```sql
------------- SQL --------------


-- Creating table in database
CREATE TABLE student (
    id INT PRIMARY KEY,
    name VARCHAR(50),
    age INT NOT NULL
);

-- Inserting values in a table/schema
INSERT INTO student VALUES(1, "Kanad", 23);
INSERT INTO student VALUES(2, "Sudip", 25);

-- Show all data from a table
SELECT * FROM student;

/*
Types of SQL commands:
--------------------

DDL (Data Definition Language): create, alter, rename, truncate and drop
DQL (Data Query Language): select
DML (Data Manipulation Language): insert, update, delete
DCL (Data Control Language): grant and revoke permissions to users
TCL (Transaction Control Language): start transaction, commit, rollback (advanced)
*/

/*
Database Related Queries:
-----------------------
CREATE DATABASE db_name;
CREATE DATABASE IF NOT EXISTS db_name;

DROP DATABASE db_name;
DROP DATABASE IF EXISTS db_name;

SHOW DATABASES;
SHOW TABLES;
*/

CREATE DATABASE IF NOT EXISTS college;
DROP DATABASE IF EXISTS company;

SHOW DATABASES;
SHOW TABLES;

/*
Table Related Queries:
---------------------
1. create
CREATE TABLE table_name (
    column_name1 datatype constraint,
    column_name2 datatype constraint,
);
*/

CREATE TABLE student(
    id INT PRIMARY KEY, -- not null and unique
    name VARCHAR(50)
);

-- 2. Select and view all columns
-- SELECT * FROM table_name;

SELECT * FROM student;

/*
3. Insert
INSERT INTO table_name(column1, column2)
VALUES
(col1_val1, col2_val1),
(col1_val2, col2_val2);
*/
-- For multiple value we run this query
INSERT INTO student (id, name)
VALUES
(1, "Kanad"),
(2, "Sudip"),
(3, "Koushik");

-- For single value
INSERT INTO student VALUES(4, "Soumya");

/*
------------------------------------------------
Practice Question:
Qs: Create a database for your company named 'xyz'.
    → Create a table inside this DB to store
    employee info(id, name, salary)
    → Add following information in the DB:
        1, "adam", 25000
        2, "bob", 30000
        3, "casey", 40000
    → Select and view all your table data.
------------------------------------------------
*/
CREATE DATABASE IF NOT EXISTS xyz;
USE xyz;
-- Ans 1: Creating table
CREATE TABLE employees_info(
    id INT PRIMARY KEY,
    name VARCHAR(100),
    salary INT
);

-- Ans 2: inserting data into that table
INSERT INTO employees_info (id, name, salary)
VALUES
(1, "adam", 25000),
(2, "bob", 30000),
(3, "casey", 40000);

-- Ans 3: showing data from a table
SELECT * FROM employees_info;


/*
KEYS:

1. Primary Key:
--------------
→ It is a column (or a set of columns) in a table that uniquely identifies each row. (a unique id).
→ There can be only 1 Primary Key and it should be NOT NULL.


2. Foreign Key:
--------------
→ A foreign key is a column (or a set of columns) in a table that refers to the primary key in an another table.
→ There can be multiple foreign keys.
→ Foreign keys can have duplicate and null values.


--- CONSTRAINTS ---
1. NOT NULL (columns cannot have a null value) col1 int NOT NULL
2. UNIQUE (all values in column are different) col2 int UNIQUE
3. PRIMARY KEY ()
*/

USE college;

CREATE TABLE temp1(
    id INT UNIQUE
);

INSERT INTO temp1 VALUES(101); -- this will run fine
INSERT INTO temp1 VALUES(101); -- but here this will give error as duplicate entry 101.

SELECT * FROM temp1;

CREATE TABLE temp2(
    id INT UNIQUE,
    name VARCHAR(50) NOT NULL,
    age INT
);

INSERT INTO temp2 VALUES(1, "Kanad", 23);
INSERT INTO temp2(id, age) VALUES(2, 25); -- error: field 'name' doesn't have a default value.
INSERT INTO temp2(id, name) VALUES(2, "Sudip"); -- this will run fine as age is not necessary.

