Table of Contents

What is Database?

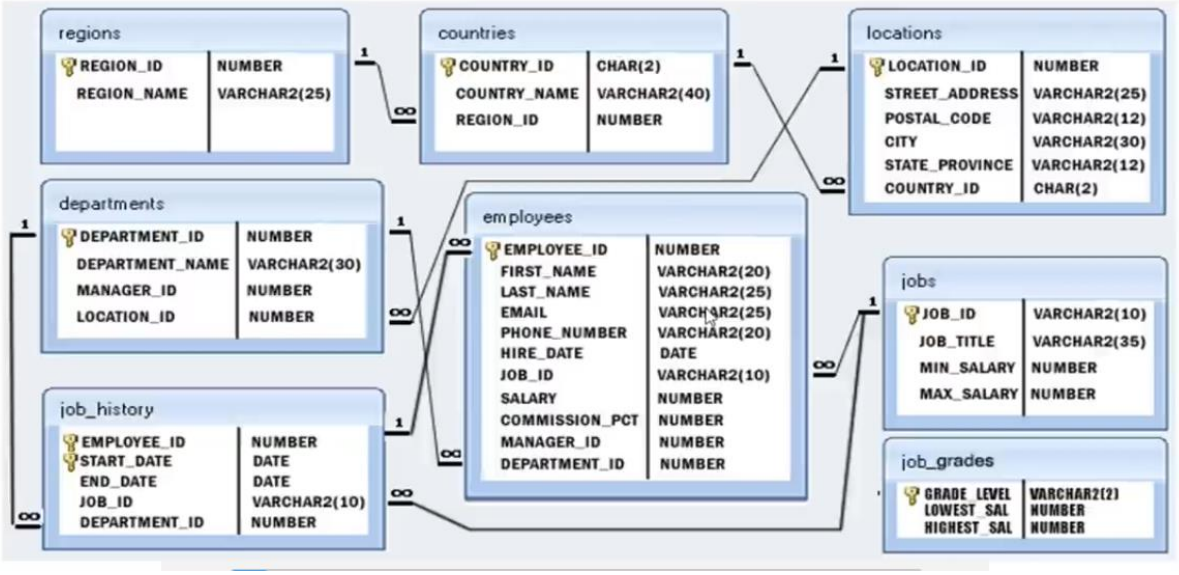
Database is a collection of information organized for easy access, management and maintenance.

- 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.

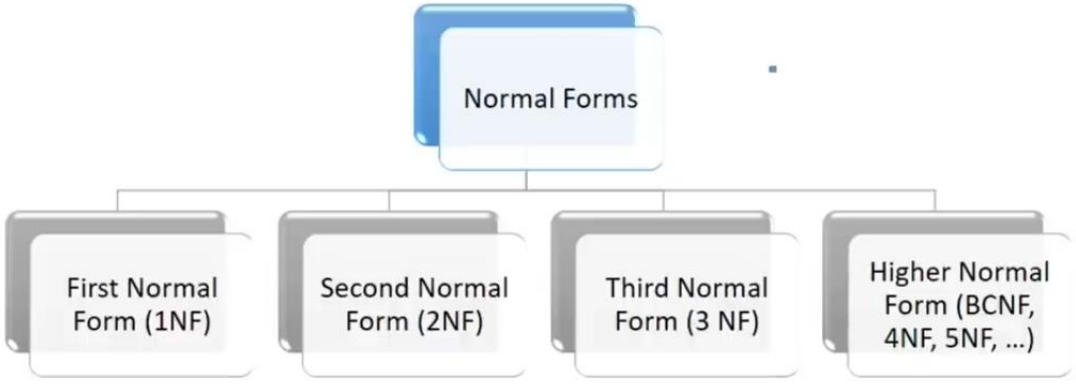
- There can be, but there will be no "relation" between the tables, like in a RDBMS. In traditional approach, data is generally stored in either a hierarchical form/navigational form. This means that a single data unit will have 0, 1 or more children nodes and 1 parent node.

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**

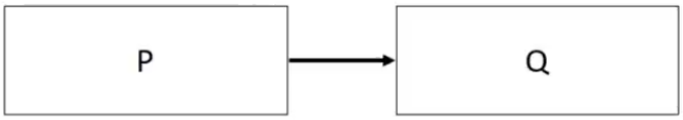
Student_Details	Course_details		Pre-requisite	Result_details	
101 Jack 11/4/1975	M1 Advance Math	17	Basic Math	03/11/2015 82	A
102 Rock 10/04/1976	P4 Advance Physics	18	Basic Physics	22/11/2015 83	A
103 Mary 11/07/1975	B3 Advance Biology	10	Basic Biology	14/11/2015 68	B
104 Roby 10/04/1976	H6 Advance 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

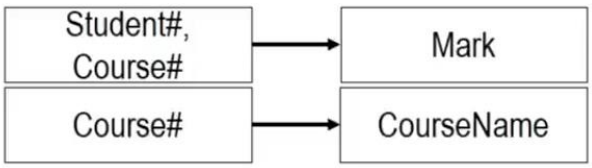
Partial Functional Dependency

Transitive Dependency

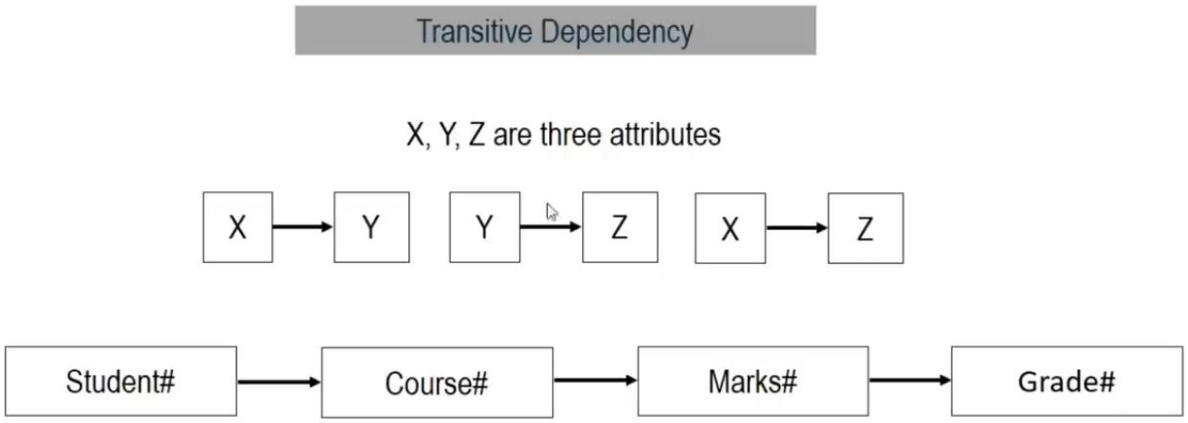
Functional Dependency Types

Partial Functional Dependency

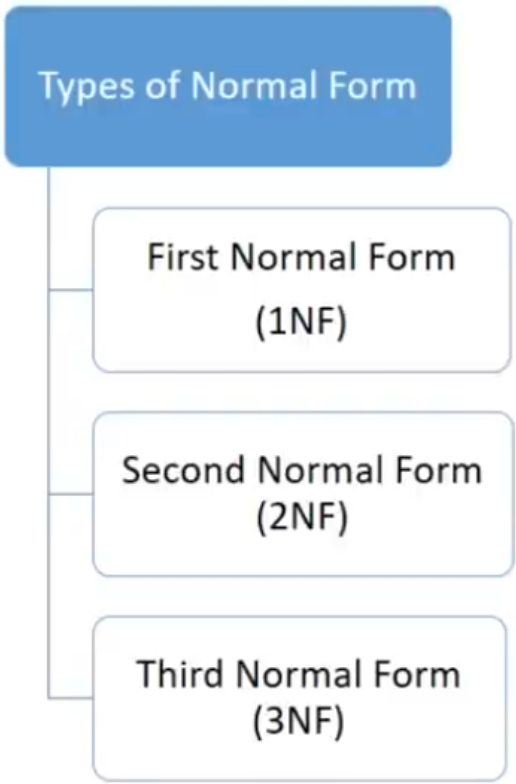
- Attribute Q is partially dependent on attribute P, if and only if it is dependent on the subset of attribute P.
- REPORT (Student#, Course#, StudentName, CourseName, Marks, Grade)



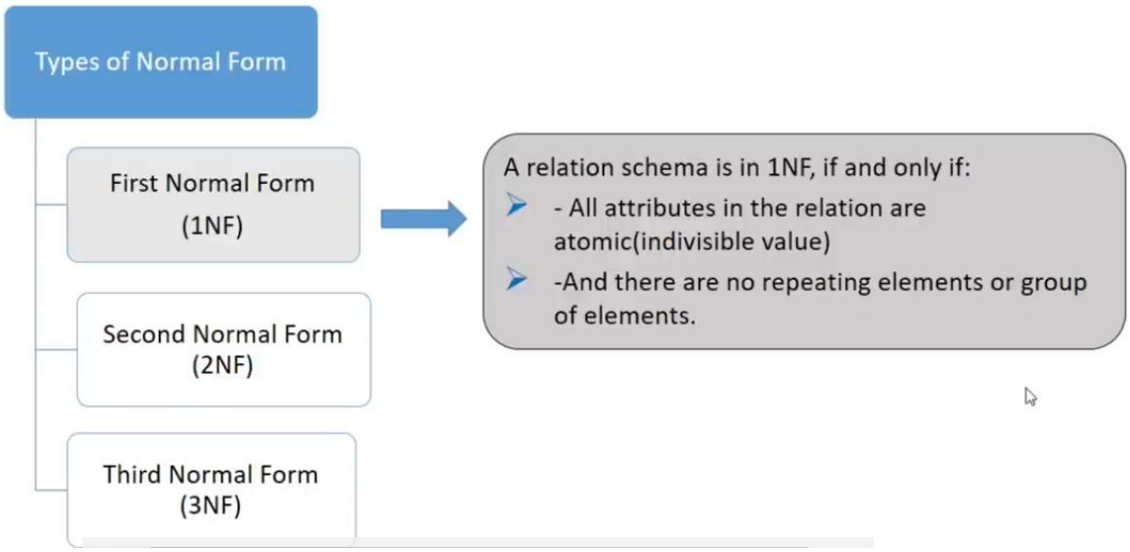
Functional Dependency Types



Normalization



First Normal Form – (1NF)



First Normal Form – (1NF)

Student Marks Table

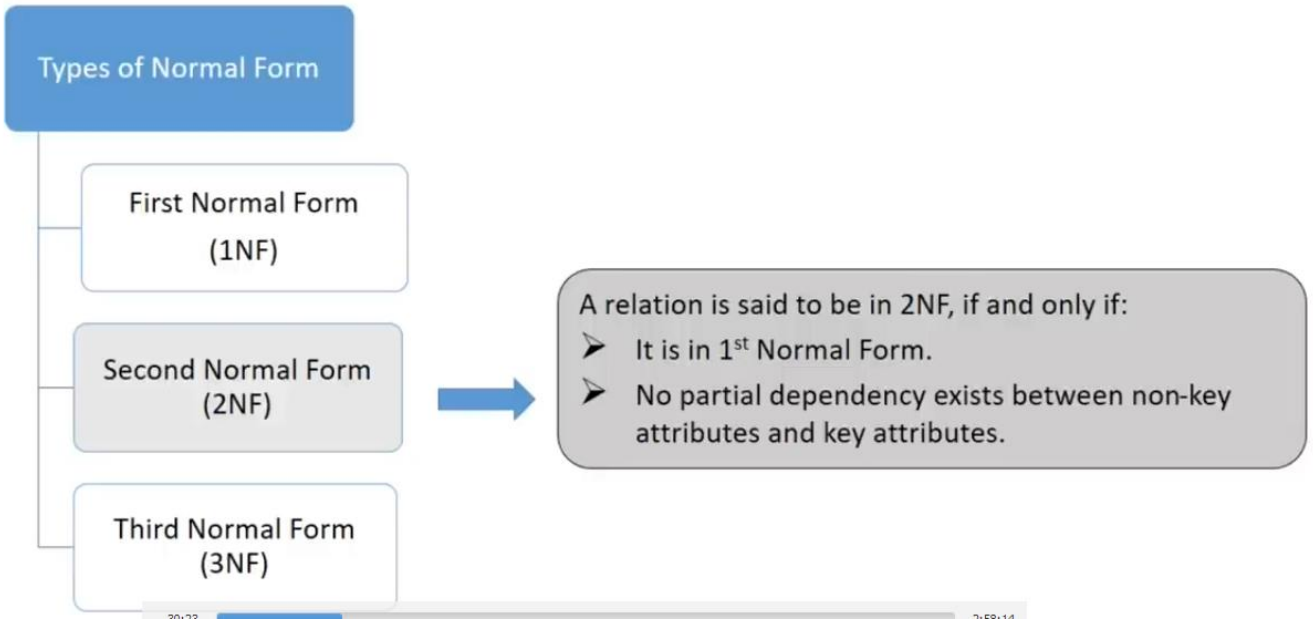
Student_Details	Course_details		Pre-requisite	Result_details	
101 Jack 11/4/1975	M1 Advance Math	17	Basic Math	03/11/2015 82	A
102 Rock 10/04/1976	P4 Advance Physics	18	Basic Physics	21/11/2015 83	A
103 Mary 11/07/1975	B3 Advance Biology	10	Basic Biology	12/11/2015 68	B
104 Roby 10/04/1976	H6 Advance 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 in1NF

