

SQL

by

APNA
COLLEGE

Database



Database is **collection of data** in a format that can be easily accessed (Digital)

A software application used to manage our DB is called DBMS (**Database Management System**)

Types of Databases

Relational

Data stored in tables



Non-relational (NoSQL)

data not stored in tables



** We use SQL to work with relational DBMS

APNA
COLLEGE

What is SQL?



Structured Query Language

SQL is a programming language used to interact with relational databases.

It is used to perform **CRUD** operations :

Create

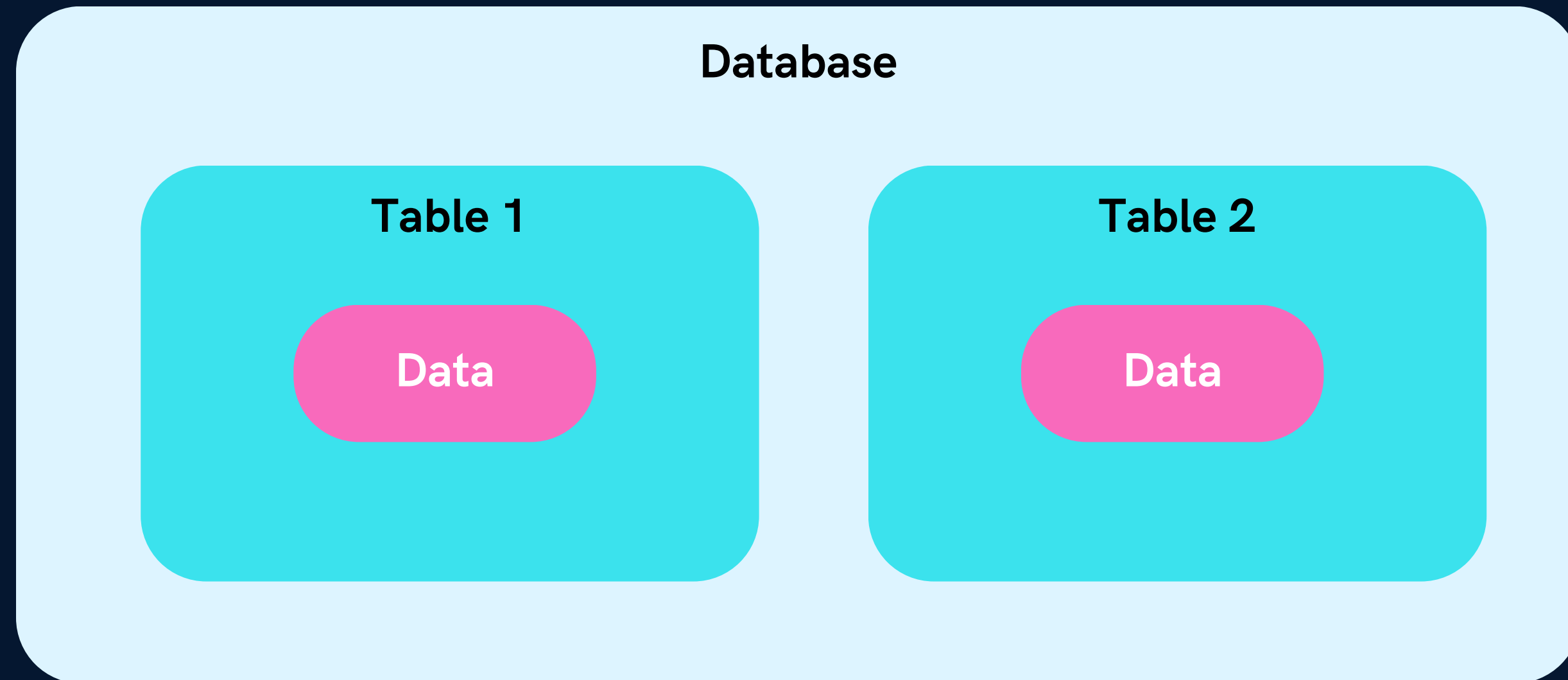
Read

Update

Delete

APNA
COLLEGE

Database Structure



What is a table?

Student table

RollNo	Name	Class	DOB	Gender	City	Marks
1	Nanda	X	1995-06-06	M	Agra	551
2	Saurabh	XII	1993-05-07	M	Mumbai	462
3	Sonal	XI	1994-05-06	F	Delhi	400
4	Trisla	XII	1995-08-08	F	Mumbai	450
5	Store	XII	1995-10-08	M	Delhi	369
6	Marisla	XI	1994-12-12	F	Dubai	250
7	Neha	X	1995-12-08	F	Moscow	377
8	Nishant	X	1995-06-12	M	Moscow	489

Creating our First Database

Our first SQL Query

```
CREATE DATABASE db_name;
```

```
DROP DATABASE db_name;
```

APNA
COLLEGE

Creating our First Table

USE *db_name*;

