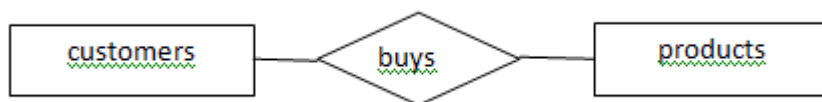
MODEL INPUT

DEFINITIONS:

The cardinality ratio: - for a binary relationship specifies the maximum number of relationships that an entity can participate in.



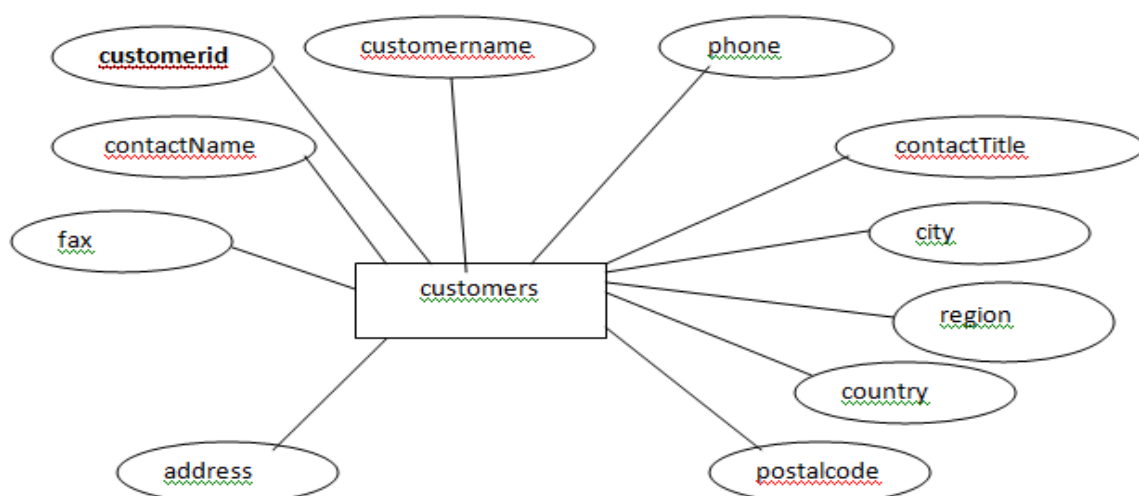
Relationship: - it is defined as an association among two or more entities.



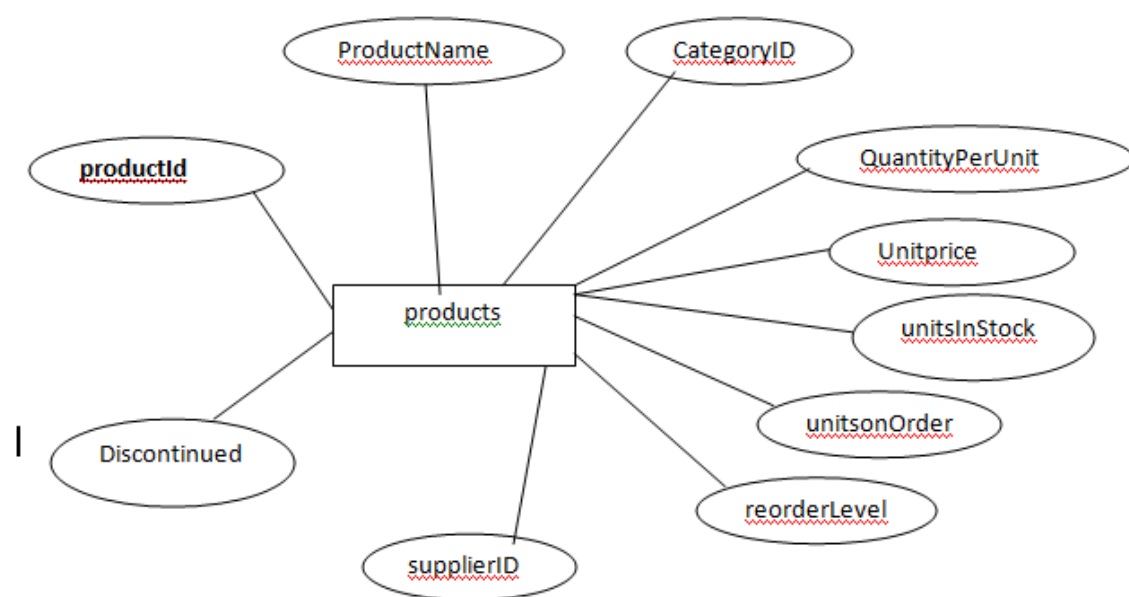
Weak and strong entity: - an entity set may not have sufficient attributes to form a primary key. Such an entity set is termed a weak entity set. An entity set that has primary key is termed a strong entity set.

EXPECTED OUTPUT

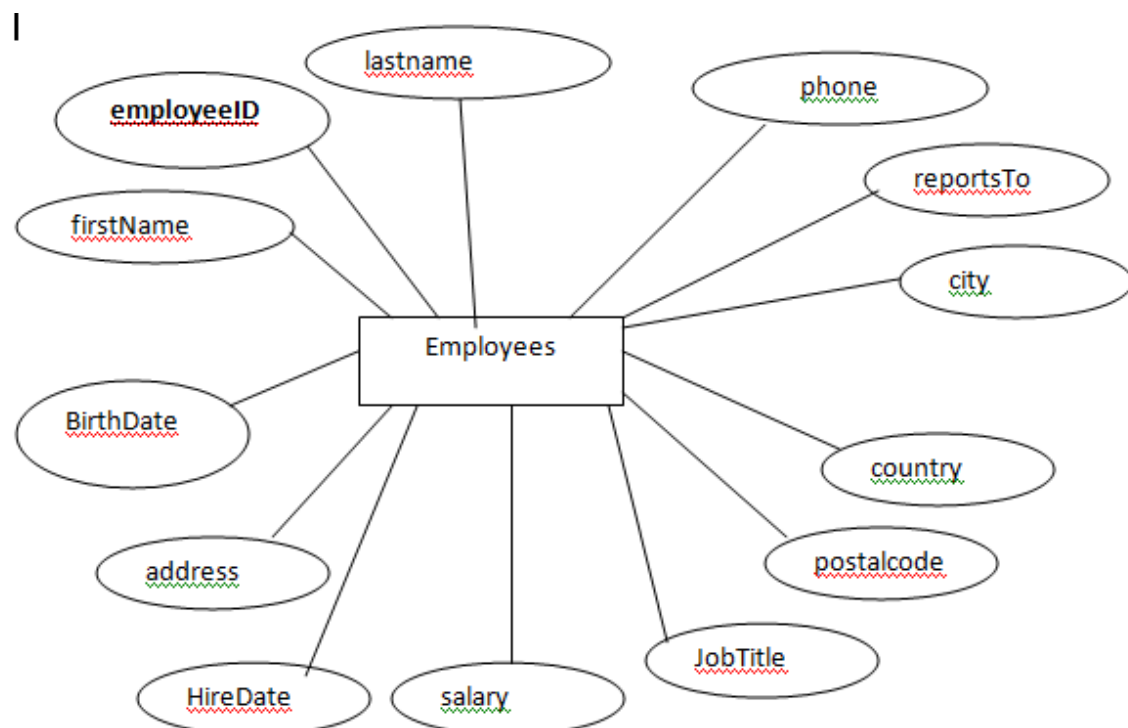
Entity diagram for Customers



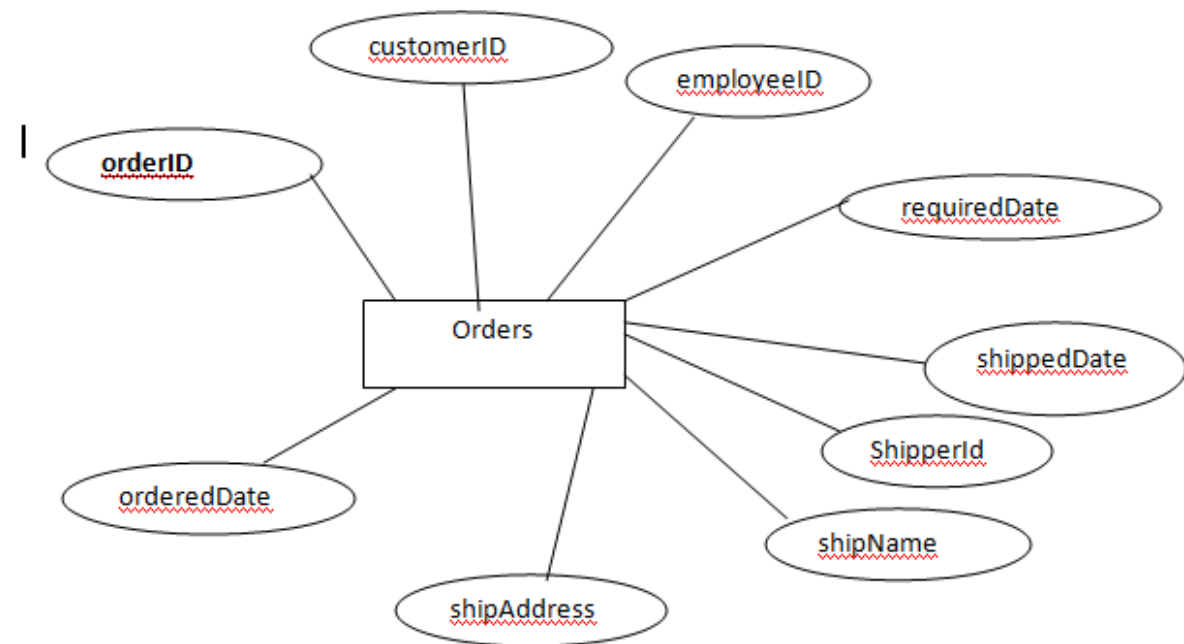
Entity diagram for Products



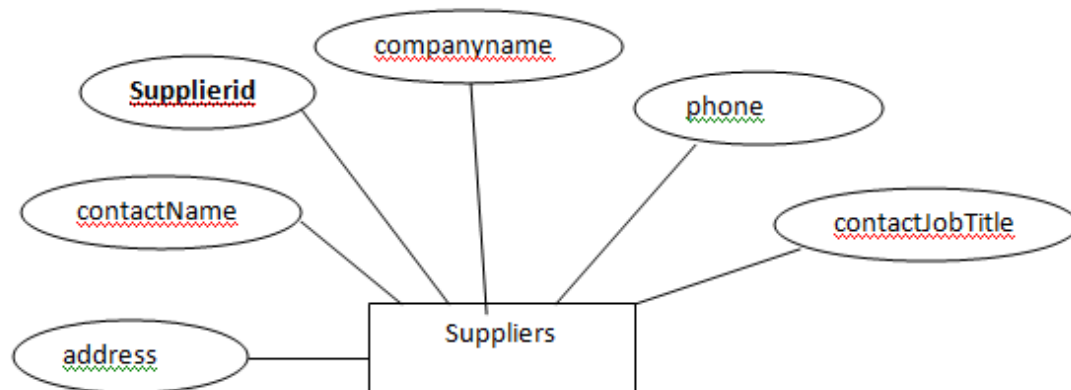
Entity diagram for Employees



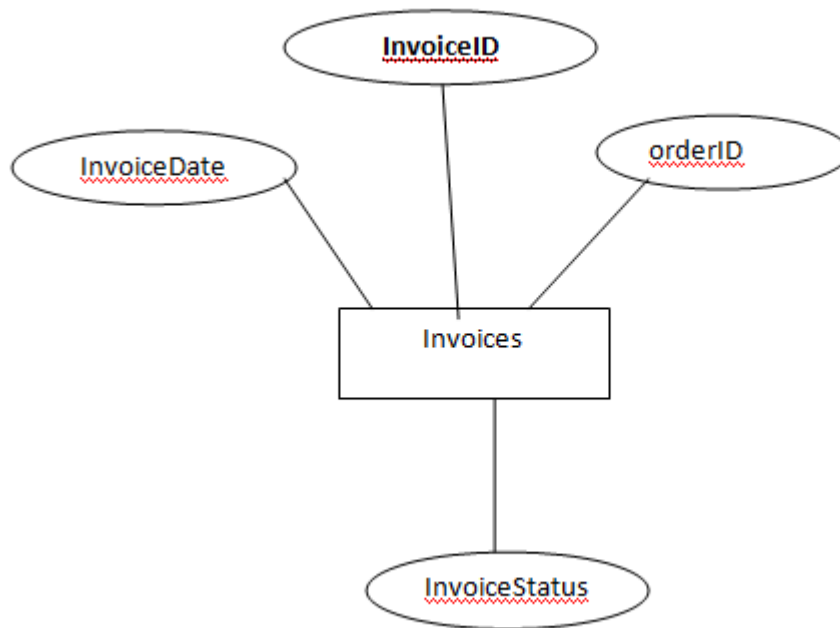
Entity diagram for Orders



Entity diagram for Suppliers:

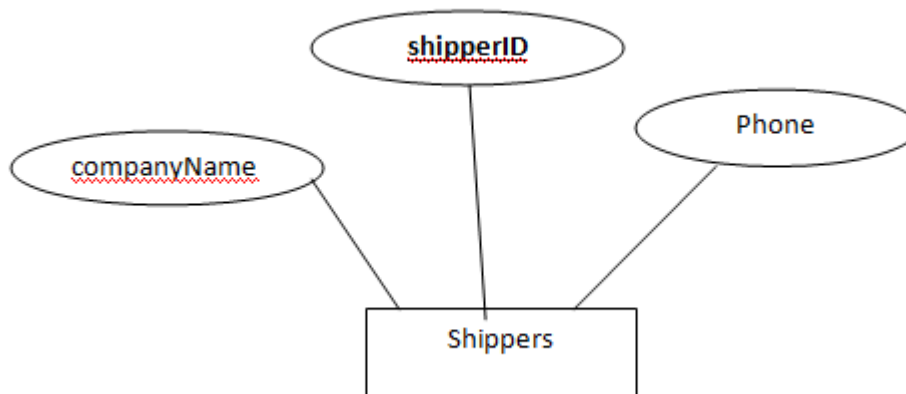


Entity Diagram of Invoices



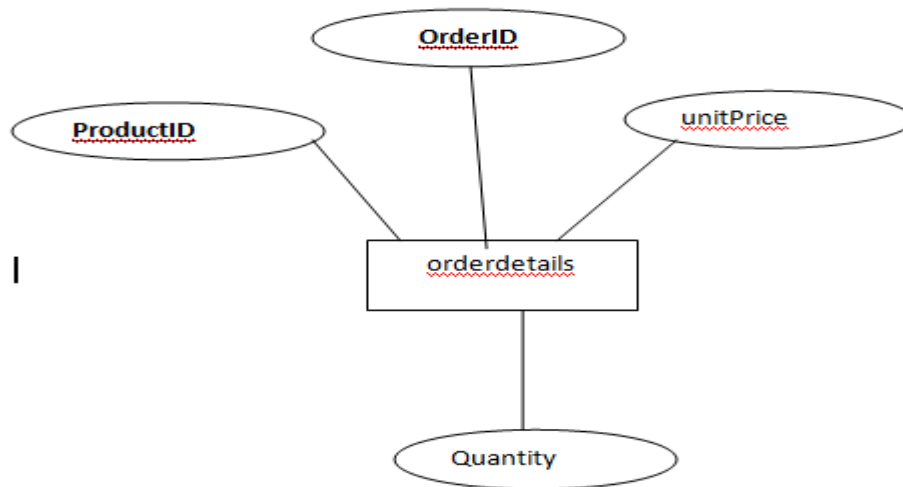
Entity

Diagram of Shippers



I

Entity Diagram of orderdetails



Normalization

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

MODEL INPUT

The normalization forms are:

1. **First Normal Form:** 1NF requires that the values in each column of a table are atomic. By atomic we mean that there are no sets of values within a column.
2. **Second Normal Form:** where the 1NF deals with atomicity of data, the 2NF deals with relationships between composite key columns and non-key columns. To achieve 2NF the tables should be in 1NF. The 2NF any non-key columns must depend on the entire primary key. In case of a composite primary key, this means that non-key column can't depend on only part of the composite key.
3. **Third Normal Form:** 3NF requires that all columns depend directly on the primary key. Tables violate the third normal form when one column depends on another column, which in turn depends on the primary key(transitive dependency). One way to identify transitive dependency is to look at your tables and see if any columns would require

updating if another column in the table was updated. If such a column exists, it probably violates 3NF.

Experiment 2

Installation & DDL

Installation of Mysql and practicing DDL commands.

Creating databases, How to create tables, altering the database or tables, dropping tables and databases if not required. You will also try truncate, rename commands etc.

Data Definition Language (DDL) : create , alter, drop.

Installation of Mysql and Practicing DDL commands:

Installation of Mysql. In this week you will learn creating databases, how to create tables, altering the tables, dropping tables and databases if not required. You will also try truncate, rename commands etc.

MODEL INPUT

Step: 1 download mysql essential from the website www.mysql.com/downloads and save the .exe file.

Step: 2 Double click on the mysql.exe file to start installation.

