SQL SHORT NOTE

19 August 2023 19:55

-- create database
-- CREATE DATABASE temp1;
-- CREATE DATABASE temp2;
-- create database college;

-- nahi ho to crate karna he
-- create database if not exists college;

-- create database college;
-- create database run;

-- delete database
-- drop database run;
-- drop database college;
-- drop database temp1;
-- drop database templ;
-- drop database temp2;

-- create table
```
create database college;
use college;
create table student(
id int primary key,
name varchar(50),
age int not null
);
```

-- insert data
```
insert into student values(1,'navnath',24);
-- insert into student values(7,'priya',24);
-- insert into student values(3,'priyap',24);
-- insert into student values(4,'navnath',24);
-- insert into student values(5,'priya',24);
```

-- display data
select* from student;

-- delete database
drop database college;
drop database if exists college;

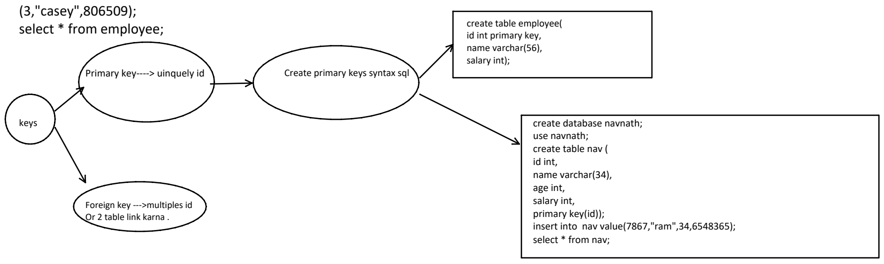
-- show the database present
show databases;

-- show tables present
show tables

-- tables status
show table status

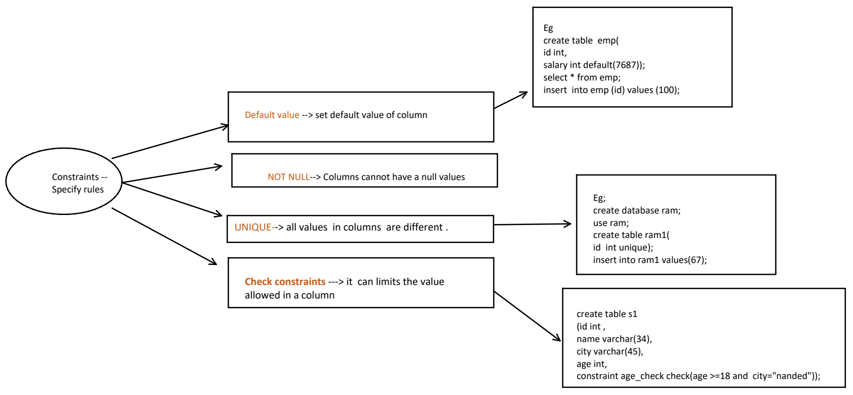
-- insert table data
```
insert into ram
(rollno ,name)
values
(11,"navnath"),
(16,"ram");
```

Q1.
ANS :
```
create database xyz_company;
use xyz_company;
create table employee(
id int primary key,
name varchar(56),
salary int);

insert into employee
(id,name,salary)
values
(1,"ram",653574),
(2,"bob",8978),
(3,"casey",806509);
select * from employee;
```


keys
Primary key----> uinquely id
Create primary keys syntax sql
Foreign key --->multiples id
Or 2 table link karna .

```
create table employee(
id int primary key,
name varchar(56),
salary int);
```

```
create database navnath;
use navnath;
create table nav (
id int,
name varchar(34),
age int,
salary int,
primary key(id));
insert into  nav value(7867,"ram",34,6548365);
select * from nav;
```


Constraints --
Specify rules
Default value --> set default value of column
NOT NULL--> Columns cannot have a null values
UNIQUE--> all values in columns are different .
Check constraints ---> it can limits the value allowed in a column

```
Eg
create table  emp(
id int,
salary int default(7687));
select * from emp;
insert  into emp (id) values (100);
```

```
Eg;
create database ram;
use ram;
create table ram1(
id  int unique);
insert into ram1 values(67);
```

```
create table s1
(id int ,
name varchar(34),
city varchar(45),
age int,
constraint age_check check(age >=18 and  city="nanded"));
```

Q2. create simple database your college ?