CREATE TABLE *table_name* (
 column_name1 datatype constraint,
 column_name2 datatype constraint,
 column_name2 datatype constraint
);

```
CREATE TABLE student (  
    id INT PRIMARY KEY,  
    name VARCHAR(50),  
    age INT NOT NULL  
);
```


SQL Datatypes

They define the **type of values** that can be stored in a column

DATATYPE	DESCRIPTION	USAGE
CHAR	string(0-255), can store characters of fixed length	CHAR(50)
VARCHAR	string(0-255), can store characters up to given length	VARCHAR(50)
BLOB	string(0-65535), can store binary large object	BLOB(1000)
INT	integer(-2,147,483,648 to 2,147,483,647)	INT
TINYINT	integer(-128 to 127)	TINYINT
BIGINT	integer(-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807)	BIGINT
BIT	can store x-bit values. x can range from 1 to 64	BIT(2)
FLOAT	Decimal number - with precision to 23 digits	FLOAT
DOUBLE	Decimal number - with 24 to 53 digits	DOUBLE
BOOLEAN	Boolean values 0 or 1	BOOLEAN
DATE	date in format of YYYY-MM-DD ranging from 1000-01-01 to 9999-12-31	DATE
YEAR	year in 4 digits format ranging from 1901 to 2155	YEAR

SQL Datatypes

Signed & Unsigned

TINYINT UNSIGNED (0 to 255)

TINYINT (-128 to 127)

APNA
COLLEGE

Types of SQL Commands

DDL (Data Definition Language) : create, alter, rename, truncate & drop

DQL (Data Query Language) : select

DML (Data Manipulation Language) : select, insert, update & delete select comes under DML also

DCL (Data Control Language) : grant & revoke permission to users

TCL (Transaction Control Language) : start transaction, commit, rollback etc.



Database related Queries

CREATE DATABASE *db_name*;

CREATE DATABASE **IF NOT EXISTS** *db_name*;

CREATE DATABASE IF NOT EXISTS college;

DROP DATABASE *db_name*;

DROP DATABASE **IF EXISTS** *db_name*;

SHOW DATABASES; Shows all databases present in the server

SHOW TABLES; shows all tables present within a database



Table related Queries

Create

```
CREATE TABLE table_name (  
    column_name1 datatype constraint,  
    column_name2 datatype constraint,  
);
```

```
CREATE TABLE student (  
    rollno INT PRIMARY KEY,  
    name VARCHAR(50)  
);
```

Table related Queries

Select & View ALL columns

```
SELECT * FROM table_name;
```

* means all

```
SELECT * FROM student;
```

APNA
COLLEGE

Table related Queries

Insert

```
INSERT INTO table_name  
(colname1, colname2);  
VALUES  
(col1_v1, col2_v1),  
(col1_v2, col2_v2);
```

```
INSERT INTO student  
(rollno, name)  
VALUES  
(101, "karan"),  
(102, "arjun");
```

Keys

Primary Key

It is a column (or set of columns) in a table that uniquely identifies each row. (a unique id)

There is only 1 PK & it should be NOT null.

If two (or more) columns form the primary key, then each column separately may have duplicate entries or null values, but combination of the two (or more columns respectively) cannot have duplicate values or null values

For example: primary key(id,name); combination of id & name is always unique & null.

Foreign Key

A foreign key is a column (or set of columns) in a table that refers to the primary key in another table.

There can be multiple FKs.

FKs can have duplicate & null values.

Keys

table1 - Student

id (PK)	name	cityId (FK)	city
101	karan	1	Pune
102	arjun	2	Mumbai
103	ram	1	Pune
104	shyam	3	Delhi

table2 - City

id (PK)	city_name
1	Pune
2	Mumbai
3	Delhi

PK: primary key
FK: foreign key

Constraints

SQL constraints are used to specify rules for data in a table.

NOT NULL columns cannot have a null value

col1 **int NOT NULL**

UNIQUE all values in column are different

col2 **int UNIQUE**

PRIMARY KEY makes a column unique & not null but used only for one

id **int PRIMARY KEY**

```
CREATE TABLE temp (  
  id int not null,  
  PRIMARY KEY (id)  
);
```

both ways acceptable, we can declare a column at the time of declaration or later also

APNA
COLLEGE

Constraints

FOREIGN KEY prevent actions that would destroy links between tables

```
CREATE TABLE temp (  
  cust_id int,  
  FOREIGN KEY (cust_id) references customer(id)  
);
```



Parent table

DEFAULT sets the default value of a column
You may see supplementary notes

only for the rows in which you don't specify the values, if you specify then that will be considered else default one will be considered

```
salary INT DEFAULT 25000
```

APNA
COLLEGE

Constraints

CHECK it can limit the values allowed in a column

```
CREATE TABLE city (  
  id INT PRIMARY KEY,  
  city VARCHAR(50),  
  age INT,  
  CONSTRAINT age_check CHECK (age >= 18 AND city="Delhi")  
);
```

it is constraint name, you may skip this

```
CREATE TABLE newTab (  
  age INT CHECK (age >= 18)  
);
```