Step: 3



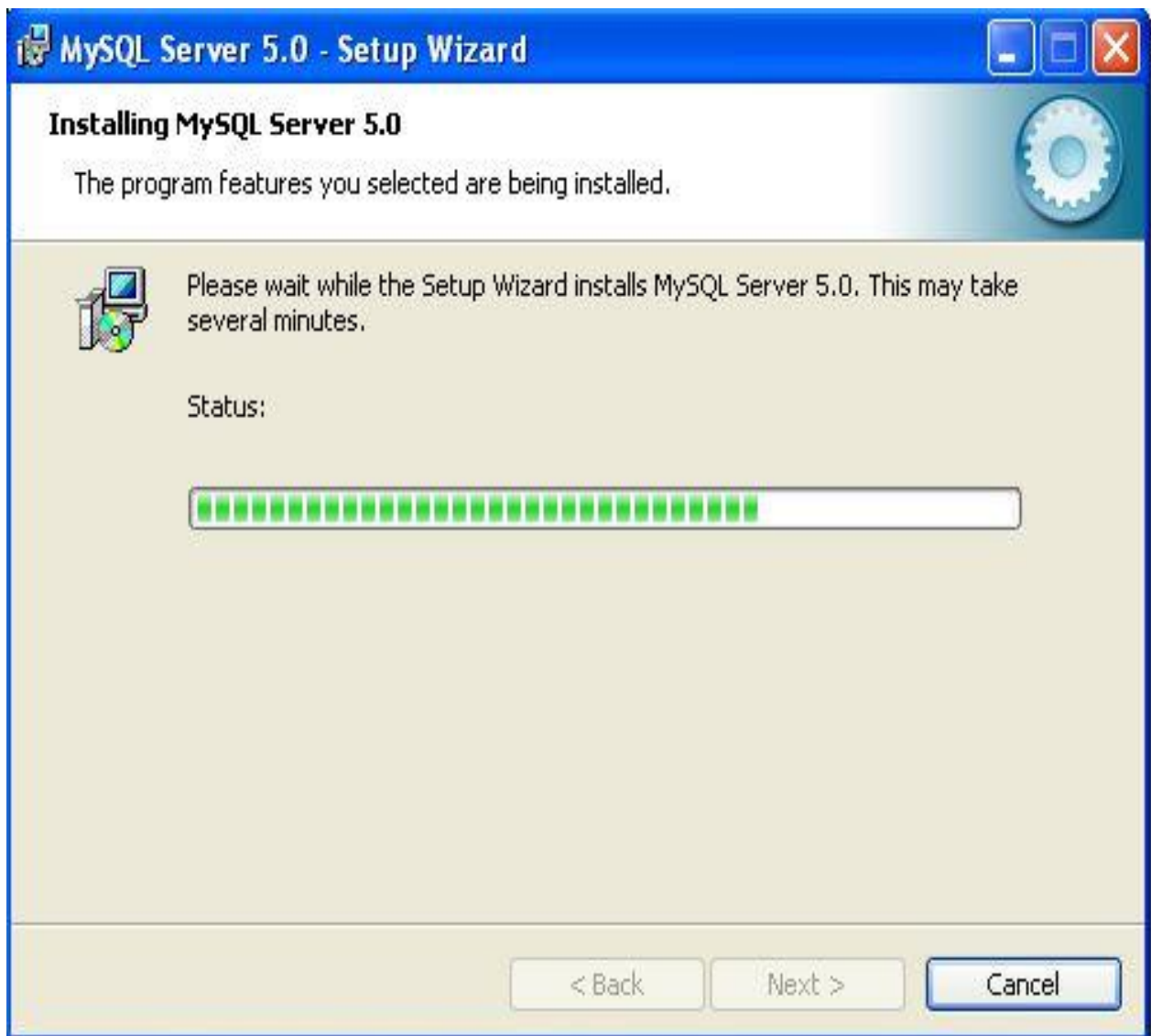
Step : 4



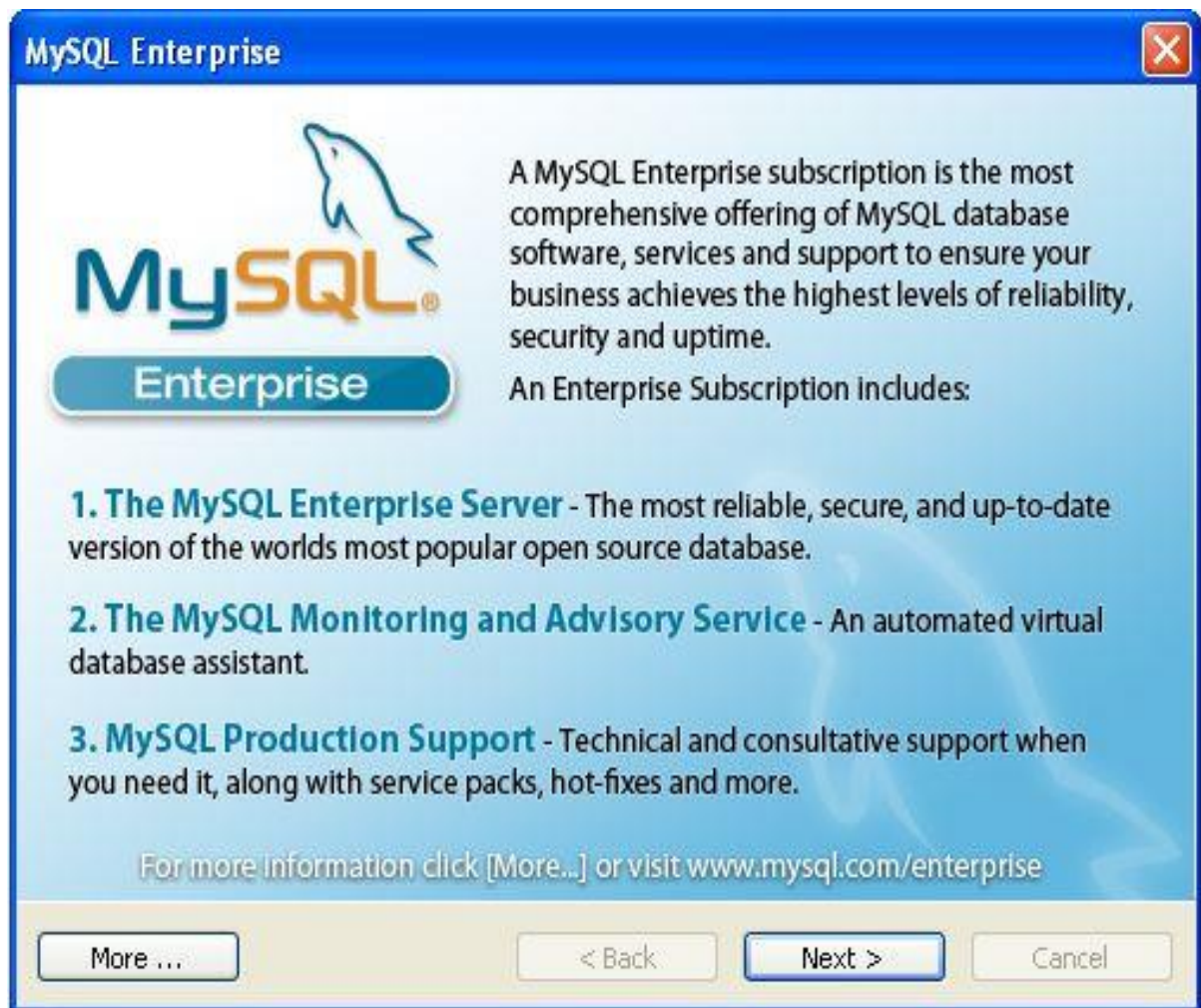
Step: 5



Step: 6



Step: 7



Step 8



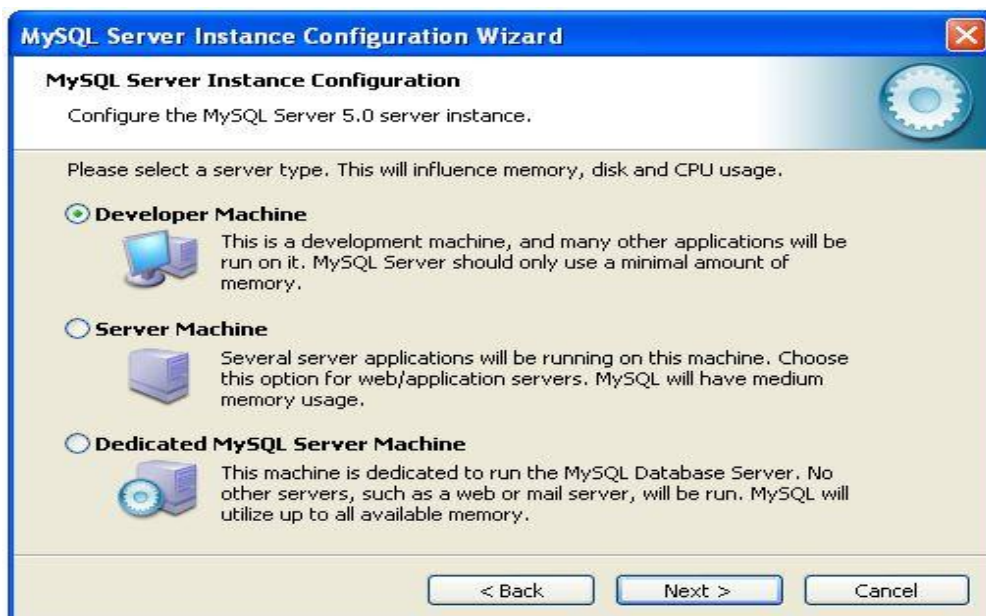
Step: 9



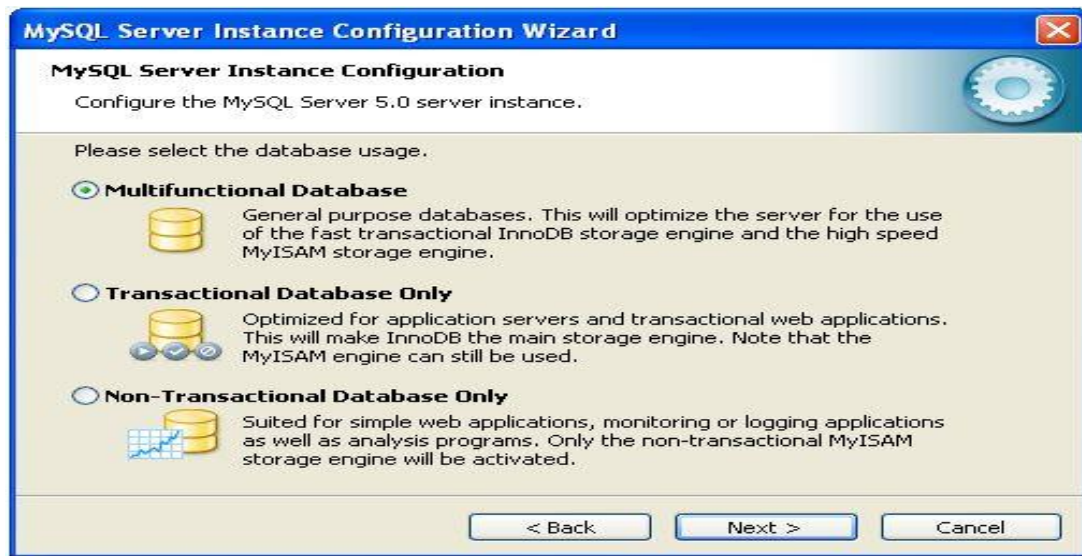
Step: 10



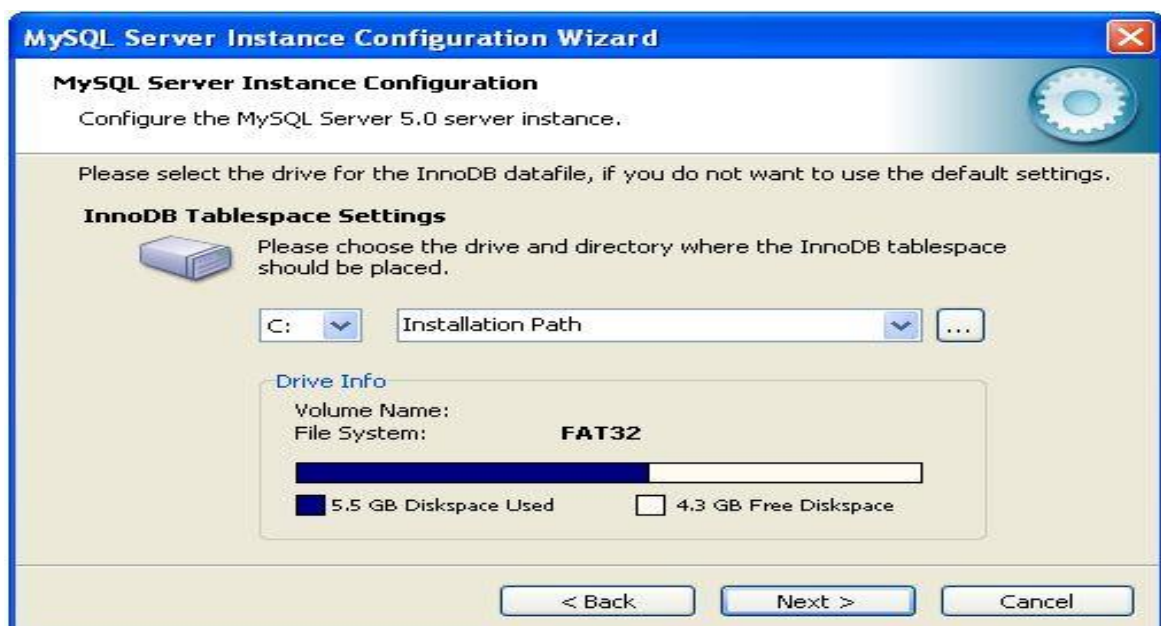
Step: 11



Step: 12



Step: 13



Step: 14



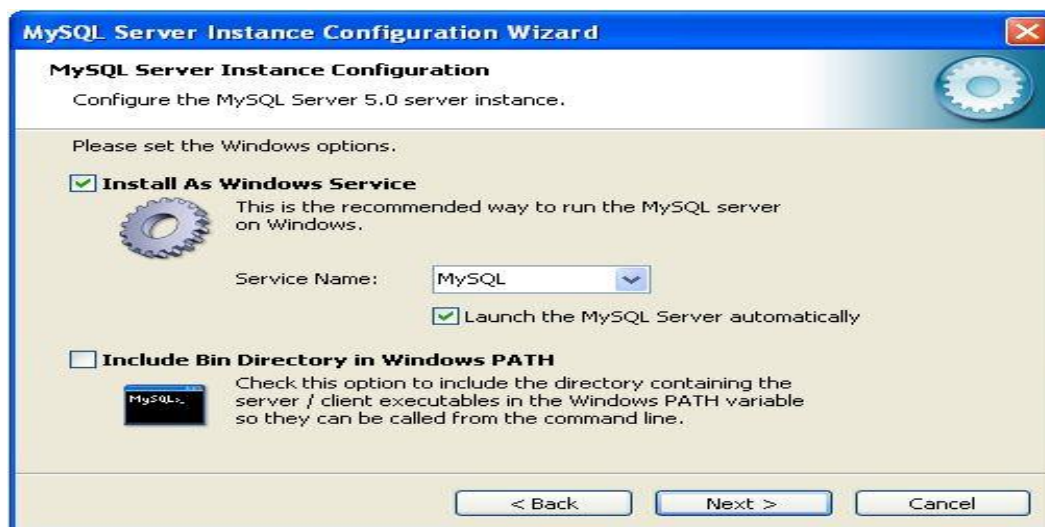
Step: 15



Step: 16



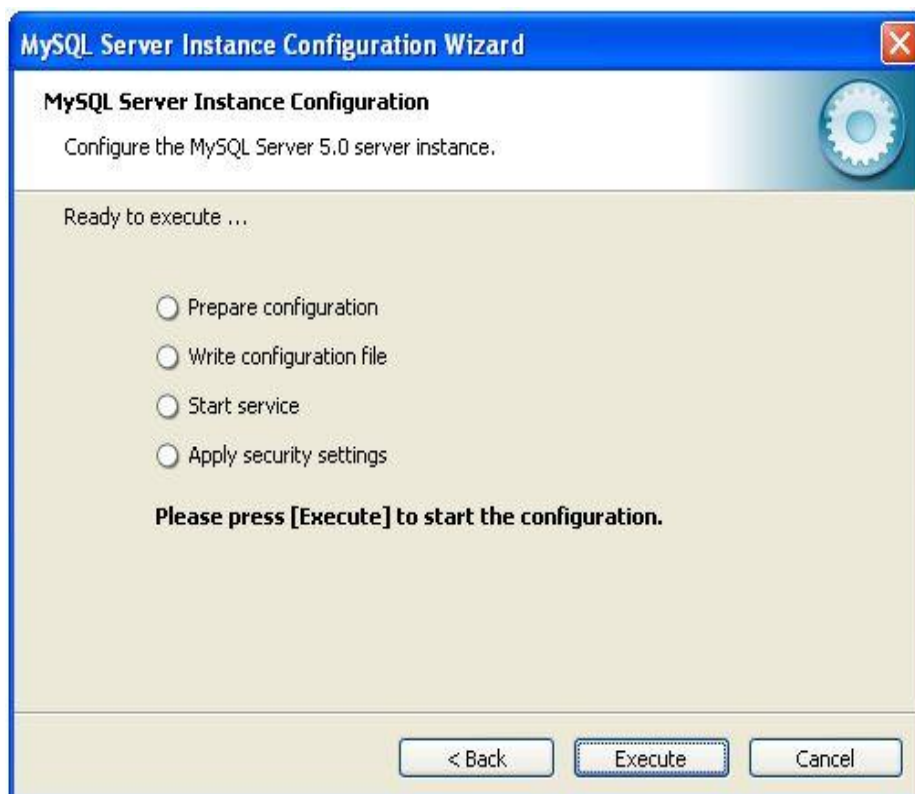
Step: 17



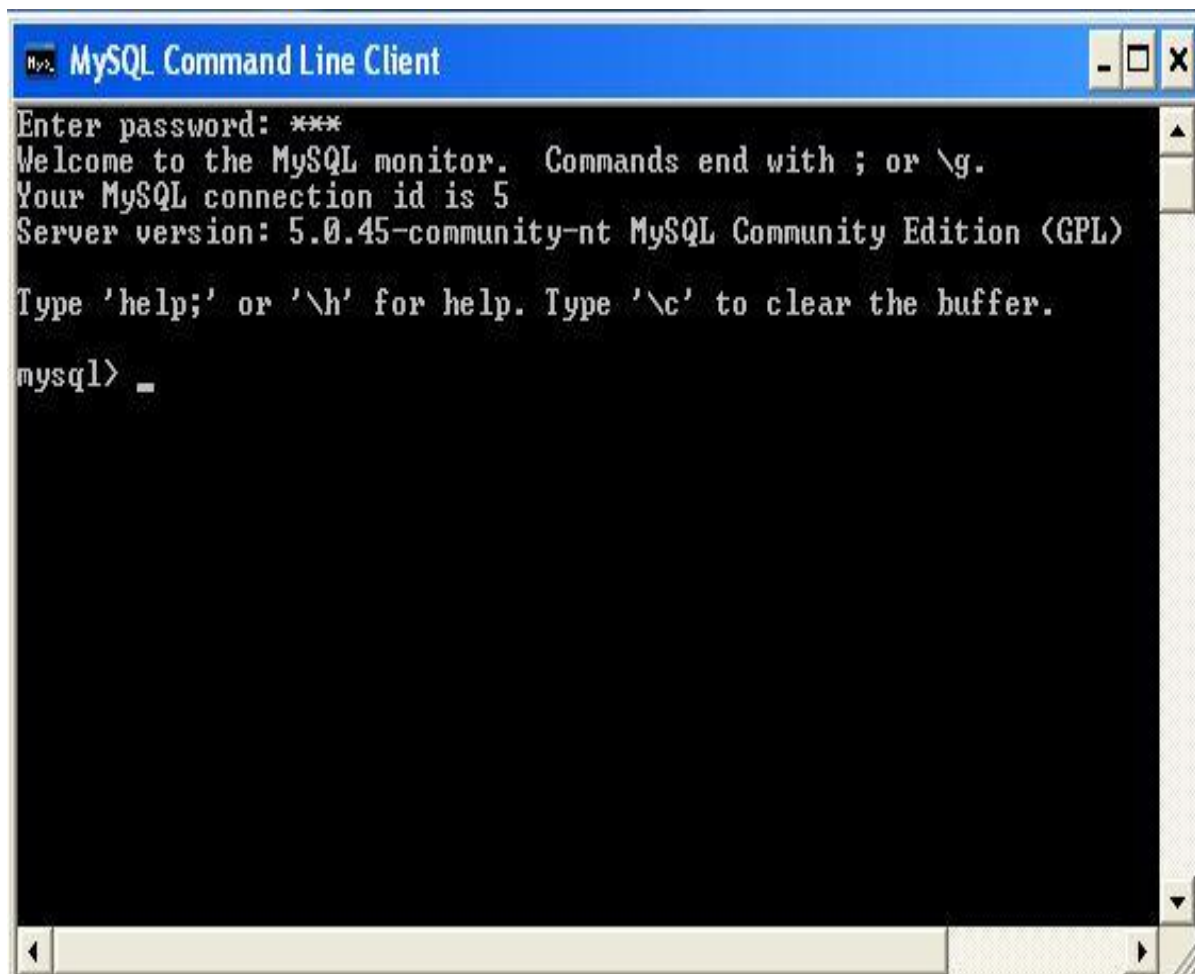
Step: 18



Step: 19



Step: 20



```
mysql> create database southwind;  
Query OK, 1 row affected (0.00 sec)
```

```
mysql> show databases;
```

```
+-----+  
| Database |  
+-----+  
| information_schema |  
| kmit |  
| mysql |  
| performance_schema |  
| southwind |  
| test |  
+-----+  
6 rows in set (0.09 sec)
```

```
mysql> drop database southwind;  
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> show databases;
```

```
+-----+  
| Database |  
+-----+  
| information_schema |  
| kmit |  
| mysql |  
| performance_schema |  
| test |  
+-----+  
5 rows in set (0.09 sec)
```

```
mysql> create database southwind;  
Query OK, 1 row affected (0.00 sec)
```

```
mysql> show databases;
```

```
+-----+  
| Database |  
+-----+  
| information_schema |  
| kmit |  
| mysql |  
| performance_schema |  
| southwind |  
| test |  
+-----+  
6 rows in set (0.09 sec)
```

```
mysql> use southwind;  
Database changed
```

```
mysql> show tables;  
Empty set (0.00 sec)
```

