

Table of Contents

- Introduction to DMBS & SQL
- Normalization
- Subsets of SQL
 - DDL
 - DML
 - DCL
 - TCL
- SQL Operators
- SQL Functions
- SQL Joins
- Lab Session

What is Database?

Database is a collection of information organized for easy access, management and maintenance.

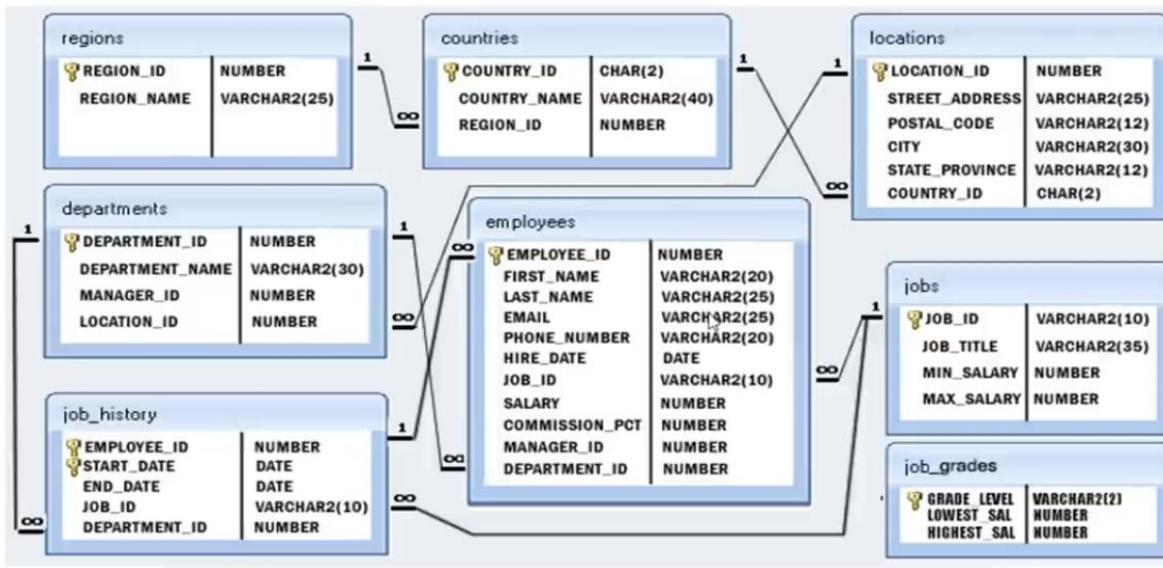
- Examples:
 - Telephone directory
 - Customer data
 - Product inventory
 - Visitors' register
 - Weather records



Types of Data Models

- **Record based logical model**
 - Hierarchical data model
 - Network data model
 - Relational data model
- **Object based logical model**
 - Entity relationship model

E/R Diagram



DBMS Operations

- Adding new files
- Inserting data
- Retrieving data
- Modifying data
- Removing data
- Removing files

Advantages of DBMS

- Sharing of data across applications
- Enhanced security mechanism
- Enforce integrity constraints
- Better transaction support
- Backup and recovery features

Introduction to RDBMS

- A relational database refers to a database that stores data in a structured format, using rows and columns.
- This makes it easier to locate and access specific values within the database.
- It is "relational" because the values within each table are related to each other. Tables may also be related to other tables.
- The relational structure makes it possible to run queries across multiple tables at once.

Features of RDBMS

- Every piece of information is stored in the form of tables
- Has primary keys for unique identification of rows
- Has foreign keys to ensure data integrity
- Provides SQL for data access
- Uses indexes for faster data retrieval
- Gives access privileges to ensure data security

RDBMS VS TRADITIONAL APPROACH

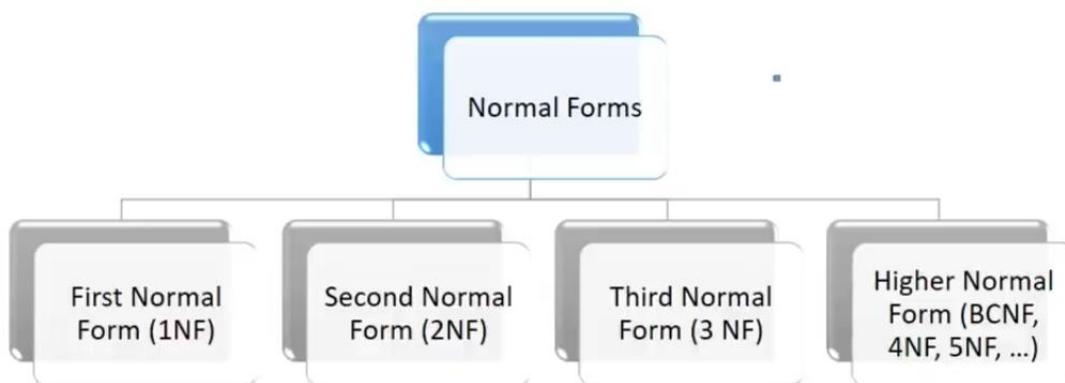
- The key difference is that RDBMS (relational database management system) applications store data in a tabular form, whereas in tradition approach, applications store data as files.
- There can be, but there will be no “relation” between the tables, like in a RDBMS. In traditional approach, data is generally stored in either a hierarchical form/navigational form. This means that a single data unit will have 0, 1 or more children nodes and 1 parent node.

Normalization

Normalization

- Decompose larger, complex table into simpler and smaller ones
- Moves from lower normal forms to higher normal forms.

Normalization and Normal Forms



Need for Normalization

- In order to produce good database design
- To ensure all database operations to be efficiently performed
- Avoid any expensive DBMS operations
- Avoid unnecessary replication of information

Need for Normalization

- RAW DATABASE

Student_Details	Course_details	Pre-requisite	Result_details
101 Jack 11/4/1975	M1 Advance Math	17	Basic Math 03/11/2015 82 A
102 Rock 10/04/1976	P4 Advance Physics	18	Basic Physics 22/11/2015 83 A
103 Mary 11/07/1975	B3 Advance Biology	10	Basic Biology 14/11/2015 68 B
104 Roby 10/04/1976	H6 Advance History	19	Basic History 22/11/2015 83 A
105 Jim 03/08/1978	C3 Advance Chemistry	12	Basic Biology 15/11/2015 50 C

Functional Dependency

- Consider the relation
 - Result (Student#, Course#, CourseName#, Marks#, Grade#)
 - Student# and course# together defines exactly one value of marks. Student#, course#, Marks
 - Student# and course# determines Marks or Marks is functionally dependent on student# and course#
 - Other functional dependencies in the relation:
 - Course# - CourseName
 - Marks# - Grade

Functional Dependency

- In a given relation R, P and Q are attributes. Attribute Q is functionally dependent on attribute P if each value of P determines exactly one value of Q.



Functional Dependency Types

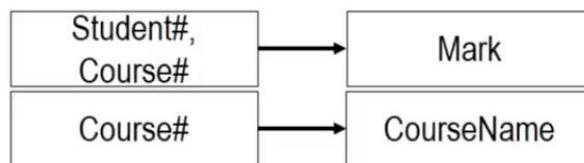
Partial Functional Dependency

Transitive Dependency

Functional Dependency Types

Partial Functional Dependency

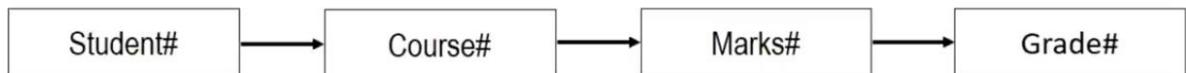
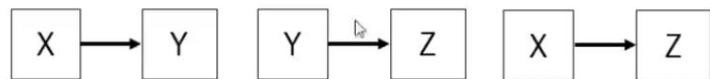
- Attribute Q is partially dependent on attribute P, if and only if it is dependent on the subset of attribute P.
- REPORT (Student#, Course#, StudentName, CourseName, Marks, Grade)