/*
→ Ways to define Primary Key:
*/

-- First syntax:
CREATE TABLE temp3(
    id INT PRIMARY KEY,
    name VARCHAR(50),
    age INT,
    city VARCHAR(30)
);

-- Second syntax:
CREATE TABLE temp3(
    id INT,
    name VARCHAR(50),
    age INT,
    city VARCHAR(30),
    PRIMARY KEY (id)
);

-- Third syntax: (combination as primary key)

CREATE TABLE temp3(
    id INT,
    name VARCHAR(50),
    age INT,
    city VARCHAR(30),
    PRIMARY KEY (id, name) -- id or name can be duplicate but (id, name) combination can't
);

/*
--- FOREIGN KEY ---
→ Prevent action that would detroy links between tables

*/
CREATE TABLE customer (
    id INT PRIMARY KEY,
    name VARCHAR(50)
);
CREATE TABLE buying_items(
    cust_id int,
    FOREIGN KEY (cust_id) REFERENCES customer(id)
);

/*
--- DEFAULT ---
→ sets the default value of a column
→ salary INT DEFAULT 20000
*/

CREATE TABLE emp (
    id INT,
    salary INT DEFAULT 25000
);

INSERT INTO emp(id) VALUES(2154); -- here salary is not mandatory
SELECT * FROM emp; -- this will show one row with | 2154 | 25000 |

/*
--- CHECK ---
→ It can limit the values allowed in a column
*/

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

INSERT INTO newtab VALUES(17); -- this shows check constraint is violated
INSERT INTO newtab VALUES(18); -- this one will run fine

SELECT * FROM newtab;

-- One more way to use CHECK:
CREATE TABLE city (
    id INT PRIMARY KEY,
    city VARCHAR(50),
    age INT,
    CONSTRAINT age_check CHECK (age >= 18 AND city="Delhi")
);

INSERT INTO city VALUES(1, "Mumbai", 21); -- age_check violated
INSERT INTO city VALUES(1, "Delhi", 17); -- age_check violated
INSERT INTO city VALUES(1, "delhi", 27); -- runs fine
INSERT INTO city VALUES(2, "Delhi", 23); -- runs fine
INSERT INTO city VALUES(3, "DELHI", 23); -- runs fine
INSERT INTO city VALUES(4, "dElHI", 23); -- runs fine




-------------------- Part 1 -----------------------

USE college;

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

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 * FROM student;

/*
------- SELECT in detail ----------
*/

-- Basic: SELECT col1, col2 FROM table_name;
SELECT name, marks FROM student; -- this will show name and marks in table
SELECT city FROM student; -- show all cities
SELECT DISTINCT city FROM student; -- shows all distinct city (for this: Pune, Mumbai, Delhi)

/*
---------- WHERE Clause ------------
→ to define some conditions
SYNTAX:
    SELECT col1, col2 FROM table_name
    WHERE condition;
*/

SELECT * FROM student WHERE marks > 80; -- showing 4 rows
SELECT * FROM student WHERE city = "Mumbai"; -- showing 2 results
SELECT * FROM student WHERE (city = "Delhi" AND grade = "A"); -- showing one result

/*
-------------- OPERATORS --------------

⇒ While using "WHERE", we use a lot of operators

1. Arithmetic Operators: +, -, *, /, %
2. Comparison Operators: =, ≠, >, >=, <, <=
3. Logical Operators: AND, OR, NOT, IN, BETWEEN, ALL, LIKE, ANY
4. Bitwise Operators: &(bitwise AND), | (bitwise OR)
*/

SELECT * FROM student WHERE (marks > 80+10 AND grade = "A"); -- showing 2 result
SELECT * FROM student WHERE (marks % 2 = 0 AND grade = "A"); -- showing 1 result
SELECT * FROM student WHERE (marks % 2 = 0 OR grade = "A"); -- showing 5 result
SELECT * FROM student WHERE marks BETWEEN 80 AND 90; -- showing 2 result (it includes 80 and 90)
SELECT * FROM student WHERE city IN ("Delhi", "Kolkata", "Mumbai", "Hyderabad"); -- showing 5 result
SELECT * FROM student WHERE city NOT IN ("Delhi", "Kolkata", "Mumbai", "Hyderabad"); -- showing 1 result


/*
---------- LIMIT Clause -------------
⇒ Sets an upper limit on number of (tuples) rows to be returned
SYNTAX:
    SELECT col1, col2 FROM table_name
    LIMIT number;
*/

SELECT * FROM student LIMIT 2; -- shows 2 rows
SELECT * FROM student WHERE marks > 75 LIMIT 10; -- shows 5 rows


/*
---------- ORDER BY Clause -------------
⇒ To sort results in ASC(ascending) or DESC (descending) order
SYNTAX:
    SELECT col1, col2 FROM table_name
    ORDER BY col_name(s) ASC;
*/

SELECT name, marks FROM student
ORDER BY city ASC; -- gives result in ascending order of city (delhi then mumbai then pune)

-- Lets find out top 3 students
SELECT * FROM student
ORDER BY marks DESC LIMIT 3; -- dhruv, bhumika, chetan


/*
------------------ Aggregate Functions ------------------------
→ Aggregate functions performs a calculation on a set of values, and return a single value

- COUNT()
- MAX()
- MIN()
- SUM()
- AVG()

⇒ For example: SELECT max(marks) FROM student;
*/

SELECT MAX(marks) FROM student; -- shows 96
SELECT MIN(marks) FROM student; -- shows 12
SELECT AVG(marks) FROM student; -- shows 74.3333
SELECT COUNT(rollNo) FROM student; -- shows 6
SELECT SUM(marks) FROM student; -- shows 446


/*
----------------- 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.