Creating Tables:-

Syntax: Create table tablename (columnname 1 datatype 1,
 Columnname 2 datatype 2,
 : :
 : :
 Columnname n datatype n ...);

Example:

```
mysql> create table student1(rollno int, name varchar(10),dob date, city varchar(10), state  
varchar(10));
```

Query OK, 0 rows affected (0.07 sec)

Table Created

Observing Table Information:-

The easiest way to get description about a table is with the DESC command:

Syntax: DESC table name;

Example:

```
mysql> desc student1;
```

```
+-----+-----+-----+-----+-----+
| Field | Type      | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
| rollno | int(11)    | YES  |     | NULL    |       |
| name   | varchar(10)| YES  |     | NULL    |       |
| dob    | date       | YES  |     | NULL    |       |
| city   | varchar(10)| YES  |     | NULL    |       |
| state  | varchar(10)| YES  |     | NULL    |       |
+-----+-----+-----+-----+-----+
```

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

Altering Tables:-

To alter a table use the alter table command:

Syntax1:

```
Alter table tablename add (columnname datatype, ...);
```

Syntax2:

```
Alter table tablename drop column columnname;
```

Example 1:

```
mysql> alter table student1 add(pin int);
```

Query OK, 0 rows affected (0.31 sec)

Records: 0 Duplicates: 0 Warnings: 0

Table altered

```
mysql> desc student1;
```

Field	Type	Null	Key	Default	Extra
rollno	int(11)	YES		NULL	
name	varchar(10)	YES		NULL	
dob	date	YES		NULL	
city	varchar(10)	YES		NULL	
state	varchar(10)	YES		NULL	
pin	int(11)	YES		NULL	


```
+-----+-----+-----+-----+-----+
```

6 rows in set (0.02 sec)

```
mysql> alter table student1 drop column pin;
```

Query OK, 0 rows affected (0.23 sec)

Records: 0 Duplicates: 0 Warnings: 0

```
mysql> desc student1;
```

```
+-----+-----+-----+-----+-----+
```

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

```
+-----+-----+-----+-----+-----+
```

```
| rollno | int(11)   | YES  |     | NULL    |      |
```

```
| name   | varchar(10) | YES  |     | NULL    |      |
```

```
| dob    | date      | YES  |     | NULL    |      |
```

```
| city   | varchar(10) | YES  |     | NULL    |      |
```

```
| state  | varchar(10) | YES  |     | NULL    |      |
```

```
+-----+-----+-----+-----+-----+
```

5 rows in set (0.01 sec)

```
mysql> alter table student1 add pin int;
```

```
Query OK, 0 rows affected (0.21 sec)
```

```
Records: 0 Duplicates: 0 Warnings: 0
```

```
mysql> desc student1;
```

```
+-----+-----+-----+-----+-----+
| Field | Type   | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
| rollno | int(11) | YES  |     | NULL    |      |
| name   | varchar(10) | YES  |     | NULL    |      |
| dob    | date    | YES  |     | NULL    |      |
| city   | varchar(10) | YES  |     | NULL    |      |
| state  | varchar(10) | YES  |     | NULL    |      |
| pin    | int(11) | YES  |     | NULL    |      |
+-----+-----+-----+-----+-----+
```

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

```
mysql> alter table student1 drop column dob;
```

```
Query OK, 0 rows affected (0.20 sec)
```

```
Records: 0 Duplicates: 0 Warnings: 0
```

```
mysql> desc student1;
```

```
+-----+-----+-----+-----+-----+
| Field | Type   | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
| rollno | int(11) | YES  |     | NULL    |      |
```

```
| name | varchar(10) | YES | | NULL | |
| city | varchar(10) | YES | | NULL | |
| state | varchar(10) | YES | | NULL | |
| pin | int(11) | YES | | NULL | |
+-----+-----+-----+-----+-----+-----+
```

5 rows in set (0.00 sec)

Dropping Tables:-

To delete a table use the following:

Syntax: Drop table tablename;

Example:

```
mysql> drop table student1;
```

Query OK, 0 rows affected (0.08 sec)

Table dropped.

```
mysql> desc student1;
```

ERROR 1146 (42S02): Table 'southwind.student1' doesn't exist

Object student does not exist

```
mysql> create table products (
productID INT UNSIGNED NOT NULL AUTO_INCREMENT,
productCode CHAR(3) NOT NULL DEFAULT "",
name VARCHAR(30) NOT NULL DEFAULT "",
quantity INT UNSIGNED NOT NULL DEFAULT 0,
price DECIMAL(7,2) NOT NULL DEFAULT 99999.99,
PRIMARY KEY (productID)
);
```

Query OK, 0 rows affected (0.24 sec)

```
mysql> desc products;
```

```
+-----+-----+-----+-----+-----+-----+
| Field      | Type          | Null | Key | Default | Extra          |
+-----+-----+-----+-----+-----+-----+
| productID  | int(10) unsigned | NO   | PRI | NULL    | auto_increment |
| productCode | char(3)         | NO   |     |         |                |
| name       | varchar(30)     | NO   |     |         |                |
| quantity   | int(10) unsigned | NO   |     | 0       |                |
| price      | decimal(7,2)    | NO   |     | 99999.99 |                |
+-----+-----+-----+-----+-----+-----+
```

5 rows in set (0.03 sec)

```
mysql> INSERT INTO products VALUES (1001, 'PEN', 'Pen Red', 5000, 1.23);
```

Query OK, 1 row affected (0.04 sec)

```
mysql> INSERT INTO products VALUES (NULL, 'PEN', 'Pen Blue', 8000, 1.25),
(NULL, 'PEN', 'Pen Black', 2000, 1.25);
```

Query OK, 2 rows affected (0.04 sec)

Records: 2 Duplicates: 0 Warnings: 0

```
mysql> INSERT INTO products (productCode, name, quantity, price) VALUES
('PEC', 'Pencil 2B', 10000, 0.48), ('PEC', 'Pencil 2H', 8000, 0.49);
```

Query OK, 2 rows affected (0.04 sec)

Records: 2 Duplicates: 0 Warnings: 0

```
mysql> INSERT INTO products (productCode, name) VALUES ('PEC', 'Pencil HB');
```

Query OK, 1 row affected (0.06 sec)

```
mysql> select * from products;
```

productID	productCode	name	quantity	price
1001	PEN	Pen Red	5000	1.23
1002	PEN	Pen Blue	8000	1.25
1003	PEN	Pen Black	2000	1.25
1004	PEC	Pencil 2B	10000	0.48
1005	PEC	Pencil 2H	8000	0.49
1006	PEC	Pencil HB	0	99999.99

6 rows in set (0.00 sec)

```
mysql> select name, price from products;
```

name	price
Pen Red	1.23
Pen Blue	1.25
Pen Black	1.25
Pencil 2B	0.48
Pencil 2H	0.49
Pencil HB	99999.99

6 rows in set (0.00 sec)

Comparison Operators

For numbers (INT, DECIMAL, FLOAT), you could use comparison operators: '=' (equal to), '<>' or '!=' (not equal to), '>' (greater than), '<' (less than), '>=' (greater than or equal to), '<=' (less than or equal to), to compare two numbers. For example, price > 1.0, quantity <= 500.

```
mysql> SELECT name, price FROM products WHERE price < 1.0;
```

name	price
Pencil 2B	0.48
Pencil 2H	0.49

2 rows in set (0.03 sec)

```
mysql> SELECT name, quantity FROM products WHERE quantity <= 2000;
```

name	quantity
Pen Black	2000
Pencil HB	0

2 rows in set (0.00 sec)

CAUTION: Do not compare FLOATs (real numbers) for equality ('=' or '<>'), as they are not precise. On the other hand, DECIMAL are precise.

For strings, you could also use '=', '<>', '>', '<', '>=', '<=' to compare two strings (e.g., productCode = 'PEC'). The ordering of string depends on the so-called *collation* chosen. For example,

```
mysql> SELECT name, price FROM products WHERE productCode = 'PEN';
```

```
+-----+-----+
| name   | price |
+-----+-----+
| Pen Red | 1.23  |
| Pen Blue | 1.25  |
| Pen Black | 1.25  |
+-----+-----+
3 rows in set (0.00 sec)
```

String Pattern Matching - LIKE and NOT LIKE

For strings, in addition to full matching using operators like '=' and '<>', we can perform *pattern matching* using operator LIKE (or NOT LIKE) with wildcard characters. The wildcard '_' matches any single character; '%' matches any number of characters (including zero). For example,

- 'abc%' matches strings beginning with 'abc';
- '%xyz' matches strings ending with 'xyz';
- '%aaa%' matches strings containing 'aaa';
- '___' matches strings containing exactly three characters; and
- 'a_b%' matches strings beginning with 'a', followed by any single character, followed by 'b', followed by zero or more characters.

```
mysql> SELECT name, price FROM products WHERE name LIKE 'PENCIL%';
```

```
+-----+-----+
| name   | price |
+-----+-----+
| Pencil 2B | 0.48  |
| Pencil 2H | 0.49  |
| Pencil HB | 99999.99 |
+-----+-----+
3 rows in set (0.00 sec)
```

```
mysql> SELECT name, price FROM products WHERE name LIKE 'P__ %';
```

```
+-----+-----+
| name   | price |
+-----+-----+
| Pen Red | 1.23  |
| Pen Blue | 1.25  |
| Pen Black | 1.25  |
+-----+-----+
3 rows in set (0.00 sec)
```

Arithmetic Operators

```
mysql> select (20+30);
```

```
+-----+  
| (20+30) |
```

```
+-----+
```

```
|    50 |
```

```
+-----+
```

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

```
mysql> select (30-20);
```

```
+-----+  
| (30-20) |
```

```
+-----+
```

```
|    10 |
```

```
+-----+
```

```
1 row in set (0.12 sec)
```

```
mysql> select (20*5);
```

```
+-----+  
| (20*5) |
```

```
+-----+
```

```
|   100 |
```

```
+-----+
```

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

```
mysql> select (20/5);
```

```

+-----+
| (20/5) |
+-----+
| 4.0000 |
+-----+
1 row in set (0.02 sec)

```

```

mysql> select (20%5);
+-----+
| (20%5) |
+-----+
|      0 |
+-----+
1 row in set (0.00 sec)

```

```

mysql> select (20 div 5);
+-----+
| (20 div 5) |
+-----+
|          4 |
+-----+
1 row in set (0.00 sec)

```

Logical Operators - AND, OR, NOT, XOR

You can combine multiple conditions with boolean operators AND, OR, XOR. You can also invert a condition using operator NOT. For examples,

```

mysql> SELECT * FROM products WHERE quantity >= 5000 AND name LIKE 'Pen %';
+-----+-----+-----+-----+-----+

```


productID	productCode	name	quantity	price
1001	PEN	Pen Red	5000	1.23
1002	PEN	Pen Blue	8000	1.25

2 rows in set (0.00 sec)

```
mysql> SELECT * FROM products WHERE quantity >= 5000 AND price < 1.24 AND name LIKE 'Pen %';
```

productID	productCode	name	quantity	price
1001	PEN	Pen Red	5000	1.23

1 row in set (0.00 sec)

```
mysql> SELECT * FROM products WHERE NOT (quantity >= 5000 AND name LIKE 'Pen %');
```

productID	productCode	name	quantity	price
1003	PEN	Pen Black	2000	1.25
1004	PEC	Pencil 2B	10000	0.48
1005	PEC	Pencil 2H	8000	0.49
1006	PEC	Pencil HB	0	99999.99

4 rows in set (0.00 sec)

IN, NOT IN

You can select from members of a set with IN (or NOT IN) operator. This is easier and clearer than the equivalent AND-OR expression.

```
mysql> SELECT * FROM products WHERE name IN ('Pen Red', 'Pen Black');
```

productID	productCode	name	quantity	price
1001	PEN	Pen Red	5000	1.23
1003	PEN	Pen Black	2000	1.25

2 rows in set (0.03 sec)

BETWEEN, NOT BETWEEN

To check if the value is within a range, you could use BETWEEN ... AND ... operator. Again, this is easier and clearer than the equivalent AND-OR expression.

```
mysql> SELECT * FROM products WHERE (price BETWEEN 1.0 AND 2.0) AND (quantity BETWEEN 1000 AND 2000);
```

```
+-----+-----+-----+-----+
| productID | productCode | name      | quantity | price |
+-----+-----+-----+-----+
| 1003 | PEN      | Pen Black | 2000 | 1.25 |
+-----+-----+-----+-----+
1 row in set (0.00 sec)
```

IS NULL, IS NOT NULL

NULL is a special value, which represent "no value", "missing value" or "unknown value". You can checking if a column contains NULL by IS NULL or IS NOT NULL. For example,

```
mysql> SELECT * FROM products WHERE productCode IS NULL;
Empty set (0.00 sec)
```

```
mysql> SELECT * FROM products WHERE productCode IS NOT NULL;
```

```
+-----+-----+-----+-----+
| productID | productCode | name      | quantity | price |
+-----+-----+-----+-----+
| 1001 | PEN      | Pen Red   | 5000 | 1.23 |
| 1002 | PEN      | Pen Blue  | 8000 | 1.25 |
| 1003 | PEN      | Pen Black | 2000 | 1.25 |
| 1004 | PEC      | Pencil 2B | 10000 | 0.48 |
| 1005 | PEC      | Pencil 2H | 8000 | 0.49 |
| 1006 | PEC      | Pencil HB | 0 | 99999.99 |
+-----+-----+-----+-----+
6 rows in set (0.00 sec)
```

ORDER BY Clause

You can order the rows selected using ORDER BY clause, with the following syntax:

```
SELECT ... FROM tableName WHERE criteria ORDER BY columnA ASC|DESC, columnB ASC|DESC, ...
```

The selected row will be ordered according to the values in *columnA*, in either ascending (ASC) (default) or descending (DESC) order. If several rows have the same value in *columnA*, it will be ordered according to *columnB*, and so on. For strings, the ordering could be case-sensitive or case-insensitive, depending on the so-called character collating sequence used. For examples,

```
mysql> SELECT * FROM products WHERE name LIKE 'Pen %' ORDER BY price DESC;
```

```
+-----+-----+-----+-----+-----+
| productID | productCode | name      | quantity | price |
+-----+-----+-----+-----+-----+
| 1002 | PEN      | Pen Blue | 8000 | 1.25 |
| 1003 | PEN      | Pen Black | 2000 | 1.25 |
| 1001 | PEN      | Pen Red  | 5000 | 1.23 |
+-----+-----+-----+-----+-----+
```

3 rows in set (0.01 sec)

```
mysql> SELECT * FROM products WHERE name LIKE 'Pen %' ORDER BY price DESC, quantity;
```

```
+-----+-----+-----+-----+-----+
| productID | productCode | name      | quantity | price |
+-----+-----+-----+-----+-----+
| 1003 | PEN      | Pen Black | 2000 | 1.25 |
| 1002 | PEN      | Pen Blue  | 8000 | 1.25 |
| 1001 | PEN      | Pen Red   | 5000 | 1.23 |
+-----+-----+-----+-----+-----+
```

3 rows in set (0.00 sec)

You can randomize the returned records via function RAND(), e.g.,

```
mysql> SELECT * FROM products ORDER BY RAND();
```

```
+-----+-----+-----+-----+-----+
| productID | productCode | name      | quantity | price |
+-----+-----+-----+-----+-----+
| 1006 | PEC      | Pencil HB | 0 | 99999.99 |
| 1001 | PEN      | Pen Red   | 5000 | 1.23 |
| 1003 | PEN      | Pen Black | 2000 | 1.25 |
| 1004 | PEC      | Pencil 2B | 10000 | 0.48 |
| 1002 | PEN      | Pen Blue  | 8000 | 1.25 |
| 1005 | PEC      | Pencil 2H | 8000 | 0.49 |
+-----+-----+-----+-----+-----+
```

6 rows in set (0.08 sec)

LIMIT Clause

A SELECT query on a large database may produce many rows. You could use the LIMIT clause to limit the number of rows displayed, e.g.,

```
mysql> SELECT * FROM products ORDER BY price LIMIT 2;
```

productID	productCode	name	quantity	price
1004	PEC	Pencil 2B	10000	0.48
1005	PEC	Pencil 2H	8000	0.49

2 rows in set (0.00 sec)

To continue to the following records , you could specify the number of rows to be skipped, followed by the number of rows to be displayed in the LIMIT clause, as follows:

```
mysql> SELECT * FROM products ORDER BY price LIMIT 2, 1;
```

productID	productCode	name	quantity	price
1001	PEN	Pen Red	5000	1.23

1 row in set (0.00 sec)

AS - Alias

You could use the keyword AS to define an *alias* for an identifier (such as column name, table name). The alias will be used in displaying the name. It can also be used as reference. For example,

```
mysql> SELECT productID AS ID, productCode AS Code,  
name AS Description, price AS `Unit Price` FROM products  
ORDER BY ID;
```

ID	Code	Description	Unit Price
1001	PEN	Pen Red	1.23
1002	PEN	Pen Blue	1.25
1003	PEN	Pen Black	1.25
1004	PEC	Pencil 2B	0.48
1005	PEC	Pencil 2H	0.49
1006	PEC	Pencil HB	99999.99

6 rows in set (0.03 sec)

Take note that the identifier "Unit Price" contains a blank and must be *back-quoted*.