Functional Dependency Types

Transitive Dependency

X, Y, Z are three attributes



Normalization

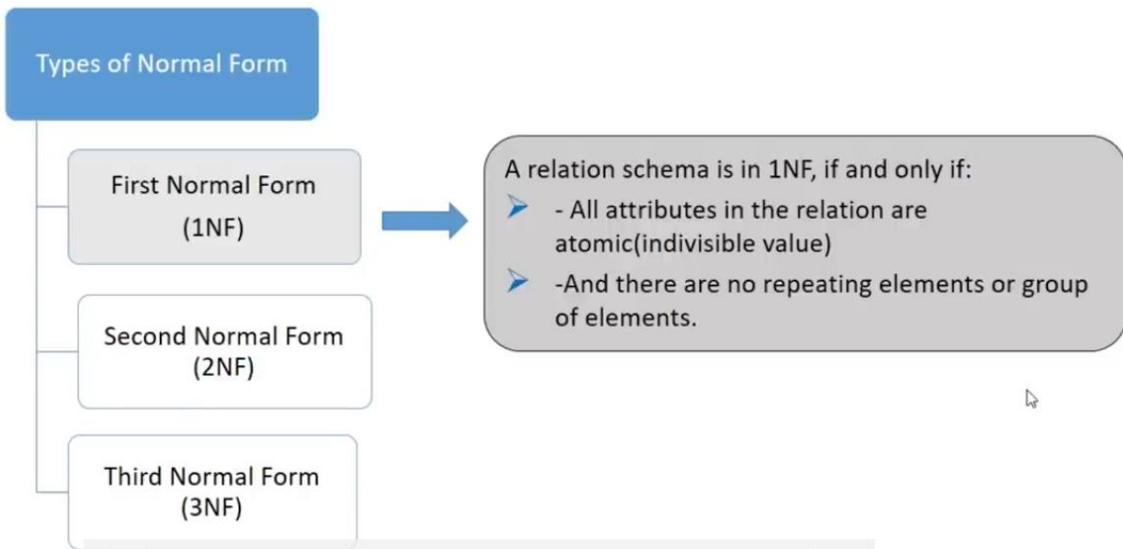
Types of Normal Form

First Normal Form
(1NF)

Second Normal Form
(2NF)

Third Normal Form
(3NF)

First Normal Form – (1NF)



First Normal Form – (1NF)

Student Marks Table

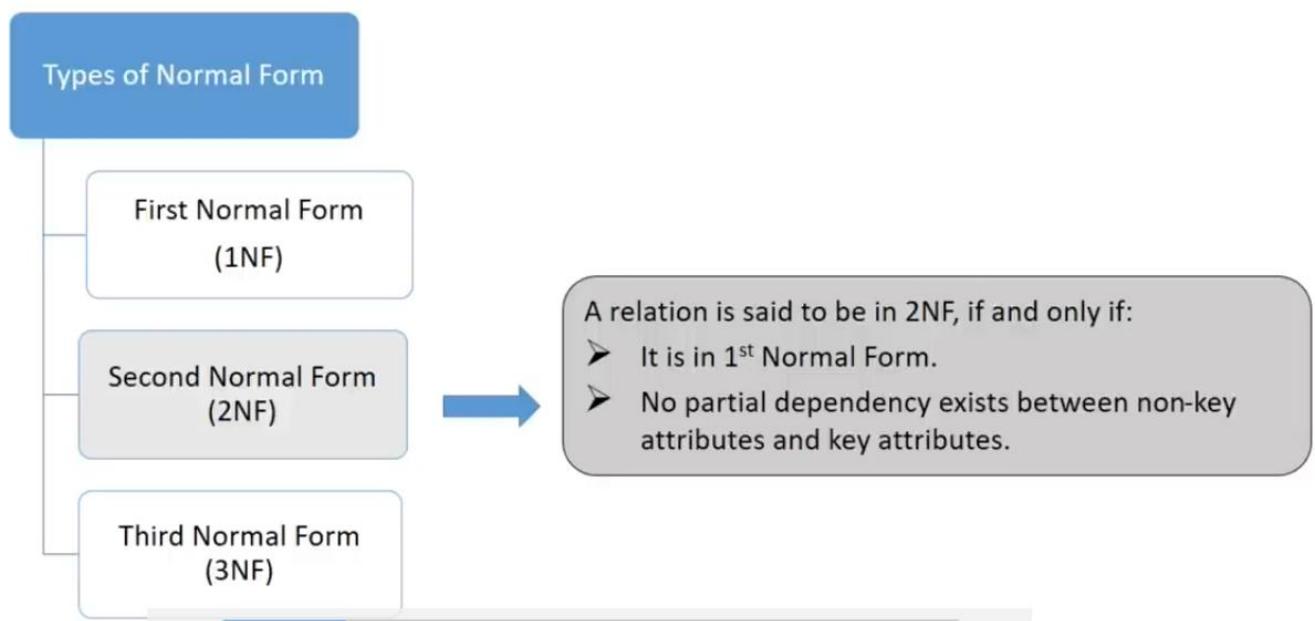
Student_Details	Course_details	Pre-requisite	Result_details
101 Jack 11/4/1975	M1 Advance Math	17	Basic Math 03/11/2015 82 A
102 Rock 10/04/1976	P4 Advance Physics	18	Basic Physics 21/11/2015 83 A
103 Mary 11/07/1975	B3 Advance Biology	10	Basic Biology 12/11/2015 68 B
104 Roby 10/04/1976	H6 Advance History	19	Basic History 21/11/2015 83 A
105 Jim 03/08/1978	C3 Advance Chemistry	12	Basic Biology 12/11/2015 50 C

First Normal Form - (1NF)

Student Marks Table in 1NF

Student #	Student_Name	Date Of Birth	Course#	CourseName	Pre-Requisite	Duration in days	Date Of Exam	Marks	Grade
101	Jack	11/4/1975	M1	Advance Math	Basic Math	17	02/11/2015	82	A
102	Roby	10/04/1976	P4	Advance Physics	Basic Physics	18	21/11/2015	83	A
103	Mary	11/07/1975	B3	Advance Biology	Basic Biology	10	12/11/2015	68	B
...

Second Normal Form – (2NF)



Second Normal Form – (2NF)

Student Marks Table in 1NF

Student#	Student_Name	DOB	Course #	CourseName	Pre Requisite	Duration in days	Date of Exam	Marks	Grade
101	Jack	11/4/1975	M1	Advance Math	Basic Math	17	02/11/2015	82	A
102	Rob	10/04/1976	P4	Advance Physics	Basic Physics	18	21/11/2015	83	A
103	Mary	11/07/1975	B3	Advance Biology	Basic Biology	10	12/11/2015	68	B

Second Normal Form – (2NF)

- Student#, Course# → Marks
- Student#, Course# → Grade
- Marks → Grade
- Student# → StudentName, DOB
- Course# → CourseName, Pre-Requisite,
- DurationDays, Date of exam

Partial
Dependency
with the Key
attribute

Split/Decompose the
tables to remove partial
dependencies

Second Normal Form – (2NF)

Student Table

Student#	Student_Name	Date Of Birth
101	Jack	11/4/1975
102	Roby	10/04/1976
103	Mary	11/07/1975