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

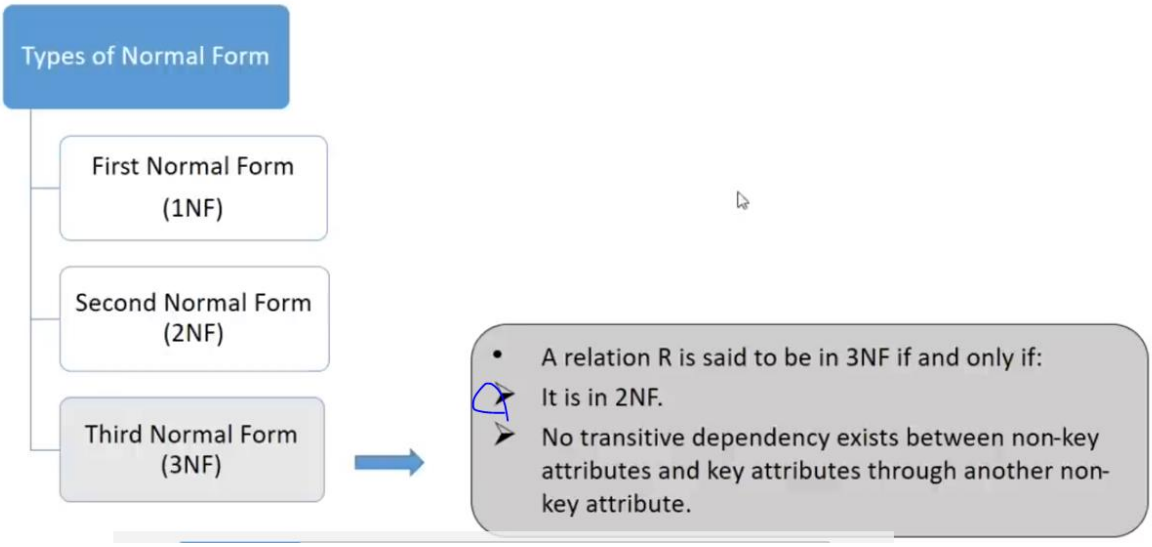
Result Table

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

Course Table

Course#	CourseName	Prerequisite	Durationindays	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

Student#	Course#	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(

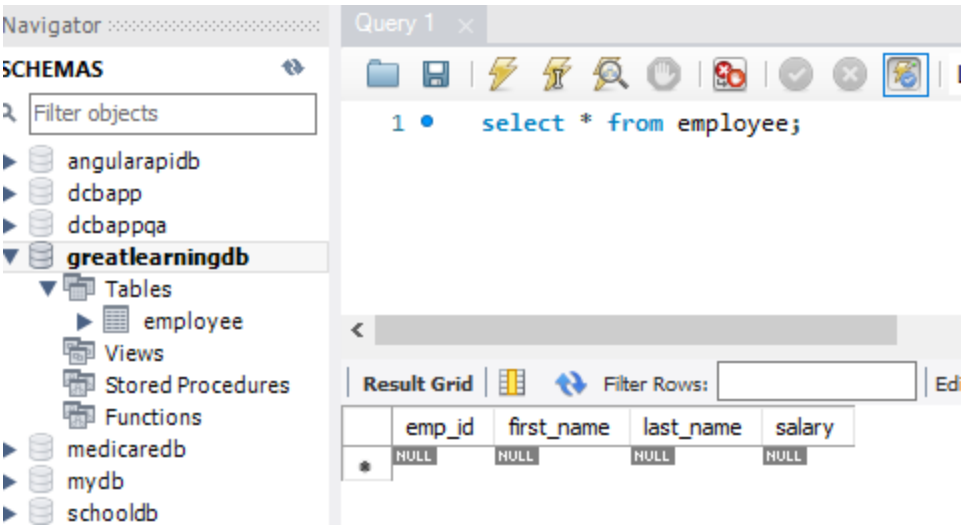
emp_id int not null,

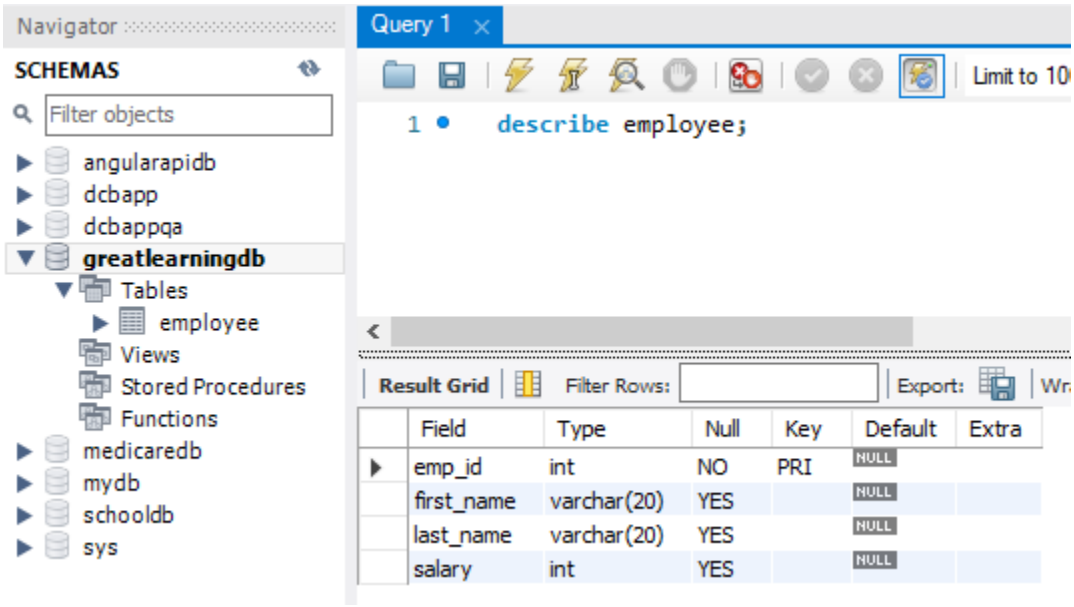
first_name varchar(20),

last_name varchar(20),

salary int,

primary key(emp_id));

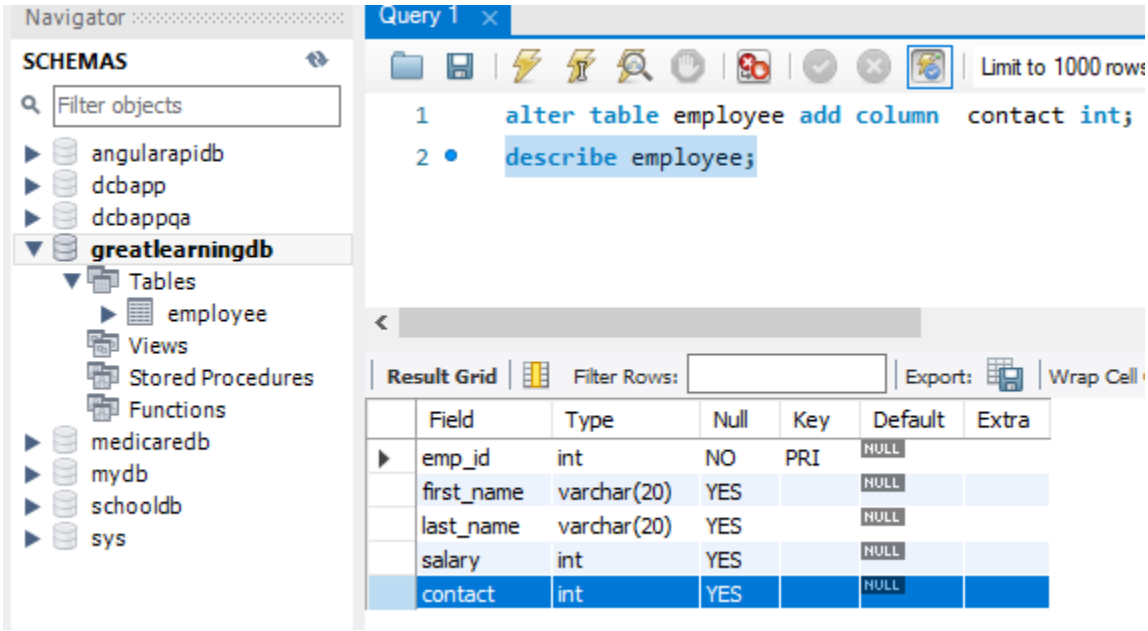

Navigator
SCHEMAS
Filter objects
angularapidb
dcbapp
dcbappqa
greatlearningdb
Tables
employee
Views
Stored Procedures
Functions
medicaredb
mydb
schooldb
Query 1
select * from employee;
Result Grid
Filter Rows:
emp_id
first_name
last_name
salary
NULL


Navigator
SCHEMAS
Filter objects
angularapidb
dcbapp
dcbappqa
greatlearningdb
Tables
employee
Views
Stored Procedures
Functions
medicaredb
mydb
schooldb
sys
Query 1
Limit to 10
describe employee;
Result Grid
Filter Rows:
Export:
Field
Type
Null
Key
Default
Extra
emp_id
int
NO
PRI
first_name
varchar(20)
YES
last_name
varchar(20)
YES
salary
int
YES
NULL

DDL - Data Definition Language – Alter Command

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

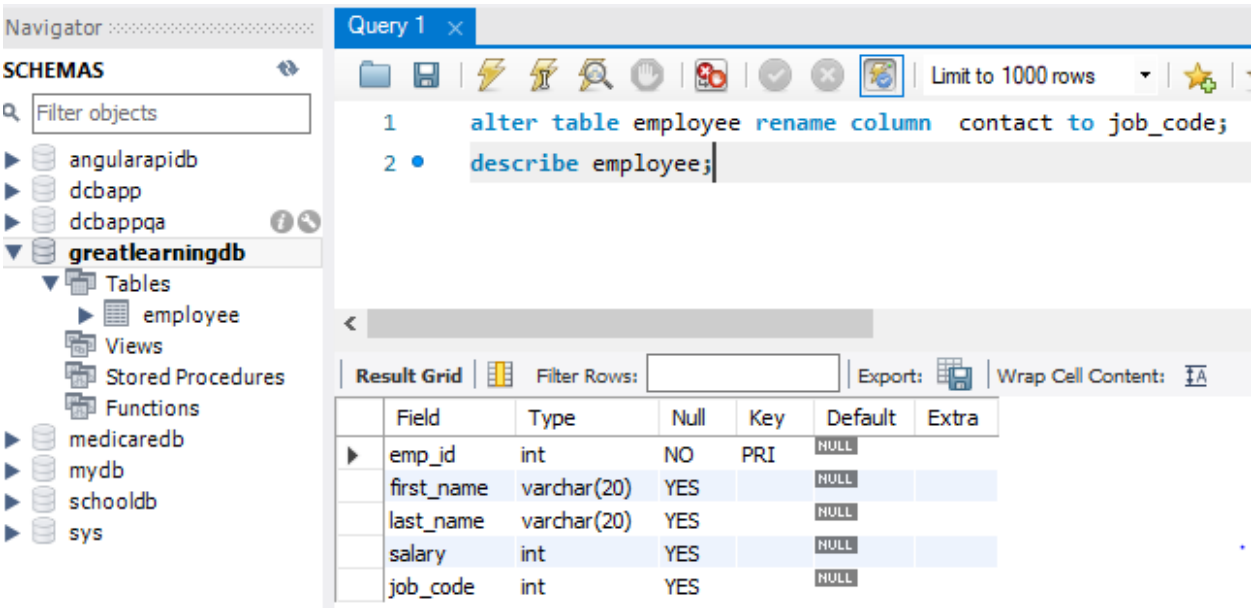
emp_id	first_name	last_name	salary	contact
101	Steven	Cohen	10000	
102	Edwin	Thom	15000	
103	Harry	Potter	20000	