Function CONCAT()

You can also concatenate a few columns as one (e.g., joining the last name and first name) using function CONCAT(). For example,

```
mysql> SELECT CONCAT(productCode, ' - ', name) AS `Product Description`, price FROM products;
```

```
+-----+-----+
| Product Description | price |
+-----+-----+
| PEN - Pen Red      | 1.23 |
| PEN - Pen Blue     | 1.25 |
| PEN - Pen Black    | 1.25 |
| PEC - Pencil 2B    | 0.48 |
| PEC - Pencil 2H    | 0.49 |
| PEC - Pencil HB    | 99999.99 |
+-----+-----+
6 rows in set (0.04 sec)
```

Experiment 3

DML

Data Manipulation Language Commands (DML) commands are used to for managing data within schema objects.

Exercising the commands using **DML** : insert, delete, update on the following tables : products, customers, suppliers, orders, , employees, order details, categories.

- INSERT – insert data into a table.
- UPDATE – updates existing data within a table.
- DELETE – deletes single or all records from a table.

Data Query Language – Select

```
mysql> CREATE TABLE Customers( CustomerID VARCHAR(5) NOT NULL, CompanyName
VARCHAR(40) NOT NULL, ContactName VARCHAR(30), ContactTitle VARCHAR(30), Address
VARCHAR(60), City VARCHAR(15), Region VARCHAR(15), PostalCode VARCHAR(10),
Country VARCHAR(15), Phone VARCHAR(24), Fax VARCHAR(24), CONSTRAINT
`PK_Customers` PRIMARY KEY (`CustomerID`));
Query OK, 0 rows affected (0.09 sec)
```

```
mysql> desc customers;
```

Field	Type	Null	Key	Default	Extra
CustomerID	varchar(5)	NO	PRI	NULL	
CompanyName	varchar(40)	NO		NULL	
ContactName	varchar(30)	YES		NULL	
ContactTitle	varchar(30)	YES		NULL	
Address	varchar(60)	YES		NULL	
City	varchar(15)	YES		NULL	
Region	varchar(15)	YES		NULL	
PostalCode	varchar(10)	YES		NULL	
Country	varchar(15)	YES		NULL	
Phone	varchar(24)	YES		NULL	
Fax	varchar(24)	YES		NULL	

```
11 rows in set (0.01 sec)
```

```
mysql> insert into Customers( CustomerID, CompanyName, ContactName, ContactTitle,
Address, City, Region, PostalCode, Country, Phone, Fax) values('1001','tcs','maria',
'sales representative','Obere Str. 57','Berlin', NULL, '12209','Germany','030-0074321','030-
0076545'),('1002','wipro','Ana ','Owner','Avda. de la Constitucin 2222','Mxico D.F.', NULL,
'05021','Mexico','(5) 555-4729','(5) 555-3745'),('1003','polaris','Antonio ','Owner',
'Mataderos 2312','Mxico D.F.', NULL, '05023','Mexico','(5) 555-3932','(5) 565-3532'),('1004',
'virtuasa','Thomas Hardy','Sales Representative','120 Hanover Sq.','London', NULL, 'WA1
1DP','UK','(171) 555-7788','(171) 555-6750'),('1005','genpact','Frdrique Citeaux','Marketing
Manager','24, place Klber','Strasbourg', NULL, '67000','France','88.60.15.31','88.60.15.32');
Query OK, 5 rows affected (0.07 sec)
Records: 5 Duplicates: 0 Warnings: 0
```

```
mysql> select * from Customers;
```

```
+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+
| CustomerID | CompanyName | ContactName | ContactTitle | Address |
City | Region | PostalCode | Country | Phone | Fax |
+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+
| 1001 | tcs | maria | sales representative | Obere Str. 57 | Berlin |
NULL | 12209 | Germany | 030-0074321 | 030-0076545 |
| 1002 | wipro | Ana | Owner | Avda. de la Constitucin 2222 | Mxico D.F.
| NULL | 05021 | Mexico | (5) 555-4729 | (5) 555-3745 |
| 1003 | polaris | Antonio | Owner | Mataderos 2312 | Mxico D.F. |
NULL | 05023 | Mexico | (5) 555-3932 | (5) 565-3532 |
| 1004 | virtuasa | Thomas Hardy | Sales Representative | 120 Hanover Sq. |
London | NULL | WA1 1DP | UK | (171) 555-7788 | (171) 555-6750 |
| 1005 | genpact | Frdrique Citeaux | Marketing Manager | 24, place Klber |
Strasbourg | NULL | 67000 | France | 88.60.15.31 | 88.60.15.32 |
+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+
5 rows in set (0.00 sec)
```

```
mysql> CREATE TABLE Employees ( EmployeeID INTEGER NOT NULL AUTO_INCREMENT,
LastName VARCHAR(20) NOT NULL,  FirstName VARCHAR(10) NOT NULL,  JobTitle
VARCHAR(30),  BirthDate DATETIME,  HireDate DATETIME,  Address VARCHAR(60),  City
VARCHAR(15),  PostalCode VARCHAR(10),  Country VARCHAR(15),  Phone VARCHAR(24),
ReportsTo INTEGER,  Salary FLOAT,  CONSTRAINT `PK_Employees` PRIMARY KEY
(`EmployeeID`));
Query OK, 0 rows affected (0.16 sec)
```

```
mysql> desc employees;
```

```
+-----+-----+-----+-----+-----+
| Field      | Type      | Null | Key | Default | Extra      |
+-----+-----+-----+-----+-----+
| EmployeeID | int(11)    | NO   | PRI | NULL     | auto_increment |
| LastName   | varchar(20)| NO   |     | NULL     |               |
| FirstName  | varchar(10)| NO   |     | NULL     |               |
| JobTitle   | varchar(30)| YES  |     | NULL     |               |
| BirthDate  | datetime   | YES  |     | NULL     |               |
| HireDate   | datetime   | YES  |     | NULL     |               |
| Address    | varchar(60)| YES  |     | NULL     |               |
| City       | varchar(15)| YES  |     | NULL     |               |
| PostalCode | varchar(10)| YES  |     | NULL     |               |
| Country    | varchar(15)| YES  |     | NULL     |               |
| Phone      | varchar(24)| YES  |     | NULL     |               |
| ReportsTo  | int(11)    | YES  |     | NULL     |               |
| Salary     | float      | YES  |     | NULL     |               |
+-----+-----+-----+-----+-----+
13 rows in set (0.01 sec)
```

```
mysql> insert into Employees(EmployeeID, LastName, FirstName, JobTitle, BirthDate, HireDate,
Address,City ,PostalCode, Country,Phone, ReportsTo, Salary ) values (5001, 'rao', 'krishna',
'Sales Representative', '1948-12-08','1992-05-01', '507 - 20th Ave. E.Apt. 2A','Seattle',
'98122','USA', '22255556',1234,30000.00);
Query OK, 1 row affected (0.16 sec)
```

```
mysql> insert into Employees(EmployeeID, LastName, FirstName, JobTitle, BirthDate, HireDate,
Address,City ,PostalCode, Country,Phone, ReportsTo, Salary ) values (5002, 'krishna', 'ram',
'Manager', '1950-10-06','1995-06-02', '307 - 15th Ave. E.Apt','Hyderabad', '500031','India',
'9849789648',1315,35000.00);
Query OK, 1 row affected (0.12 sec)
```

```
mysql> insert into Employees(EmployeeID, LastName, FirstName, JobTitle, BirthDate, HireDate,
Address,City ,PostalCode, Country,Phone, ReportsTo, Salary ) values (5003, 'hare', 'ram',
```



```
'Accountant', '1975-10-05','1997-09-01', '107 Apt', 'Secunderabad', '500031','India',
'9849755648',1415,45000.00);
Query OK, 1 row affected (0.05 sec)
```

```
mysql> insert into Employees(EmployeeID, LastName, FirstName, JobTitle, BirthDate, HireDate,
Address,City ,PostalCode, Country,Phone, ReportsTo, Salary ) values (5004, 'hare', 'krishna',
'Assistant Manager', '1980-08-06','2002-10-05', '407 Apt', 'Vijayawada', '520031','India',
'9888755648',2415,40000.00);

Query OK, 1 row affected (0.12 sec)
```

```
mysql> insert into Employees(EmployeeID, LastName, FirstName, JobTitle, BirthDate, HireDate,
Address,City ,PostalCode, Country,Phone, ReportsTo, Salary ) values (5005, 'murali', 'mohan', '
Manager', '1982-08-07','2007-06-01', '237 Apt', 'Hyderabad', '500031','India',
'9888695648',2695,45000.00);
Query OK, 1 row affected (0.05 sec)
```

```
mysql> select * from Employees;
```

EmployeeID	LastName	FirstName	JobTitle	BirthDate	HireDate	Address	City	PostalCode	Country	Phone	ReportsTo	Salary
5001	rao	krishna	Sales Representative	1948-12-08 00:00:00	1992-05-01 00:00:00	507 - 20th Ave. E.Apt. 2A	Seattle	98122	USA	22255556	1234	30000
5002	krishna	ram	Manager	1950-10-06 00:00:00	1995-06-02 00:00:00	307 - 15th Ave. E.Apt	Hyderabad	500031	India	9849789648	1315	35000
5003	hare	ram	Accountant	1975-10-05 00:00:00	1997-09-01 00:00:00	107 Apt	Secunderabad	500031	India	9849755648	1415	45000
5004	hare	krishna	Assistant Manager	1980-08-06 00:00:00	2002-10-05 00:00:00	407 Apt	Vijayawada	520031	India	9888755648	2415	40000
5005	murali	mohan	Manager	1982-08-07 00:00:00	2007-06-01 00:00:00	237 Apt	Hyderabad	500031	India	9888695648	2695	45000

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

```
mysql> CREATE TABLE Orders ( OrderID INTEGER NOT NULL AUTO_INCREMENT, CustomerID
VARCHAR(5), EmployeeID INTEGER, OrderDate DATETIME, RequiredDate DATETIME,
ShippedDate DATETIME, ShipId INTEGER, ShipName VARCHAR(40), ShipAddress
VARCHAR(60), CONSTRAINT `PK_Orders` PRIMARY KEY (`OrderID`));
Query OK, 0 rows affected (0.17 sec)
```

```
mysql> desc orders;
```

```
+-----+-----+-----+-----+-----+
| Field      | Type      | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
| OrderID    | int(11)   | NO   | PRI | NULL    | auto_increment |
| CustomerID | varchar(5) | YES  |     | NULL    |               |
| EmployeeID | int(11)   | YES  |     | NULL    |               |
| OrderDate  | datetime  | YES  |     | NULL    |               |
| RequiredDate | datetime  | YES  |     | NULL    |               |
| ShippedDate | datetime  | YES  |     | NULL    |               |
| ShipId     | int(11)   | YES  |     | NULL    |               |
| ShipName   | varchar(40) | YES  |     | NULL    |               |
| ShipAddress | varchar(60) | YES  |     | NULL    |               |
+-----+-----+-----+-----+-----+
```

```
9 rows in set (0.01 sec)
```

```
mysql> insert into Orders (OrderID, CustomerID, EmployeeID,OrderDate,RequiredDate,
ShippedDate,ShipId,ShipName,ShipAddress) values(9001, '1001',5001, '1996-07-04
00:00:00.000','1996-08-01 00:00:00.000','1996-07-16 00:00:00.000',3,'Reims','France');
Query OK, 1 row affected (0.07 sec)
```

```
mysql> insert into Orders values (9002, '1002',5002, '1996-07-05 00:00:00.000','1996-08-16
00:00:00.000','1996-07-10 00:00:00.000',1,'Mnster','Germany');
Query OK, 1 row affected (0.05 sec)
```

```
mysql> insert into Orders values (9003, '1003',5003, '1998-11-06 00:00:00.000','1998-12-04
00:00:00.000','1998-11-08 00:00:00.000',3,'SP','Brazil');
```

```
Query OK, 1 row affected (0.03 sec)
```

```
mysql> insert into Orders values (9004, '1004',5004, '2002-11-12 00:00:00.000','2002-11-26
00:00:00.000','2002-11-18 00:00:00.000',2,'Lisboa','Portugal');
```

```
Query OK, 1 row affected (0.12 sec)
```

```
mysql> insert into Orders values (9005, '1005',5005, '2008-11-25 00:00:00.000','2008-12-23
00:00:00.000','2008-11-28 00:00:00.000',4, 'Marseille','France');
```

Query OK, 1 row affected (0.03 sec)

```
mysql> select * from Orders;
```

OrderID	CustomerID	EmployeeID	OrderDate	RequiredDate	ShippedDate	ShipId	ShipName	ShipAddress
9001	1001	5001	1996-07-04 00:00:00	1996-08-01 00:00:00	1996-07-16 00:00:00	3	Reims	France
9002	1002	5002	1996-07-05 00:00:00	1996-08-16 00:00:00	1996-07-10 00:00:00	1	Mnster	Germany
9003	1003	5003	1998-11-06 00:00:00	1998-12-04 00:00:00	1998-11-08 00:00:00	3	SP	Brazil
9004	1004	5004	2002-11-12 00:00:00	2002-11-26 00:00:00	2002-11-18 00:00:00	2	Lisboa	Portugal
9005	1005	5005	2008-11-25 00:00:00	2008-12-23 00:00:00	2008-11-28 00:00:00	4	Marseille	France

5 rows in set (0.00 sec)

```
mysql> CREATE TABLE Categories ( CategoryID INTEGER NOT NULL AUTO_INCREMENT,  
CategoryName VARCHAR(15) NOT NULL, CONSTRAINT `PK_Categories` PRIMARY KEY  
(`CategoryID`));
```

Query OK, 0 rows affected (0.09 sec)

```
mysql> desc categories;
```

Field	Type	Null	Key	Default	Extra
CategoryID	int(11)	NO	PRI	NULL	auto_increment
CategoryName	varchar(15)	NO		NULL	

2 rows in set (0.01 sec)

```
mysql> insert into Categories(CategoryID,CategoryName) values  
(7001,'electronics'),(7002,'groceries'),(7003,'clothing'),(7004,'utensils'),(7005,'bags');  
Query OK, 5 rows affected (0.06 sec)  
Records: 5 Duplicates: 0 Warnings: 0
```

```
mysql> select * from Categories;
```

```
+-----+-----+
| CategoryID | CategoryName |
+-----+-----+
| 7001 | electronics |
| 7002 | groceries |
| 7003 | clothing |
| 7004 | utensils |
| 7005 | bags |
+-----+-----+
```

5 rows in set (0.00 sec)

```
mysql> CREATE TABLE Products ( ProductID INTEGER NOT NULL AUTO_INCREMENT,
ProductName VARCHAR(40) NOT NULL, SupplierID INTEGER, CategoryID INTEGER,
QuantityPerUnit VARCHAR(20), UnitPrice DECIMAL(10,4) DEFAULT 0, UnitsInStock
SMALLINT(2) DEFAULT 0, UnitsOnOrder SMALLINT(2) DEFAULT 0, ReorderLevel SMALLINT(2)
DEFAULT 0, Discontinued BIT NOT NULL DEFAULT 0, CONSTRAINT `PK_Products` PRIMARY
KEY (`ProductID`));
```

Query OK, 0 rows affected (0.05 sec)

```
mysql> desc products;
```

```
+-----+-----+-----+-----+-----+-----+
| Field      | Type      | Null | Key | Default | Extra      |
+-----+-----+-----+-----+-----+-----+
| ProductID  | int(11)   | NO   | PRI | NULL    | auto_increment |
| ProductName | varchar(40) | NO   |     | NULL    |               |
| SupplierID | int(11)   | YES  |     | NULL    |               |
| CategoryID | int(11)   | YES  |     | NULL    |               |
| QuantityPerUnit | varchar(20) | YES  |     | NULL    |               |
| UnitPrice  | decimal(10,4) | YES  |     | 0.0000  |               |
| UnitsInStock | smallint(2) | YES  |     | 0       |               |
| UnitsOnOrder | smallint(2) | YES  |     | 0       |               |
| ReorderLevel | smallint(2) | YES  |     | 0       |               |
| Discontinued | bit(1)    | NO   |     | b'0'    |               |
+-----+-----+-----+-----+-----+-----+
```

10 rows in set (0.01 sec)

```
mysql> insert into Products ( ProductID, ProductName , SupplierID , CategoryID,
QuantityPerUnit ,UnitPrice, UnitsInStock , UnitsOnOrder, ReorderLevel , Discontinued)
values (6001,'computer',2001,7001,'25 computers',25000.00,20,10,5,1),
(6002,'laptop',2002,7002,'20 laptops',45000.00,10,5,3,0), (6003,'pendrive',2003,7003,'35
pendrives',1000.00,30,10,5,0), (6004,'memory card',2004,7004,'45 memory
cards',500.00,50,20,10,1), (6005,'headphones',2005,7005,'20
headphones',1000.00,30,15,10,0);
Query OK, 5 rows affected (0.07 sec)
Records: 5 Duplicates: 0 Warnings: 0
```

```
mysql> select * from products;
```

ProductID	ProductName	SupplierID	CategoryID	QuantityPerUnit	UnitPrice	UnitsInStock	UnitsOnOrder	ReorderLevel	Discontinued
6001	computer	2001	7001	25 computers	25000.0000	20	10	5	1
6002	laptop	2002	7002	20 laptops	45000.0000	10	5	3	0
6003	pendrive	2003	7003	35 pendrives	1000.0000	30	10	5	0
6004	memory card	2004	7004	45 memory cards	500.0000	50	20	10	1
6005	headphones	2005	7005	20 headphones	1000.0000	30	15	10	0