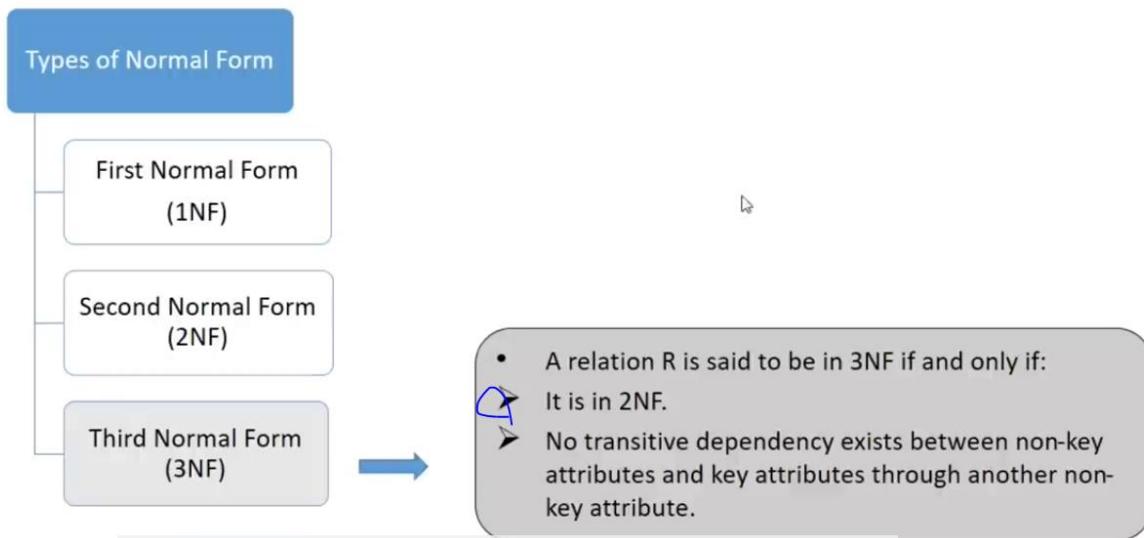
Result Table

Student#	Course#	Marks	Grade
101	M1	82	A
102	P4	83	A
103	B3	68	B

Course Table

Course#	CourseName	Prerequisite	Durationin days	Date Of Exam
M1	Advance Maths	Basic Math	17	02/11/2015
P4	Advance Physics	Basic Physics	18	21/11/2015
B3	Advance Biology	Basic Biology	10	12/11/2015

Third Normal Form – (3NF)



Third Normalization – (3NF)

Result_table

Student#	Course#	Marks	Grade
101	M1	82	A
102	P4	83	A
103	B3	68	B

Student# ,Course# → Marks

Student#, Course# → Grade

Marks → Grade

Student#,Course#→ Marks→ Grade:TD



Remove

Third Normalization – (3NF)

Result Table

<u>Student#</u>	<u>Course#</u>	Marks
101	M1	82
102	P4	83
103	B3	68

Marks Grade Table

Marks	Grade
82	A
83	A
68	B

What is SQL ?

Programming language specifically designed for working with Database to...

- CREATE
- MANIPULATE
- SHARE/ACCESS

Why SQL?

SQL is widely popular because it offers the following advantages:

- Allows users to communicate i.e, access and manipulate the database.
- Allows users to retrieve data from a database.
- Allows users to create, update, modify and delete the database

SQL is a language for defining the structure of a database.



SQL Terms

Data

Data is defined as facts or figures, or information that's stored in or used by a computer.

Database

A database is a organized collection of data/information so that it can be easily accessed, managed and updated.

SQL Data Types

1. Numeric – bit, tinyint, smallint, int, bigint, decimal, numeric, float, real
2. Character/String - Char, Varchar, Text
3. Date/Time - Date, Time, Datetime, Timestamp, Year
4. Miscellaneous- Json, XML

SQL Constraints

Constraint	Description
Not Null	Ensures that a column does not have a NULL value.
Default	Provides a default value for a column when none is specified.
Unique	Ensures that all the values in a column are different.
Primary	Identifies each row/record in a database table uniquely.
Check	Ensures that all values in a column satisfy certain conditions.
Index	Creates and retrieves data from the database very quickly.

Subsets of SQL

SQL Command Groups

- **DDL** (Data Definition Language) : creation of objects
- **DML** (Data Manipulation Language) : manipulation of data
- **DCL** (Data Control Language) : assignment and removal of permissions
- **TCL** (Transaction Control Language) : saving and restoring changes to a database

DDL – Data Definition Language

Command	Description
Create	Creates objects in the database/database objects
Alter	Alters the structures of the database/ database objects
Drop	Deletes objects from the database
Truncate	Removes all records from a table permanently
Rename	Renames an object

DDL - Data Definition Language – Create Command

```
CREATE TABLE employees (
    emp_id INT (10) NOT NULL,
    first_name VARCHAR(20),
    last_name VARCHAR(20) NOT NULL,
    salary int(10) NOT NULL,
    PRIMARY KEY (emp_id));
```

emp_id	first_name	last_name	salary
			→

```
create table employee(
    emp_id int not null,
    first_name varchar(20),
    last_name varchar(20),
    salary int,
    primary key(emp_id));
```

Navigator

SCHEMAS

- Filter objects
- angularapidb
- dcbapp
- dcbappqa
- greatlearningdb**
- Tables
- employee
- Views
- Stored Procedures
- Functions

medicaredb

mydb

schooldb

Query 1

```
1 • select * from employee;
```

Result Grid | Filter Rows: Ed

	emp_id	first_name	last_name	salary
*	NULL	NULL	NULL	NULL

Navigator

SCHEMAS

- Filter objects
- angularapidb
- dcbapp
- dcbappqa
- greatlearningdb**
- Tables
- employee
- Views
- Stored Procedures
- Functions

medicaredb

mydb

schooldb

sys

Query 1

```
1 • describe employee;
```

Result Grid | Filter Rows: Export: Wr

	Field	Type	Null	Key	Default	Extra
▶	emp_id	int	NO	PRI	NULL	
	first_name	varchar(20)	YES		NULL	
	last_name	varchar(20)	YES		NULL	
	salary	int	YES		NULL	

DDL - Data Definition Language – Alter Command

```
ALTER TABLE employees ADD COLUMN contact  
INT(10);
```

↓

emp_id	first_name	last_name	salary	contact
101	Steven	Cohen	10000	
102	Edwin	Thom	15000	
103	Harry	Potter	20000	

Navigator :::::::::::::::::::::

SCHEMAS

- Filter objects
- angularapidb
- dcbapp
- dcbappqa
- ▼ ► greatlearningdb
 - Tables
 - employee
 - Views
 - Stored Procedures
 - Functions
- medicaredb
- mydb
- schooldb
- sys

Query 1 ×

Folder | Back | Refresh | Save | Run | Stop | Help | Limit to 1000 rows

```
1 alter table employee add column contact int;
2 • describe employee;
```

Result Grid | Filter Rows: [] | Export: [] | Wrap Cell []

	Field	Type	Null	Key	Default	Extra
►	emp_id	int	NO	PRI	NULL	
	first_name	varchar(20)	YES		NULL	
	last_name	varchar(20)	YES		NULL	
	salary	int	YES		NULL	
	contact	int	YES		NULL	

DDL - Data Definition Language – Rename Command

```
ALTER TABLE employee RENAME  
COLUMN contact TO job_code;
```

empl_id	first_name	last_name	salary	job_code
101	Steven	Cohen	10000	
102	Edwin	Thomas	15000	
103	Harry	Potter	20000	

The screenshot shows the MySQL Workbench interface. The Navigator pane on the left lists databases: angularapidb, dcbbapp, dcbbappqa, greatlearningdb (selected), medicaredb, mydb, schooldb, and sys. Under greatlearningdb, it shows Tables (employee), Views, Stored Procedures, and Functions. The Query 1 editor at the top contains the following SQL code:

```
1 alter table employee rename column contact to job_code;  
2 • describe employee;
```

The Result Grid below displays the structure of the employee table:

	Field	Type	Null	Key	Default	Extra
▶	emp_id	int	NO	PRI	NULL	
	first_name	varchar(20)	YES		NULL	
	last_name	varchar(20)	YES		NULL	
	salary	int	YES		NULL	
	job_code	int	YES		NULL	

Drop column command:-

The screenshot shows the MySQL Workbench interface. On the left, the Navigator pane displays a tree view of databases: angularapidb, dcbapp, dcbaapqa, greatlearningdb (selected), medicaredb, mydb, schooldb, and sys. Under the greatlearningdb database, there are sub-folders for Tables, Views, Stored Procedures, and Functions. The Tables folder contains an entry for employee. The Query Editor pane at the top has a title 'Query 1' and contains the following SQL code:

```
1 • alter table employee drop column job_code;
2 • select * from employee
3
```

Below the code, the Result Grid pane shows the structure of the employee table with columns: emp_id, first_name, last_name, and salary. A single row is present with all values set to NULL.

	emp_id	first_name	last_name	salary
*	NULL	NULL	NULL	NULL

Truncate command is used to empty(delete all records from the table) the table:

DDL - Data Definition Language – Truncate Command

```
TRUNCATE TABLE employees;
```

→

emp_id	first_name	last_name	salary
101	Steven	Cohen	10000
102	Edwin	Thomas	15000
103	Harry	Potter	20000

The screenshot shows a database management interface with two main panes: Navigator and Query 1.