Create this sample table

```
CREATE DATABASE college;
USE college;

CREATE TABLE student (
  rollno INT PRIMARY KEY,
  name VARCHAR(50),
  marks INT NOT NULL,
  grade VARCHAR(1),
  city VARCHAR(20)
);
```

Insert this data

```
INSERT INTO student
(rollno, name, marks, grade, city)
VALUES
(101, "anil", 78, "C", "Pune"),
(102, "bhumika", 93, "A", "Mumbai"),
(103, "chetan", 85, "B", "Mumbai"),
(104, "dhruv", 96, "A", "Delhi"),
(105, "emanuel", 12, "F", "Delhi"),
(106, "farah", 82, "B", "Delhi");
```

Select in Detail

used to select any data from the database

Basic Syntax

```
SELECT col1, col2 FROM table_name;
```

To Select ALL

```
SELECT * FROM table_name;
```



Where Clause

To define some conditions

```
SELECT col1, col2 FROM table_name  
WHERE conditions;
```

```
SELECT * FROM student WHERE marks > 80;  
SELECT * FROM student WHERE city = "Mumbai";
```

Both conditions can be clubbed also:
select *
from student
where marks > 80 and city="Mumbai";

Where Clause

Using Operators in WHERE

Arithmetic Operators : +(addition) , -(subtraction), *(multiplication), /(division), %(modulus)

Comparison Operators : = (equal to), != (not equal to), > , >=, <, <=

Logical Operators : AND, OR , NOT, IN, BETWEEN, ALL, LIKE, ANY

Bitwise Operators : & (Bitwise AND), | (Bitwise OR)



Operators

AND (to check for both conditions to be true)

```
SELECT * FROM student WHERE marks > 80 AND city = "Mumbai";
```

OR (to check for one of the conditions to be true)

```
SELECT * FROM student WHERE marks > 90 OR city = "Mumbai";
```

Operators

Between (selects for a given range)

```
SELECT * FROM student WHERE marks BETWEEN 80 AND 90;
```

 80 and 90 inclusive

In (matches **any value in the list**)

```
SELECT * FROM student WHERE city IN ("Delhi", "Mumbai");
```

^{IN}
NOT (to negate the given condition)

```
SELECT * FROM student WHERE city NOT IN ("Delhi", "Mumbai");
```

Limit Clause

Sets an upper limit on number of (tuples)rows to be returned

```
SELECT * FROM student LIMIT 3;
```

only data of first 3 student comes in the output

```
SELECT col1, col2 FROM table_name  
LIMIT number;
```

Note: we can use combination of different clause in a statement

Order By Clause

To sort in ascending (ASC) or descending order (DESC)

```
SELECT * FROM student  
ORDER BY city ASC;
```

if we don't mention asc or desc it orders in ascending order by default

```
SELECT col1, col2 FROM table_name  
ORDER BY col_name(s) ASC;
```

Aggregate Functions

Aggregate functions perform a calculation on a set of values, and return a single value.

- COUNT() used to determine the number of rows that match a specified condition
- MAX()
- MIN()
- SUM()
- AVG()

Get Maximum Marks

```
SELECT max(marks)
FROM student;
```

↩ or name

select count(rollno)
from student;
gives total no. of student present in
the class

Get Average marks

```
SELECT avg(marks)
FROM student;
```

APNA
COLLEGE

Group By Clause

Groups rows that have the same values into summary rows.

It collects data from multiple records and groups the result by one or more column.

*Generally we use group by with some *aggregation function*.

see supplementary notes also

Count number of students in each city

```
SELECT city, count(name)
FROM student
GROUP BY city;
```

Having Clause

Similar to Where i.e. applies some condition on rows.

Used when we want to apply any **condition after grouping**.

Count number of students in each city where max marks cross 90.

```
SELECT count(name), city
FROM student
GROUP BY city
HAVING max(marks) > 90;
```

where is used to apply conditions on normal rows but
having is used to apply conditions on formed groups

General Order

SELECT *column(s)*

FROM *table_name*

WHERE *condition*

GROUP BY *column(s)*

HAVING *condition*

ORDER BY *column(s)* **ASC;**

where applies conditons on rows
having applies conditions on groups

where written before group by
having written after group by

Having Clause

Similar to Where i.e. applies some condition on rows.

Used when we want to apply any **condition after grouping**.

Count number of students in each city where max marks cross 90.

```
SELECT count(name), city
FROM student
GROUP BY city
HAVING max(marks) > 90;
```

Table related Queries

Update (to update existing rows)

```
UPDATE table_name  
SET col1 = val1, col2 = val2  
WHERE condition;
```

```
UPDATE student  
SET grade = "0"  
WHERE grade = "A";
```

Table related Queries

Delete (to delete existing rows)

DELETE FROM *table_name*
WHERE *condition*;

```
DELETE FROM student  
WHERE marks < 33;
```

Cascading for FK

Foreign key

CASCADE is used to update or remove an entry from both the parent and child tables at the same time
cascade means if we apply change in parent table it should be reflected in child table as well automatically.