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

```
mysql> CREATE TABLE Shippers( ShipperID INTEGER NOT NULL AUTO_INCREMENT,
CompanyName VARCHAR(40) NOT NULL, Phone VARCHAR(24), CONSTRAINT `PK_Shippers`
PRIMARY KEY (`ShipperID`));
Query OK, 0 rows affected (0.18 sec)
```

```
mysql> desc shippers;
```

Field	Type	Null	Key	Default	Extra
ShipperID	int(11)	NO	PRI	NULL	auto_increment
CompanyName	varchar(40)	NO		NULL	
Phone	varchar(24)	YES		NULL	

```
3 rows in set (0.01 sec)
```

```
mysql> insert into Shippers(ShipperID, CompanyName, Phone) values
(1,'samsung','900000001'),(2,'lg','6258215545'),(3,'redme','6454545454'),(4,'nokia','895633552'),
(5,'oppo','789658258');
```

Query OK, 5 rows affected (0.08 sec)
Records: 5 Duplicates: 0 Warnings: 0

```
mysql> select * from Shippers;
```

ShipperID	CompanyName	Phone
1	samsung	90000001
2	lg	6258215545
3	redme	6454545454
4	nokia	895633552
5	oppo	789658258

5 rows in set (0.00 sec)


```
mysql> CREATE TABLE Suppliers( SupplierID INTEGER NOT NULL AUTO_INCREMENT,
CompanyName VARCHAR(40) NOT NULL, ContactName VARCHAR(30), ContactJobTitle
VARCHAR(30), Address VARCHAR(60), Phone VARCHAR(24), CONSTRAINT `PK_Suppliers`
PRIMARY KEY (`SupplierID`));
```

Query OK, 0 rows affected (0.42 sec)

```
mysql> desc suppliers;
```

Field	Type	Null	Key	Default	Extra
SupplierID	int(11)	NO	PRI	NULL	auto_increment
CompanyName	varchar(40)	NO		NULL	
ContactName	varchar(30)	YES		NULL	
ContactJobTitle	varchar(30)	YES		NULL	
Address	varchar(60)	YES		NULL	
Phone	varchar(24)	YES		NULL	

6 rows in set (0.01 sec)

```
mysql> insert into Suppliers (SupplierID,CompanyName,ContactName,ContactJobTitle,Address,
Phone ) values (2001,'samsung','abc','Salesman','hyderabad','987456321'),
(2002,'lg','xyz','manager','delhi','879654123'),(2003,'redme','pqr','accountant','agra','6587492
34'),(2004,'nokia','uvw','assistant manager','vijayawada','896547213'),
(2005,'oppo','abcd','executive','secunderabad','985462317');
```

Query OK, 5 rows affected (0.05 sec)

Records: 5 Duplicates: 0 Warnings: 0

```
mysql> CREATE TABLE OrderDetails ( OrderID INTEGER NOT NULL, ProductID INTEGER NOT
NULL, UnitPrice DECIMAL(10,4) NOT NULL DEFAULT 0, Quantity SMALLINT(2) NOT NULL
DEFAULT 1, CONSTRAINT `PK_Order Details` PRIMARY KEY (`OrderID`, `ProductID`));
```

Query OK, 0 rows affected (0.07 sec)

```
mysql> desc orderdetails;
```

Field	Type	Null	Key	Default	Extra
OrderID	int(11)	NO	PRI	NULL	
ProductID	int(11)	NO	PRI	NULL	
UnitPrice	decimal(10,4)	NO		0.0000	
Quantity	smallint(2)	NO		1	

```
+-----+-----+-----+-----+
```

```
4 rows in set (0.01 sec)
```

```
mysql> insert into OrderDetails(OrderID ,ProductID,UnitPrice,Quantity)
```

```
values(9001,6001,20000.00,10),(9002,6002,25000.00,5),(9003,6003,30000.00,10),(9004,6004,23000.00,5),(9005,6005,25000.00,10);
```

```
Query OK, 5 rows affected (0.07 sec)
```

```
Records: 5 Duplicates: 0 Warnings: 0
```

```
mysql> select * from orderDetails;
```

OrderID	ProductID	UnitPrice	Quantity
9001	6001	20000.0000	10
9002	6002	25000.0000	5
9003	6003	30000.0000	10
9004	6004	23000.0000	5
9005	6005	25000.0000	10

```
+-----+-----+-----+-----+
```

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

```
mysql> select * from products;
```

```
+-----+-----+-----+-----+-----+-----+-----+-----+
-----+-----+
| ProductID | ProductName | SupplierID | CategoryID | QuantityPerUnit | UnitPrice |
UnitsInStock | UnitsOnOrder | ReorderLevel | Discontinued |
+-----+-----+-----+-----+-----+-----+-----+-----+
-----+-----+
| 6001 | computer | 2001 | 7001 | 25 computers | 25000.0000 | 20 |
10 | 5 | ☺ |
| 6002 | laptop | 2002 | 7002 | 20 laptops | 45000.0000 | 10 | 5 |
3 | |
| 6003 | pendrive | 2003 | 7003 | 35 pendrives | 1000.0000 | 30 | 10
| 5 | |
| 6004 | memory card | 2004 | 7004 | 45 memory cards | 500.0000 | 50 |
20 | 10 | ☺ |
| 6005 | headphones | 2005 | 7005 | 20 headphones | 1000.0000 | 30 |
15 | 10 | |
+-----+-----+-----+-----+-----+-----+-----+-----+
-----+-----+
5 rows in set (0.00 sec)
```

```
mysql> desc products;
```

```
+-----+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| ProductID | int(11) | NO | PRI | NULL | auto_increment |
| ProductName | varchar(40) | NO | | NULL | |
| SupplierID | int(11) | YES | | NULL | |
| CategoryID | int(11) | YES | | NULL | |
| QuantityPerUnit | varchar(20) | YES | | NULL | |
| UnitPrice | decimal(10,4) | YES | | 0.0000 | |
| UnitsInStock | smallint(2) | YES | | 0 | |
| UnitsOnOrder | smallint(2) | YES | | 0 | |
| ReorderLevel | smallint(2) | YES | | 0 | |
| Discontinued | bit(1) | NO | | b'0' | |
+-----+-----+-----+-----+-----+-----+
10 rows in set (0.00 sec)
```

Altering Tables:-

To alter a table use the alter table command:

Syntax1:

Alter table tablename add (columnname datatype, ...);

Syntax2:

Alter table tablename drop column columnname;

```
mysql> alter table products add(productno int);
```

```
Query OK, 5 rows affected (0.25 sec)
```

```
Records: 5 Duplicates: 0 Warnings: 0
```

```
mysql> select * from products;
```

```
+-----+-----+-----+-----+-----+-----+-----+-----+
-----+-----+-----+
| ProductID | ProductName | SupplierID | CategoryID | QuantityPerUnit | UnitPrice |
UnitsInStock | UnitsOnOrder | ReorderLevel | Discontinued | productno |
+-----+-----+-----+-----+-----+-----+-----+-----+
-----+-----+-----+
| 6001 | computer | 2001 | 7001 | 25 computers | 25000.0000 | 20 |
10 | 5 | ☺ | NULL |
| 6002 | laptop | 2002 | 7002 | 20 laptops | 45000.0000 | 10 | 5 |
3 | | NULL |
| 6003 | pendrive | 2003 | 7003 | 35 pendrives | 1000.0000 | 30 | 10
| 5 | | NULL |
| 6004 | memory card | 2004 | 7004 | 45 memory cards | 500.0000 | 50 |
20 | 10 | ☺ | NULL |
| 6005 | headphones | 2005 | 7005 | 20 headphones | 1000.0000 | 30 |
15 | 10 | | NULL |
+-----+-----+-----+-----+-----+-----+-----+-----+
-----+-----+-----+
5 rows in set (0.00 sec)
```

```
mysql> alter table products drop column productno;
```

```
Query OK, 5 rows affected (0.35 sec)
```

```
Records: 5 Duplicates: 0 Warnings: 0
```

```
mysql> select * from products;
```

```
+-----+-----+-----+-----+-----+-----+-----+-----+
-----+-----+
| ProductID | ProductName | SupplierID | CategoryID | QuantityPerUnit | UnitPrice |
UnitsInStock | UnitsOnOrder | ReorderLevel | Discontinued |
+-----+-----+-----+-----+-----+-----+-----+-----+
-----+-----+
| 6001 | computer | 2001 | 7001 | 25 computers | 25000.0000 | 20 |
10 | 5 | ☺ |
| 6002 | laptop | 2002 | 7002 | 20 laptops | 45000.0000 | 10 | 5 |
3 | |
| 6003 | pendrive | 2003 | 7003 | 35 pendrives | 1000.0000 | 30 | 10
| 5 | |
| 6004 | memory card | 2004 | 7004 | 45 memory cards | 500.0000 | 50 |
20 | 10 | ☺ |
| 6005 | headphones | 2005 | 7005 | 20 headphones | 1000.0000 | 30 |
15 | 10 | |
+-----+-----+-----+-----+-----+-----+-----+-----+
-----+-----+
5 rows in set (0.00 sec)
```

```
mysql> select ProductID,ProductName from Products;
```

```
+-----+-----+
| ProductID | ProductName |
+-----+-----+
| 6001 | computer |
| 6002 | laptop |
| 6003 | pendrive |
| 6004 | memory card |
| 6005 | headphones |
+-----+-----+
5 rows in set (0.00 sec)
```

```
mysql> select ProductID,ProductName from Products where unitprice<=20000;
```

```
+-----+-----+
| ProductID | ProductName |
+-----+-----+
| 6003 | pendrive |
| 6004 | memory card |
| 6005 | headphones |
+-----+-----+
3 rows in set (0.00 sec)
```

```
mysql> select ProductID,QuantityPerUnit from Products where UnitsInStock=30;
```

```
+-----+-----+  
| ProductID | QuantityPerUnit |  
+-----+-----+  
|    6003 | 35 pendrives   |  
|    6005 | 20 headphones  |  
+-----+-----+
```

```
2 rows in set (0.10 sec)
```

```
mysql> select ProductID,ProductName from Products WHERE ProductName LIKE 'P_%';
```

```
+-----+-----+  
| ProductID | ProductName |  
+-----+-----+  
|    6003 | pendrive   |  
+-----+-----+
```

```
1 row in set (0.02 sec)
```

Experiment 4

Querying

Practice queries on **Aggregate functions** like count, max , min ,avg ,sum Practice queries like nested queries/co-related queries using ANY, ALL, IN,Exists, NOT EXISTS, UNION, INTERSECT, groupby ,having etc.

Joins: Join , Left Outer Join, Right Outer Join, Self Join

The SQL ANY and ALL Operators

The ANY and ALL operators allow you to perform a comparison between a single column value and a range of other values.

The SQL ANY Operator

The ANY operator:

- returns a boolean value as a result
- returns TRUE if ANY of the subquery values meet the condition

ANY means that the condition will be true if the operation is true for any of the values in the range.

ANY Syntax

```
SELECT column_name(s) FROM table_name WHERE column_name operator ANY  
(SELECT column_name FROM table_name WHERE condition);
```

```
mysql> SELECT ProductName FROM Products WHERE ProductID = ANY (SELECT ProductID  
FROM OrderDetails WHERE Quantity = 10);
```

```
+-----+  
| ProductName |  
+-----+  
| computer   |  
| pendrive   |  
| headphones |
```

+-----+

3 rows in set (0.08 sec)

The SQL ALL Operator

The ALL operator:

- returns a boolean value as a result
- returns TRUE if ALL of the subquery values meet the condition
- is used with SELECT, WHERE and HAVING statements

ALL means that the condition will be true only if the operation is true for all values in the range.

ALL Syntax With SELECT

SELECT ALL *column_name(s)* FROM *table_name* WHERE *condition*;

ALL Syntax With WHERE or HAVING

SELECT *column_name(s)* FROM *table_name* WHERE *column_name* operator ALL
(SELECT *column_name* FROM *table_name* WHERE *condition*);

```
mysql> SELECT ProductName FROM Products WHERE ProductID = ANY  
(SELECT ProductID FROM OrderDetails WHERE Quantity > 1);
```

```
+-----+  
| ProductName |  
+-----+  
| computer   |  
| laptop     |  
| pendrive   |  
| memory card |  
| headphones |  
+-----+  
5 rows in set (0.00 sec)
```

IN, NOT IN

You can select from members of a set with IN (or NOT IN) operator. This is easier and clearer than the equivalent AND-OR expression.

```
mysql> select * from products;
```

ProductID	ProductName	SupplierID	CategoryID	QuantityPerUnit	UnitPrice	UnitsInStock	UnitsOnOrder	ReorderLevel	Discontinued
6001	computer	2001	7001	25 computers	25000.0000	20	10	5	☺
6002	laptop	2002	7002	20 laptops	45000.0000	10	5	3	
6003	pendrive	2003	7003	35 pendrives	1000.0000	30	10	5	
6004	memory card	2004	7004	45 memory cards	500.0000	50	20	10	☺
6005	headphones	2005	7005	20 headphones	1000.0000	30	15	10	

5 rows in set (0.02 sec)

```
mysql> select * from products where ProductName IN ('computer','laptop');
```

ProductID	ProductName	SupplierID	CategoryID	QuantityPerUnit	UnitPrice	UnitsInStock	UnitsOnOrder	ReorderLevel	Discontinued
6001	computer	2001	7001	25 computers	25000.0000	20	10	5	☺
6002	laptop	2002	7002	20 laptops	45000.0000	10	5	3	

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

2 rows in set (0.02 sec)

```
mysql> select * from products where ProductName NOT IN ('computer','laptop');
```

```
+-----+-----+-----+-----+-----+-----+-----+-----+
----+

| ProductID | ProductName | SupplierID | CategoryID | QuantityPerUnit | UnitPrice | UnitsInStock |
UnitsOnOrder | ReorderLevel | Discontinued |

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

| 6003 | pendrive | 2003 | 7003 | 35 pendrives | 1000.0000 | 30 | 10 | 5 |
|
| 6004 | memory card | 2004 | 7004 | 45 memory cards | 500.0000 | 50 | 20 | 10
| ☺ |
| 6005 | headphones | 2005 | 7005 | 20 headphones | 1000.0000 | 30 | 15 | 10
|
+-----+-----+-----+-----+-----+-----+-----+-----+
----+
```

3 rows in set (0.00 sec)

Exists:

The EXISTS condition in SQL is used to check whether the result of a correlated nested query is empty (contains no tuples) or not. The result of EXISTS is a boolean value True or False. It can be used in a SELECT, UPDATE, INSERT or DELETE statement.

Syntax:

SELECT column_name(s)**FROM** table_name **WHERE EXISTS** (**SELECT** column_name(s)
FROM table_name **WHERE** condition);

```
mysql> SELECT CompanyName FROM Customers WHERE EXISTS (SELECT *
FROM Orders WHERE Customers.CustomerID = Orders. CustomerID);
```

```
+-----+
| CompanyName |
+-----+
| tcs |
| wipro |
| polaris |
| virtuasa |
| genpact |
+-----+
```

5 rows in set (0.00 sec)

Using NOT with EXISTS

```
mysql> SELECT CompanyName FROM Customers WHERE NOT EXISTS (SELECT *  
FROM Orders WHERE Customers.CustomerID = Orders. CustomerID);  
Empty set (0.00 sec)
```

Differences between UNION and INTERSECT Operators:

1. The SET operators are mainly used to combine the result of more than 1 select statement and return a single result set to the user.
2. The set operators work on complete rows of the queries, so the results of the queries must have the same column name, same column order and the types of columns must be compatible.

There are the following 4 set operators in SQL Server:

1. **UNION**: Combine two or more result sets into a single set, without duplicates.
2. **UNION ALL**: Combine two or more result sets into a single set, including all duplicates.
3. **INTERSECT**: Takes the data from both result sets which are in common.

Rules on Set Operations:

1. The result sets of all queries must have the same number of columns.
2. In every result set the data type of each column must be compatible (well matched) to the data type of its corresponding column in other result sets.
3. In order to sort the result, an ORDER BY clause should be part of the last select statement. The column names or aliases must be found out by the first select statement.

Understand the differences between these operators with examples.

Use below SQL Script to create and populate the two tables that we are going to use in our examples.

```
mysql> create table colors_a(color_name varchar(20));
```

Query OK, 0 rows affected (0.17 sec)

```
mysql> create table colors_b(color_name varchar(20));
```

Query OK, 0 rows affected (0.11 sec)

```
mysql> insert into colors_a(color_name)values('red'),('green'),('orange'),('yellow'),('violet');
```

Query OK, 5 rows affected (0.09 sec)

Records: 5 Duplicates: 0 Warnings: 0

```
mysql> insert into colors_b(color_name)values('white'),('red'),('peach'),('orange');
```

Query OK, 4 rows affected (0.08 sec)

Records: 4 Duplicates: 0 Warnings: 0

```
mysql> select * from colors_a;
```

```
+-----+
| color_name |
+-----+
| red      |
| green    |
| orange   |
| yellow   |
| violet   |
+-----+
```

5 rows in set (0.00 sec)

```
mysql> select * from colors_b;
```

```
+-----+
| color_name |
+-----+
| white      |
| red        |
| peach      |
| orange     |
+-----+
```

4 rows in set (0.00 sec)

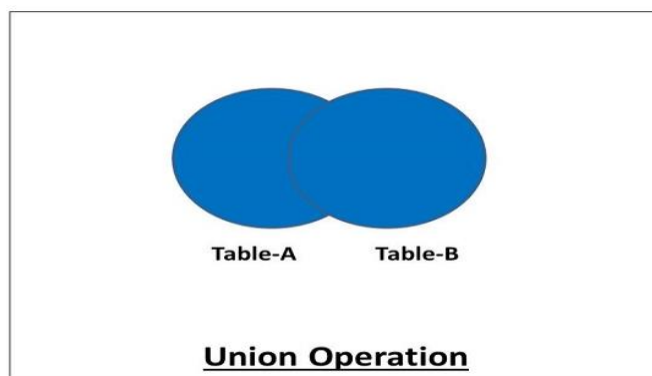
UNION

The Union is a binary set operator in DBMS. It is used to combine the result set of two select queries. Thus, It combines two result sets into one. In other words, the result set obtained after union operation is the collection of the result set of both the tables.

But two necessary conditions need to be fulfilled when we use the union command. These are:

1. Both SELECT statements should have an equal number of fields in the same order.
2. The data types of these fields should either be the same or compatible with each other.