For ex: Count number of students in each city
*/

SELECT city, COUNT(rollNo)
FROM student
GROUP BY city;
/*
→ Group the data based on cities
Pune    1
Mumbai  2
Delhi   3
*/

SELECT city, name, COUNT(rollNo)
FROM student
GROUP BY city, name;
/*
→ Group the data based on cities and name
Pune        anil        1
Mumbai      bhumika     1
Mumbai      chetan      1
Delhi       dhruv       1
Delhi       emanuel     1
Delhi       farah       1
*/

-- Lets get the average marks based on city
SELECT city, AVG(marks)
FROM student
GROUP BY city;
/*
Pune    78.0000
Mumbai  89.0000
Delhi   63.3333
*/

/*
---------------------------------------------------------------------
Practice Qs:
Qs: Write the query to find avg marks in each city in ascending order
*/

SELECT city, AVG(marks)
FROM student
GROUP BY city
ORDER BY city ASC;
/*
Delhi   63.3333
Mumbai  89.0000
Pune    78.0000
*/

SELECT city, AVG(marks)
FROM student
GROUP BY city
ORDER BY AVG(marks) ASC;
/*
Delhi   63.3333
Pune    78.0000
Mumbai  89.0000
*/
----------------------------------------------------------

/*
----------------------------------------------------------
Practice Qs:
→ For a given table, find the total payment according to each payment method.

--------------------------------------------------------------------------
customer id    -        customer      -      mode       -     city         -
--------------------------------------------------------------------------
101            -     Olivia Barrett    -  Netbanking     -     Portland     -
102            -     Ethan Sinclair    -  Credit Card    -     Miami        -
103            -     Maya Hernandez    -  Credit Card    -     Seattle      -
104            -     Liam Donovan      -  Netbanking     -     Denver       -
105            -     Sophia Nguyen     -  Credit Card    -     New Orleans  -
106            -     Caleb Foster      -  Debit Card     -     Minneapolis  -
107            -     Ava Patel         -  Debit Card     -     Phoenix      -
108            -     Lucas Carter      -  Netbanking     -     Boston       -
109            -     Isabella Martinez -  Netbanking     -     Nashville    -
110            -     Jackson Brooks    -  Credit Card    -     Boston       -
--------------------------------------------------------------------------
*/

-- Create the table
CREATE TABLE customers (
    customer_id INT PRIMARY KEY,
    customer_name VARCHAR(100),
    payment_mode VARCHAR(20),
    city VARCHAR(50)
);

-- Insert the data
INSERT INTO customers (customer_id, customer_name, payment_mode, city) VALUES
(101, 'Olivia Barrett', 'Netbanking', 'Portland'),
(102, 'Ethan Sinclair', 'Credit Card', 'Miami'),
(103, 'Maya Hernandez', 'Credit Card', 'Seattle'),
(104, 'Liam Donovan', 'Netbanking', 'Denver'),
(105, 'Sophia Nguyen', 'Credit Card', 'New Orleans'),
(106, 'Caleb Foster', 'Debit Card', 'Minneapolis'),
(107, 'Ava Patel', 'Debit Card', 'Phoenix'),
(108, 'Lucas Carter', 'Netbanking', 'Boston'),
(109, 'Isabella Martinez', 'Netbanking', 'Nashville'),
(110, 'Jackson Brooks', 'Credit Card', 'Boston');

SELECT payment_mode, COUNT(customer_name)
FROM customers
GROUP BY payment_mode;

/*
Result:
-------
Netbanking      4
Credit Card     4
Debit Card      2
*/

----------------------

-- Q: Lets take the above student table and write a query to group students
-- according to their grade.