On Delete Cascade

When we create a foreign key using this option, it deletes the referencing rows in the child table when the referenced row is deleted in the parent table which has a primary key.

On Update Cascade

When we create a foreign key using UPDATE CASCADE the referencing rows are updated in the child table when the referenced row is updated in the parent table which has a primary key.

```
CREATE TABLE student (  
  id INT PRIMARY KEY,  
  courseID INT,  
  FOREIGN KEY(courseID) REFERENCES course(id)  
  ON DELETE CASCADE  
  ON UPDATE CASCADE  
);
```

if you want u can use cascade

Table related Queries

Alter (to change the schema)

ADD Column

ALTER TABLE *table_name*

ADD COLUMN *column_name datatype constraint;*

DROP Column

ALTER TABLE *table_name*

DROP COLUMN *column_name;*

RENAME Table

ALTER TABLE *table_name*

RENAME TO *new_table_name;*



Table related Queries

CHANGE Column (rename)

ALTER TABLE *table_name*

CHANGE COLUMN *old_name new_name new_datatype new_constraint;*

MODIFY Column (modify datatype/ constraint)

ALTER TABLE *table_name*

MODIFY *col_name new_datatype new_constraint;*



ADD Column

```
ALTER TABLE student  
ADD COLUMN age INT NOT NULL DEFAULT 19;
```

DROP Column

```
ALTER TABLE student  
DROP COLUMN stu_age;
```

MODIFY Column

```
ALTER TABLE student  
MODIFY age VARCHAR(2);
```

RENAME Table

```
ALTER TABLE student  
RENAME TO stu;
```

CHANGE Column (rename)

```
ALTER TABLE student  
CHANGE age stu_age INT;
```

APNA
COLLEGE

Table related Queries

Truncate (to delete table's data)

TRUNCATE TABLE *table_name* ;

drop deletes whole table

truncate only deletes the data of table, table and its schemas intact, if we want to add new data we can do that

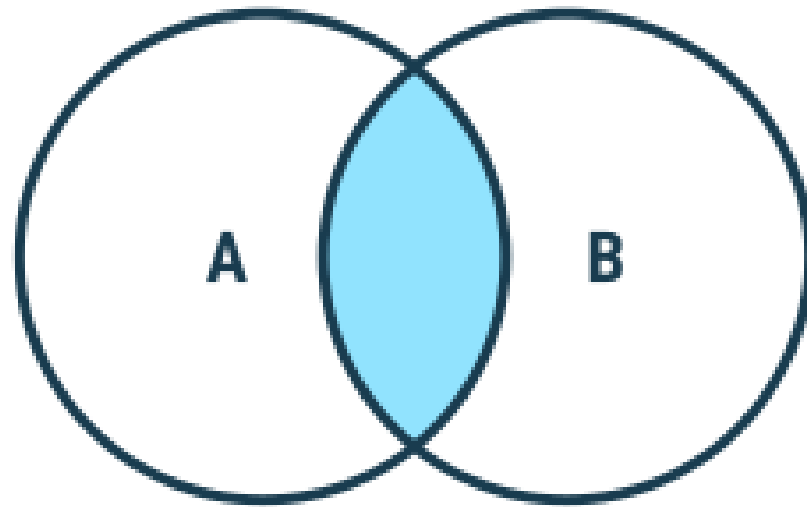
```
UPDATE student  
SET grade = "0"  
WHERE grade = "A";
```