Ans :

```
create database college;
use college;
create table college(
rollno int  primary key,
id int ,
name varchar(34),
mark int not null ,
grade  varchar(1),
```

i. city varchar(20)

```
);
insert into college (rollno,id,name,mark,grade,city)
values
(1,765437, "ram",56,"A","nanded"),
(23,34545,"navnath",45,"A","pune");
select * from college;
```

```
-- this command used for specific value find
    select  city from college;

-- this command used for not  duplicate   statements words .
    select distinct  city from college;

-- this command   used for  where clause
    select * from college where mark >35
```

Where clause operation

1. Arithmetic operation : +,-,*,/,%

Code:

```
-- Create a sample table
CREATE TABLE Numbers (
   id INT PRIMARY KEY,
   num1 INT,
   num2 INT
);

-- Insert some sample data
INSERT INTO Numbers (id, num1, num2)
VALUES (1, 10, 5),
       (2, 20, 8),
       (3, 15, 3);

-- Perform arithmetic operations and retrieve results
SELECT id,
       num1,
       num2,
       num1 + num2 AS addition,
       num1 - num2 AS subtraction,
       num1 * num2 AS multiplication,
       num1 / num2 AS division
FROM Numbers;
```

2. Assignment Operators:

```
-- Create a sample table
CREATE TABLE Employees (
   id INT PRIMARY KEY,
   name VARCHAR(50),
   salary DECIMAL(10, 2)
);

-- Insert some sample data
INSERT INTO Employees (id, name, salary)
VALUES (1, 'Alice', 50000),
       (2, 'Bob', 60000),
       (3, 'Charlie', 75000);

-- Update an employee's salary using the assignment operator
UPDATE Employees
SET salary = 65000
WHERE id = 2;

-- Display updated employee information
SELECT * FROM Employees;
```

3. Relational Operators:

```
-- Create a sample table
CREATE TABLE Students (
   id INT PRIMARY KEY,
   name VARCHAR(50),
   age INT,
   score INT
);

-- Insert some sample data
INSERT INTO Students (id, name, age, score)
VALUES (1, 'Alice', 18, 85),
       (2, 'Bob', 17, 92),
       (3, 'Charlie', 19, 78),
      (4, 'David', 16, 95),
      (5, 'Eve', 18, 70);

-- Retrieve students based on relational operators
-- Get students older than 17
SELECT * FROM Students WHERE age > 17;
```

4. Ternary operator :

```
-- Create a sample table
CREATE TABLE Employees (
   id INT PRIMARY KEY,
   name VARCHAR(50),
   salary DECIMAL(10, 2)
);

-- Insert some sample data
INSERT INTO Employees (id, name, salary)
VALUES (1, 'Alice', 50000),
      (2, 'Bob', 60000),
      (3, 'Charlie', 75000),
      (4, 'David', 45000);

-- Simulate a ternary operation to categorize employees' salary
SELECT id,
      name,
      salary,
      CASE
         WHEN salary > 60000 THEN 'High'
         WHEN salary > 50000 THEN 'Medium'
         ELSE 'Low'
      END AS salary_category
FROM Employees;
```

5. Comparison operation : =,!=,>,=<,>=

```
-- Create a sample table
CREATE TABLE Students (
   id INT PRIMARY KEY,
   name VARCHAR(50),
   age INT,
   grade CHAR(1)
);

-- Insert some sample data
INSERT INTO Students (id, name, age, grade)
VALUES (1, 'Alice', 18, 'A'),
      (2, 'Bob', 17, 'B'),
      (3, 'Charlie', 19, 'C'),
      (4, 'David', 16, 'A'),
      (5, 'Eve', 18, 'B');

-- Retrieve students based on comparison operators
-- Get students older than 17
SELECT * FROM Students WHERE age > 17;

-- Get students with grade 'A'
SELECT * FROM Students WHERE grade = 'A';

-- Get students between the ages of 16 and 18
SELECT * FROM Students WHERE age BETWEEN 16 AND 18;

-- Get students whose names start with 'A'
SELECT * FROM Students WHERE name LIKE 'A%';
```