Navigator: On the left, it displays a tree view of schemas. The 'greatlearningdb' schema is expanded, showing its tables ('employee'), views, stored procedures, and functions. Other schemas listed include 'angularapidb', 'dcbapp', 'dcbappqa', 'medicaredb', 'mydb', 'schooldb', and 'sys'.

Query 1: On the right, there is a query editor and a result grid.

Query Editor: The SQL code entered is:

```
1 •  select * from employee;
2 •  truncate table employee;
3 •  select * from employee;
```

Result Grid: The result grid shows the state of the 'employee' table after the truncate command. It has 6 columns: emp_id, first_name, last_name, salary, and job_code. There is one row with all values set to NULL.

*	emp_id	first_name	last_name	salary	job_code
*	NULL	NULL	NULL	NULL	NULL

Drop table command is used to delete the entire table

DDL - Data Definition Language – Drop Command

```
DROP TABLE table_name;
```

```
DROP TABLE employees;
```

DML – Data Manipulation Language

Command	Description
Insert	Insert data into a table
Update	Updates existing data within a table
Delete	Deletes specified/all records from a table

DML – Data Manipulation Language – INSERT Command

```
INSERT INTO employees  
(emp_id,first_name,last_name,salary) VALUES  
(101, 'Steven', 'King', 10000);
```

```
INSERT INTO employees  
(emp_id,first_name,last_name,salary) VALUES  
(102, 'Edwin', 'Thomas', 15000 );
```

```
INSERT INTO employees  
(emp_id,first_name,last_name,salary) VALUES  
(103, 'Harry', 'Potter', 20000);
```

emp_id	first_name	last_name	salary
101	Steven	King	10000
102	Edwin	Thomas	15000
103	Harry	Potter	20000

Screenshot of a database management tool showing the execution of an INSERT command and its results.

Navigator: Shows the schema tree with the **greatlearningdb** database selected, containing **Tables**, **Views**, **Stored Procedures**, and **Functions**.

Query 1: The query window displays the following SQL code:

```
1 • insert into employee(emp_id,first_name,last_name,salary)  
2     values(1,'jennifer','aniston',100000),  
3             (2,'Charlie','sheen',200000),  
4             (3,'Matt','Damon',300000),  
5             (4,'jennifer','lopx',400000),  
6             (5,'charlie','harper',500000),  
7             (6,'matt','leblanc',600000);  
8 • select * from employee
```

Result Grid: The result grid shows the data inserted into the **employee** table:

emp_id	first_name	last_name	salary
1	jennifer	aniston	100000
2	Charlie	sheen	200000
3	Matt	Damon	300000
4	jennifer	lopx	400000
5	charlie	harper	500000
6	matt	leblanc	600000
*	NULL	NULL	NULL

DML – Data Manipulation Language – UPDATE Command

```
UPDATE employees  
SET last_name='Cohen'  
WHERE emp_id=101;
```

emp_id	first_name	last_name	salary
101	Steven	Cohen	10000
102	Edwin	Thomas	15000
103	Harry	Potter	20000

The screenshot shows the MySQL Workbench interface. On the left is the Navigator pane with a tree view of schemas: angularapidb, dcapp, dcappqa, greatlearningdb (selected), medicaredb, mydb, schooldb, and sys. Under greatlearningdb, there are tables (employee), views, stored procedures, and functions. The central area is the Query 1 editor with the following content:

```
1 • update employee set last_name='hathaway' where emp_id=1;  
2 • select * from employee  
3
```

Below the editor is the Result Grid, which displays the following data:

	emp_id	first_name	last_name	salary
1	1	jennifer	hathaway	100000
2	2	Charlie	sheen	200000
3	3	Matt	Damon	300000
4	4	jennifer	lopx	400000
5	5	charlie	harper	500000
6	6	matt	leblanc	600000
*	NULL	NULL	NULL	NULL

DML – Data Manipulation Language - DELETE Command

```
DELETE FROM employees WHERE emp_id IN  
(101,103);
```

emp_i

Query 1

```
1 • delete from employee where emp_id in(1,6);
2 • select * from employee
3
```

Result Grid | Filter Rows: | Edit: |

	emp_id	first_name	last_name	salary
▶	2	Charlie	sheen	200000
	3	Matt	Damon	300000
	4	jennifer	lopx	400000
	5	charlie	harper	500000
*	NULL	NULL	NULL	NULL

Delete where command

Query 1

```
1 • delete from employee where emp_id=5;
2 • select * from employee
3
```

Result Grid | Filter Rows: | Edit: |

	emp_id	first_name	last_name	salary
▶	2	Charlie	sheen	200000
	3	Matt	Damon	300000
	4	jennifer	lopx	400000
*	NULL	NULL	NULL	NULL

DCL – Data Control Language

Command	Description
Grant	Gives access privileges to database
Revoke	Withdraws access privileges given with the grant command

GRANT <Privilege list> ON
<Relation Name> TO
<USER>

REVOKE <Privilege list> ON
<Relation Name> TO
<USER>

TCL – Transaction Control

Command	Description
Commit	Saves the work done
Rollback	Restores database to origin state since the last commit
Savepoint	Identify a point in a transaction to which you can roll back later

SQL Operators

SQL Operators - Filter

WHERE Clause :

- Used to specify a condition while fetching the data from a single table or by joining with multiple tables.
- Not only used in the SELECT statement, but it is also used in the UPDATE, DELETE statement, etc.,

e.g.

```
SELECT * FROM employees WHERE  
emp_id=101;
```

The example mentioned above extracts all the columns from the table 'employees' whose emp_id=101

emp_id	first_name	last_name	salary
101	Steven	Cohen	10000
102	Edwin	Thomas	15000
103	Harry	Potter	20000

emp_id	first_name	last_name	salary
101	Steven	Cohen	10000

Query 1

The screenshot shows the MySQL Workbench interface. The top bar has tabs for 'Query 1' and other database objects. Below the tabs is a toolbar with various icons for operations like select, insert, update, delete, and search. The main area contains a query editor with the following SQL code:

```
1 •  select * from employee where emp_id=3
2
```

Below the query editor is a results grid titled 'Result Grid'. It has columns for emp_id, first_name, last_name, and salary. There are two rows of data displayed:

	emp_id	first_name	last_name	salary
▶	3	Matt	Damon	300000
*	NULL	NULL	NULL	NULL

SQL Operators – Logical

Operator	Illustrative Example	Result
AND	(5<2) AND (5>3)	FALSE
OR	(5<2) OR (5>3)	TRUE
NOT	NOT(5<2)	TRUE

Sample Queries:

```
SELECT * FROM employees WHERE first_name = 'Steven' and salary = 15000;
```

```
SELECT * FROM employees WHERE first_name = 'Steven' OR salary = 15000;
```

```
SELECT * FROM employees WHERE first_name = 'Steven' and salary != 10000;
```

Query 1 x

File Edit View Insert Tools Options Help

Folder Open Save All Close Find Replace Undo Redo Refresh Limit to 1000 rows Filter Results Star Save As Print

1 • select * from employee where first_name='jennifer' **and** salary=100000
2

Result Grid | Filter Rows: | Edit: | Export/Import: |

emp_id	first_name	last_name	salary
1	jennifer	aniston	100000
*	NULL	NULL	NULL

Or keyword:-

The screenshot shows the MySQL Workbench interface. At the top, there's a toolbar with various icons. Below it is a query editor window titled "Query 1" containing the following SQL code:

```
1 •    select * from employee where first_name='jennifer' or salary=100000
2
```

Below the query editor is a results grid titled "Result Grid". The grid has columns: emp_id, first_name, last_name, and salary. It displays two rows of data:

	emp_id	first_name	last_name	salary
▶	4	jennifer	lopx	400000
	1	jennifer	aniston	100000
*	NULL	NULL	NULL	NULL