APNA
COLLEGE

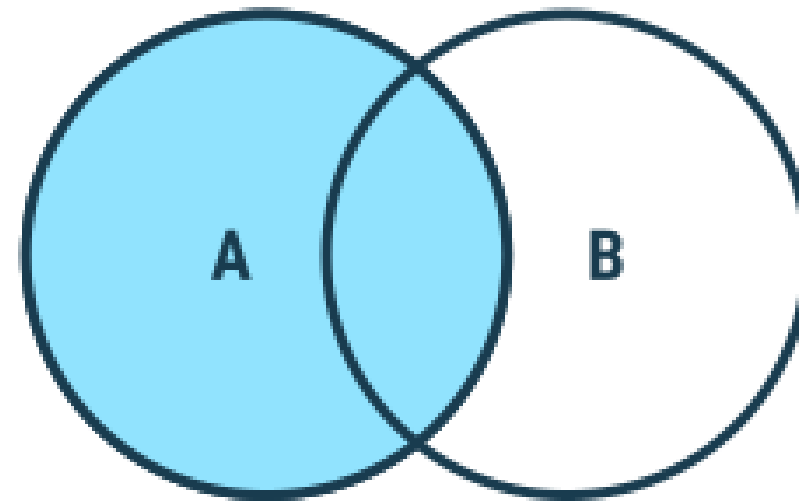
Joins in SQL

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

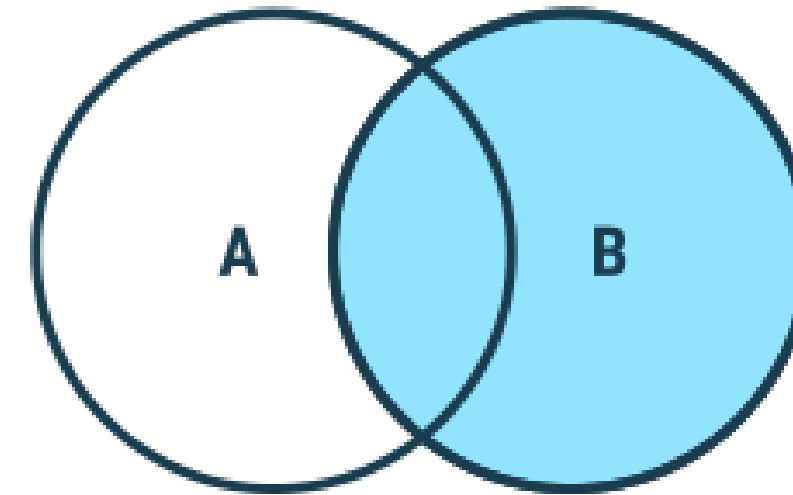
Types of Joins



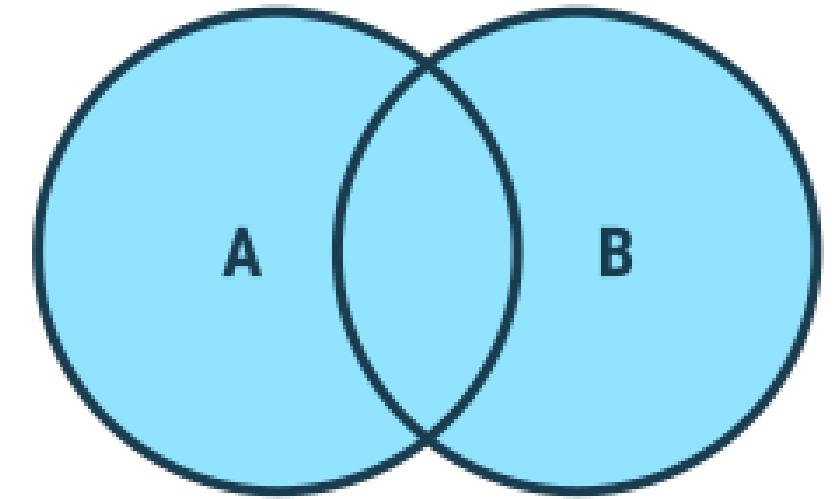
Inner Join



Left Join



Right Join



Full Join



Outer Joins

Venn diagram

APNA
COLLEGE

Inner Join

Returns records that have matching values in both tables

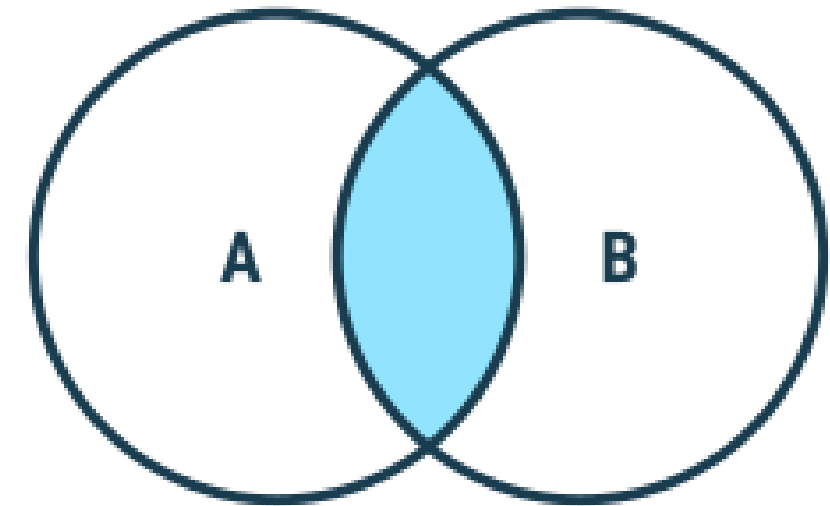
Syntax

SELECT *column(s)*

FROM *tableA*

INNER JOIN *tableB*

ON *tableA.col_name = tableB.col_name;*



in inner join you can write table B first and table A second (order is not an issue), no impact on the result but this is not true for other joins

Inner Join

Example

student

student_id	name
101	adam
102	bob
103	casey

course

student_id	course
102	english
105	math
103	science
107	computer science

```
SELECT *  
FROM student  
INNER JOIN course  
ON student.student_id = course.student_id;
```

These column
name may or may
not be same

you could have written like this also, "as" is used to provide alternate name or alias

Result

student_id	name	course
102	bob	english
103	casey	science

```
SELECT *  
FROM student as s  
INNER JOIN course as c  
ON s.student_id = c.student_id
```

APNA
COLLEGE

Left Join

Returns all records from the left table, and the matched records from the right table