6. Logical operation : AND , OR, NOT IN BETWEEN ALL LIKE ANY

```
CREATE TABLE Products (
   id INT PRIMARY KEY,
   name VARCHAR(50),
   price DECIMAL(10, 2),
   in_stock BOOLEAN
);

-- Insert some sample data
INSERT INTO Products (id, name, price, in_stock)
VALUES (1, 'Widget A', 10.99, true),
      (2, 'Widget B', 15.99, false),
      (3, 'Widget C', 8.49, true),
      (4, 'Widget D', 20.00, true),
      (5, 'Widget E', 12.75, false);

-- Retrieve products based on logical operators
-- Get products that are in stock and priced below $15
SELECT * FROM Products WHERE in_stock = true AND price < 15.00;

-- Get products that are either in stock or priced below $10
SELECT * FROM Products WHERE in_stock = true OR price < 10.00;

-- Get products that are not in stock
SELECT * FROM Products WHERE NOT in_stock;
```

7. Bitwise operation : & ,|

```
-- Create a sample table
CREATE TABLE BitwiseExample (
   id INT PRIMARY KEY,
   value1 INT,
   value2 INT
);

-- Insert some sample data
INSERT INTO BitwiseExample (id, value1, value2)
VALUES (1, 5, 3),
      (2, 10, 6),
      (3, 15, 9);

-- Perform bitwise operations and retrieve results
-- Bitwise AND
SELECT id, value1, value2, (value1 & value2) AS bitwise_and_result
FROM BitwiseExample;

-- Bitwise OR
SELECT id, value1, value2, (value1 | value2) AS bitwise_or_result
FROM BitwiseExample;

-- Bitwise XOR
SELECT id, value1, value2, (value1 ^ value2) AS bitwise_xor_result
FROM BitwiseExample;

-- Bitwise NOT
SELECT id, value1, ~value1 AS bitwise_not_result
FROM BitwiseExample;
```

LIMTS CLAUSE ---> SETS OF UPPER LIMITS ON NUMBER OF TUPPLES ROWS TO BE RETURENTED.
Syntax :
```
 Select *from student limit 3;
Select *from college  where mark >40 limit 3;
```

ORDER BY CLAUSE ---> To sort ascending order or descending order
[DESE]
SYNTAX:
```
Select * from college order by city asc;----> ascending order
Select * from college order by city desc;--->descending order
```

GROUP BY CLAUSE :
- groups rows that have same values into summary rows.
- It collects data from multiple records and group the result by one or more column.
- Generally we use group by with some aggregation function.

Syntax :
```
select city  from college group by  city;
select city, count(rollno) from college group by city;
select city, name ,count(rollno) from college group by city , name;
select city ,name,mark, count(rollno) from college group by city ,name,mark;
select city ,name , grade,mark,count(rollno) from college group by
city ,name,mark,grade;
select city, name,id,grade,mark,count(rollno) from college group by
city ,name,id,mark,grade;
```

Aggregate function ----> TO perform a calculation on a set of values and return a single value .
- Count()
- Max()
- Min()
- Sum()
- Avg()

Eg:
```
select  max(mark) from college;
select min(mark) from college;
select sum(mark) from college;
select avg(mark) from college;
```

Q3. write the query to find avg mark in each city in ascending order?
Ans:
```
create database college;
use college;
create table college(
rollno int  primary key,
id int ,
name varchar(34),
mark int not null ,
grade  varchar(1),
city varchar(20)
);
insert into college (rollno,id,name,mark,grade,city)
values
-- (1,765437, "ram",56,"A","nanded"),
-- (23,34545,"navnath",45,"A","pune"),
(31,765437, "ram",96,"A","nanded"),
(53,34545,"navnath",45,"A","pune"),
(66,765437, "ram",96,"A","nanded"),
(13,34545,"navnath",85,"A","pune");
select * from college;
select city,avg(mark)
from college group by city;
```