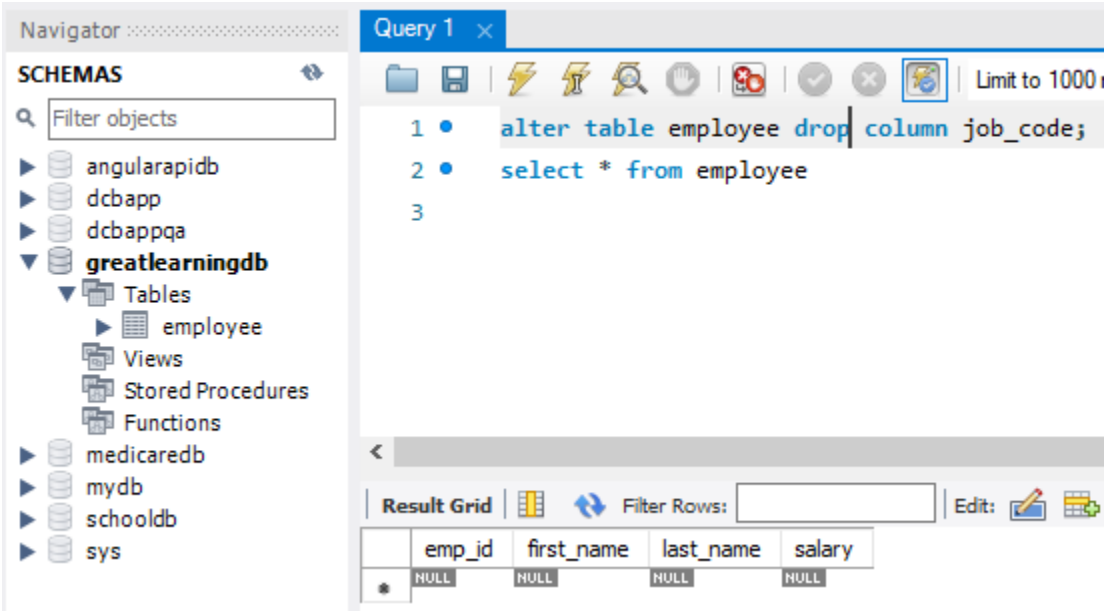
DDL - Data Definition Language – Rename Command

```
ALTER TABLE employeeb 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	



Drop column command:-

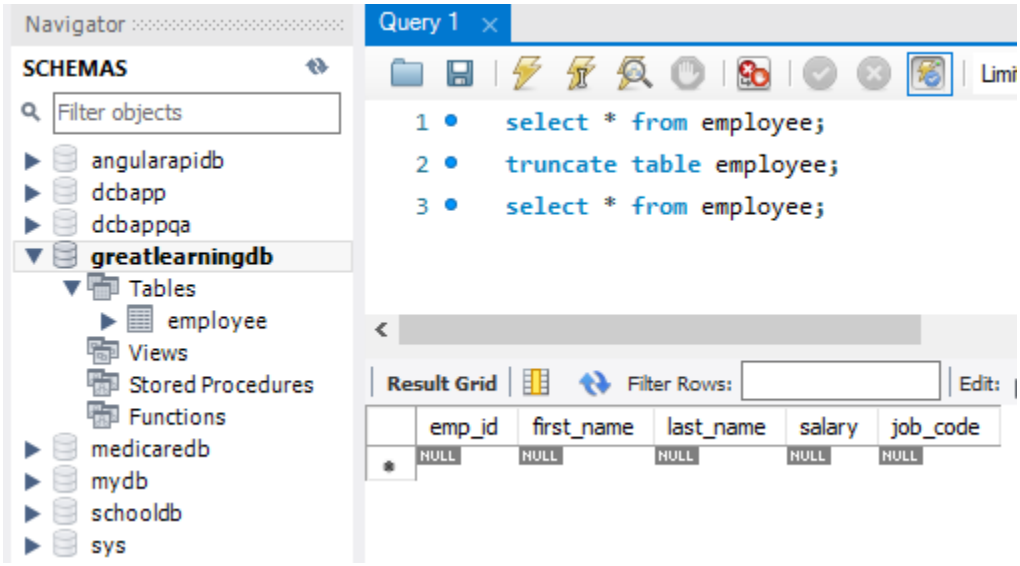


Truncate command is used to empty(delete all records from the table) the table:

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



Drop table command is used to delete the entire table

DDL - Data Definition Language – Drop Command

```
DROP TABLE table_name;

DROP TABLE employees;
```

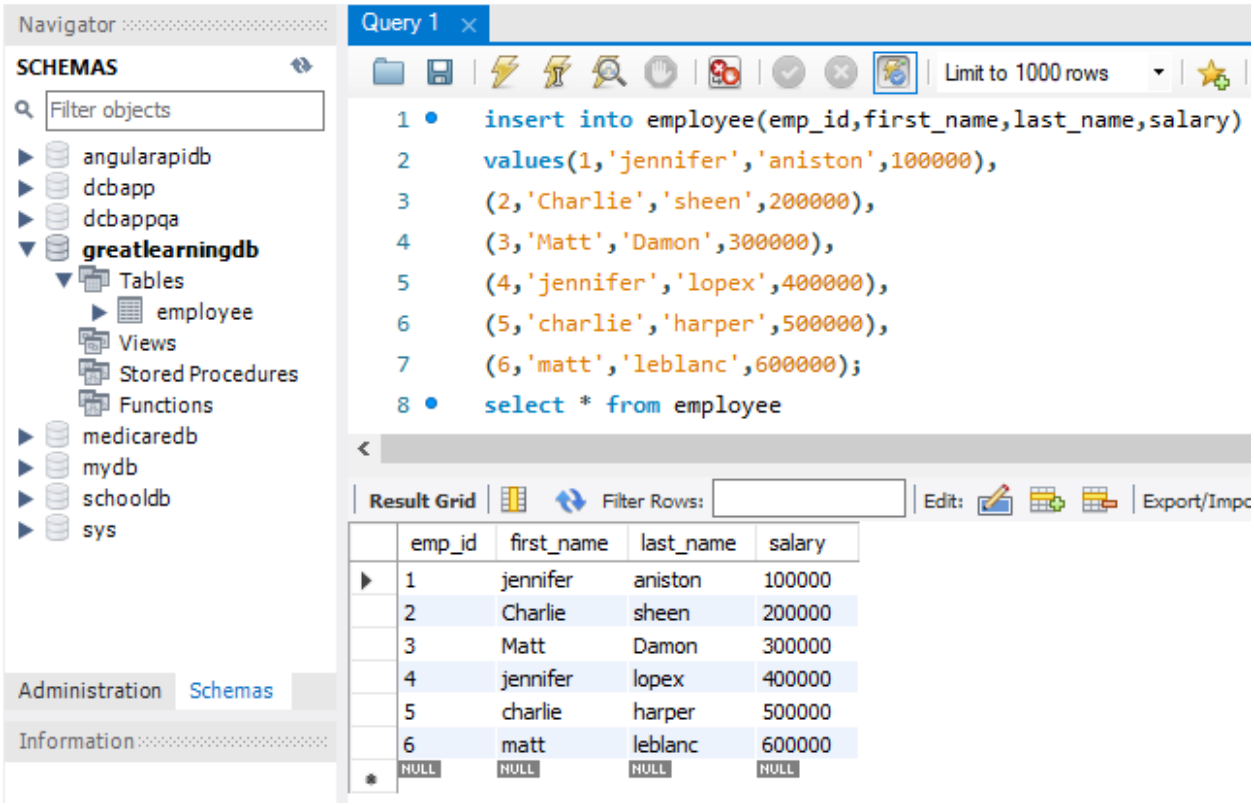
DML – Data Manipulation Language

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);
```

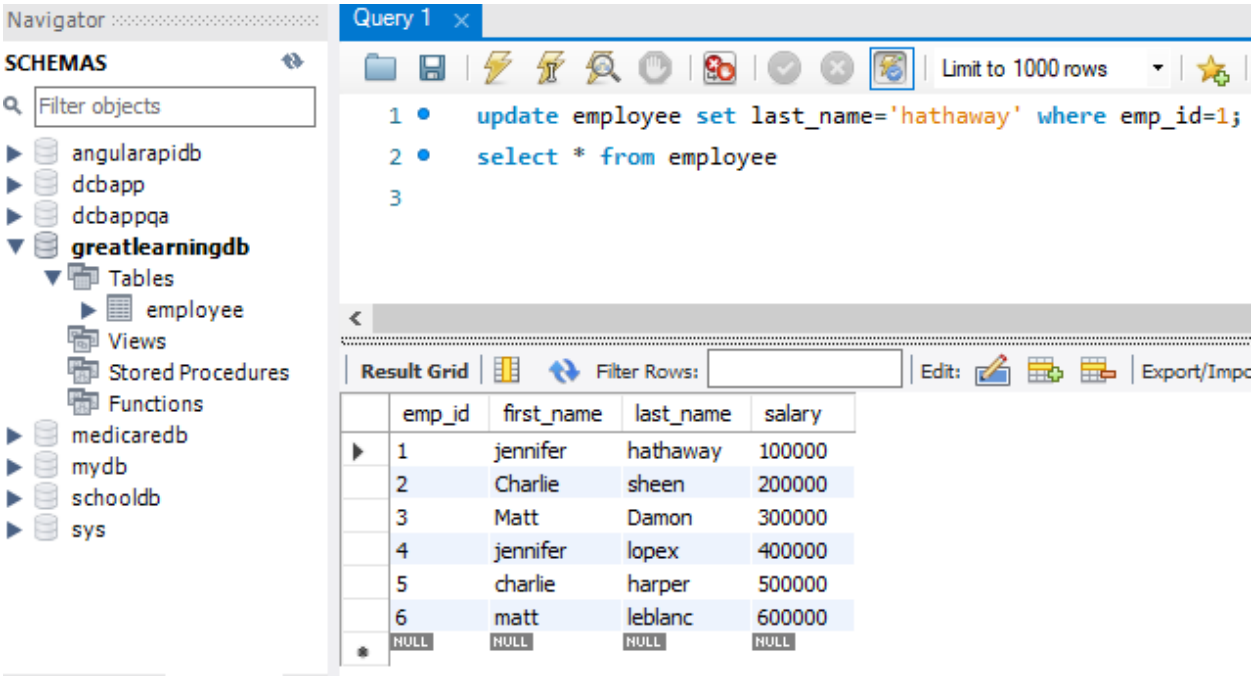
emp_id	first_name	last_name	salary
101	Steven	King	10000
102	Edwin	Thomas	15000
103	Harry	Potter	20000



DML – Data Manipulation Language – UPDATE Command

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

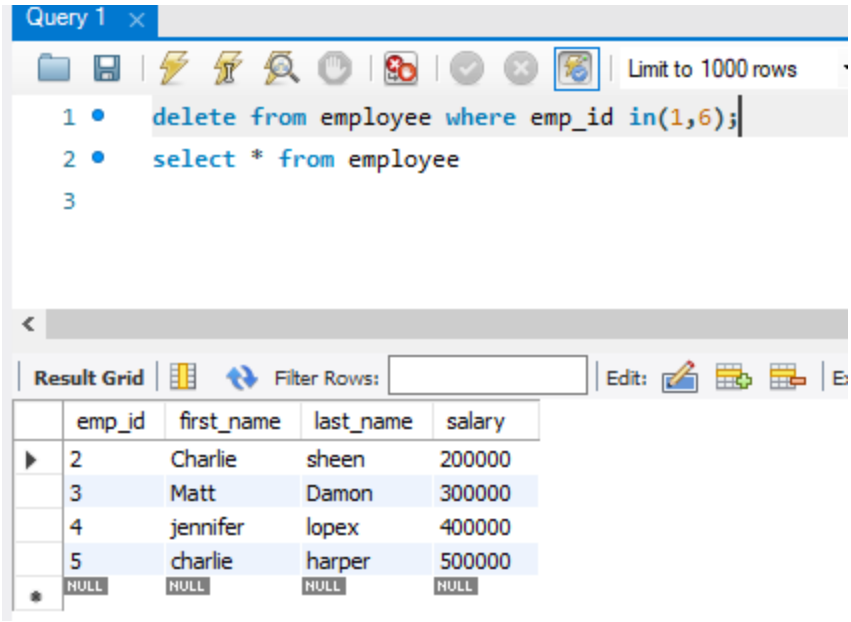
emp_id	first_name	last_name	salary
101	Steven	Cohen	10000
102	Edwin	Thomas	15000
103	Harry	Potter	20000