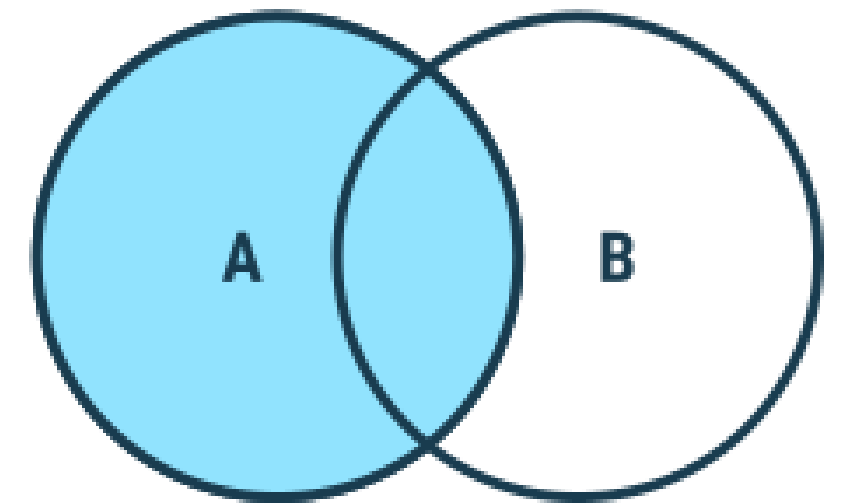
Syntax

SELECT *column(s)*

FROM *tableA*

LEFT JOIN *tableB*

ON *tableA.col_name = tableB.col_name;*



APNA
COLLEGE

Left Join

Example

student

student_id	name
101	adam
102	bob
103	casey

course

student_id	course
102	english
105	math
103	science
107	computer science

Result

student_id	name	course
101	adam	<i>null</i>
102	bob	english
103	casey	science

```
SELECT *  
FROM student as s  
LEFT JOIN course as c  
ON s.student_id = c.student_id;
```



Right Join

Returns all records from the right table, and the matched records from the left table

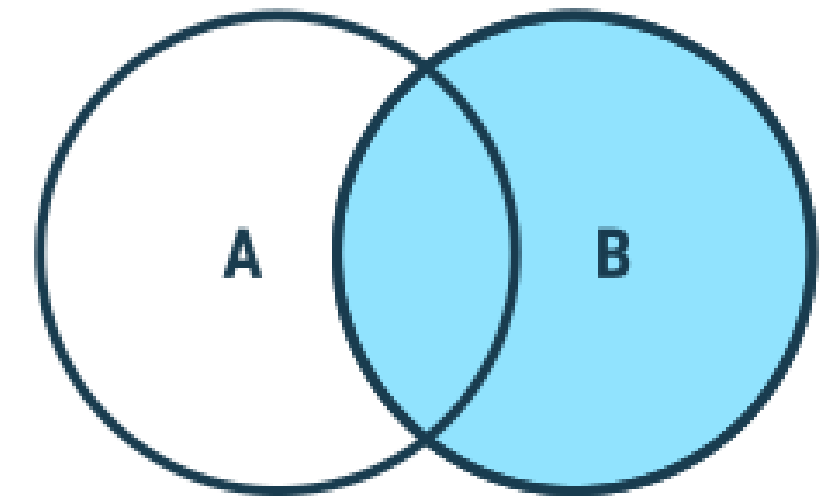
Syntax

SELECT *column(s)*

FROM *tableA*

RIGHT JOIN *tableB*

ON *tableA.col_name = tableB.col_name;*



APNA
COLLEGE

Right Join

Example

student

student_id	name
101	adam
102	bob
103	casey

course

student_id	course
102	english
105	math
103	science
107	computer science

```
SELECT *  
FROM student as s  
RIGHT JOIN course as c  
ON s.student_id = c.student_id;
```

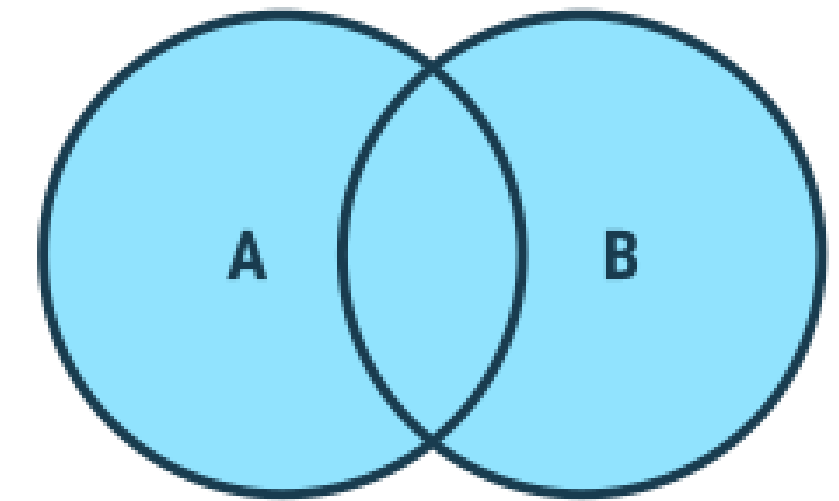
Result

student_id	course	name
102	english	bob
105	math	<i>null</i>
103	science	casey
107	computer science	<i>null</i>