Q4.for the given table find payment according to each payment method ?
Ans:
```
create database payment11 ;
use payment11;
create table payment11(
customer_id int,
customer_name varchar(29),
payment_mode varchar(34),
city varchar(120));
select * from payment11;
insert into payment11 (customer_id, customer_name,payment_mode, city)
values
(101,"ram","netbanking","portland"),
(102,"ram","credit card","portland"),
(103,"ram","credit card","portland"),
(104,"ram","netbanking","portland"),
(105,"ram","credit card","portland"),
(106,"ram","debit card","portland"),
(107,"ram","debit card","portland"),
(108,"ram","netbanking","portland"),
(109,"ram","netbanking","portland"),
(110,"ram","credit card","portland");
select * from payment11;
select payment_mode, count(customer_name) from payment11 group by
payment_mode;
```

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 student in each city where max mark cross 90.
Syntax :
```
select count(name),city from priya group by city having max(mark)>90;
```

General order :
```
Select columns(s)
From table_name
Where condition
Group by columns(s)
Having condition
Order by columns(s) asc;
```

Eg:
```
Select city
From priya
Where grade="A"
Group by city
Having max(marks)>=90
Order by city asc;
```

TABLE related queries---> update (to update existing rows)
Syntax:
```
Update table_name
Set column= var1, col2=val2
Where condition
```
Eg1:
```
Update college
Set grade="o"
Where grade="A";
```
Eg2:
```
update priya1
set mark="90"
where rollno="23";
```
Eg3:
```
update priya1
set grade="A"
where mark between 80 and 90;
```
Eg4:
```
update priya1
set mark = mark+1
```

Update karte time this commans used

★ --THIS method used for **safe mode off or closed**
```
SET SQL_SAFE_UPDATES=0;
```

★ --THIS method used for **safe mode on or open**
```
SET SQL_SAFE_UPDATES=1;
```

Table related queries ----> delete (to delete existing rows)
Syntax:
```
Delete from table_name
Where condition
```
Eg1:
```
DELETE FROM COLLEGE
WHERE MARK<44;
```

REVISITING[Foreign key] ------> link between child table to parents table

Eg:
```
create database engineering;
use engineering;
create table dep(
id int primary key,
name varchar(20)
);

create table tech (
id int primary key,
name varchar(20),
dep_id int,
foreign key (dep_id) references dep(id)
);
```

Cascading means one places change to all other places changes

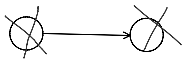
Cascading for foreign key

other places changes

Cascading for foreign key

On delete cascade---> jar delete zal tar both department delete other database.

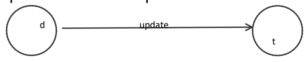
Eg.



when we create a foreign key using this option it delete the referencing rows in the child class 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 table when the referenced rows is update in parent table which has a primary key.

Eg, jar department made update zal tar teacher made pan update hote



Code :

```
create database  engineering1;
use engineering1;
create table dep(
id int primary key,
name varchar(20)
);
insert  into dep values
(1243,"IT"),
(1233,"computer");
update dep
set id =101
where  id =1234;

update dep
set id =102
where  id =1243;

select * from dep;

create table tech (
id int primary key,
name varchar(20),
dep_id int,
foreign key (dep_id) references dep(id)
on update cascade
on delete cascade
);
insert  into tech
values
(1243,"ram",1243),
(1233,"navnath",1233);

select * from tech;
```

Table related queries example:

```
create database  engineering3;
use engineering3;
create table engineering3(
rollno int  primary key,
id int ,
name varchar(34),
mark int not null ,
grade  varchar(1),
city varchar(20)
);
insert into engineering3 (rollno,id,name,mark,grade,city)
values
(1,765437, "RANI",56,"A","nanded"),
(23,34545,"SHALINI",45,"A","pune"),
(31,765437, "KAVITA",96,"A","latur"),
(53,34545,"navnath",45,"A","loha"),
(66,765437, "POOJA",96,"A","nashik"),
(13,34545,"AARTI",85,"A","satara");
select *from engineering3;
-- add column
alter  table  engineering3
add column  age int not null default 19;
select *from engineering3;

--  modifiy column
alter  table  engineering3
modify column age varchar(2);
select *from engineering3;

-- change table name
alter table  engineering3
change age  age_st int;

--  insert data
insert into engineering3
(rollno,id,name,mark,grade,city,age_st)
values
(103,87568," shri",46,"B","amravati",20);

-- delete data or column
alter table  engineering3
drop column age_st;

-- rename table
alter table  engineering3
rename to student;

-- rename table
alter table  student
rename to  engineering3 ;

-- delete data into the table
truncate table engineering3;
```