The Union operation can be demonstrated as follows:



The syntax for the union operation is as follows:

```
SELECT (column_names) from table1 [WHERE condition] UNION SELECT (column_names) from table2 [WHERE condition];
```

The MySQL query for the union operation can be as follows:

```
mysql> select color_name from colors_a union select color_name from colors_b;
```

The returned values for the above query is as follows:

```
+-----+
| color_name |
+-----+
| red       |
```

```
| green  |
| orange |
| yellow |
| violet |
| white  |
| peach  |
```

```
+-----+
```

7 rows in set (0.00 sec)

The Union operation gives us distinct values. If we want to allow the duplicates in our result set, we'll have to use the '**Union-All**' operation.

Union All operation is also similar to the union operation. The only difference is that it allows duplicate values in the result set.

The syntax for the union all operation is as follows:

```
SELECT (column_names) from table1 [WHERE condition] UNION ALL SELECT (column_names) from table2
[WHERE condition];
```

The MySQL query for the union all operation can be as follows:

```
mysql> select color_name from colors_a union all select color_name from colors_b;
```

The returned values for the above query is as follows:

```
+-----+
| color_name |
+-----+
| red       |
| green     |
| orange    |
```

```
| yellow |
| violet |
| white  |
| red    |
| peach  |
| orange |
```

```
+-----+
```

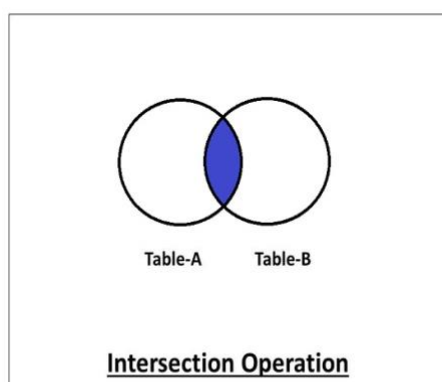
9 rows in set (0.00 sec)

INTERSECT

Intersect is a binary set operator in DBMS. The intersection operation between two selections returns only the common data sets or rows between them. It should be noted that the intersection operation always returns the distinct rows. The duplicate rows will not be returned by the intersect operator.

Here also, the above conditions of the union and minus are followed, i.e., the number of fields in both the SELECT statements should be the same, with the same data type, and in the same order for the intersection.

The intersection operation can be demonstrated as follows:



The syntax for the intersection operation is as follows:

```
SELECT (column_names) from table1[WHERE condition] INTERSECT SELECT (column_names) from
table2 [WHERE condition];
```

Note: It is to be noted that the intersect operator is not present in MySQL. But we can make use of 'IN' operator for performing an intersection operation in MySQL.

Here, we are using the 'IN' clause for demonstrating the examples.

The MySQL query for the intersection operation using the 'IN' operator can be as follows:

```
mysql> select color_name from colors_a where color_name in(select color_name from colors_b);
```

The returned values for the above query is as follows:

```
+-----+
```

```
| color_name |
```

```
+-----+
```

```
| red      |
```

```
| orange   |
```

```
+-----+
```

```
2 rows in set (0.00 sec)
```

BETWEEN, NOT BETWEEN

To check if the value is within a range, you could use BETWEEN ... AND ... operator. Again, this is easier and clearer than the equivalent AND-OR expression.

```
mysql> select * from Products where (CategoryId between 7001 and 7003) and (UnitsInStock between 10 and 30);
```

ProductID	ProductName	SupplierID	CategoryId	QuantityPerUnit	UnitPrice	UnitsInStock	UnitsOnOrder	ReorderLevel	Discontinued
6001	computer	2001	7001	25 computers	25000.0000	20	10	5	
6002	laptop	2002	7002	20 laptops	45000.0000	10	5	3	
6003	pendrive	2003	7003	35 pendrives	1000.0000	30	10	5	

3 rows in set (0.00 sec)

```
mysql> select * from Products where (CategoryId not between 7001 and 7003) and (UnitsInStock not between 10 and 30);
```

ProductID	ProductName	SupplierID	CategoryId	QuantityPerUnit	UnitPrice	UnitsInStock	UnitsOnOrder	ReorderLevel	Discontinued
6004	memory card	2004	7004	45 memory cards	500.0000	50	20	10	

1 row in set (0.00 sec)

IS NULL, IS NOT NULL

NULL is a special value, which represent "no value", "missing value" or "unknown value". You can checking if a column contains NULL by IS NULL or IS NOT NULL. For example,

```
mysql> select * from Products where ProductId is NULL;
```

Empty set (0.00 sec)

```
mysql> select * from Products where ProductId is NOT NULL;
```

```
+-----+-----+-----+-----+-----+-----+-----+-----+
-----+-----+
```

```
| ProductID | ProductName | SupplierID | CategoryID | QuantityPerUnit | UnitPrice |
UnitsInStock | UnitsOnOrder | ReorderLevel | Discontinued |
```

```
+-----+-----+-----+-----+-----+-----+-----+-----+
-----+-----+
```

```
| 6001 | computer | 2001 | 7001 | 25 computers | 25000.0000 | 20 |
10 | 5 | ☺ |
```

```
| 6002 | laptop | 2002 | 7002 | 20 laptops | 45000.0000 | 10 | 5 |
3 | |
```

```
| 6003 | pendrive | 2003 | 7003 | 35 pendrives | 1000.0000 | 30 | 10 |
5 | |
```

```
| 6004 | memory card | 2004 | 7004 | 45 memory cards | 500.0000 | 50 |
20 | 10 | ☺ |
```

```
| 6005 | headphones | 2005 | 7005 | 20 headphones | 1000.0000 | 30 |
15 | 10 | |
```

```
+-----+-----+-----+-----+-----+-----+-----+-----+
-----+-----+
```

5 rows in set (0.01 sec)

ORDER BY Clause

You can order the rows selected using ORDER BY clause, with the following syntax:

```
SELECT ... FROM tableName WHERE criteria ORDER BY columnA ASC|DESC, columnB ASC|DESC, ...
```

The selected row will be ordered according to the values in *columnA*, in either ascending (ASC) (default) or descending (DESC) order. If several rows have the same value in *columnA*, it will be ordered according to *columnB*, and so on. For strings, the ordering could be case-sensitive or case-insensitive, depending on the so-called character collating sequence used. For examples,

```
mysql> select * from Products where ProductName like 'l%' order by UnitPrice desc;
```

```
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| ProductID | ProductName | SupplierID | CategoryID | QuantityPerUnit | UnitPrice | UnitsInStock | UnitsOnOrder | ReorderLevel | Discontinued |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 6002 | laptop | 2002 | 7002 | 20 laptops | 45000.0000 | 10 | 5 | 3 | |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
1 row in set (0.00 sec)
```


You can randomize the returned records via function RAND(), e.g.,

```
mysql> SELECT * FROM products ORDER BY RAND();
```

```
+-----+-----+-----+-----+-----+-----+-----+-----+
----+

| ProductID | ProductName | SupplierID | CategoryID | QuantityPerUnit | UnitPrice | UnitsInStock | UnitsOnOrder | ReorderLevel | Discontinued |
+-----+-----+-----+-----+-----+-----+-----+-----+
----+

| 6001 | computer | 2001 | 7001 | 25 computers | 25000.0000 | 20 | 10 | 5 |
☺ |
| 6003 | pendrive | 2003 | 7003 | 35 pendrives | 1000.0000 | 30 | 10 | 5 |
|
| 6004 | memory card | 2004 | 7004 | 45 memory cards | 500.0000 | 50 | 20 | 10 |
☺ |
| 6005 | headphones | 2005 | 7005 | 20 headphones | 1000.0000 | 30 | 15 | 10 |
|
| 6002 | laptop | 2002 | 7002 | 20 laptops | 45000.0000 | 10 | 5 | 3 |
|
+-----+-----+-----+-----+-----+-----+-----+-----+
----+
```

5 rows in set (0.02 sec)

```
mysql> select distinct ProductId from Products;
```

```
+-----+
| ProductId |
+-----+
| 6001 |
| 6002 |
| 6003 |
| 6004 |
| 6005 |
```

+-----+

5 rows in set (0.01 sec)

GROUP BY Clause

The GROUP BY clause allows you to *collapse* multiple records with a common value into groups. For example,

```
mysql> select * from Products group by ProductID;
```

```
+-----+-----+-----+-----+-----+-----+-----+-----+
-----+-----+
```

```
| ProductID | ProductName | SupplierID | CategoryID | QuantityPerUnit | UnitPrice |
UnitsInStock | UnitsOnOrder | ReorderLevel | Discontinued |
```

```
+-----+-----+-----+-----+-----+-----+-----+-----+
-----+-----+
```

```
| 6001 | computer | 2001 | 7001 | 25 computers | 25000.0000 | 20 |
10 | 5 | ☺ |
```

```
| 6002 | laptop | 2002 | 7002 | 20 laptops | 45000.0000 | 10 | 5 |
3 | |
```

```
| 6003 | pendrive | 2003 | 7003 | 35 pendrives | 1000.0000 | 30 | 10 |
5 | |
```

```
| 6004 | memory card | 2004 | 7004 | 45 memory cards | 500.0000 | 50 |
20 | 10 | ☺ |
```

```
| 6005 | headphones | 2005 | 7005 | 20 headphones | 1000.0000 | 30 |
15 | 10 | |
```

```
+-----+-----+-----+-----+-----+-----+-----+-----+
-----+-----+
```

5 rows in set (0.00 sec)

GROUP BY Aggregate Functions: COUNT, MAX, MIN, AVG, SUM:

```
mysql> select count(*), max(UnitPrice),min(UnitPrice),  
sum(UnitPrice),avg(UnitPrice),STD(UnitPrice) from Products;
```

```
+-----+-----+-----+-----+-----+-----+  
| count(*) | max(UnitPrice) | min(UnitPrice) | sum(UnitPrice) | avg(UnitPrice) | STD(UnitPrice) |  
+-----+-----+-----+-----+-----+-----+  
|      5 | 45000.0000 | 500.0000 | 72500.0000 | 14500.00000000 | 17894.13311675 |
```

```
+-----+-----+-----+-----+-----+-----+
```

1 row in set (0.03 sec)

Join Operations:

In DBMS, a join statement is mainly used to combine two tables based on a specified common field between them. If we talk in terms of Relational algebra, it is the cartesian product of two tables followed by the selection operation. Thus, we can execute the product and selection process on two tables using a single join statement. We can use either 'on' or 'using' clause in MySQL to apply predicates to the join queries.

A Join can be broadly divided into two types:

1. **Inner Join**
2. **Outer Join**

For all the examples, we will consider the below-mentioned employee and department table.

```
mysql> create table employee (empId int,empName varchar(20),deptId int);
```

Query OK, 0 rows affected (0.16 sec)

```
mysql> insert into employee (empId,empName,deptId) values  
(1,'Harry',2),(2,'Tom',3),(3,'Joy',5),(4,'Roy',8);
```

Query OK, 4 rows affected (0.06 sec)

Records: 4 Duplicates: 0 Warnings: 0

```
mysql> create table department(deptId int, deptName varchar(20));
```

Query OK, 0 rows affected (0.14 sec)

```
mysql> insert into department(deptId,deptName) values (1,'CSE'),(2,'Mech'),(3,'IT');
```

Query OK, 3 rows affected (0.07 sec)

Records: 3 Duplicates: 0 Warnings: 0

```
mysql> select * from employee;
```

empId	empName	deptId
1	Harry	2
2	Tom	3
3	Joy	5
4	Roy	8

4 rows in set (0.05 sec)

```
mysql> select * from department;
```

deptId	deptName
--------	----------

	1	CSE	
--	---	-----	--

	2	Mech	
--	---	------	--

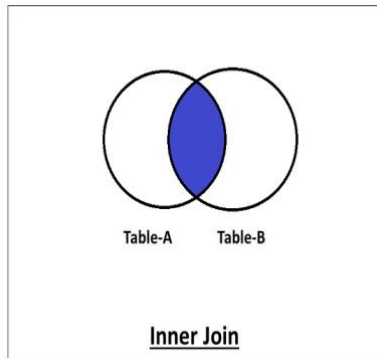
	3	IT	
--	---	----	--

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

3 rows in set (0.00 sec)

Inner Join

Inner Join is a join that can be used to return all the values that have matching values in both the tables. Inner Join can be depicted using the below diagram.



The inner join can be further divided into the following types:

1. **Equi Join**
2. **Natural Join**

1. Equi Join

Equi Join is an inner join that uses the equivalence condition for fetching the values of two tables.

Query:

```
mysql> Select employee.empId, employee.empName, department.deptName from employee  
inner join department on employee.deptId = department.deptId;
```

The returned values for the above query is as follows:

```
+-----+-----+-----+  
| empId | empName | deptName |  
+-----+-----+-----+  
| 1 | Harry | Mech |  
| 2 | Tom | IT |  
+-----+-----+-----+
```

2 rows in set (0.00 sec)

2. Natural Join

Natural Join is an inner join that returns the values of the two tables on the basis of a common attribute that has the same name and domain. It does not use any comparison operator. It also removes the duplicate attribute from the results.

Query:

```
mysql> select * from employee natural join department;
```

Output:

```
+-----+-----+-----+-----+
| deptId | empId | empName | deptName |
+-----+-----+-----+-----+
| 2 | 1 | Harry | Mech |
| 3 | 2 | Tom | IT |
+-----+-----+-----+-----+
```

2 rows in set (0.00 sec)

The above query will return the values of tables removing the duplicates.
If we want to specify the attribute names, the query will be as follows:

Query:

```
mysql> Select employee.empId, employee.empName, department.deptId,
department.deptName from employee natural join department;
```

Output:

```
+-----+-----+-----+-----+
| empId | empName | deptId | deptName |
+-----+-----+-----+-----+
| 1 | Harry | 2 | Mech |
| 2 | Tom | 3 | IT |
+-----+-----+-----+-----+
```

2 rows in set (0.00 sec)

Outer Join

Outer Join is a join that can be used to return the records in both the tables whether it has matching records in both the tables or not.

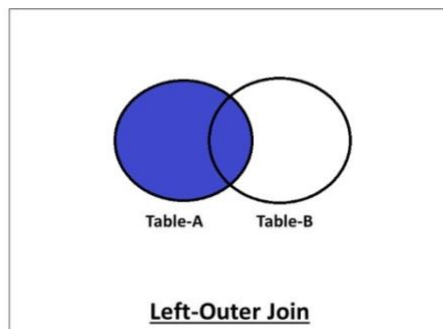
The outer join can be further divided into three types:

1. **Left-Outer Join**
2. **Right-Outer Join**
3. **Full-Outer Join**

1. Left-Outer Join:

The **Left-Outer Join** is an outer join that returns all the values of the left table, and the values of the right table that has matching values in the left table.

If there is no matching result in the right table, it will return null values in that field. The Left-Outer Join can be depicted using the below diagram.



Query:

```
mysql> select employee.empId, employee.empName, department.deptName from  
employee left outer join department on employee.deptId = department.deptId;
```

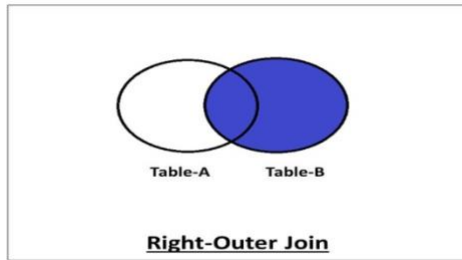
Output:

```
+-----+-----+-----+  
| empId | empName | deptName |  
+-----+-----+-----+  
|  1  | Harry  | Mech    |  
|  2  | Tom    | IT       |  
|  3  | Joy    | NULL     |  
|  4  | Roy    | NULL     |  
+-----+-----+-----+  
4 rows in set (0.06 sec)
```

2. Right-Outer Join:

The **Right-Outer Join** is an outer join that returns all the values of the right table, and the values of the left table that has matching values in the right table.

The Right-Outer Join can be depicted using the below diagram.



Query:

```
mysql> select employee.empId, employee.empName, department.deptName from employee
right outer join department on employee.deptId = department.deptId;
```

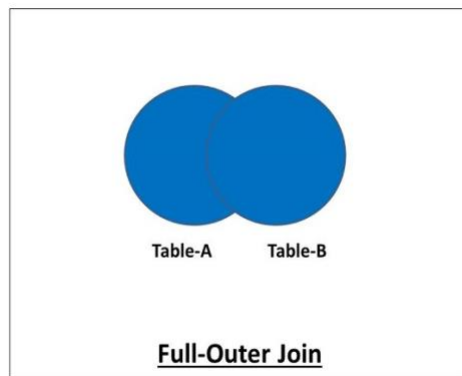
Output:

```
+-----+-----+-----+
| empId | empName | deptName |
+-----+-----+-----+
| NULL  | NULL    | CSE      |
| 1     | Harry   | Mech     |
| 2     | Tom     | IT       |
+-----+-----+-----+
3 rows in set (0.00 sec)
```

3. Full-Outer Join:

The Full-Outer join contains all the values of both the tables whether they have matching values in them or not.

The Full-Outer Join can be depicted using the below diagram.



Query:

```
mysql> select * from employee full join department;
```

Output:

```
+-----+-----+-----+-----+-----+
| empId | empName | deptId | deptId | deptName |
+-----+-----+-----+-----+-----+
| 1 | Harry | 2 | 1 | CSE |
| 1 | Harry | 2 | 2 | Mech |
| 1 | Harry | 2 | 3 | IT |
| 2 | Tom | 3 | 1 | CSE |
| 2 | Tom | 3 | 2 | Mech |
| 2 | Tom | 3 | 3 | IT |
| 3 | Joy | 5 | 1 | CSE |
| 3 | Joy | 5 | 2 | Mech |
| 3 | Joy | 5 | 3 | IT |
| 4 | Roy | 8 | 1 | CSE |
| 4 | Roy | 8 | 2 | Mech |
| 4 | Roy | 8 | 3 | IT |
+-----+-----+-----+-----+-----+
```

12 rows in set (0.00 sec)

Experiment 5

Querying(continued...)