SELECT grade, COUNT(rollNo)
FROM student
GROUP BY grade
ORDER BY grade ASC;
/*
Result:
------
A   2
B   2
C   1
F   1
*/

/*
-------------- Having Clause --------------
               ------------
→ Similar to Where i.e. applies some condition on rows.
→ Used when we want to apply any condition after grouping.

→ For ex: Count number of students in each city where max marks cross 90/
*/

SELECT city, COUNT(rollNo)
FROM student
GROUP BY city;
/*
Result:
------
Pune    1
Mumbai  2
Delhi   3
*/

-- Now, after applying having clause on this query,
SELECT city, COUNT(rollNo)
FROM student
GROUP BY city
HAVING MAX(marks) > 90;
/*
Result:
------
Mumbai  2
Delhi   3
*/

/*
-------------- General Order -------------
               -----------
SELECT column(s)
FROM table_name
WHERE condition
GROUP BY condition
HAVING condition
ORDER BY column(s) ASC;
*/

SELECT city
FROM student
WHERE grade = "A"
GROUP BY city
HAVING MAX(marks) > 90
ORDER BY city DESC;
/*
Result:
------
Mumbai
Delhi
*/

/*
---------- Table Related Queries ------------

Update (to update existing rows)
-------------------------------
SYNTAX:-
------
    UPDATE table_name
    SET col1 = val1, col2 = val2
    WHERE condition;
*/

-- Sometime SQL strict us to update any table. So in that case we have to run one command
SET SQL_SAFE_UPDATES = 0; -- mostly this problem visible in SQL Workbench

UPDATE student
SET grade = "O"
WHERE grade = "A";

SELECT * FROM student; -- here the grade "A" changed to grade "O".

-- Update student emanuel marks to 82 and grade A

UPDATE student
SET marks = 82, grade = "B"
WHERE rollNo = 105;


/*
Lets say we have to increment each student marks by 1
*/

UPDATE student
SET marks = marks + 1; -- this will increment each student marks by 1

/*
---------------- Delete ---------------
                 ------
→ To delete existing rows
SYNTAX:-
    DELETE FROM table_name
    WHERE condition;
*/

UPDATE student
SET marks = 12
WHERE rollNo = 105; -- this will again set emanuel marks to 12

-- Now we can apply a delete query
-- Delete the student whose marks < 33

DELETE FROM student
WHERE marks < 33; -- this has deleted emanuel's row



------------------- Part 2 -----------------------


/*
------------ Foreign Key in Detail ---------------

*/


USE college;
CREATE TABLE dept (
    id INT PRIMARY KEY,
    subject_name VARCHAR(50)
);

CREATE TABLE teacher (
    id INT PRIMARY KEY,
    name VARCHAR(50),
    dept_id INT,
    FOREIGN KEY (dept_id) REFERENCES dept(id)
);

-- We can see the connection between these two table in the er diagram provided in github

-- dept → parent table
-- teacher → child table

--------------------------------------------------------------

---------------- Cascading for Foreign Key ---------------------
                 ------------------------
/*
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
*/

-- so we have to write this create table in this way
CREATE TABLE teacher (
    id INT PRIMARY KEY,
    name VARCHAR(50),
    dept_id INT,
    FOREIGN KEY (dept_id) REFERENCES dept(id)
    ON UPDATE CASCADE
    ON DELETE CASCADE
);

INSERT INTO dept
VALUES
(101, "CSE"),
(102, "IT");

SELECT * FROM dept;

INSERT INTO teacher
VALUES
(1001, "jayanta", 101),
(1002, "arindam", 101),
(1003, "sagar", 102),
(1004, "tanaya", 101),
(1005, "yusuf", 102);

SELECT * FROM teacher;

UPDATE dept
SET id = 103
WHERE subject_name = "CSE" AND id = 101;
-- this will not just update dept table it will also update teacher table's dept_id

---------------------------------------------------
------------- 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;

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_contraint;

*/

ALTER TABLE student
ADD COLUMN age INT; -- this will add a new column age in student

ALTER TABLE student
DROP COLUMN age; -- this will delete the age column from student

ALTER TABLE student
RENAME TO stu; -- this will rename student table to stu

ALTER TABLE stu
ADD COLUMN age INT; -- adding new col age

ALTER TABLE stu
CHANGE age stu_age INT; -- change column name from age to stu_age

ALTER TABLE stu
MODIFY stu_age VARCHAR(2); -- change column datatype int to varchar

SELECT * FROM stu;

/*
---------------------------- TRUNCATE ---------------------------
                             ----------
→ DROP TABLE deletes whole table with data whereas
→ TRUNCATE TABLE table_name delete only table's data not the table
*/

TRUNCATE TABLE stu; -- this will delete all the data but not the table


/*
-------------------------- PRACTICE Qs: --------------------------------
Qs: In the student table:
    → change the name of the column "name" to "full_name".