Table related queries :

Alter ---> to change the schema -->schema means design ---> columns

Add column --->add column
Alter table table_name
Add column column_name datatype;

Drop column-->delete
Alter table table_name
drop column column_name datatype;

Rename table --->rename table
Alter table table_name
Rename column column_name datatype;

Change column(rename)
Alter table table_name
Change column old_name new_name new data_types new_constraint;

Modify column (modifiy datatype/constraint)
Alter table table_name
Modifiy col_name new data_types new_constraint;

Q5. in the student table :

A. change the name of column "name" to full_name

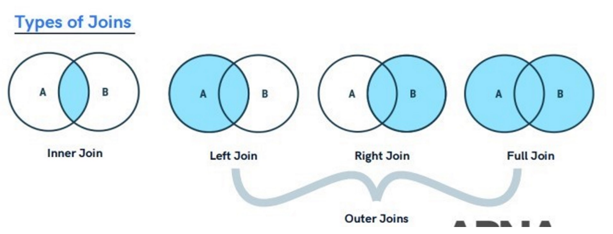
B. Delete all the student who scored mark is less then 80

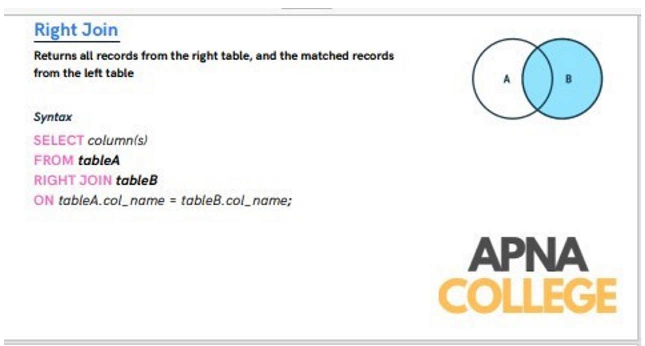
C. delete the column for grades.

Ans:

```
create database  engineering4;
use engineering4;
create table engineering4(
rollno int  primary key,
id int ,
name varchar(34),
mark int not null ,
grade  varchar(1),
city varchar(20)
);
insert into engineering4 (rollno,id,name,mark,grade,city)
values
(1,765437, "RANI",56,"A","nanded"),
(23,34545,"SHALINI",45,"A","pune"),
(31,765437, "KAVITA",96,"A","latur"),
(53,34545,"navnath",45,"A","loha"),
(66,765437, "POOJA",96,"A","nashik"),
(13,34545,"AARTI",85,"A","satara");
select *from engineering4;

-- rename the table
alter table engineering4
change name full_name varchar(20);

--  safe mode off
SET SQL_SAFE_UPDATES=0;

--  delete data for user condition
 delete from engineering4
 where mark < 80;

 alter table engineering4
 drop column grade
```

JOIN IN SQL ---> JOIN is used to combine rows from two or more tables, based on related on column between them.

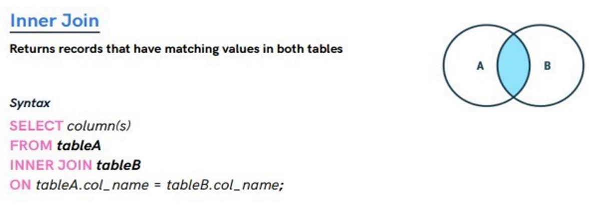

Types of Joins


Right Join

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

Syntax

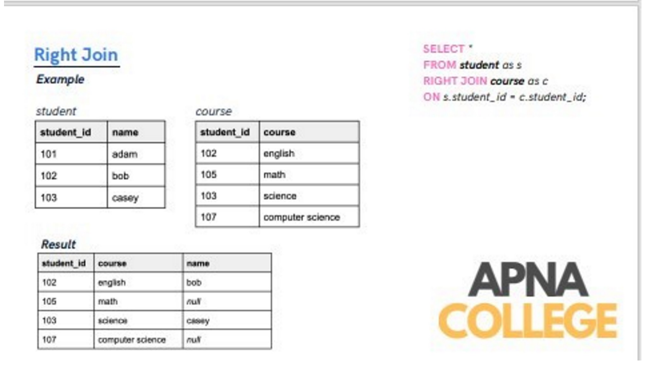
```
SELECT column(s)
FROM tableA
RIGHT JOIN tableB
ON tableA.col_name = tableB.col_name;
```