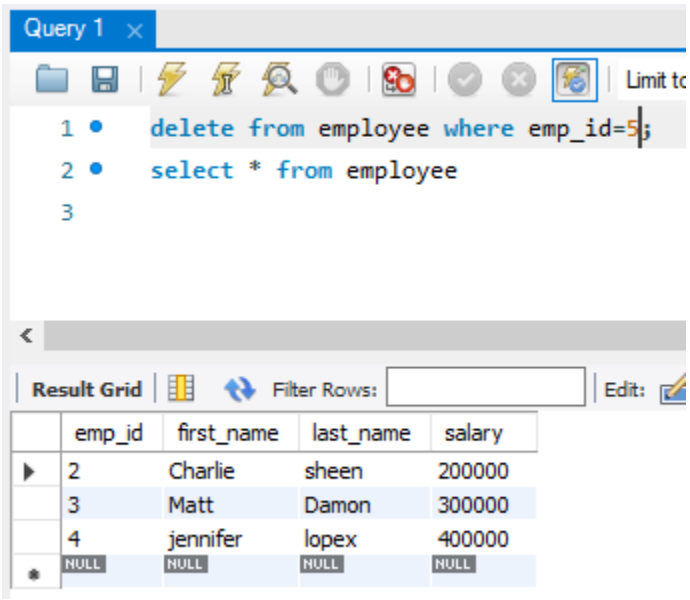
DML – Data Manipulation Language - DELETE Command

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

emp_i



Delete where command



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.,

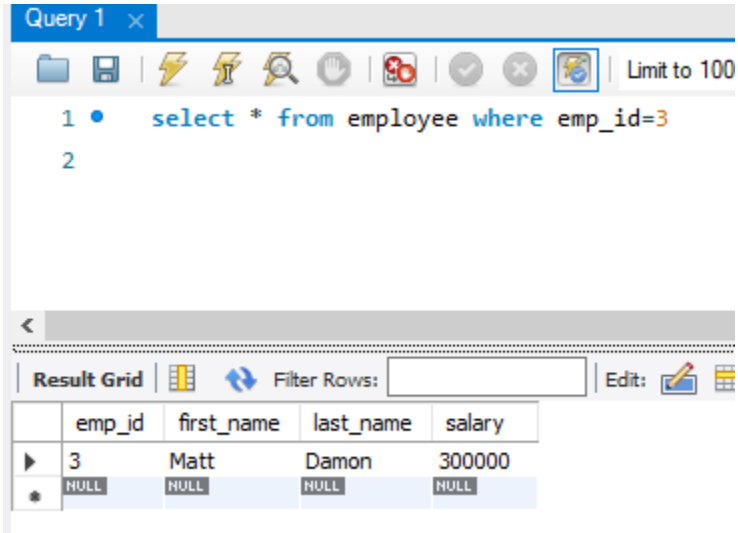
e.g.

```
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

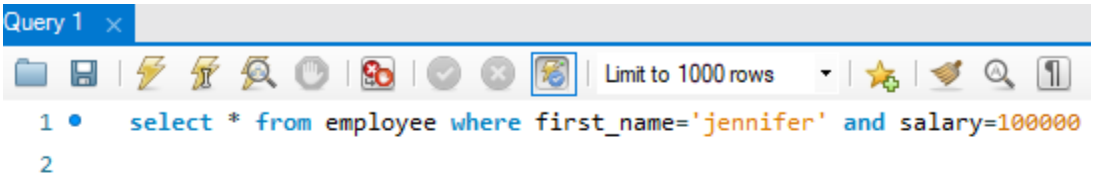


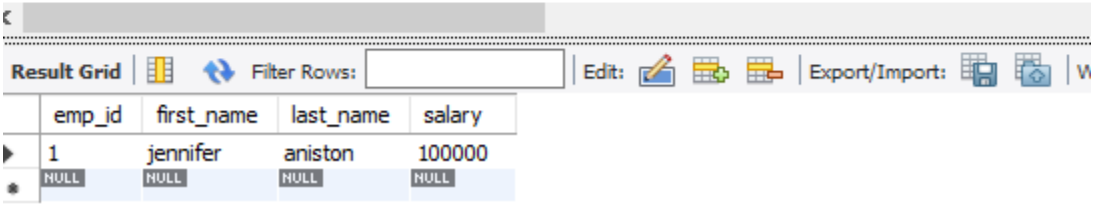
SQL Operators – Logical

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

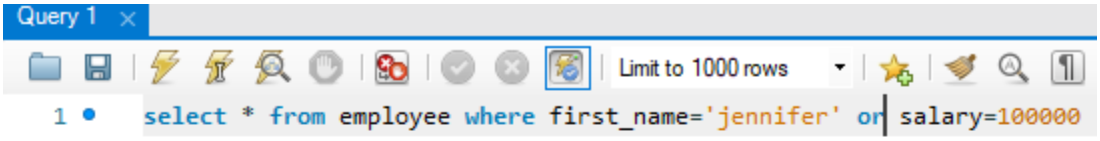
Sample Queries:

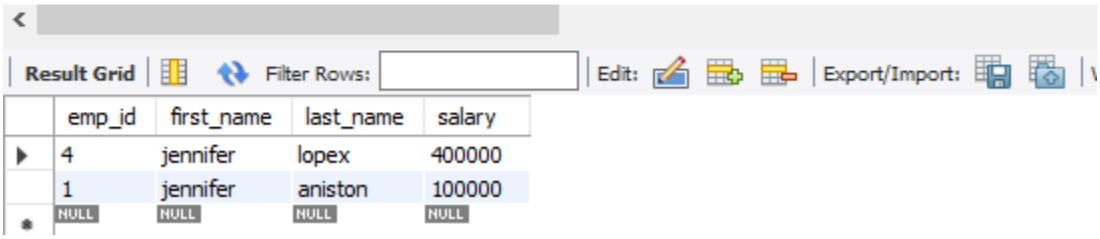
```
SELECT * FROM employees WHERE first_name = 'Steven' and salary = 15000;

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

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





Or keyword:-





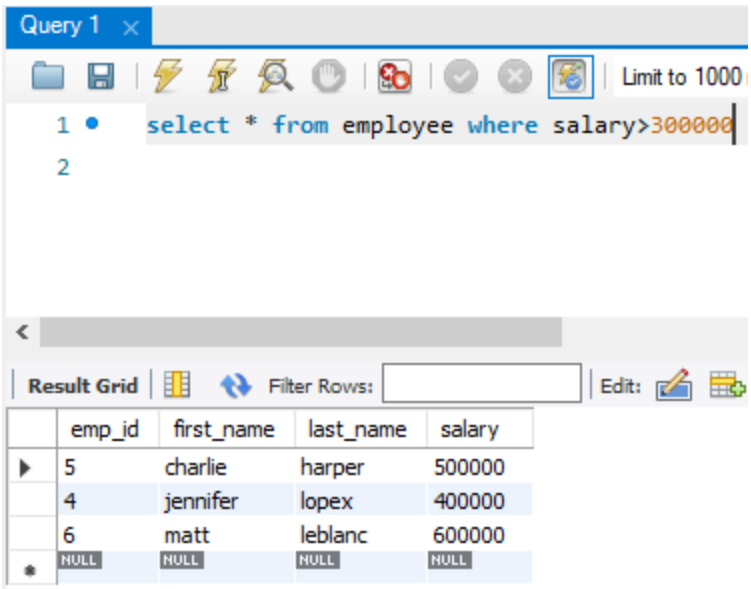
emp_id	first_name	last_name	salary
4	jennifer	lopex	400000
1	jennifer	aniston	100000
NULL	NULL	NULL	NULL

SQL Operators – Comparison

Comparison Operators	
Symbol	Meaning
=	Equal to
>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to
<> or !=	Not equal to

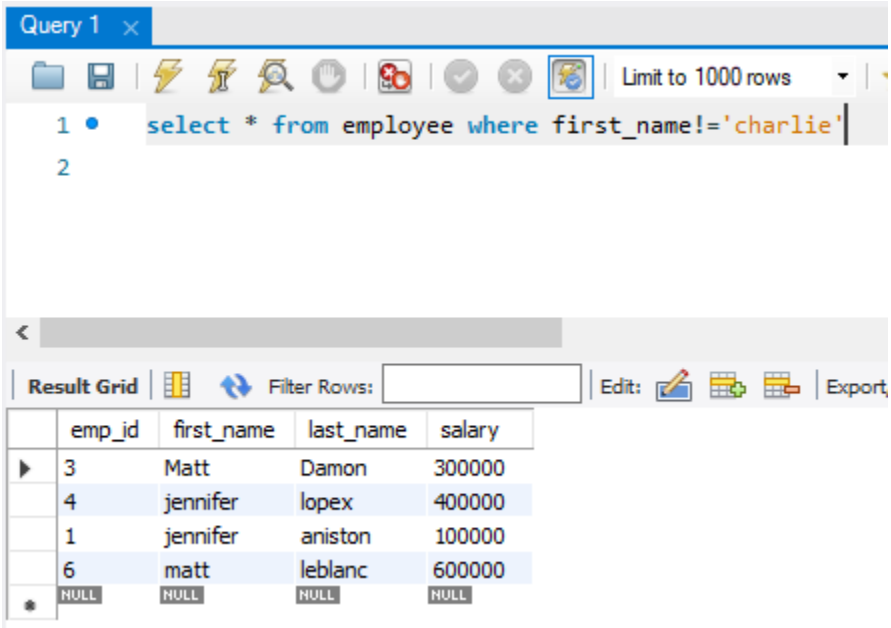
Sample Queries:

```
SELECT * FROM employees WHERE first_name = 'Steven'
AND salary <=10000;

SELECT * FROM employees WHERE first_name = 'Steven'
OR salary >=10000;
```



Result Grid | Filter Rows: | Edit:

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



Result Grid | Filter Rows: | Edit: | Export

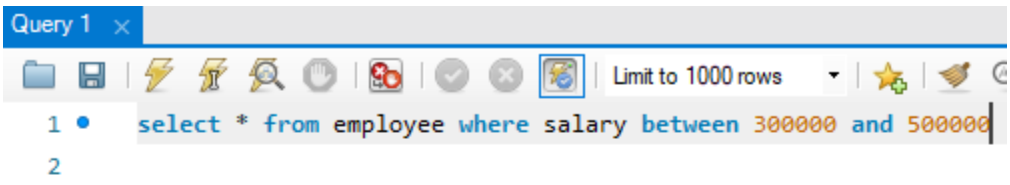
emp_id	first_name	last_name	salary
3	Matt	Damon	300000
4	jennifer	lopex	400000
1	jennifer	aniston	100000
6	matt	leblanc	600000
NULL	NULL	NULL	NULL

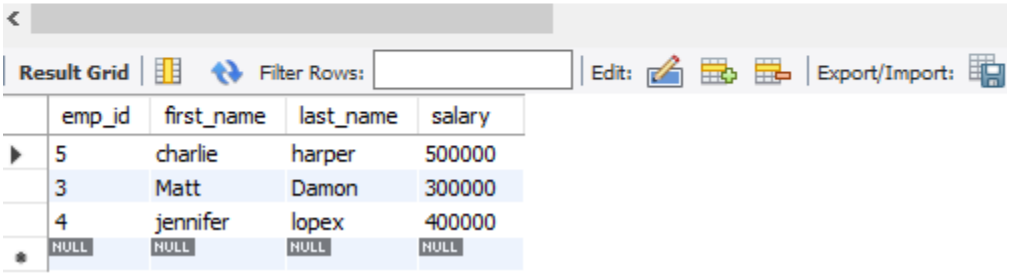
SQL Operators – Special

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

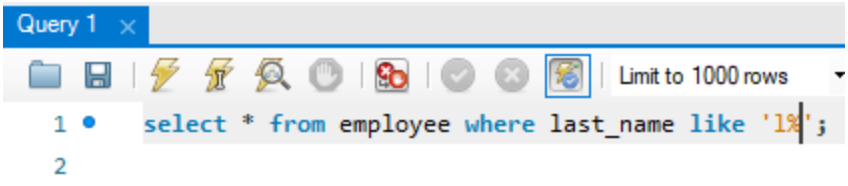
Sample Queries:

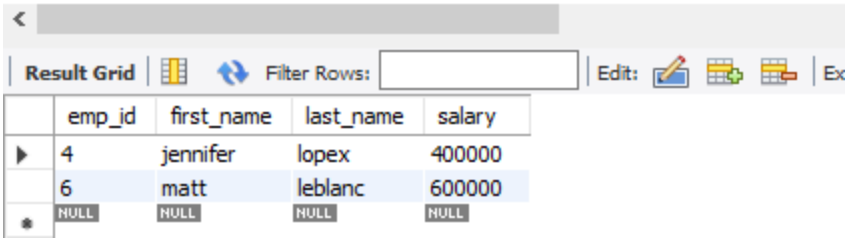
```
SELECT * FROM employees WHERE salary
between 10000 and 20000;

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;
```

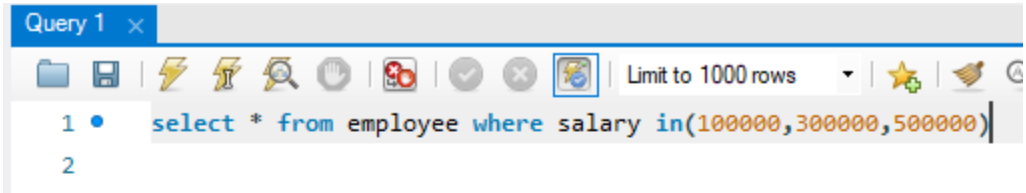


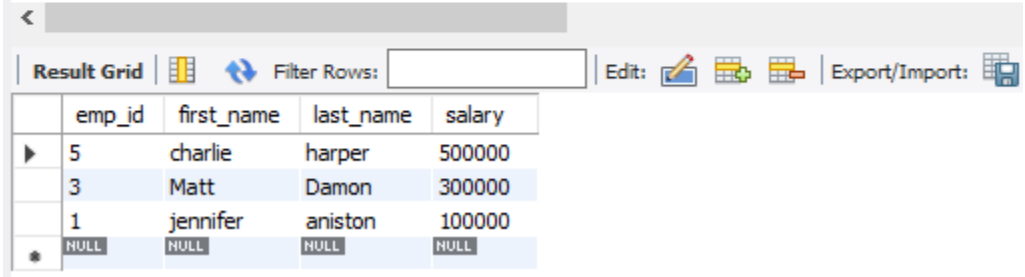


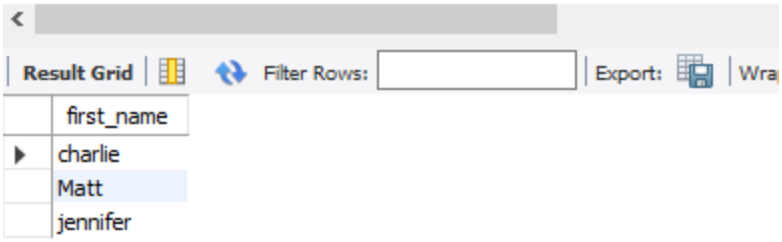
emp_id	first_name	last_name	salary
5	charlie	harper	500000
3	Matt	Damon	300000
4	jennifer	lopex	400000
NULL	NULL	NULL	NULL


Query 1
Limit to 1000 rows
1 select * from employee where last_name like 'l%';
2


Result Grid
Filter Rows:
Edit:
emp_id first_name last_name salary
4 jennifer lopex 400000
6 matt leblanc 600000
NULL NULL NULL NULL


Query 1
Limit to 1000 rows
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

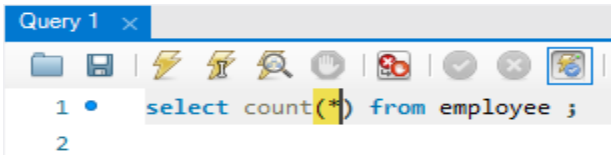


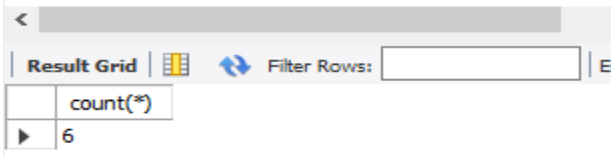
SQL Operators – Aggregations

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

Limit to 1000 rows

```
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

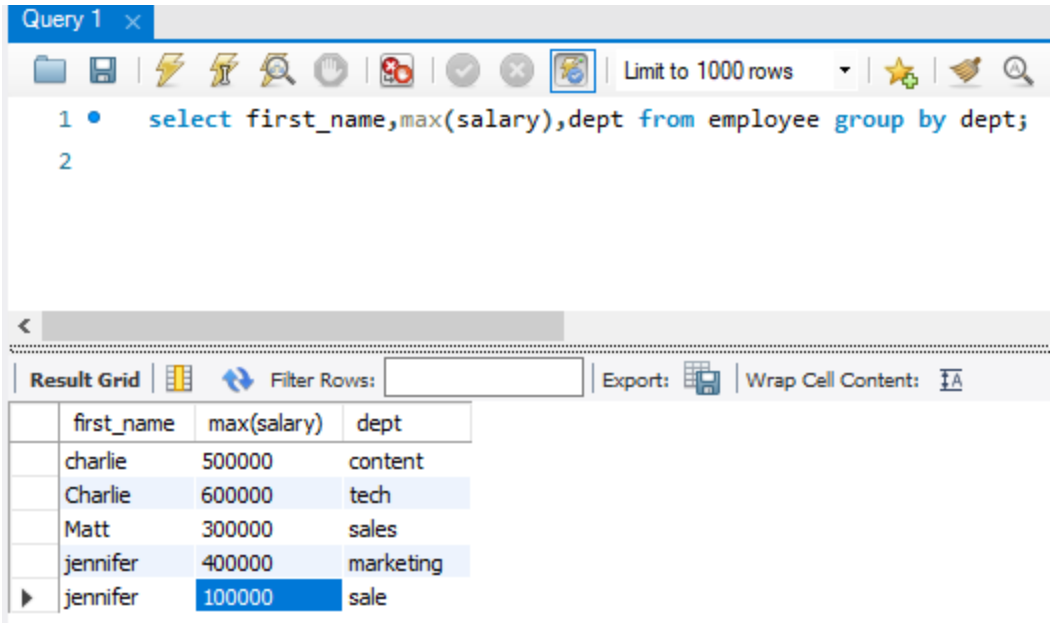
SQL GROUP BY Clause

- 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



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

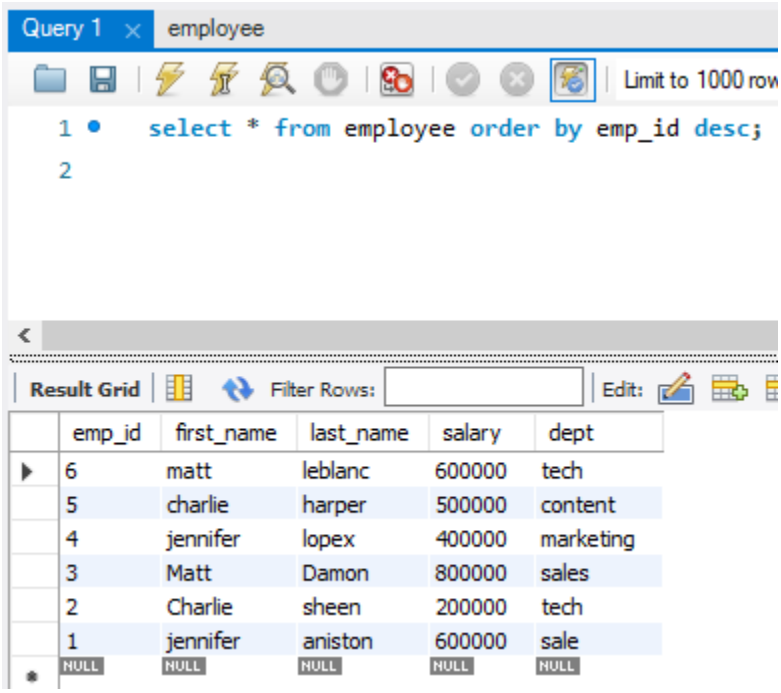
- Used to sort output of SELECT statement
- Default is to sort in ASC (Ascending)
- Can Sort in Reverse (Descending) Order with "DESC" after the column name

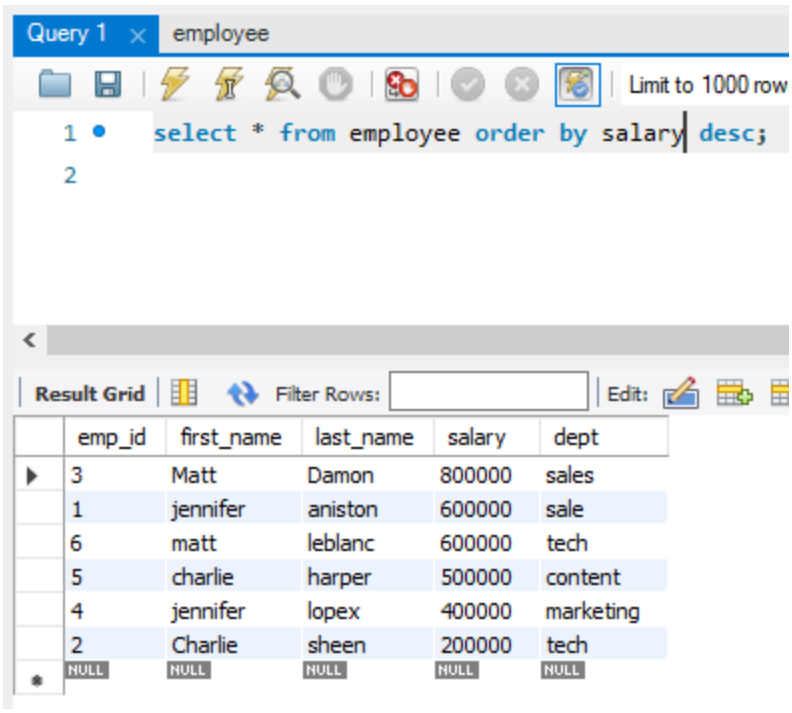
employee_id	first_name	last_name	salary
101	Steven	Cohen	10000
102	Edwin	Thomas	15000
103	Harry	Potter	20000

e.g.,

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

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





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	lopex	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

CATEGORY_ID	PRODUCT_NAME
1	Nokia
2	Samsung
3	HP
6	Nikon

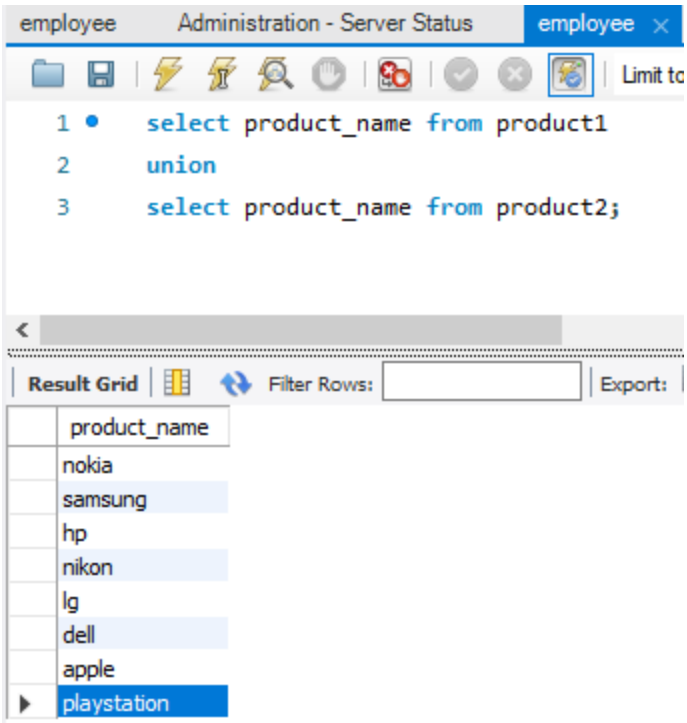
Product2

CATEGORY_ID	PRODUCT_NAME
1	Samsung
2	LG
3	HP
5	Dell
6	Apple
10	Playstation

e.g.,

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

PRODUCT_NAME
Nokia
Samsung
HP
Nikon
LG
Dell
Apple
Playstation

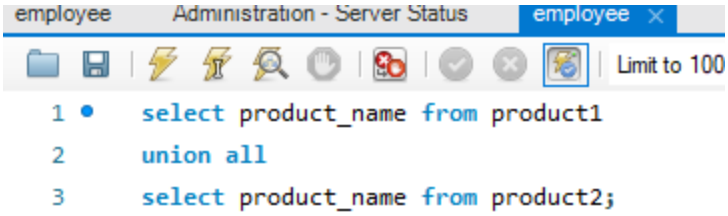


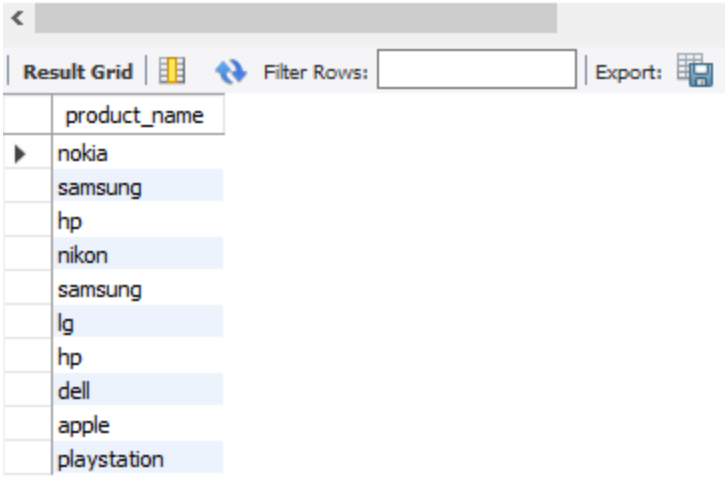
SQL UNION ALL

- 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;
```