SQL Operators – Comparison

Comparison Operators		Sample Queries:
Symbol	Meaning	
=	Equal to	SELECT * FROM employees WHERE first_name = 'Steven' AND salary <=10000;
>	Greater than	SELECT * FROM employees WHERE first_name = 'Steven'
>=	Greater than or equal to	OR salary >=10000;
<	Less than	
<=	Less than or equal to	
<> or !=	Not equal to	

Query 1 x

Folder | New | Save | Refresh | Help | Limit to 1000

```
1 • select * from employee where salary>300000
2
```

Result Grid | Filter Rows: [] | Edit: | Export

	emp_id	first_name	last_name	salary
▶	5	charlie	harper	500000
	4	jennifer	lopxex	400000
	6	matt	leblanc	600000
*	NULL	NULL	NULL	NULL

Query 1 x

Folder | New | Save | Refresh | Help | Limit to 1000 rows ▾

```
1 • select * from employee where first_name!='charlie'
2
```

Result Grid | Filter Rows: [] | Edit: | Export

	emp_id	first_name	last_name	salary
▶	3	Matt	Damon	300000
	4	jennifer	lopxex	400000
	1	jennifer	aniston	100000
	6	matt	leblanc	600000
*	NULL	NULL	NULL	NULL

SQL Operators – Special

Special Operators	
BETWEEN	Checks an attribute value within range
LIKE	Checks an attribute value matches a given string pattern
IS NULL	Checks an attribute value is NULL [↳]
IN	Checks an attribute value matches any value within a value list
DISTINCT	Limits values to unique values

Sample Queries:

```
SELECT * FROM employees WHERE salary  
between 10000 and 20000;
```

```
SELECT * FROM employees WHERE first_name  
like 'Steven';
```

```
SELECT * FROM employees WHERE salary is  
null;
```

```
SELECT * FROM employees where salary in  
(10000,12000,20000);
```

```
SELECT DISTINCT(first_name) from  
employees;
```

Query 1 x

```
1 •  select * from employee where salary between 300000 and 500000
2
```

	emp_id	first_name	last_name	salary
▶	5	charlie	harper	500000
	3	Matt	Damon	300000
	4	jennifer	lopx	400000
*	NULL	NULL	NULL	NULL

Query 1

```
1 • select * from employee where last_name like 'l%';
2
```

Result Grid | Filter Rows: [] | Edit: [] | Ex

	emp_id	first_name	last_name	salary
▶	4	jennifer	lopx	400000
	6	matt	leblanc	600000
*	NULL	NULL	NULL	NULL

Query 1

```
1 • select * from employee where salary in(100000,300000,500000);
2
```

Result Grid | Filter Rows: [] | Edit: [] | Export/Import: []

	emp_id	first_name	last_name	salary
▶	5	charlie	harper	500000
	3	Matt	Damon	300000
	1	jennifer	aniston	100000
*	NULL	NULL	NULL	NULL

Query 1 ×

```
1 •   select distinct(first_name) from employee ;  
2
```

Result Grid | Filter Rows: [] | Export: [] | Wrap

first_name
charlie
Matt
jennifer

SQL Operators – Aggregations

Aggregation Functions

Avg()	Returns the average value from specified columns
Count()	Returns number of table rows
Max()	Returns largest value among the records
Min()	Returns smallest value among the records
Sum()	Returns the sum of specified column values

Sample Queries:

```
SELECT avg(salary) FROM employees;  
SELECT count(*) FROM employees;  
SELECT min(salary) FROM employees;  
SELECT max(salary) FROM employees;  
SELECT sum(salary) FROM employees;
```

Query 1 ×

```
1 •   select count(*) from employee ;  
2
```

Result Grid | Filter Rows: [] | E

count(*)
6

Query 1

```

1 •  select * from employee where salary=(select max(salary) from employee);
2

```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap |

	emp_id	first_name	last_name	salary
▶	6	matt	leblanc	600000
*	NULL	NULL	NULL	NULL

SQL GROUP BY Clause

- Arrange identical data into groups.

e.g.,
 SELECT max(salary), dept_id
 FROM employees
 GROUP BY dept_id

emp_id	first_name	last_name	salary	dept_id
103	Harry	Potter	20000	12
102	Edwin	Thomas	15000	11
101	Steven	Cohen	10000	10
100	Erik	John	10000	12

Query 1

```

1 •  select first_name,max(salary),dept from employee group by dept;
2

```

	first_name	max(salary)	dept
	charlie	500000	content
	Charlie	600000	tech
	Matt	300000	sales
▶	jennifer	400000	marketing
▶	jennifer	100000	sale

SQL HAVING Clause

- Used with aggregate functions due to its non-performance in the WHERE clause.
- Must follow the GROUP BY clause in a query and must also precede the ORDER BY clause if used.

employee_id	first_name	last_name	salary	dept_id
103	Harry	Potter	20000	12
102	Edwin	Thomas	15000	11
101	Steven	Cohen	10000	10
100	Erik	John	10000	12

e.g.,

```

SELECT AVG(salary),dept_id
FROM employees
GROUP BY dept_id
HAVING count(dept_id)>=2

```

SQL ORDER BY Clause

- Used to sort output of SELECT statement
- Default is to sort in ASC (Ascending)
- Can Sort in Reverse (Descending) Order with "DESC" after the column name

e.g.,

```
SELECT * FROM employees  
ORDER BY salary DESC;
```

employee_id	first_name	last_name	salary
101	Steven	Cohen	10000
102	Edwin	Thomas	15000
103	Harry	Potter	20000

employee_id	first_name	last_name	salary
103	Harry	Potter	20000
102	Edwin	Thomas	15000
101	Steven	Cohen	10000

The screenshot shows the MySQL Workbench interface. The top window is titled 'Query 1' and contains the SQL query:select * from employee order by emp_id desc;The bottom window is titled 'Result Grid' and displays the sorted employee data:

	emp_id	first_name	last_name	salary	dept
▶	6	matt	leblanc	600000	tech
	5	charlie	harper	500000	content
	4	jennifer	lopx	400000	marketing
	3	Matt	Damon	800000	sales
	2	Charlie	sheen	200000	tech
	1	jennifer	aniston	600000	sale
*	NULL	NULL	NULL	NULL	NULL

Query 1 × employee

Limit to 1000 row

```
1 • select * from employee order by salary| desc;
2
```

Result Grid | Filter Rows: | Edit: |

	emp_id	first_name	last_name	salary	dept
▶	3	Matt	Damon	800000	sales
	1	jennifer	aniston	600000	sale
	6	matt	leblanc	600000	tech
	5	charlie	harper	500000	content
	4	jennifer	lopx	400000	marketing
	2	Charlie	sheen	200000	tech
*	NULL	NULL	NULL	NULL	NULL

SQL UNION

- Used to combine the result-set of two or more SELECT statements removing duplicates
- Each SELECT statement within the UNION must have the same number of columns
- The selected columns must be of similar data types and must be in the same order in each SELECT statement

greatlearning
Learning for Life

SQL UNION

Product1		Product2		PRODUCT_NAME
CATEGORY_ID	PRODUCT_NAME	CATEGORY_ID	PRODUCT_NAME	PRODUCT_NAME
1	Nokia	1	Samsung	
2	Samsung	2	LG	
3	HP	3	HP	
6	Nikon	5	Dell	
e.g.,		6	Apple	
		10	Playstation	

```
SELECT product_name FROM product1
UNION
SELECT product_name FROM
product2;
```

The screenshot shows a MySQL Workbench interface. The top bar has tabs for 'employee' and 'Administration - Server Status'. Below the tabs is a toolbar with icons for file operations, search, and refresh. The main area contains a SQL editor with the following code:

```
1 •  select product_name from product1
2      union
3      select product_name from product2;
```

Below the SQL editor is a results grid titled 'Result Grid'. The grid has one column labeled 'product_name' and contains the following data:

product_name
nokia
samsung
hp
nikon
lg
dell
apple
playstation

SQL UNION ALL

- Used to combine the results of two SELECT statements including duplicate rows.
- The same rules that apply to the UNION clause will apply to the UNION ALL operator.