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;
```


Right Join

Example

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

student

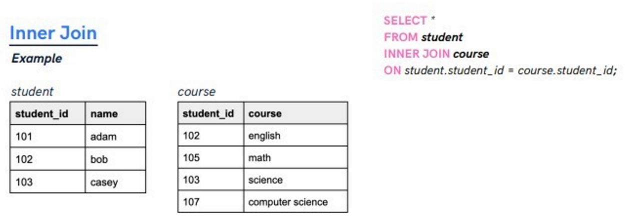
student_id	name
101	adam
102	bob
103	casey

course

student_id	course
102	english
105	math
103	science
107	computer science

Result

student_id	course	name
102	english	bob
105	math	null
103	science	casey
107	computer science	null


Inner Join

Example

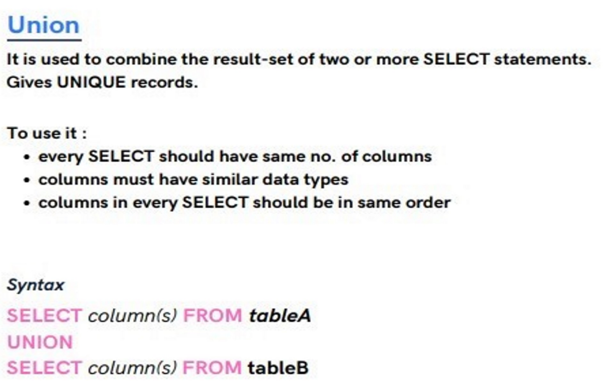
```
SELECT *
FROM student
INNER JOIN course
ON student.student_id = course.student_id;
```

student

student_id	name
101	adam
102	bob
103	casey

course

student_id	course
102	english
105	math
103	science
107	computer science


Union

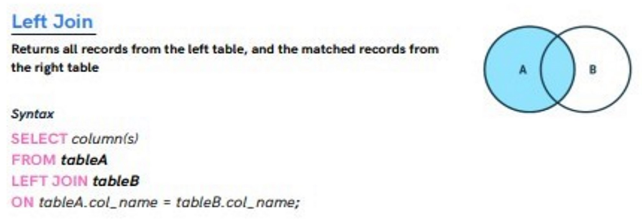
It is used to combine the result-set of two or more SELECT statements. Gives UNIQUE records.

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

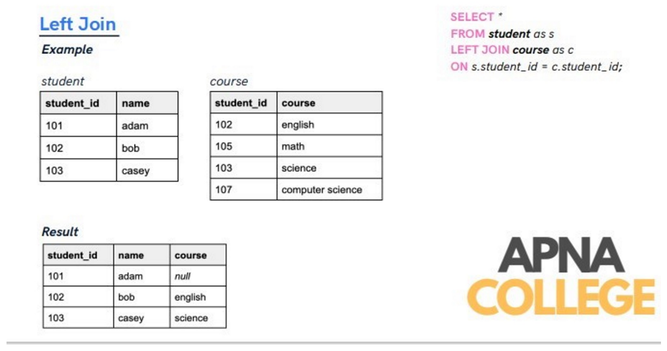
```
SELECT column(s) FROM tableA
UNION
SELECT column(s) FROM tableB
```


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;
```