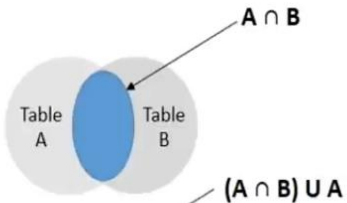
product_name
nokia
samsung
hp
nikon
samsung
lg
hp
dell
apple
playstation

SQL Joins

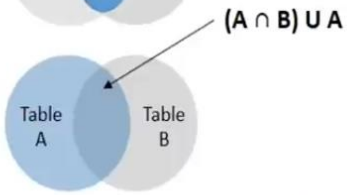
SQL JOINS

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

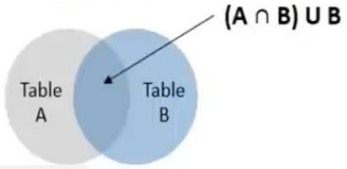
- **INNER JOIN** – Returns rows when there is a match in both tables.



- **LEFT JOIN** – Returns all rows from the left table, even if there are no matches in the right table.

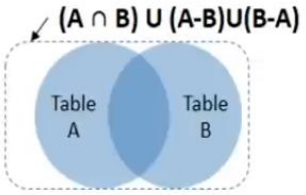


- **RIGHT JOIN** – Returns all rows from the right table, even if there are no matches in the left table.



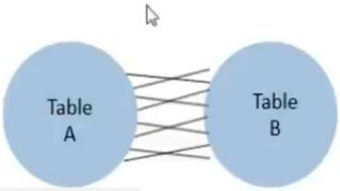
SQL JOINS

- **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;
```

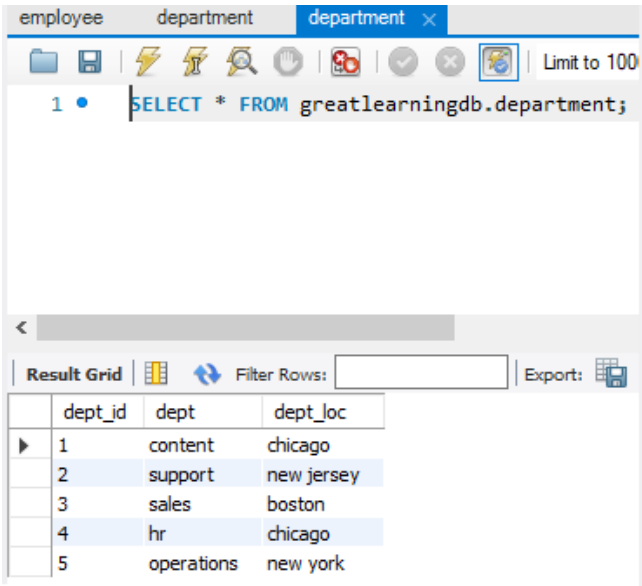
SQL INNER JOIN

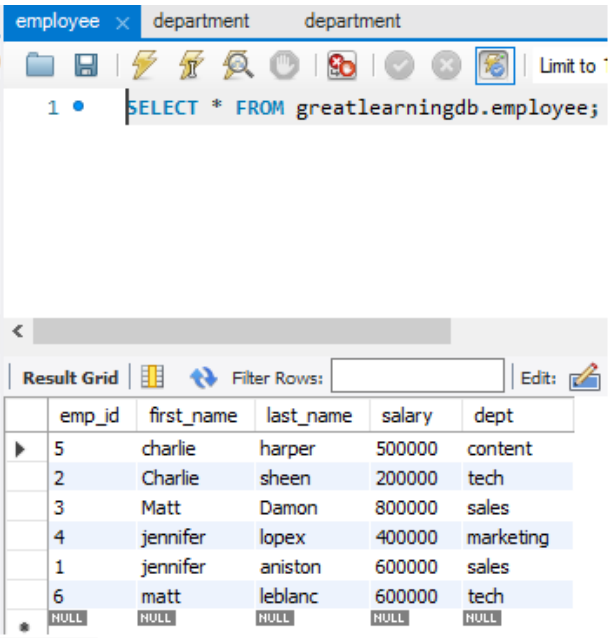
emp_id	first_name	last_name	salary	dept_id
103	Harry	Potter	20000	12
102	Edwin	Thomas	15000	11
101	Steven	Cohen	10000	10
100	Erik	John	10000	12

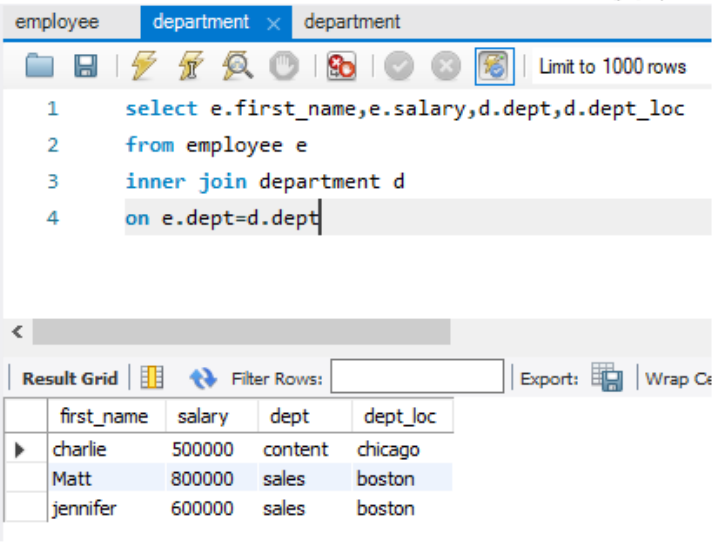
dept_id	dept_name	manager_id	location_id
10	IT	200	1700
11	Marketing	201	1800
13	Resources	203	2400
14	Shipping	121	1500

```
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
Limit to 100
1 SELECT * FROM greatlearningdb.department;
Result Grid
Filter Rows:
Export:
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
Limit to
1 SELECT * FROM greatlearningdb.employee;
Result Grid
Filter Rows:
Edit:
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
Limit to 1000 rows
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
Result Grid
Filter Rows:
Export:
Wrap Ce
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;
```

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

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

employee | department | department

Limit to 1000 rows

```
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
```

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

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;
```

greatlearning
Learning for Life

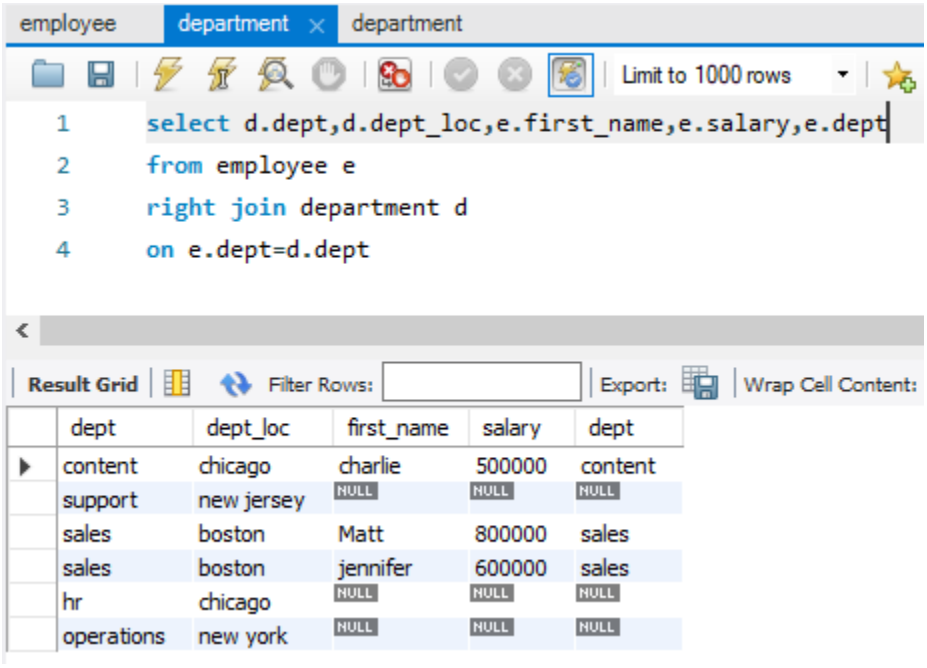
SQL RIGHT JOIN

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
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



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_e
101	Steven	Cohen	10	IT
102	Edwin	Thomas	11	Marke
103	Harry	Potter	Null	Null
100	Erik	John	Null	Null
Null	Null	Null	13	Resou

employee | department | department

Limit to 1000 rows

```
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
```