    → Delete all the students who scored marks less than 80.
    → Delete the column for grades
*/

SELECT * FROM stu;

--- Ans: Q1
ALTER TABLE stu
CHANGE name full_name VARCHAR(50);

--- Ans: Q2

DELETE FROM stu
WHERE marks < 80;

--- Ans: Q3
ALTER TABLE stu
DROP COLUMN grade;

-----------------------------------------------------------------
```

```
/*
------------------ JOINS in SQL ---------------------
                   ------------------

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

         Employee                          Salary
    id           name                   id      salary
    -----------------                   -----------
    101          john                   102     25000
    102          bob                    103     50000

    So, here from the above columns we can join name and salary based on related id 102.


----------------- Types of Joins -----------------

1. Inner Join
       A          B
    [   ]---[   ]
       \    /
        \__/       <- Only matching records (A ∩ B)


2. Outer Join
   ------------
   i) Left Join, ii) Right Join, iii) Full Outer Join

--- Left Join ---
    ---------
      A         B
    [   ]---[   ]
    [_______]
       \___/         <- All from A + matching from B

--- Right Join ---
    ----------
      A         B
    [   ]---[   ]
         [_______]
            \___/    <- All from B + matching from A

--- Full Outer Join ---
    ---------------
      A         B
    [   ]   [   ]
    [___________]    <- All from A + B (matched or not)


For better understanding see the diagram in google or in this repo.

*/


------------ INNER JOIN ------------
             ------------
/*
-- Return records that have matching values in  both tables
SYNTAX:
------
        SELECT column(s)
        FROM tableA
        INNER JOIN tableB
        ON tableA.col_name = tableB.col_name

-> For ex:
This is our student table

+---------+-----------+-------+--------+
| rollNo | full_name | marks | city   |
+---------+-----------+-------+--------+
|  102   | bhumika   |  93   | Mumbai |
|  103   | chetan    |  85   | Mumbai |
|  104   | dhruv     |  96   | Delhi  |
|  106   | farah     |  82   | Delhi  |
+---------+-----------+-------+--------+

and course table is:
+---------+------------------+
| rollNo | course           |
+---------+------------------+
|  102   | english          |
|  103   | math             |
|  104   | science          |
|  106   | computer science |
+---------+------------------+

and now after performing inner join on student and course table with respect to rollNo and want to
see full_name, city and marks column then the result will be:

+-----------+--------+-------+
| full_name | city   | marks |
+-----------+--------+-------+
| bhumika   | Mumbai |  93   |
| chetan    | Mumbai |  85   |
| dhruv     | Delhi  |  96   |
| farah     | Delhi  |  82   |
+-----------+--------+-------+


*/

SELECT * FROM stu;

CREATE TABLE course(
    rollNo INT,
    course VARCHAR(50),
    FOREIGN KEY (rollNo) REFERENCES stu(rollNo)
);

INSERT INTO course
VALUES
(102, "english"),
(103, "math"),
(104, "science"),
(106, "computer science");

SELECT * FROM course;

---- Inner Join ----
SELECT full_name, city, marks
FROM stu
INNER JOIN course
ON stu.rollNo = course.rollNo;

/*
RESULT:
------

+-----------+--------+-------+
| full_name | city   | marks |
+-----------+--------+-------+
| bhumika   | Mumbai |  93   |
| chetan    | Mumbai |  85   |
| dhruv     | Delhi  |  96   |
| farah     | Delhi  |  82   |
+-----------+--------+-------+

*/


------------ 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

-> For ex:
This is our student table

+---------+-----------+-------+--------+
| rollNo | full_name | marks | city   |
+---------+-----------+-------+--------+
|  102   | bhumika   |  93   | Mumbai |
|  103   | chetan    |  85   | Mumbai |
|  104   | dhruv     |  96   | Delhi  |
|  106   | farah     |  82   | Delhi  |
+---------+-----------+-------+--------+

and course table is:
+---------+------------------+