Full Join

Returns all records when there is a match in either left or right table

don't give duplicate records



Syntax in MySQL

```
SELECT * FROM student as a  
LEFT JOIN course as b  
ON a.id = b.id  
UNION
```

```
SELECT * FROM student as a  
RIGHT JOIN course as b  
ON a.id = b.id;
```

LEFT JOIN
UNION

RIGHT JOIN

there is no direct syntax for full join in mysql, we do this indirectly with the help of union. In oracle sql or postgres sql there is direct syntax.

in oracle:
SELECT *
FROM student as s
FULL OUTER JOIN course as c
ON s.student_id = c.student_id;

APNA
COLLEGE

Full Join

Example

student

student_id	name
101	adam
102	bob
103	casey

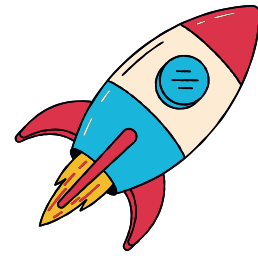
course

student_id	course
102	english
105	math
103	science
107	computer science

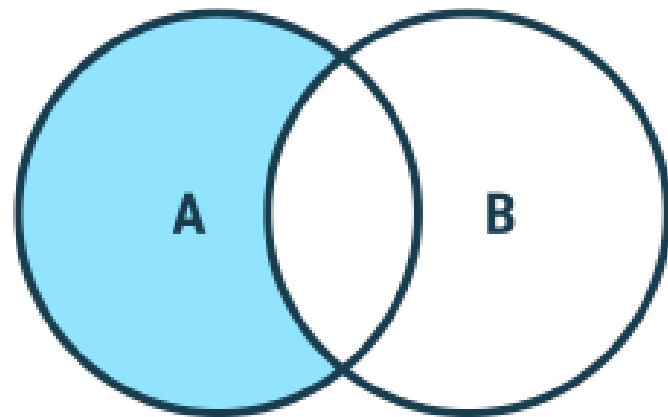
Result

student_id	name	course
101	adam	<i>null</i>
102	bob	english
103	casey	science
105	<i>null</i>	math
107	<i>null</i>	computer science

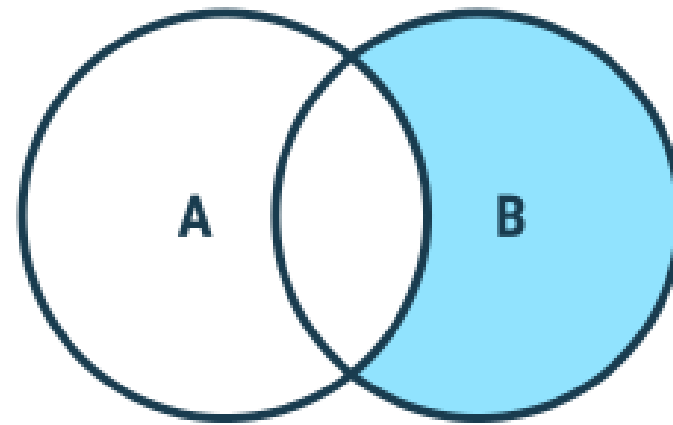
Think & Ans



Qs: Write SQL commands to display the right exclusive join :



Left Exclusive Join



Right Exclusive Join

```
SELECT *  
FROM student as a  
LEFT JOIN course as b  
ON a.id = b.id  
WHERE b.id IS NULL;
```

```
SELECT *  
FROM student as a  
RIGHT JOIN course as b  
ON a.id = b.id  
where a.id is null;
```

Self Join

It is a regular join but the table is joined with itself.

Syntax

SELECT *column(s)*

FROM *table as a*

JOIN *table as b*

ON *a.col_name = b.col_name;*

Self Join

Example

Employee

id	name	manager_id
101	adam	103
102	bob	104
103	casey	<i>null</i>
104	donald	103

Result

```
SELECT a.name as manager_name, b.name  
FROM employee as a  
JOIN employee as b  
ON a.id = b.manager_id;
```

must see supplementary notes

Union

It is used to combine the result-set of two or more SELECT statements.
Gives UNIQUE records.

To use it :

- every SELECT should have same no. of columns
- columns must have similar data types
- columns in every SELECT should be in same order

Syntax

SELECT *column(s)* **FROM** *tableA*

UNION

SELECT *column(s)* **FROM** *tableB*



SQL Sub Queries

A Subquery or Inner query or a Nested query is a query within another SQL query.