SYNTAX:

```
SELECT col1,col2... FROM table1
UNION ALL
SELECT col1,col2... FROM table2;
```

employee Administration - Server Status employee x

Limit to 100

```
1 • select product_name from product1
2 union all
3 select product_name from product2;
```

Result Grid | Filter Rows: [] | Export: []

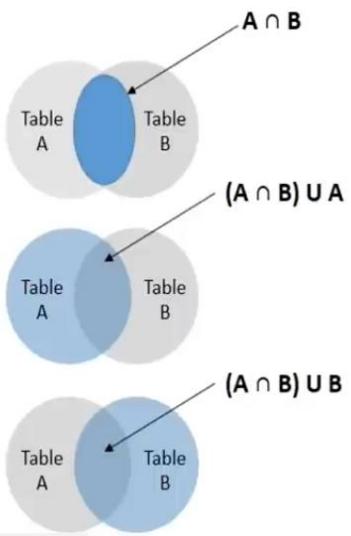
product_name
nokia
samsung
hp
nikon
samsung
lg
hp
dell
apple
playstation

SQL Joins

SQL JOINS

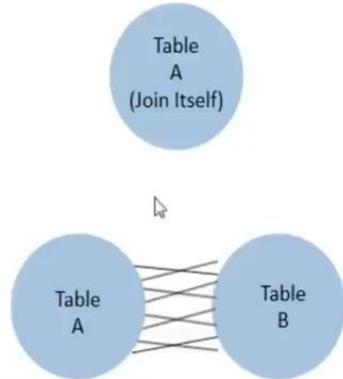
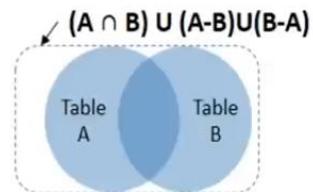
Combine rows/columns from two or more tables, based on a related column between them in a database

- **INNER JOIN** – Returns rows when there is a match in both tables.
- **LEFT JOIN** – Returns all rows from the left table, even if there are no matches in the right table.
- **RIGHT JOIN** – Returns all rows from the right table, even if there are no matches in the left table.



SQL JOINS

- **FULL OUTER JOIN** – Returns rows when there is a match in one of the tables.
- **SELF JOIN** – Used to join a table to itself as if the table were two tables, temporarily renaming at least one table in the SQL statement.
- **CARTESIAN JOIN (CROSS JOIN)** – Returns the Cartesian product of the sets of records from the two or more joined tables.



SQL INNER JOIN

The INNER JOIN creates a new result table by combining column values of two tables (table1 and table2) based upon the join-predicate. The query compares each row of table1 with each row of table2 to find all pairs of rows which satisfy the join-predicate.

SYNTAX :

```
SELECT table1.col1, table2.col2,..., table1.coln  
FROM table1  
INNER JOIN table2  
ON table1.commonfield = table2.commonfield;
```

SQL INNER JOIN

emp_id	first_name	last_name	salary	dept_id
103	Harry	Potter	20000	12
102	Edwin	Thomas	15000	11
101	Steven	Cohen	10000	10
100	Erik	John	10000	12

dept_id	dept_name	manager_id	location_id
10	IT	200	1700
11	Marketing	201	1800
13	Resources	203	2400
14	Shipping	121	1500

```
SELECT e.emp_id, e.first_name,  
e.last_name, d.dept_id,  
d.dept_name  
FROM employees e  
INNER JOIN departments d  
ON e.dept_id=d.dept_id;
```

emp_id	first_name	last_name	dept_id	dept_name
101	Steven	Cohen	10	IT
102	Edwin	Thomas	11	Marketing

employee department department

```
1 • SELECT * FROM greatlearningdb.department;
```

	dept_id	dept	dept_loc
▶	1	content	chicago
	2	support	new jersey
	3	sales	boston
	4	hr	chicago
	5	operations	new york

employee department department

```
1 • SELECT * FROM greatlearningdb.employee;
```

	emp_id	first_name	last_name	salary	dept
▶	5	charlie	harper	500000	content
	2	Charlie	sheen	200000	tech
	3	Matt	Damon	800000	sales
	4	jennifer	lopex	400000	marketing
	1	jennifer	aniston	600000	sales
	6	matt	leblanc	600000	tech
*	NULL	NULL	NULL	NULL	NULL

employee department department

```
1 • select e.first_name,e.salary,d.dept,d.dept_loc
2   from employee e
3     inner join department d
4       on e.dept=d.dept|
```

	first_name	salary	dept	dept_loc
▶	charlie	500000	content	chicago
	Matt	800000	sales	boston
	jennifer	600000	sales	boston

SQL LEFT JOIN

The LEFT JOIN returns all the values from the left table, plus matched values from the right table or NULL in case of no matching join predicate.

SYNTAX :

```
SELECT table1.col1, table2.col2,..., table1.coln  
FROM table1  
LEFT JOIN table2  
ON table1.commonfield = table2.commonfield;
```

greatlearning
Learning for Life

SQL LEFT JOIN

emp_id	first_name	last_name	salary	dept_id
103	Harry	Potter	20000	12
102	Edwin	Thomas	15000	11
101	Steven	Cohen	10000	10
100	Erik	John	10000	12

dept_id	dept_name	manager_id	location_id
10	IT	200	1700
11	Marketing	201	1800
13	Resources	203	2400
14	Shipping	121	1500

```
SELECT e.employee_id,  
e.first_name, e.last_name,  
d.dept_id, d.dept_name  
FROM employees e  
LEFT OUTER JOIN departments d  
ON e.dept_id = d.dept_id;
```

emp_id	first_name	last_name	dept_id	dept_name
101	Steven	Cohen	10	IT
102	Edwin	Thomas	11	Marketing
103	Harry	Potter	Null	Null
100	Erik	John	Null	Null

The screenshot shows a MySQL Workbench interface. At the top, there are tabs for 'employee' and 'department'. Below the tabs is a toolbar with various icons. A SQL editor window contains the following code:

```
1 select e.first_name, e.salary, e.dept, d.dept, d.dept_loc
2 from employee e
3 left join department d
4 on e.dept=d.dept
```

Below the code is a result grid titled 'Result Grid'. It has columns: first_name, salary, dept, dept, and dept_loc. The data is as follows:

	first_name	salary	dept	dept	dept_loc
▶	charlie	500000	content	content	chicago
	Charlie	200000	tech	NULL	NULL
	Matt	800000	sales	sales	boston
	jennifer	400000	marketing	NULL	NULL
	jennifer	600000	sales	sales	boston
	matt	600000	tech	NULL	NULL

SQL RIGHT JOIN

- The RIGHT JOIN returns all the values from the right table, plus matched values from the left table or NULL in case of no matching join predicate.

SYNTAX :

```
SELECT table1.col1, table2.col2,..., table1.coln
FROM table1
RIGHT JOIN table2
ON table1.commonfield = table2.commonfield;
```

SQL RIGHT JOIN

emp_id	first_name	last_name	salary	dept_id	dept_id	dept_name	manager_id	location_id
103	Harry	Potter	20000	12	10	IT	200	1700
102	Edwin	Thomas	15000	11	11	Marketing	201	1800
101	Steven	Cohen	10000	10	13	Resources	203	2400
100	Erik	John	10000	12	14	Shipping	121	1500

```
SELECT e.emp_id, e.first_name, e.last_name,
       d.dept_id, d.dept_name
  FROM employees e
RIGHT JOIN departments d
ON e.dept_id=d.dept_id;
```

emp_id	first_name	last_name	dept_id	dept_name
101	Steven	Cohen	10	IT
102	Edwin	Thomas	11	Marketing
Null	Null	Null	13	Resources
Null	Null	Null	14	Shipping

The screenshot shows the MySQL Workbench interface with three tabs: 'employee', 'department', and 'department'. The 'department' tab is active. A SQL query is entered in the query editor:

```
1 select d.dept,d.dept_loc,e.first_name,e.salary,e.dept
2 from employee e
3 right join department d
4 on e.dept=d.dept
```

The results are displayed in a 'Result Grid' table:

	dept	dept_loc	first_name	salary	dept
▶	content	chicago	charlie	500000	content
	support	new jersey	NULL	NULL	NULL
	sales	boston	Matt	800000	sales
	sales	boston	jennifer	600000	sales
	hr	chicago	NULL	NULL	NULL
	operations	new york	NULL	NULL	NULL

SQL FULL OUTER JOIN

The FULL OUTER JOIN combines the results of both left and right outer joins. The joined table will contain all records from both the tables and fill in NULLs for missing matches on either side.