| rollNo | course           |
+---------+------------------+
|  102   | english          |
|  111   | math             |
|  114   | science          |
|  106   | computer science |
+---------+------------------+

and now after performing left join on student and course table with respect to rollNo and want to
see full_name, city and marks column then the result will be:

+---------+-----------+-------+--------+--------+------------------+
| rollNo | full_name | marks | city   | rollNo | course           |
+---------+-----------+-------+--------+--------+------------------+
|  102   | bhumika   |  93   | Mumbai | 102    | english          |
|  103   | chetan    |  85   | Mumbai | (NULL) | (NULL)           |
|  104   | dhruv     |  96   | Delhi  | (NULL) | (NULL)           |
|  106   | farah     |  82   | Delhi  | 106    | computer science |
+---------+-----------+-------+--------+--------+------------------+

*/

DROP TABLE course;

CREATE TABLE course(
    rollNo INT,
    course VARCHAR(50)
);

INSERT INTO course
VALUES
(102, "english"),
(111, "math"),
(114, "science"),
(106, "computer science");

SELECT * FROM course;

SELECT *
FROM stu as s
LEFT JOIN course as c
ON s.rollNo = c.rollNo;

/*
RESULT:
------

+---------+-----------+-------+--------+--------+------------------+
| rollNo | full_name | marks | city   | rollNo | course           |
+---------+-----------+-------+--------+--------+------------------+
|  102   | bhumika   |  93   | Mumbai | 102    | english          |
|  103   | chetan    |  85   | Mumbai | (NULL) | (NULL)           |
|  104   | dhruv     |  96   | Delhi  | (NULL) | (NULL)           |
|  106   | farah     |  82   | Delhi  | 106    | computer science |
+---------+-----------+-------+--------+--------+------------------+

*/

------------ 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

-> For ex:
This is our student table

+---------+-----------+-------+--------+
| rollNo | full_name | marks | city   |
+---------+-----------+-------+--------+
|  102   | bhumika   |  93   | Mumbai |
|  103   | chetan    |  85   | Mumbai |
|  104   | dhruv     |  96   | Delhi  |
|  106   | farah     |  82   | Delhi  |
+---------+-----------+-------+--------+

and course table is:
+---------+------------------+
| rollNo | course           |
+---------+------------------+
|  102   | english          |
|  111   | math             |
|  114   | science          |
|  106   | computer science |
+---------+------------------+

and now after performing left join on student and course table with respect to rollNo and want to
see full_name, city and marks column then the result will be:

+---------+-----------+-------+--------+--------+------------------+
| rollNo | full_name | marks | city   | rollNo | course           |
+---------+-----------+-------+--------+--------+------------------+
|  102   | bhumika   |  93   | Mumbai | 102    | english          |
| (NULL) | (NULL)    | NULL  | NULL   | 111    | math             |
| (NULL) | (NULL)    | NULL  | NULL   | 114    | science          |
|  106   | farah     |  82   | Delhi  | 106    | computer science |
+---------+-----------+-------+--------+--------+------------------+


*/

SELECT *
FROM stu as s
RIGHT JOIN course as c
ON s.rollNo = c.rollNo;

/*
RESULT:
------

+---------+-----------+-------+--------+--------+------------------+
| rollNo | full_name | marks | city   | rollNo | course           |
+---------+-----------+-------+--------+--------+------------------+
|  102   | bhumika   |  93   | Mumbai | 102    | english          |
| (NULL) | (NULL)    | NULL  | NULL   | 111    | math             |
| (NULL) | (NULL)    | NULL  | NULL   | 114    | science          |
|  106   | farah     |  82   | Delhi  | 106    | computer science |
+---------+-----------+-------+--------+--------+------------------+

*/

/*
------------------ FULL JOIN / FULL OUTER JOIN ------------------
                   ---------------------------
-> Returns all records when there is a match in either left or right table.