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

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;
```

greatlearning
Learning for Life

SQL CROSS JOIN

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

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

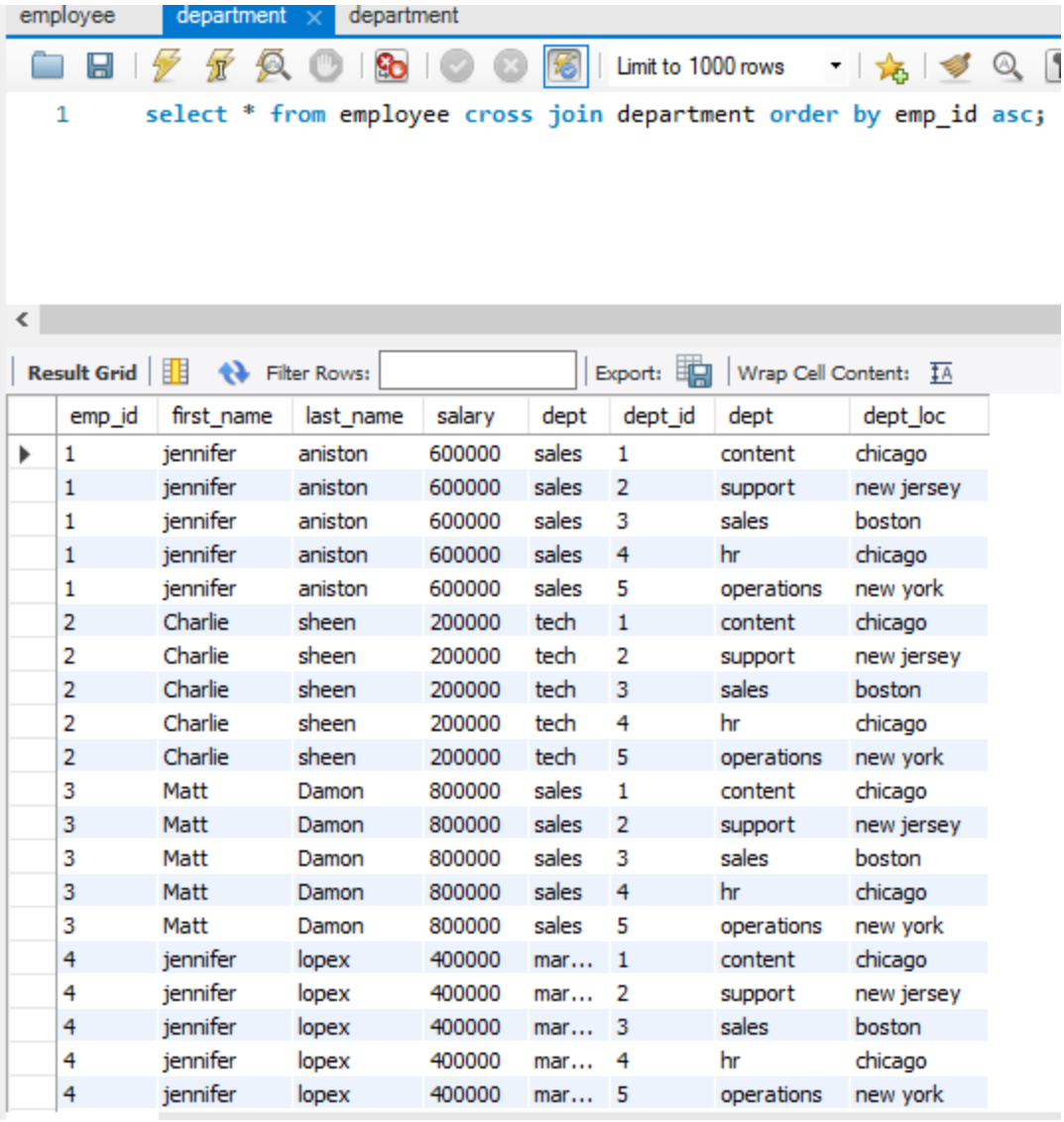
Table-1

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



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

```
-- ------------------------------------------------------
# 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:-

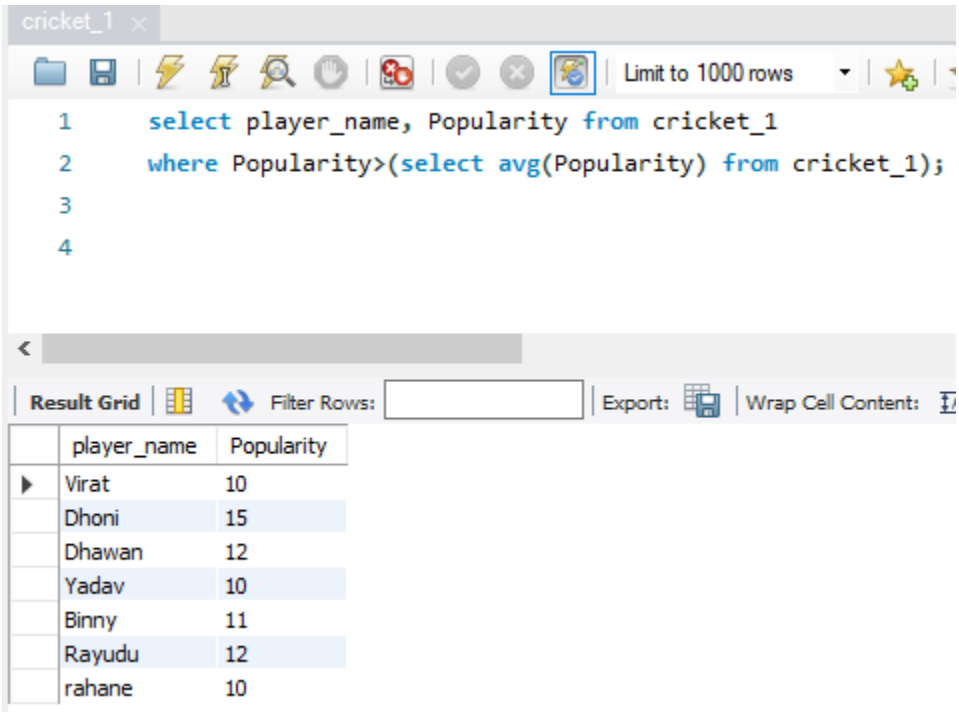
cricket_1 cricket_2

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

Result Grid | Filter Rows:

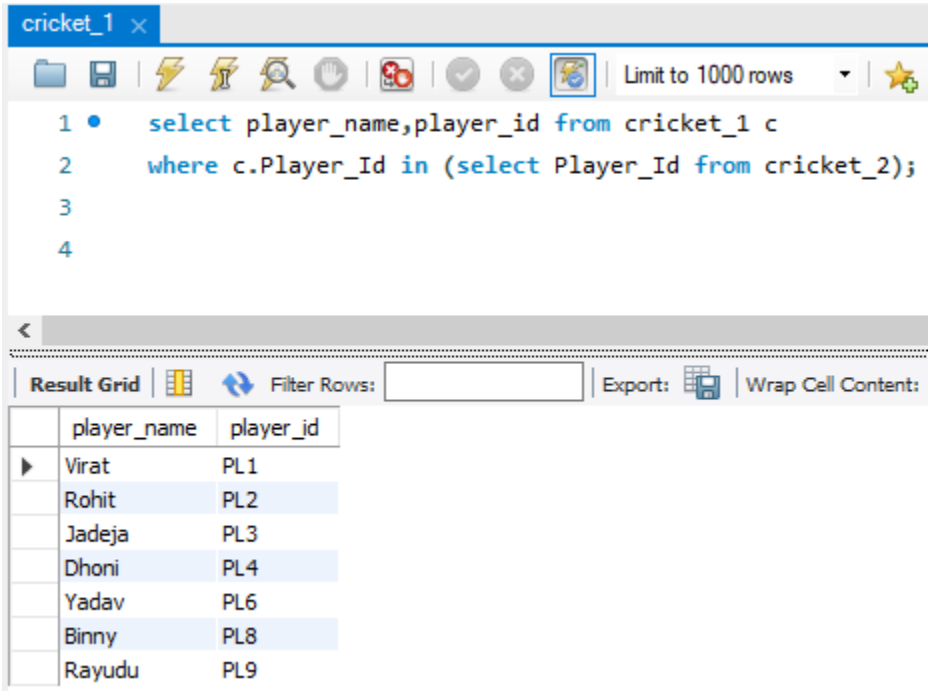
Player_Name
Virat
Rohit
Jadeja
Dhoni
Dhawan
Yadav
Raina
Binny
Rayudu
rahane
A. Patel
B. Kumar
Yuvraaj
Tendulkar
Dravid
Yusuf

Q-2:-



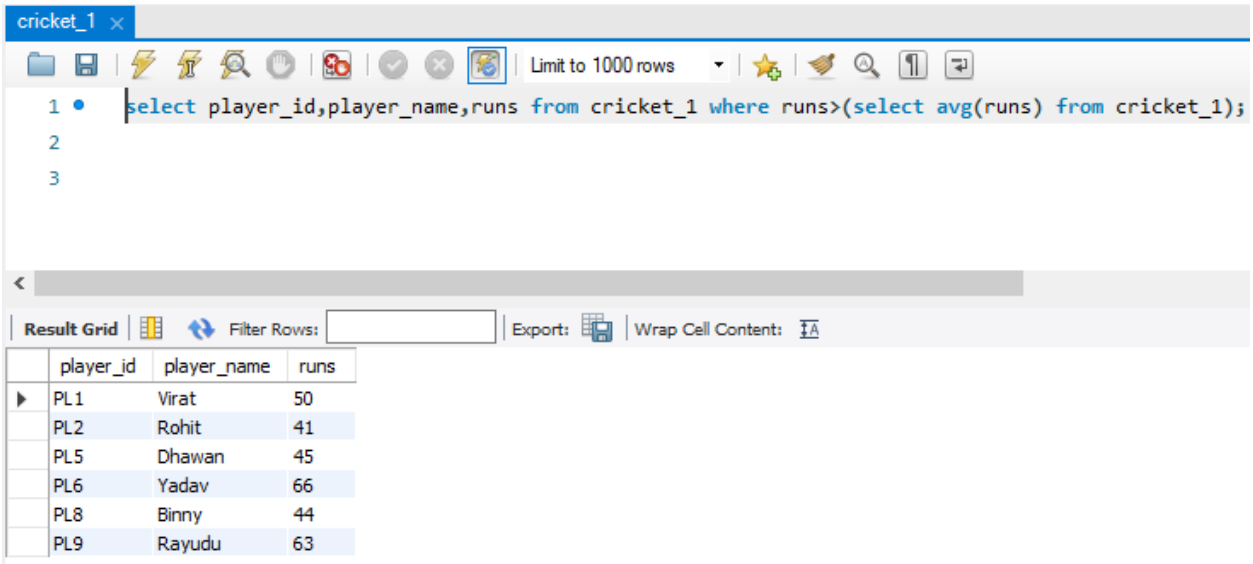
player_name	Popularity
Virat	10
Dhoni	15
Dhawan	12
Yadav	10
Binny	11
Rayudu	12
rahane	10

Q-3:-



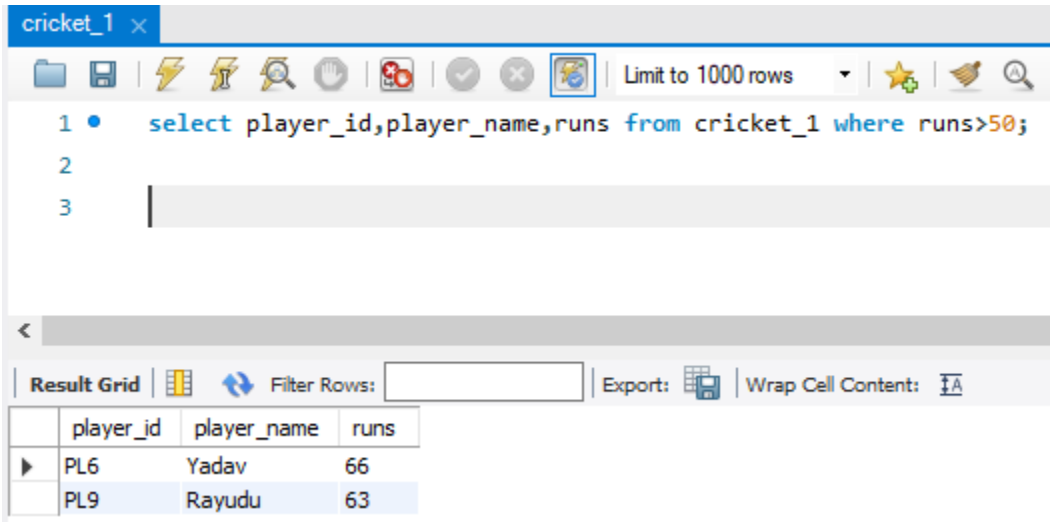
player_name	player_id
Virat	PL1
Rohit	PL2
Jadeja	PL3
Dhoni	PL4
Yadav	PL6
Binny	PL8
Rayudu	PL9