SYNTAX :

```
SELECT table1.col1, table2.col2,..., table1.coln  
FROM table1  
Left JOIN table2  
ON table1.commonfield = table2.commonfield;  
Union  
SELECT table1.col1, table2.col2,..., table1.coln  
FROM table1  
Right JOIN table2  
ON table1.commonfield = table2.commonfield;
```

greatlearning
Learning for Life

SQL FULL OUTER JOIN

emp_id	first_name	last_name	salary	dept_id
103	Harry	Potter	20000	12
102	Edwin	Thomas	15000	11
101	Steven	Cohen	10000	10
100	Erik	John	10000	12

dept_id	dept_name	manager_id	location_id
10	IT	200	1700
11	Marketing	201	1800
13	Resources	203	2400
14	Shipping	121	1500

```
SELECT e.emp_id, e.first_name, e.last_name, d.dept_id,  
d.dept_name  
FROM employees e  
LEFT JOIN departments d  
ON e.dept_id=d.dept_id  
UNION  
SELECT e.emp_id, e.first_name, e.last_name, d.dept_id,  
d.dept_name  
FROM employees e  
RIGHT JOIN departments d  
ON e.dept_id=d.dept_id;
```

emp_id	first_name	last_name	dept_id	dept_name
101	Steven	Cohen	10	IT
102	Edwin	Thomas	11	Market
103	Harry	Potter	12	Null
100	Erik	John	Null	Null
Null	Null	Null	13	Resou

The screenshot shows the MySQL Workbench interface. At the top, there are tabs for 'employee', 'department', and 'department'. Below the tabs is a toolbar with various icons. The main area contains a multi-line text editor with the following SQL code:

```

1 •  select e.first_name,e.salary,e.dept,d.dept,d.dept_loc
2   from employee e
3   left join department d
4   on e.dept=d.dept
5   union
6   select e.first_name,e.salary,e.dept,d.dept,d.dept_loc
7   from employee e
8   right join department d
9   on e.dept=d.dept

```

Below the code is a result grid titled 'Result Grid' with the following data:

	first_name	salary	dept	dept	dept_loc
	charlie	500000	content	content	chicago
	Charlie	200000	tech	NULL	NULL
	Matt	800000	sales	sales	boston
	jennifer	400000	marketing	NULL	NULL
	jennifer	600000	sales	sales	boston
	matt	600000	tech	NULL	NULL
	NULL	NULL	NULL	support	new jersey
▶	NULL	NULL	NULL	hr	chicago
	NULL	NULL	NULL	operations	new york

SQL CROSS JOIN

- The CROSS JOIN produces a result set with the number of rows in the first table multiplied by the number of rows in the second.

SYNTAX:

```

SELECT table1.col1, table2.col2,..., table1.coln
FROM table1
CROSS JOIN table2;

```

SQL CROSS JOIN

- Assume there are 4 records in table1 and 3 records in table2

```
SELECT * FROM table1  
CROSS JOIN table2;
```

Table-1

alpha
A
B
C
D

Table-2

Num
1
2
3



A	1
A	2
A	3
B	1
B	2
B	3
C	1
C	2
C	3
D	1
D	2
D	3

employee department x department

1 select * from employee cross join department order by emp_id asc;

Result Grid | Filter Rows: | Export: | Wrap Cell Content: |

	emp_id	first_name	last_name	salary	dept	dept_id	dept	dept_loc
▶	1	jennifer	aniston	600000	sales	1	content	chicago
	1	jennifer	aniston	600000	sales	2	support	new jersey
	1	jennifer	aniston	600000	sales	3	sales	boston
	1	jennifer	aniston	600000	sales	4	hr	chicago
	1	jennifer	aniston	600000	sales	5	operations	new york
	2	Charlie	sheen	200000	tech	1	content	chicago
	2	Charlie	sheen	200000	tech	2	support	new jersey
	2	Charlie	sheen	200000	tech	3	sales	boston
	2	Charlie	sheen	200000	tech	4	hr	chicago
	2	Charlie	sheen	200000	tech	5	operations	new york
	3	Matt	Damon	800000	sales	1	content	chicago
	3	Matt	Damon	800000	sales	2	support	new jersey
	3	Matt	Damon	800000	sales	3	sales	boston
	3	Matt	Damon	800000	sales	4	hr	chicago
	3	Matt	Damon	800000	sales	5	operations	new york
	4	jennifer	lopex	400000	mar...	1	content	chicago
	4	jennifer	lopex	400000	mar...	2	support	new jersey
	4	jennifer	lopex	400000	mar...	3	sales	boston
	4	jennifer	lopex	400000	mar...	4	hr	chicago
	4	jennifer	lopex	400000	mar...	5	operations	new york

Exercise

File Edit Format View Help

grea
Lea

```
-- -----  
# Datasets Used: cricket_1.csv, cricket_2.csv  
-- cricket_1 is the table for cricket test match 1.  
-- cricket_2 is the table for cricket test match 2.  
-- -----  
  
# Q1. Find all the players who were present in the test match 1 or test match 2;  
SELECT * FROM cricket_1  
UNION  
SELECT * FROM cricket_2;  
  
# Q2. Write a MySQL query to find the players from the test match 1 having popularity higher than the average popularity.  
select player_name , Popularity from cricket_1 WHERE Popularity > (SELECT AVG(Popularity) FROM cricket_1);  
  
# Q3. Find player_id and player name that are common in the test match 1 and test match 2.  
SELECT player_id , player_name FROM cricket_1  
WHERE cricket_1.player_id IN (SELECT player_id FROM cricket_2);  
  
# Q4. Retrieve player_id, runs, and player_name from cricket_1 table and display list of the players where the runs are more than the average runs.  
SELECT player_id , runs , player_name FROM cricket_1 WHERE runs>(SELECT AVG(runs) FROM cricket_1);  
  
# Q5. Write a query to extract the player_id, runs and player_name from the table "cricket_1" where the runs are greater than 50.  
SELECT player_id , runs , player_name FROM cricket_1  
WHERE runs > 50 ;  
  
# Q6. Write a query to extract all the columns from cricket_1 where player_name starts with 'y' and ends with 'v'.  
SELECT * FROM cricket_1 WHERE player_name LIKE 'y%v';  
  
# Q7. Write a query to extract all the columns from cricket_1 where player_name does not end with 't'.  
SELECT * FROM cricket_1 WHERE player_name NOT LIKE '%t';
```

Answers:-

Q-1:-

The screenshot shows the MySQL Workbench interface. At the top, there are two tabs: 'cricket_1' and 'cricket_2'. Below the tabs is a toolbar with various icons. The main area contains a SQL query:

```
1 select Player_Name from cricket_1
2 union
3 select Player_Name from cricket_2;
```

Below the query results is a 'Result Grid' section. It has a header row with a single column labeled 'Player_Name'. The data rows are as follows:

Player_Name
Virat
Rohit
Jadeja
Dhoni
Dhawan
Yadav
Raina
Binny
Rayudu
rahane
A. Patel
B. Kumar
Yuvraaj
Tendulkar
Dravid
Yusuf

Q-2:-

cricket_1 x

The screenshot shows a MySQL Workbench interface with a query editor and a results grid. The query is:

```
1 select player_name, Popularity from cricket_1
2 where Popularity > (select avg(Popularity) from cricket_1);
3
4
```

The results grid displays the following data:

player_name	Popularity
Virat	10
Dhoni	15
Dhawan	12
Yadav	10
Binny	11
Rayudu	12
rahane	10

Q-3:-

cricket_1 x

The screenshot shows a MySQL Workbench interface with a query editor and a results grid. The query is:

```
1 • select player_name, player_id from cricket_1 c
2 where c.Player_Id in (select Player_Id from cricket_2);
3
4
```