Practice the following Queries :

Display all the order details of given a customer.

```
mysql> select * from Customers,Orders where  
Customers.CustomerID=orders.CustomerID;
```

```
+-----+-----+-----+-----+-----+-----+-----+-----+  
+-----+-----+-----+-----+-----+-----+-----+-----+  
+-----+-----+-----+-----+-----+-----+-----+-----+  
| CustomerID | CompanyName | ContactName | ContactTitle | Address |  
City | Region | PostalCode | Country | Phone | Fax | OrderID | CustomerID |  
EmployeeID | OrderDate | RequiredDate | ShippedDate | ShipId | ShipName |  
ShipAddress |  
+-----+-----+-----+-----+-----+-----+-----+-----+  
+-----+-----+-----+-----+-----+-----+-----+-----+  
| 1001 | tcs | maria | sales representative | Obere Str. 57 | Berlin |  
NULL | 12209 | Germany | 030-0074321 | 030-0076545 | 9001 | 1001 | 5001  
| 1996-07-04 00:00:00 | 1996-08-01 00:00:00 | 1996-07-16 00:00:00 | 3 | Reims | France  
|  
| 1002 | wipro | Ana | Owner | Avda. de la Constitucin 2222 | Mxico  
D.F. | NULL | 05021 | Mexico | (5) 555-4729 | (5) 555-3745 | 9002 | 1002 |  
5002 | 1996-07-05 00:00:00 | 1996-08-16 00:00:00 | 1996-07-10 00:00:00 | 1 | Mnster |  
Germany |  
| 1003 | polaris | Antonio | Owner | Mataderos 2312 | Mxico  
D.F. | NULL | 05023 | Mexico | (5) 555-3932 | (5) 565-3532 | 9003 | 1003 |  
5003 | 1998-11-06 00:00:00 | 1998-12-04 00:00:00 | 1998-11-08 00:00:00 | 3 | SP |  
Brazil |  
| 1004 | virtuasa | Thomas Hardy | Sales Representative | 120 Hanover Sq. |  
London | NULL | WA1 1DP | UK | (171) 555-7788 | (171) 555-6750 | 9004 | 1004  
| 5004 | 2002-11-12 00:00:00 | 2002-11-26 00:00:00 | 2002-11-18 00:00:00 | 2 | Lisboa  
| Portugal |  
| 1005 | genpact | Frdrique Citeaux | Marketing Manager | 24, place Klber |  
Strasbourg | NULL | 67000 | France | 88.60.15.31 | 88.60.15.32 | 9005 | 1005 |  
5005 | 2008-11-25 00:00:00 | 2008-12-23 00:00:00 | 2008-11-28 00:00:00 | 4 | Marseille |  
France |  
+-----+-----+-----+-----+-----+-----+-----+-----+  
+-----+-----+-----+-----+-----+-----+-----+-----+  
+-----+-----+-----+-----+-----+-----+-----+-----+  
5 rows in set (0.00 sec)
```

- **Display all the products.**

```
mysql> select * from Products;
```

```

+-----+-----+-----+-----+-----+-----+-----+
| ProductID | ProductName | SupplierID | CategoryID | QuantityPerUnit | UnitPrice | UnitsInStock | UnitsOnOrder | ReorderLevel | Discontinued |
+-----+-----+-----+-----+-----+-----+-----+
| 6001 | computer | 2001 | 7001 | 25 computers | 25000.0000 | 20 | 10 | 5 | ☺ |
| 6002 | laptop | 2002 | 7002 | 20 laptops | 45000.0000 | 10 | 3 | 3 | |
| 6003 | pendrive | 2003 | 7003 | 35 pendrives | 1000.0000 | 30 | 10 | 5 | |
| 6004 | memory card | 2004 | 7004 | 45 memory cards | 500.0000 | 50 | 20 | 10 | ☺ |
| 6005 | headphones | 2005 | 7005 | 20 headphones | 1000.0000 | 30 | 15 | 10 | |
+-----+-----+-----+-----+-----+-----+-----+
5 rows in set (0.00 sec)
```


- **Get the highest sold product from given supplierID**

```
mysql> select max(UnitsInStock) from Products;
```

```
+-----+  
| max(UnitsInStock) |
```

```
+-----+  
|          50 |
```

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

- List all products grouped by category

```
mysql> select * from Products group by ProductID;
```

```
+-----+-----+-----+-----+-----+-----+-----+-----+
-----+-----+
```

```
| ProductID | ProductName | SupplierID | CategoryID | QuantityPerUnit | UnitPrice |
UnitsInStock | UnitsOnOrder | ReorderLevel | Discontinued |
```

```
+-----+-----+-----+-----+-----+-----+-----+-----+
-----+-----+
```

```
| 6001 | computer | 2001 | 7001 | 25 computers | 25000.0000 | 20 |
10 | 5 | ☺ |
```

```
| 6002 | laptop | 2002 | 7002 | 20 laptops | 45000.0000 | 10 | 5 |
3 | |
```

```
| 6003 | pendrive | 2003 | 7003 | 35 pendrives | 1000.0000 | 30 | 10 |
5 | |
```

```
| 6004 | memory card | 2004 | 7004 | 45 memory cards | 500.0000 | 50 |
20 | 10 | ☺ |
```

```
| 6005 | headphones | 2005 | 7005 | 20 headphones | 1000.0000 | 30 |
15 | 10 | |
```

```
+-----+-----+-----+-----+-----+-----+-----+-----+
-----+-----+
```

5 rows in set (0.00 sec)

- **List the products , whose products unit price is greater then all the products of average.**

```
mysql> SELECT PRODUCTID, ProductName, UnitPrice FROM Products WHERE  
UnitPrice IN (SELECT UnitPrice > AVG(UnitPrice) FROM Products);  
Empty set (0.00 sec)
```

- List Details of order and customer of each order

```
mysql> SELECT * FROM Customers JOIN Orders ON Customers.CustomerID = Orders.CustomerID;
```

CustomerID	CompanyName	ContactName	ContactTitle	Address	City	Region	PostalCode	Country	Phone	Fax	OrderID	CustomerID	EmployeeID	OrderDate	RequiredDate	ShippedDate	ShipId	ShipName	ShipAddress
1001	tcs	maria	sales representative	Obere Str. 57	Berlin			Germany	030-0074321	030-0076545	9001	1001		1996-07-04 00:00:00	1996-08-01 00:00:00	1996-07-16 00:00:00	3	Reims	France
1002	wipro	Ana	Owner	Avda. de la Constitucion 2222	Mxico D.F.			Mexico	(5) 555-4729	(5) 555-3745	9002	1002		1996-07-05 00:00:00	1996-08-16 00:00:00	1996-07-10 00:00:00	1	Mnster	Germany
1003	polaris	Antonio	Owner	Mataderos 2312	Mxico D.F.			Mexico	(5) 555-3932	(5) 565-3532	9003	1003		1998-11-06 00:00:00	1998-12-04 00:00:00	1998-11-08 00:00:00	3	SP	Brazil
1004	virtuasa	Thomas Hardy	Sales Representative	120 Hanover Sq.	London			UK	(171) 555-7788	(171) 555-6750	9004	1004		2002-11-12 00:00:00	2002-11-26 00:00:00	2002-11-18 00:00:00	2	Lisboa	Portugal
1005	genpact	Frdrique Citeaux	Marketing Manager	24, place Klber	Strasbourg			France	88.60.15.31	88.60.15.32	9005	1005		2008-11-25 00:00:00	2008-12-23 00:00:00	2008-11-28 00:00:00	4	Marseille	France

5 rows in set (0.00 sec)

List the products which were sold in year 1997 Display categoryName and productName

```
mysql> SELECT  Categories.CategoryName,
Products.ProductName,
Sum(ROUND(OrderDetails.UnitPrice*Quantity)) AS ProductSales
FROM Categories
JOIN Products On Categories.CategoryID = Products.CategoryID
JOIN OrderDetails on Products.ProductID = OrderDetails.ProductID
JOIN `Orders` on Orders.OrderID = OrderDetails.OrderID
WHERE Orders.ShippedDate BETWEEN '1997-01-01' And '1997-12-31'
GROUP BY Categories.CategoryName, Products.ProductName;
```

Empty set (0.00 sec)

```
mysql> SELECT  Categories.CategoryName,  Products.ProductName,
Sum(ROUND(OrderDetails.UnitPrice*Quantity)) AS ProductSales FROM Categories
JOIN Products On Categories.CategoryID = Products.CategoryID
JOIN OrderDetails on Products.ProductID = OrderDetails.ProductID
JOIN `Orders` on Orders.OrderID = OrderDetails.OrderID
WHERE Orders.ShippedDate BETWEEN '1997-01-01' And '2000-12-31'
GROUP BY Categories.CategoryName, Products.ProductName;
```

```
+-----+-----+-----+
| CategoryName | ProductName | ProductSales |
+-----+-----+-----+
| clothing    | pendrive   | 300000      |
+-----+-----+-----+
1 row in set (0.00 sec)
```

- **Display the total amount for each order**

```
mysql> SELECT OrderDetails.OrderID, Sum(ROUND(OrderDetails.UnitPrice*Quantity)) AS Subtotal
FROM OrderDetails GROUP BY OrderDetails.OrderID;
```

```
+-----+-----+
| OrderID | Subtotal |
+-----+-----+
| 9001 | 200000 |
| 9002 | 125000 |
| 9003 | 300000 |
| 9004 | 115000 |
| 9005 | 250000 |
+-----+-----+
5 rows in set (0.00 sec)
```

- **Display Order Details for given an orderID**
Order Detials: productname and unitprice for given orderID

```
mysql> select a.ProductName,b.UnitPrice from Products a,OrderDetails b where b.ProductID=a.ProductID and
b.OrderID=9003;
```

```
+-----+-----+
| ProductName | UnitPrice |
+-----+-----+
| pendrive | 30000.0000 |
+-----+-----+
1 row in set (0.00 sec)
```

Experiment 6

Stored Procedures

Create a Procedure to display order details of given customerID like ordered, orderDate , RequiredDate, ShippedDate

```
mysql> DELIMITER ;
mysql> DELIMITER $$
mysql> CREATE PROCEDURE CustOrders(IN AtCustomerID VARCHAR(5))
BEGIN
SELECT
OrderID,
OrderDate,
RequiredDate,
ShippedDate
FROM Orders
WHERE CustomerID = AtCustomerID
ORDER BY OrderID;
END $$
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> DELIMITER ;
```

```
mysql> call CustOrders(1003);
```

OrderID	OrderDate	RequiredDate	ShippedDate
9003	1998-11-06 00:00:00	1998-12-04 00:00:00	1998-11-08 00:00:00

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

```
Query OK, 0 rows affected (0.01 sec)
```

Create a procedure to accept a customerID and display the customer order history(productname and how much quantity ordered for that particular product)

Ex: productname , Total_quantity he/she ordered.

```
mysql> DELIMITER $$
mysql> CREATE PROCEDURE CustOrderHist(IN AtCustomerID VARCHAR(5))
  BEGIN
  SELECT
  ProductName,
  SUM(Quantity) as TOTAL
  FROM Products
  INNER JOIN OrderDetails USING(ProductID)
  INNER JOIN Orders USING (OrderID)
  INNER JOIN Customers USING (CustomerID)
  WHERE Customers.CustomerID = AtCustomerID
  GROUP BY ProductName;
  END $$
Query OK, 0 rows affected (0.04 sec)
```

```
mysql> DELIMITER ;
```

```
mysql> call CustOrderHist(1003);
```

```
+-----+-----+
| ProductName | TOTAL |
+-----+-----+
| pendrive   |    10 |
+-----+-----+
1 row in set (0.02 sec)
```

```
Query OK, 0 rows affected (0.02 sec)
```

```
mysql> call CustOrderHist(1002);
```

```
+-----+-----+
| ProductName | TOTAL |
+-----+-----+
| laptop      |     5 |
+-----+-----+
1 row in set (0.00 sec)
```

```
Query OK, 0 rows affected (0.00 sec)
```


- **Create a procedure to display Ten Most Expensive Products**
Columns should be displayed Productname & Unit price

```
mysql> DROP PROCEDURE IF EXISTS TenMostExpensiveProducts;
Query OK, 0 rows affected, 1 warning (0.05 sec)
```

```
mysql> DELIMITER $$
mysql> CREATE PROCEDURE TenMostExpensiveProducts()
BEGIN
SELECT
Products.ProductName AS TenMostExpensiveProducts, Products.UnitPrice FROM Products
ORDER BY Products.UnitPrice DESC LIMIT 10;
END $$
Query OK, 0 rows affected (0.02 sec)
```

```
mysql> DELIMITER ;
```

```
mysql> CALL TenMostExpensiveProducts ();
```

```
+-----+-----+
| TenMostExpensiveProducts | UnitPrice |
+-----+-----+
| laptop                   | 45000.0000 |
| computer                 | 25000.0000 |
| pendrive                 | 1000.0000  |
| headphones               | 1000.0000  |
| memory card              | 500.0000   |
+-----+-----+
```

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

```
Query OK, 0 rows affected (0.01 sec)
```

Experiment 7

Views

Create a view to display the current product list which are available(not discontinued)

```
mysql> CREATE VIEW CurrentProductList AS SELECT ProductID, ProductName FROM Products
WHERE Discontinued = 0;
Query OK, 0 rows affected (0.05 sec)
```

```
mysql> select * from CurrentProductList;
```

```
+-----+-----+
| ProductID | ProductName |
+-----+-----+
| 6002 | laptop |
| 6003 | pendrive |
| 6005 | headphones |
+-----+-----+
3 rows in set (0.00 sec)
```

Create a view to display the products by category

Display productname, quantityPerUnit,unitsInStock,Discontinued

```
mysql> CREATE VIEW ProductsbyCategory AS SELECT Categories.CategoryName,
Products.ProductName, Products.QuantityPerUnit, Products.UnitsInStock, Products.Discontinued
FROM Categories INNER JOIN Products ON Categories.CategoryID = Products.CategoryID
WHERE Products.Discontinued = 0;
Query OK, 0 rows affected (0.05 sec)
```

```
mysql> select * from ProductsbyCategory;
```

```
+-----+-----+-----+-----+-----+
| CategoryName | ProductName | QuantityPerUnit | UnitsInStock | Discontinued |
+-----+-----+-----+-----+-----+
| groceries | laptop | 20 laptops | 10 | 0 |
| clothing | pendrive | 35 pendrives | 30 | 0 |
| bags | headphones | 20 headphones | 30 | 0 |
+-----+-----+-----+-----+-----+
3 rows in set (0.02 sec)
```

Create a view as “Invoices” to display all the information from order, customer, shipper for each OrderDetails

- All information (order, customer, shipper)
- for each `Order Details` line.
- An invoice is supposed to be per order?!

```
mysql> CREATE VIEW Invoices AS SELECT Orders.OrderID, Orders.CustomerID,
Orders.EmployeeID, Orders.OrderDate, Orders.RequiredDate, Orders.ShippedDate, Orders.ShipId,
Orders.ShipName, Orders.ShipAddress, Customers.CompanyName, Customers.Address, Customers.City,
Customers.Region, Customers.PostalCode, Customers.Country FROM Customers INNER JOIN Orders
ON Customers.CustomerID = Orders.CustomerID;
Query OK, 0 rows affected (0.07 sec)
```

```
mysql> select * from Invoices;
```

```
+-----+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+-----+
| OrderID | CustomerID | EmployeeID | OrderDate      | RequiredDate    | ShippedDate      | ShipId
| ShipName | ShipAddress | CompanyName | Address          | City            | Region | PostalCode |
Country |
+-----+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+-----+
| 9001 | 1001 | 5001 | 1996-07-04 00:00:00 | 1996-08-01 00:00:00 | 1996-07-16 00:00:00 |
3 | Reims | France | tcs | Obere Str. 57 | Berlin | NULL | 12209 | Germany |
| 9002 | 1002 | 5002 | 1996-07-05 00:00:00 | 1996-08-16 00:00:00 | 1996-07-10 00:00:00 |
1 | Mnster | Germany | wipro | Avda. de la Constitucin 2222 | Mxico D.F. | NULL | 05021 |
Mexico |
| 9003 | 1003 | 5003 | 1998-11-06 00:00:00 | 1998-12-04 00:00:00 | 1998-11-08 00:00:00 |
3 | SP | Brazil | polaris | Mataderos 2312 | Mxico D.F. | NULL | 05023 | Mexico |
| 9004 | 1004 | 5004 | 2002-11-12 00:00:00 | 2002-11-26 00:00:00 | 2002-11-18 00:00:00 |
2 | Lisboa | Portugal | virtuasa | 120 Hanover Sq. | London | NULL | WA1 1DP | UK |
| 9005 | 1005 | 5005 | 2008-11-25 00:00:00 | 2008-12-23 00:00:00 | 2008-11-28 00:00:00 |
4 | Marseille | France | genpact | 24, place Klber | Strasbourg | NULL | 67000 | France
|
+-----+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+-----+
5 rows in set (0.00 sec)
```

Experiment 8

Triggers

Demonstrate Create Trigger, Alter Trigger, Drop Trigger, Row Level , Table Level triggers, Before Insert ,AfterInsert, Before Update, After Update, Before Delete, After Delete

Trigger in DBMS

Triggers are the SQL statements that are **automatically executed** when there is any change in the database. The triggers are executed **in response to certain events**(INSERT, UPDATE or DELETE) in a particular table. These triggers help in maintaining the integrity of the data by changing the data of the database in a systematic fashion.

A trigger is called a special procedure because it cannot be called directly like a stored procedure. The main difference between the trigger and procedure is that a trigger is called automatically when a data modification event is made against a table. In contrast, a stored procedure must be called explicitly.

Generally, **triggers are of two types** according to the [SQL](#) standard: row-level triggers and statement-level triggers.

Row-Level Trigger: It is a trigger, which is activated for each row by a triggering statement such as insert, update, or delete. For example, if a table has inserted, updated, or deleted multiple rows, the row trigger is fired automatically for each row affected by the [insert](#), [update](#), or [delete statement](#).

Statement-Level Trigger: It is a trigger, which is fired once for each event that occurs on a table regardless of how many rows are inserted, updated, or deleted.

Types of Triggers in MySQL:

We can define the maximum six types of actions or events in the form of triggers:

1. [Before Insert](#): It is activated before the insertion of data into the table.
2. [After Insert](#): It is activated after the insertion of data into the table.
3. [Before Update](#): It is activated before the update of data in the table.
4. [After Update](#): It is activated after the update of the data in the table.
5. [Before Delete](#): It is activated before the data is removed from the table.
6. [After Delete](#): It is activated after the deletion of data from the table.

When we use a statement that does not use INSERT, UPDATE or DELETE query to change the data in a table, the triggers associated with the trigger will not be invoked.

Create Trigger:

Syntax

create trigger **Trigger_name** (before | after) [insert | update | delete] on [table_name] [**for each row**] [trigger_body];

1. **CREATE TRIGGER:** These two keywords specify that a triggered block is going to be declared.
2. **TRIGGER_NAME:** It creates or replaces an existing trigger with the Trigger_name. The trigger name should be unique.
3. **BEFORE | AFTER:** It specifies when the trigger will be initiated i.e. before the ongoing event or after the ongoing event.
4. **INSERT | UPDATE | DELETE:** These are the [DML operations](#) and we can use either of them in a given trigger.
5. **ON[TABLE_NAME]:** It specifies the name of the table on which the trigger is going to be applied.
6. **FOR EACH ROW:** Row-level trigger gets executed when any row value of any column changes.
7. **TRIGGER BODY:** It consists of queries that need to be executed when the trigger is called.