It involves 2 select statements.

subquery can be written in either of 3 ways based on whether subquery is written in 1. select, 2. from or 3. in where clause

Syntax

SELECT *column(s)*

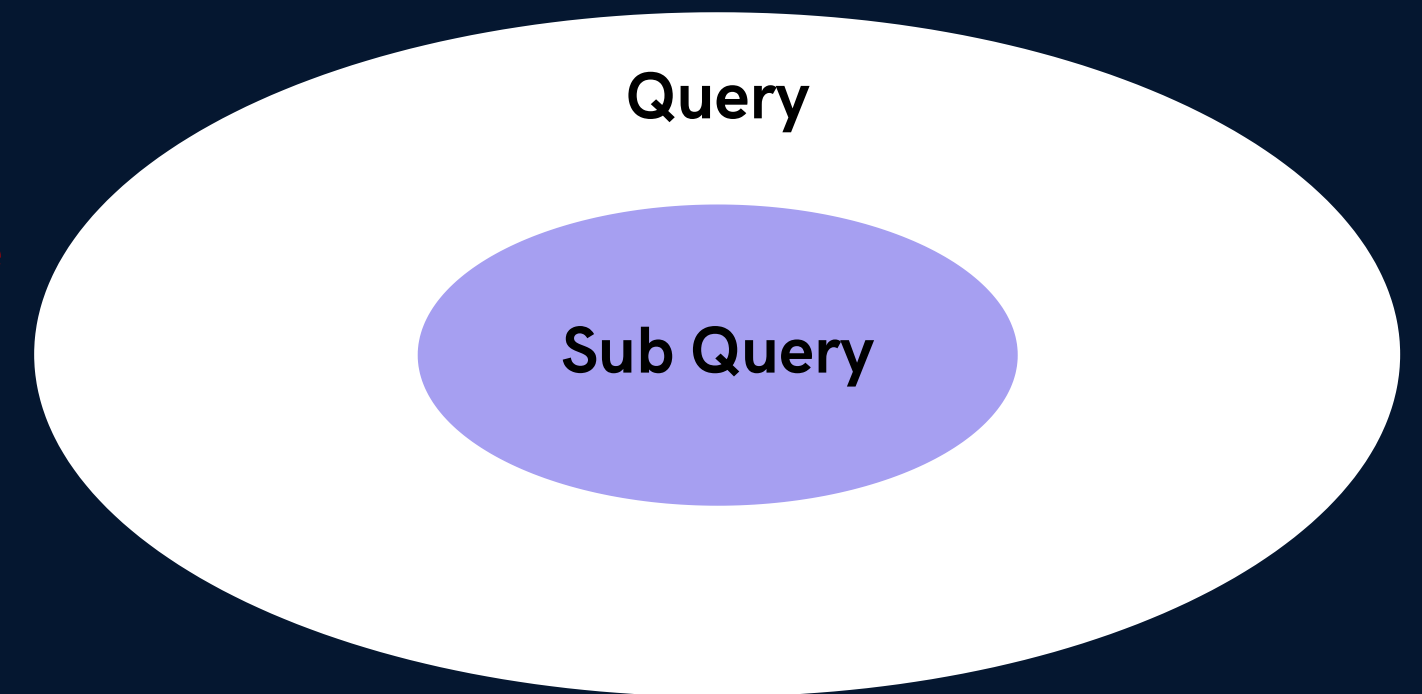
FROM *table_name*

WHERE *col_name operator*

(*subquery*);

examples of some operator: between, in, >, <, and so on

subquery is also a normal query having another select



APNA
COLLEGE

SQL Sub Queries

Example

Get names of all students who scored more than class average.

Step 1. Find the avg of class

Step 2. Find the names of students with marks > avg

rollno	name	marks
101	anil	78
102	bhumika	93
103	chetan	85
104	dhruv	96
105	emanuel	92
106	farah	82

SQL Sub Queries

Example

Find the names of all students with even roll numbers.

Step 1. Find the even roll numbers

Step 2. Find the names of students with even roll no

rollno	name	marks
101	anil	78
102	bhumika	93
103	chetan	85
104	dhruv	96
105	emanuel	92
106	farah	82

SQL Sub Queries

Example with *FROM*

Find the max marks from the students of Delhi

Step 1. Find the students of Mumbai

Step 2. Find their max marks using the sublist in step 1

rollno	name	marks	city
101	anil	78	Pune
102	bhumika	93	Mumbai
103	chetan	85	Mumbai
104	dhruv	96	Delhi
105	emanuel	92	Delhi
106	farah	82	Delhi

MySQL Views

A view is a virtual table based on the result-set of an SQL statement.

```
CREATE VIEW view1 AS  
SELECT rollno, name FROM student;
```

After creating the virtual table (view table), we can perform operations on this virtual (not real) table

```
SELECT * FROM view1;
```

Suppose a company has a database of customer, containing all the information about the customer. A sales team may require the name, Phone number, email id of the customer. Sales team may not be interested with the credit card number of the customer. So we can provide the sales team a table (which is virtual) containing only the required details instead of providing whole customer table database.

*A view always shows up-to-date data. The database engine recreates the view, every time a user queries it.