The results grid displays the following data:

player_name	player_id
Virat	PL1
Rohit	PL2
Jadeja	PL3
Dhoni	PL4
Yadav	PL6
Binny	PL8
Rayudu	PL9

Q-4

cricket_1 ×

1 • select player_id,player_name,runs from cricket_1 where runs>(select avg(runs) from cricket_1);
2
3

Result Grid | Filter Rows: Export: Wrap Cell Content:

	player_id	player_name	runs
▶	PL1	Virat	50
	PL2	Rohit	41
	PL5	Dhawan	45
	PL6	Yadav	66
	PL8	Binny	44
	PL9	Rayudu	63

Q-5

cricket_1 ×

1 • select player_id,player_name,runs from cricket_1 where runs>50;
2
3

Result Grid | Filter Rows: Export: Wrap Cell Content:

	player_id	player_name	runs
▶	PL6	Yadav	66
	PL9	Rayudu	63

Q-6

cricket_1 ×

1 • select * from cricket_1 where Player_Name like 'y%v';
2
3

< Result Grid Filter Rows: Export: Wrap Cell Content:

Player_Id	Player_Name	Runs	Popularity
PL6	Yadav	66	10

Q-7

cricket_1 ×

1 • select * from cricket_1 where Player_Name not like '%t';
2
3

< Result Grid Filter Rows: Export: Wrap Cell Content:

Player_Id	Player_Name	Runs	Popularity
PL3	Jadeja	33	6
PL4	Dhoni	35	15
PL5	Dhawan	45	12
PL6	Yadav	66	10
PL7	Raina	32	9
PL8	Binny	44	11
PL9	Rayudu	63	12
PL10	rahane	21	10
PL11	A. Patel	12	9
PL12	B. Kumar	30	7

cricket_1 ×

1 • select player_id, substr(player_id,3) as Modified_Player_Id from cricket_1;

2

3

Result Grid | Filter Rows: [] | Export: [] | Wrap Cell Content: []

	player_id	Modified_Player_Id
▶	PL1	1
	PL2	2
	PL3	3
	PL4	4
	PL5	5
	PL6	6
	PL7	7
	PL8	8
	PL9	9
	PL10	10
	PL11	11
	PL12	12

-- Dataset Used: new_cricket.csv

Q11. Extract the Player_Id and Player_name of the players where the charisma value is null.
SELECT Player_Id , Player_Name FROM new_cricket WHERE Charisma IS NULL;

Q12. Separate all Player_Id into single numeric ids (example PL1 = 1).
SELECT Player_Id, SUBSTR(Player_Id,3)
FROM new_cricket;

Q13. Write a MySQL query to extract Player_Id , Player_Name and charisma where the charisma is greater than 25.
SELECT Player_Id , Player_Name , charisma FROM new_cricket WHERE charisma > 25;]

```
# Question 1:  
# 1) Create a Database bank  
/* Solution */  
CREATE DATABASE bank;  
use bank;  
  
# Question 2:  
# 2) Create a table with the name "bank_details" with the following columns  
-- Product with string data type  
-- Quantity with numerical data type  
-- Price with real number data type  
-- purchase_cost with decimal data type  
-- estimated_sale_price with data type float  
/* Solution */  
Create table bank_details(  
Product CHAR(10) ,  
quantity INT,  
price Real,  
purchase_cost Decimal(6,2),  
estimated_sale_price Float);
```

Create table bank_details

The screenshot shows the MySQL Workbench interface with a connection named 'cricket_1'. In the SQL tab, the following code is executed:

```
1  create table bank_details(
2      Product char(10),
3      quantity int,
4      price real,
5      purchase_cost decimal(6,2),
6      estimated_sale_price float);
7
8 •    describe bank_details;
```

Below the SQL tab is a Result Grid showing the table structure:

Field	Type	Null	Key	Default	Extra
Product	char(10)	YES		NULL	
quantity	int	YES		NULL	
price	double	YES		NULL	
purchase_cost	decimal(6,2)	YES		NULL	
estimated_sale_price	float	YES		NULL	

Insert two records:-

The screenshot shows the MySQL Workbench interface with a connection named 'cricket_1'. In the SQL tab, the following code is executed:

```
1      insert into bank_details
2          values('paycard',3,330,8008,9009),
3          ('paypoints',4,300,8000,6800);
4
5 •    select * from bank_details;
```

Below the SQL tab is a Result Grid showing the inserted data:

	Product	quantity	price	purchase_cost	estimated_sale_price
▶	paycard	3	330	8008.00	9009
	paypoints	4	300	8000.00	6800

Adding the new column geo location:-

The screenshot shows the MySQL Workbench interface with a connection named 'cricket_1'. In the SQL tab, two queries are run:

```
1 • alter table bank_details add column geo_location varchar(20);
2 • describe bank_details;
```

The results of the 'describe' query are displayed in a Result Grid:

Field	Type	Null	Key	Default	Extra
Product	char(10)	YES		NULL	
quantity	int	YES		NULL	
price	double	YES		NULL	
purchase_cost	decimal(6,2)	YES		NULL	
estimated_sale_price	float	YES		NULL	
geo_location	varchar(20)	YES		NULL	

Getting the geo location of the paycard product:-

The screenshot shows the MySQL Workbench interface with a connection named 'cricket_1'. In the SQL tab, a query is run:

```
1 • select geo_location from bank_details where product='paycard';
```

The results of the query are displayed in a Result Grid:

geo_location
NULL

cricket_1 x bank_details

```
1 •  select char_length(Product) from bank_details where Product='paycard';
2 |
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content: |

	char_length(Product)
▶	7

Altering the table column's datatype

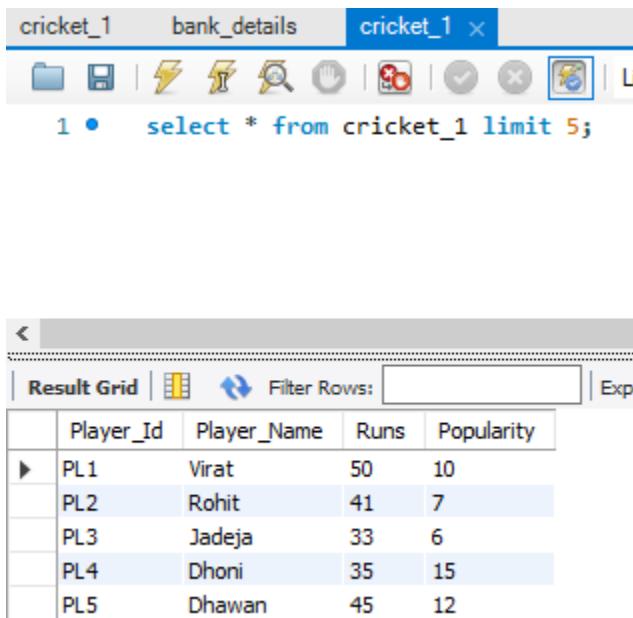
cricket_1 x bank_details

```
1 •  describe bank_details;
2 •  alter table bank_details modify product varchar(20);
3 |
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content: |

	Field	Type	Null	Key	Default	Extra
▶	product	varchar(20)	YES		NULL	
	quantity	int	YES		NULL	
	price	double	YES		NULL	
	purchase_cost	decimal(6,2)	YES		NULL	
	estimated_sale_price	float	YES		NULL	
	geo_location	varchar(20)	YES		NULL	

Displaying only five records



The screenshot shows a MySQL Workbench interface. The top bar has tabs for 'cricket_1' and 'bank_details', with 'cricket_1' currently selected. Below the tabs is a toolbar with various icons. A query editor window is open with the following SQL code:

```
1 •  select * from cricket_1 limit 5;
```

Below the query editor is a results grid titled 'Result Grid'. The grid has columns: Player_Id, Player_Name, Runs, and Popularity. It displays the following data:

	Player_Id	Player_Name	Runs	Popularity
▶	PL1	Virat	50	10
	PL2	Rohit	41	7
	PL3	Jadeja	33	6
	PL4	Dhoni	35	15
	PL5	Dhawan	45	12