CREATE [OR REPLACE]

```
CREATE [OR REPLACE ] TRIGGER trigger_name {BEFORE | AFTER | INSTEAD OF }  
{INSERT [OR] | UPDATE [OR] | DELETE} [OF col_name] ON table_name  
[REFERENCING OLD AS o NEW AS n] [FOR EACH ROW] WHEN (condition)  
  
DECLARE Declaration-statements BEGIN Executable-statements  
  
EXCEPTION Exception-handling-statements END;
```

Drop Trigger:

To drop trigger, use DROP command.

The syntax is as follows –

DROP TRIGGER IF EXISTS TriggerName;

To understand the above syntax, you need to have a trigger in your current database.

To check the trigger is present or not, you can use below query. We have a trigger in our database –

```
mysql> create table student(student_id int,name varchar(20), address varchar(20),marks int);
Query OK, 0 rows affected (0.12 sec)
```

```
mysql> insert into student(student_id,name,address,marks) values
(1,'billie','ny',220),(2,'eilish','london',190),(3,'ariana','miami',180);
Query OK, 3 rows affected (0.05 sec)
Records: 3 Duplicates: 0 Warnings: 0
```

```
mysql> select * from student;
+-----+-----+-----+-----+
| student_id | name | address | marks |
+-----+-----+-----+-----+
| 1 | billie | ny | 220 |
| 2 | eilish | london | 190 |
| 3 | ariana | miami | 180 |
+-----+-----+-----+-----+
3 rows in set (0.00 sec)
```

```
mysql> create or replace trigger add_marks before insert on student for each row set
new.marks=new.marks+100;
Query OK, 0 rows affected (0.20 sec)
```

```
mysql> select * from student;
+-----+-----+-----+-----+
| student_id | name | address | marks |
+-----+-----+-----+-----+
| 1 | billie | ny | 220 |
| 2 | eilish | london | 190 |
| 3 | ariana | miami | 180 |
| 4 | alizeh | maldives | 210 |
+-----+-----+-----+-----+
4 rows in set (0.00 sec)
```

```
mysql> show triggers;
```

The following is the output –

```
mysql> show triggers;
```

Trigger	Event	Table	Statement	Timing	Created	sql_mode	Definer	character_set_client	collation_connection	Database Collation
add_marks	INSERT	student	set new.marks=new.marks+100	BEFORE	NULL	STRICT_TRANS_TABLES,NO_AUTO_CREATE_USER,NO_ENGINE_SUBSTITUTION	root@localhost	latin1	latin1_swedish_ci	latin1_swedish_ci

```
1 row in set (0.01 sec)
```

```
mysql> drop trigger if exists add_marks;
```

```
Query OK, 0 rows affected (0.01 sec)
```

Use the show triggers command to check whether the trigger is present or not. The query is as follows –

```
mysql> show triggers;
```

```
Empty set (0.00 sec)
```

Look at the above result now, the trigger is not present the database. We removed it by using drop.

Types of Triggers in MySQL:

Examples:

```
mysql> create table Students ( sid CHAR(20) , name char (30) , login char(20) , age int,  
    gpa REAL, UNIQUE (name, age), UNIQUE(login), CONSTRAINT StudentsKey  
PRIMARY KEY (sid) );
```

Query OK, 0 rows affected (0.19 sec)

```
mysql> create table Enrolled (sid CHAR(20), cid CHAR(20), grade CHAR(2), PRIMARY  
KEY (sid,cid), FOREIGN KEY (sid) REFERENCES Students(sid) );
```

Query OK, 0 rows affected (0.10 sec)

```
mysql> INSERT INTO Students (sid, name,login,age,gpa) VALUES  
(53666,'Jones','jones@cs',18,3.4);
```

Query OK, 1 row affected (0.10 sec)

```
mysql> INSERT INTO Students (sid, name,login,age,gpa) VALUES  
(53688,'Smith','Smith@ee',18,3.2);
```

Query OK, 1 row affected (0.02 sec)

```
mysql> INSERT INTO Students (sid, name,login,age,gpa) VALUES  
(53667,'Raj','Raj@cs',20,4.4);
```

Query OK, 1 row affected (0.05 sec)

```
mysql> INSERT INTO Students (sid, name,login,age,gpa) VALUES (53650, 'Smith',  
'smith@math',19,3.8);
```

Query OK, 1 row affected (0.03 sec)


```
mysql> select * from students;
```

```
+-----+-----+-----+-----+-----+
| sid  | name  | login   | age  | gpa  |
+-----+-----+-----+-----+-----+
| 53650 | Smith | smith@math | 19 | 3.8 |
| 53666 | Jones | jones@cs   | 18 | 3.4 |
| 53667 | Raj   | Raj@cs     | 20 | 4.4 |
| 53688 | Smith | Smith@ee   | 18 | 3.2 |
+-----+-----+-----+-----+-----+
```

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

```
mysql> INSERT INTO Enrolled(sid, cid, grade) VALUES (53666, 'History105', 'B');
```

```
Query OK, 1 row affected (0.05 sec)
```

```
mysql> INSERT INTO Enrolled(sid, cid, grade) VALUES (53650, 'Topology112', 'A');
```

```
Query OK, 1 row affected (0.04 sec)
```

```
mysql> INSERT INTO Enrolled(sid, cid, grade) VALUES (53688, 'Reggae203', 'B');
```

```
Query OK, 1 row affected (0.04 sec)
```

```
mysql> INSERT INTO Enrolled(sid, cid, grade) VALUES (53688, 'Carnatic101', 'C');
```

```
Query OK, 1 row affected (0.05 sec)
```

```
mysql> select * from Enrolled;
```

```
+-----+-----+-----+
| sid  | cid      | grade |
+-----+-----+-----+
| 53650 | Topology112 | A    |
| 53666 | History105  | B    |
| 53688 | Carnatic101 | C    |
| 53688 | Reggae203   | B    |
+-----+-----+-----+
```

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

Before insert triggers

```
mysql> delimiter //
```

```
mysql> Create Trigger before_insert_students BEFORE INSERT ON Students FOR EACH
ROW
```

```
begin
if new.age<18 then set new.age=18;
end if;
end//
```

```
Query OK, 0 rows affected (0.08 sec)
```

```
mysql> insert into Students values(55555,'xq','xq@we',15,3.5);
```

```
Query OK, 1 row affected (0.10 sec)
```

```
mysql> select * from Students;
```

```
+-----+-----+-----+-----+
| sid  | name  | login   | age  | gpa  |
+-----+-----+-----+-----+
| 53650 | Smith | smith@math | 19 | 3.8 |
| 53666 | Jones | jones@cs   | 18 | 3.4 |
| 53667 | Raj   | Raj@cs    | 20 | 4.4 |
| 53688 | Smith | Smith@ee   | 18 | 3.2 |
| 55555 | xq    | xq@we     | 18 | 3.5 |
+-----+-----+-----+-----+
```

5 rows in set (0.00 sec)

After insert triggers

L*

Write a query to invoke trigger after inserting a row in students table such that the newly inserted sid in students table should even reflect in enrolled table as shown in output.

***L**

```
mysql> drop trigger if exists after_insert_details;
```

Query OK, 0 rows affected (1.42 sec)

```
mysql> DELIMITER //
```

```
mysql> Create Trigger after_insert_details
```

```
    AFTER INSERT ON Students FOR EACH ROW
```

```
    BEGIN
```

```
    INSERT INTO Enrolled VALUES (new.sid, 'History', 'A');
```

```
    END //
```

Query OK, 0 rows affected (0.25 sec)

```
mysql> insert into Students values(55556, 'xq1', 'xq1@we',15,3.5);
```

Query OK, 1 row affected (0.40 sec)

```
mysql> select * from Students;
```

```
+-----+-----+-----+-----+-----+
| sid  | name  | login   | age  | gpa  |
+-----+-----+-----+-----+-----+
| 53650 | Smith | smith@math | 19 | 3.8 |
| 53666 | Jones | jones@cs   | 18 | 3.4 |
| 53667 | Raj   | Raj@cs     | 20 | 4.4 |
| 53688 | Smith | Smith@ee   | 18 | 3.2 |
| 55555 | xq    | xq@we      | 18 | 3.5 |
| 55556 | xq1   | xq1@we     | 18 | 3.5 |
+-----+-----+-----+-----+-----+
```

6 rows in set (0.00 sec)

```
mysql> select * from Enrolled;
```

```
+-----+-----+-----+
| sid  | cid      | grade |
+-----+-----+-----+
| 53650 | Topology112 | A    |
| 53666 | History105  | B    |
| 53688 | Carnatic101 | C    |
| 53688 | Reggae203   | B    |
| 55556 | History     | A    |
+-----+-----+-----+
```

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

Before update trigger

/* Write a query to invoke trigger before updating a row in students table such that the newly updated gpa in students table if exceeds 10 then make it 10.

```
*/
```

```
mysql> drop trigger if exists before_update_students;
```

```
Query OK, 0 rows affected (1.42 sec)
```

```
mysql> DELIMITER //
```

```
mysql> CREATE TRIGGER before_update_students
```

```
BEFORE UPDATE
```

```
ON Students FOR EACH ROW
```

```
BEGIN
```

```
IF new.gpa > 10 THEN
```

```
set new.gpa=10;
```

```
END IF;
```

```
END //
```

```
Query OK, 0 rows affected (0.10 sec)
```

```
mysql> update Students set gpa=20 where sid=55555;
```

```
Query OK, 1 row affected (0.07 sec)
```

```
Rows matched: 1 Changed: 1 Warnings: 0
```

```
mysql> select * from Students;
```

```
+-----+-----+-----+-----+-----+
| sid  | name  | login   | age  | gpa  |
+-----+-----+-----+-----+-----+
| 53650 | Smith | smith@math | 19 | 3.8 |
| 53666 | Jones | jones@cs   | 18 | 3.4 |
| 53667 | Raj   | Raj@cs    | 20 | 4.4 |
| 53688 | Smith | Smith@ee   | 18 | 3.2 |
| 55555 | xq    | xq@we      | 18 | 10  |
| 55556 | xq1   | xq1@we     | 18 | 3.5 |
+-----+-----+-----+-----+-----+
```

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

After update trigger

/* Write a query to invoke trigger after updating a row in students table such that the newly updated sid in students table should even reflect in enrolled table as shown in output.

```
mysql> drop trigger if exists after_update_students;
```

Query OK, 0 rows affected (1.42 sec)

```
mysql> DELIMITER //  
mysql> SET FOREIGN_KEY_CHECKS = 0; //  
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> Create Trigger after_update_students  
AFTER UPDATE ON Students FOR EACH ROW  
BEGIN  
UPDATE Enrolled SET sid=new.sid where sid=old.sid;  
END //  
Query OK, 0 rows affected (0.15 sec)
```

```
mysql> update Students set sid=66666 where sid=55556;
```

Query OK, 1 rows affected (0.15 sec)

Rows matched: 1 Changed: 1 Warnings: 0

```
mysql> select * from Students;
```

```
+-----+-----+-----+-----+-----+
| sid  | name  | login   | age  | gpa  |
+-----+-----+-----+-----+-----+
| 53650 | Smith | smith@math | 19 | 3.8 |
| 53666 | Jones | jones@cs   | 18 | 3.4 |
| 53667 | Raj   | Raj@cs     | 20 | 4.4 |
| 53688 | Smith | Smith@ee   | 18 | 3.2 |
| 55555 | xq    | xq@we      | 18 | 10  |
| 66666 | xq1   | xq1@we     | 15 | 3.5 |
+-----+-----+-----+-----+-----+
```

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



```
mysql> select * from Enrolled;
```

```
+-----+-----+-----+  
| sid  | cid      | grade |  
+-----+-----+-----+  
| 53650 | Topology112 | A    |  
| 53666 | History105  | B    |  
| 53688 | Carnatic101 | C    |  
| 53688 | Reggae203   | B    |  
| 66666 | History     | A    |  
+-----+-----+-----+
```

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

Trigger Before Deleting:

/* Write a query to invoke trigger before deleting a row in students table such

that the newly deleted row in students table should be added to new table called studentsdeleted.

```
mysql> delete from Enrolled where sid=55555;
```

Query OK, 0 rows affected (0.00 sec)

```
mysql> drop table if exists studentsdeleted;
```

Query OK, 0 rows affected, 1 warning (0.00 sec)

```
mysql> CREATE TABLE studentsdeleted(sid CHAR(20),name CHAR (30),login  
CHAR(20),age INTEGER,gpa REAL, UNIQUE (name, age), UNIQUE(login),CONSTRAINT  
StudentsKey PRIMARY KEY (sid));
```

Query OK, 0 rows affected (0.11 sec)

```
mysql> drop trigger if exists before_delete_students;
```

Query OK, 0 rows affected, 1 warning (0.00 sec)

```
mysql> DELIMITER //
```

```
mysql> CREATE TRIGGER before_delete_students
```

```
BEFORE DELETE
```

```
ON Students FOR EACH ROW
```

```
BEGIN
```

```
INSERT INTO studentsdeleted values(old.sid,old.name,old.login,old.age,old.gpa);
```

```
END //
```

Query OK, 0 rows affected (0.10 sec)

```
mysql> delete from Students where sid=55555;
```

Query OK, 1 row affected (0.06 sec)

```
mysql> select * from Students;
```

sid	name	login	age	gpa
53650	Smith	smith@math	19	3.8
53666	Jones	jones@cs	18	3.4
53667	Raj	Raj@cs	20	4.4
53688	Smith	Smith@ee	18	3.2
55556	xq1	xq1@we	18	3.5

5 rows in set (0.00 sec)

```
mysql> select * from studentsdeleted;
```

```
+-----+-----+-----+-----+-----+
```

```
| sid | name | login | age | gpa |
```

```
+-----+-----+-----+-----+-----+
```

```
| 55555 | xq | xq@we | 18 | 10 |
```

```
+-----+-----+-----+-----+-----+
```

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

Trigger after delete

**/* Write a query to invoke trigger after deleting a row in students table
such that the deleted sid in students table should even be deleted in enrolled table.**

```
mysql> drop trigger if exists after_delete_students;
```

```
Query OK, 0 rows affected, 1 warning (0.00 sec)
```

```
mysql> DELIMITER //
```

```
mysql> SET FOREIGN_KEY_CHECKS = 0;
```

```
Create Trigger after_delete_students
```

```
AFTER DELETE ON Students FOR EACH ROW
```

```
BEGIN
```

```
DELETE FROM Enrolled where sid=old.sid;
```

```
END //
```

```
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> DELETE FROM Students where sid=66666;
```

```
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> select * from Students;
```

```
+-----+-----+-----+-----+-----+
| sid  | name  | login   | age  | gpa  |
+-----+-----+-----+-----+-----+
| 53650 | Smith | smith@math | 19 | 3.8 |
| 53666 | Jones | jones@cs   | 18 | 3.4 |
| 53667 | Raj   | Raj@cs     | 20 | 4.4 |
| 53688 | Smith | Smith@ee   | 18 | 3.2 |
| 55556 | xq1   | xq1@we     | 18 | 3.5 |
+-----+-----+-----+-----+-----+
```

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

```
mysql> select * from Enrolled;
```

```
+-----+-----+-----+  
| sid  | cid      | grade |  
+-----+-----+-----+  
| 53650 | Topology112 | A    |  
| 53666 | History105  | B    |  
| 53688 | Carnatic101 | C    |  
| 53688 | Reggae203   | B    |  
| 55556 | History     | A    |  
+-----+-----+-----+
```

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

Experiment 9

Demonstrate the role of DBA using DCL commands

The *Data Control Language* is a subset of the Structured Query Language. Database administrators use DCL to configure security access to relational databases. It complements the *Data Definition Language*, which adds and deletes database objects, and the *Data Manipulation Language*, which retrieves, inserts, and modifies the contents of a database.

DCL is the simplest of the SQL subsets, as it consists of only three commands: GRANT, REVOKE, and DENY. Combined, these three commands provide administrators with the flexibility to set and remove database permissions in granular fashion.

Adding Permissions With the GRANT Command

The GRANT command adds new permissions to a database user. It has a very simple syntax, defined as follows:

```
GRANT [privilege] ON [object] TO [user] [WITH GRANT OPTION]
```

Here's the rundown on each of the parameters you can supply with this command:

- **Privilege** — can be either the keyword ALL (to grant a wide variety of permissions) or a specific database permission or set of permissions. Examples include CREATE DATABASE, SELECT, INSERT, UPDATE, DELETE, EXECUTE and CREATE VIEW.
- **Object** — can be any database object. The valid privilege options vary based on the type of database object you include in this clause. Typically, the object will be either a database, function, stored procedure, table or view.
- **User** — can be any database user. You can also substitute a role for the user in this clause if you wish to make use of role-based database security.
- If you include the optional **WITH GRANT OPTION** clause at the end of the GRANT command, you not only grant the specified user the permissions defined in the SQL statement but also give the user permission to further grant those same permissions to *other* database users. For this reason, use this clause with care.

For example, assume you wish to grant the user *Joe* the ability to retrieve information from the *employee* table in a database called *HR*. Use the following SQL command:

```
GRANT SELECT ON HR.employees TO Joe
```

Joe can retrieve information from the employees' table. He will not, however, be able to grant other users permission to retrieve information from that table because the DCL script did not include the WITH GRANT OPTION clause.

Revoking Database Access

The REVOKE command removes database access from a user previously granted such access. The syntax for this command is defined as follows:

```
REVOKE [GRANT OPTION FOR] [permission] ON [object] FROM [user]
[CASCADE]
```

Here's the rundown on the parameters for the REVOKE command:

- **Permission** — specifies the database permissions to remove from the identified user. The command revokes both GRANT and DENY assertions previously made for the identified permission.
- **Object** — can be any database object. The valid privilege options vary based on the type of database object you include in this clause. Typically, the object will be either a database, function, stored procedure, table, or view.
- **User** — can be any database user. You can also substitute a role for the user in this clause if you wish to make use of role-based database security.
- The **GRANT OPTION FOR** clause removes the specified user's ability to grant the specified permission to other users. If you include the **GRANT OPTION FOR** clause in a REVOKE statement, the primary permission is not revoked. This clause revokes only the granting ability.
- The **CASCADE** option also revokes the specified permission from any users that the specified user granted the permission.

The following command revokes the permission granted to Joe in the previous example:

```
REVOKE SELECT ON HR.employees FROM Joe
```