Q-4



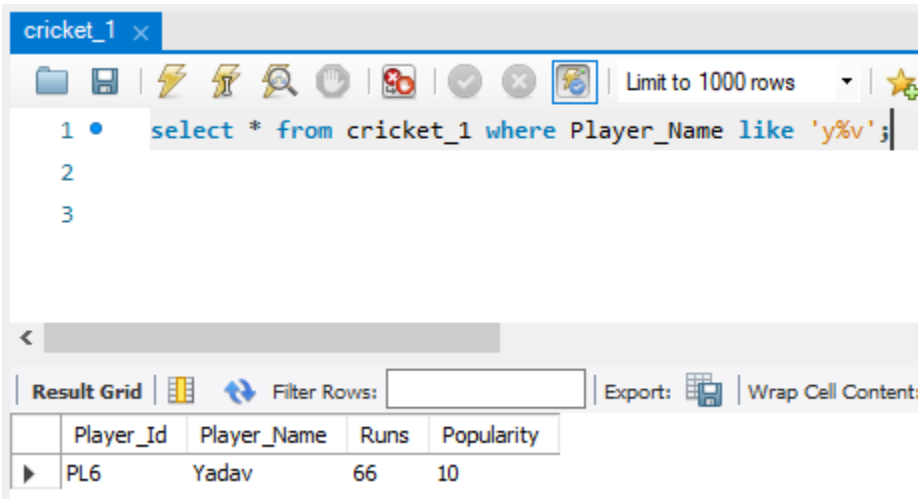
player_id	player_name	runs
PL1	Virat	50
PL2	Rohit	41
PL5	Dhawan	45
PL6	Yadav	66
PL8	Binny	44
PL9	Rayudu	63

Q-5



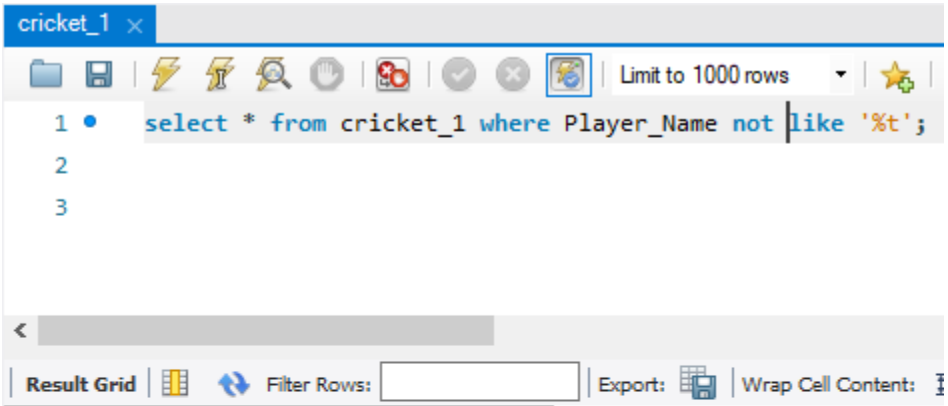
player_id	player_name	runs
PL6	Yadav	66
PL9	Rayudu	63

Q-6



Player_Id	Player_Name	Runs	Popularity
PL6	Yadav	66	10

Q-7



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

cricket_1

Limit to 1000 rows

```
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;
```

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

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:-

cricket_1

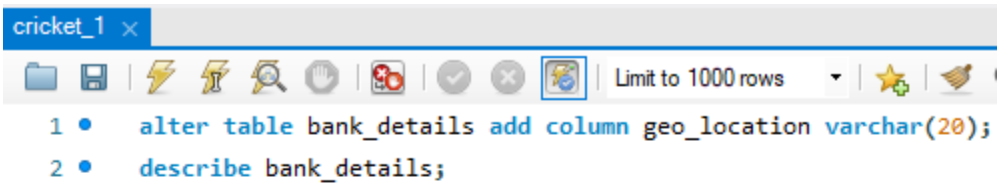
Limit to 1000 rows

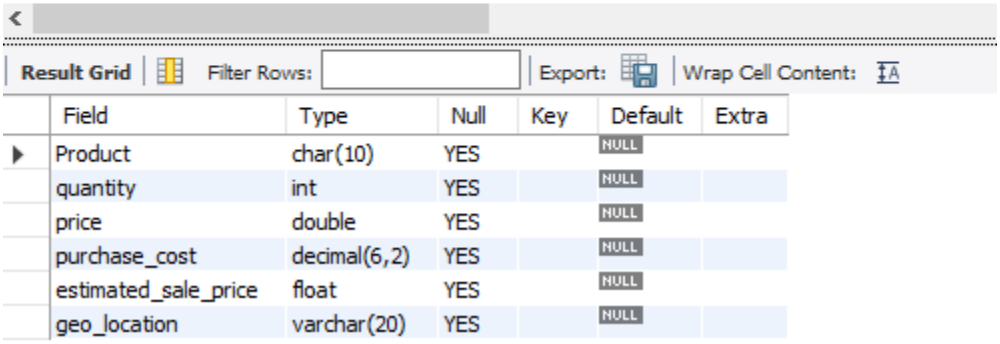
```
1  insert into bank_details
2  values('paycard',3,330,8008,9009),
3  ('paypoints',4,300,8000,6800);
4
5  select * from bank_details;
```

Result Grid | Filter Rows: | Export: | Wrap Ce

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

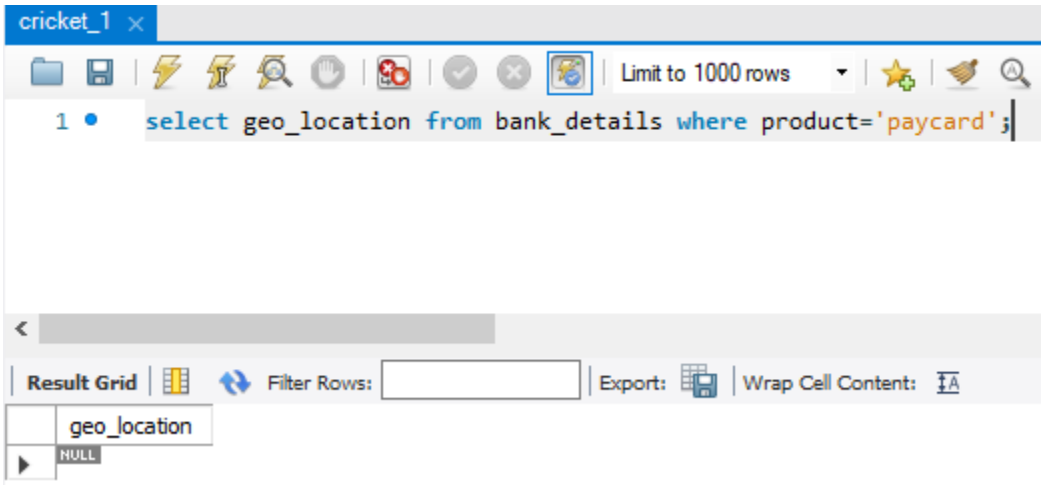
Adding the new column geo location:-

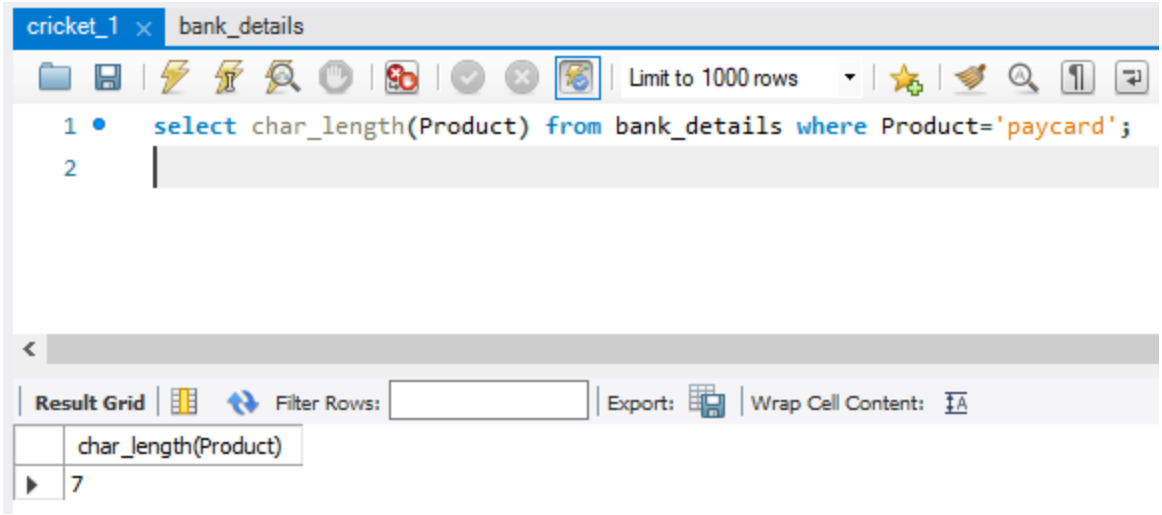




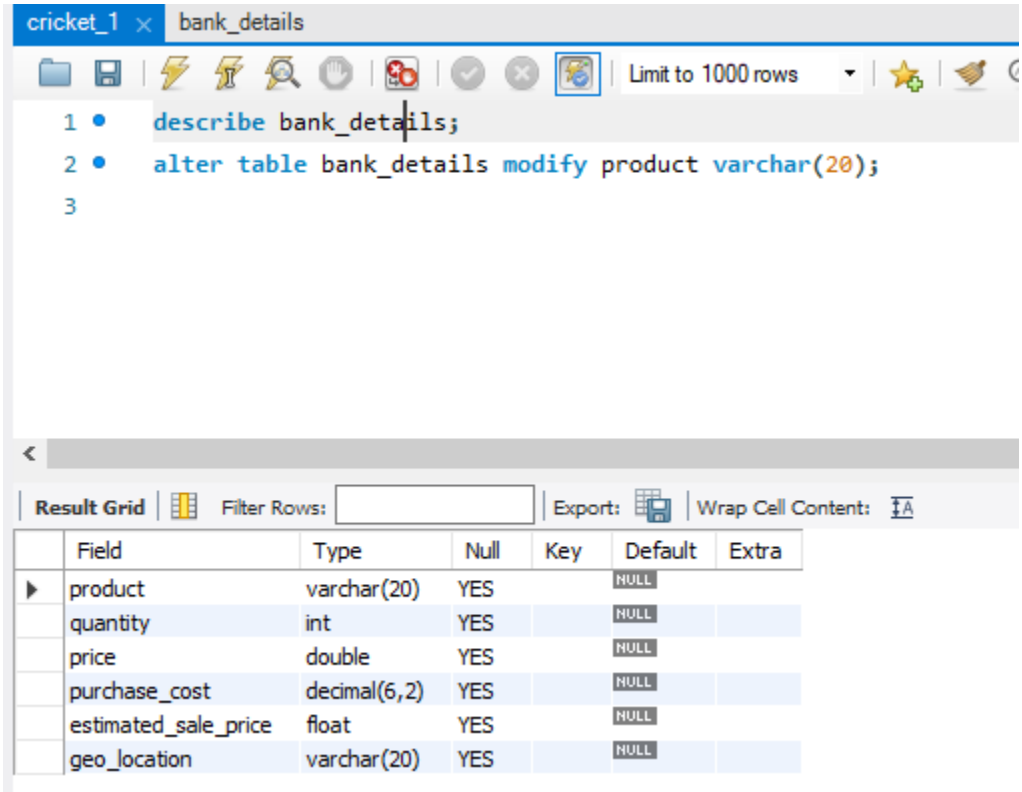
Field	Type	Null	Key	Default	Extra
Product	char(10)	YES		NULL	
quantity	int	YES		NULL	
price	double	YES		NULL	
purchase_cost	decimal(6,2)	YES		NULL	
estimated_sale_price	float	YES		NULL	
geo_location	varchar(20)	YES		NULL	

Getting the geo location of the paycard product:-

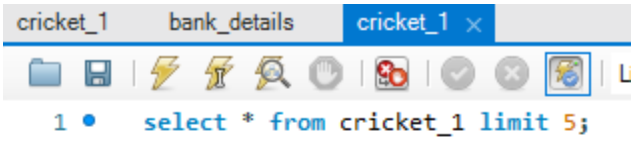


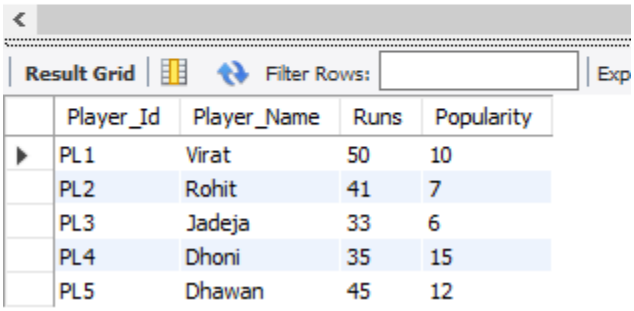


Altering the table column's datatype



Displaying only five records





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