Left Join

Example

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

student

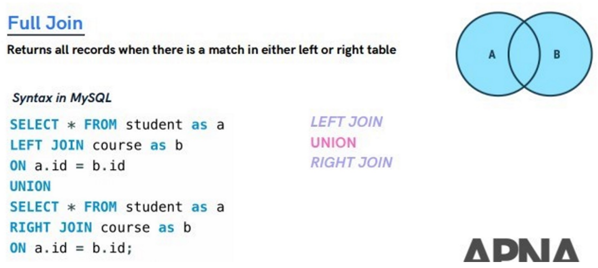
student_id	name
101	adam
102	bob
103	casey

course

student_id	course
102	english
105	math
103	science
107	computer science

Result

student_id	name	course
101	adam	null
102	bob	english
103	casey	science

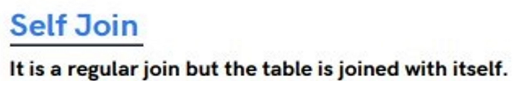

Full Join

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

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


Self Join

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

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	*null*
104	donald	103

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

```
RIGHT JOIN course as b
ON a.id = b.id;
```

Full Join

Example

student

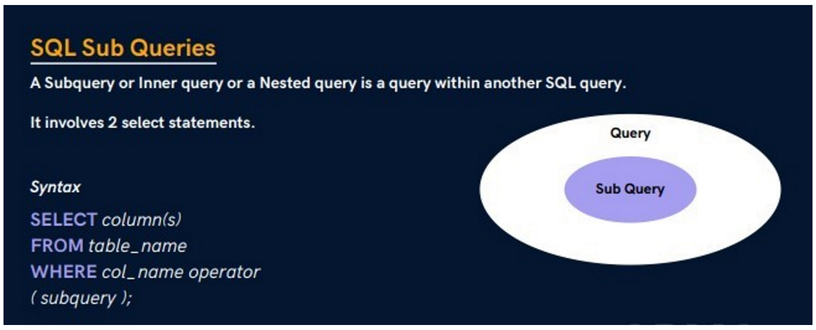
student_id	name
101	adam
102	bob
103	casey

course

student_id	course
102	english
105	math
103	science
107	computer science

Result

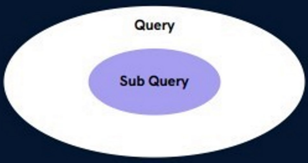
student_id	name	course
101	adam	*null*
102	bob	english
103	casey	science
105	*null*	math
107	*null*	computer science



SQL Sub Queries

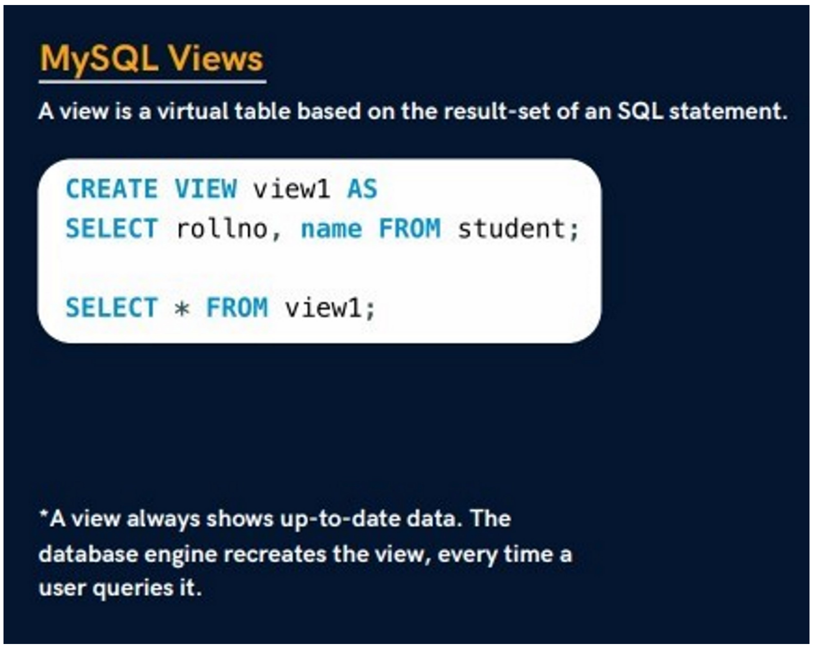
A Subquery or Inner query or a Nested query is a query within another SQL query.

It involves 2 select statements.



Syntax

SELECT *column(s)*
FROM *table_name*
WHERE *col_name operator*
(subquery);



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;

SELECT * FROM view1;
```

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