*** In MySQL, Full Join does not work. But in Oracle database it does.

so, the general syntax for FULL JOIN is:
SYNTAX:
------
        SELECT column(s)
        FROM tableA
        FULL JOIN tableB
        ON tableA.col_name = tableB.col_name

But in MySQL, we can use Union to perform full join

SYNTAX:
-------
        SELECT *
        FROM tableA
        LEFT JOIN tableB
        ON tableA.rollNo = tableB.rollNo

        UNION

        SELECT *
        FROM tableA
        RIGHT JOIN tableB
        ON tableA.rollNo = tableB.rollNo;

*/

SELECT *
FROM stu as s
LEFT JOIN course as c
ON s.rollNo = c.rollNo

UNION

SELECT *
FROM stu as s
RIGHT JOIN course as c
ON s.rollNo = c.rollNo;

/*
RESULT:
------

+---------+-----------+-------+--------+--------+------------------+
| rollNo | full_name | marks | city   | rollNo | course           |
+---------+-----------+-------+--------+--------+------------------+
|  102   | bhumika   |  93   | Mumbai | 102    | english          |
|  103   | chetan    |  85   | Mumbai | NULL   | NULL             |
|  104   | dhruv     |  96   | Delhi  | NULL   | NULL             |
|  106   | farah     |  82   | Delhi  | 106    | computer scier   |
|  NULL  | NULL      | NULL  | NULL   | 111    | math             |
|  NULL  | NULL      | NULL  | NULL   | 114    | science          |
+---------+-----------+-------+--------+--------+------------------+

*/

/*
There are also some another types of joins. Lets see them one by one.
*/

----------  Left Exclusive Join ------------
            --------------------

SELECT *
FROM stu as s
LEFT JOIN course as c
ON s.rollNo = c.rollNo
WHERE c.rollNo IS NULL;

/*
RESULT:
------

+---------+-----------+-------+--------+--------+------------------+
| rollNo | full_name | marks | city   | rollNo | course           |
+---------+-----------+-------+--------+--------+------------------+
|  103   | chetan    |  85   | Mumbai | (NULL) | (NULL)           |
|  104   | dhruv     |  96   | Delhi  | (NULL) | (NULL)           |
+---------+-----------+-------+--------+--------+------------------+
*/

----------  Right Exclusive Join ------------
            ---------------------

SELECT *
FROM stu as s
RIGHT JOIN course as c
ON s.rollNo = c.rollNo
WHERE s.rollNo IS NULL;

/*
RESULT:
------

+---------+-----------+-------+--------+--------+------------------+
| rollNo | full_name | marks | city   | rollNo | course           |
+---------+-----------+-------+--------+--------+------------------+
| (NULL) | (NULL)    | NULL  | NULL   | 111    | math             |
| (NULL) | (NULL)    | NULL  | NULL   | 114    | science          |
+---------+-----------+-------+--------+--------+------------------+
*/


----------  Full Exclusive Join ------------
            --------------------

SELECT *
FROM stu as s
LEFT JOIN course as c
ON s.rollNo = c.rollNo
WHERE c.rollNo IS NULL

UNION

SELECT *
FROM stu as s
RIGHT JOIN course as c
ON s.rollNo = c.rollNo
WHERE s.rollNo IS NULL;

/*
RESULT:
------

+---------+-----------+-------+--------+--------+------------------+
| rollNo | full_name | marks | city   | rollNo | course           |
+---------+-----------+-------+--------+--------+------------------+
|  103   | chetan    |  85   | Mumbai | (NULL) | (NULL)           |
|  104   | dhruv     |  96   | Delhi  | (NULL) | (NULL)           |
| (NULL) | (NULL)    | NULL  | NULL   | 111    | math             |
| (NULL) | (NULL)    | NULL  | NULL   | 114    | science          |
+---------+-----------+-------+--------+--------+------------------+
*/

----------  Self Join ------------
            ---------
/*
It is a regular join but the table is joined with itself.

SYNTAX:
------
        SELECT column(s)
        FROM tableA AS a
        JOIN tableB AS b
        ON a.col_name = b.col_name;
*/


CREATE TABLE employee(
    id INT PRIMARY KEY,
    name VARCHAR(50),
    manager_id INT
);

INSERT INTO employee (id, name, manager_id)
VALUES
(101, "adam", 103),
(102, "bob", 104),
(103, "casey", NULL),
(104, "donald", 103);

SELECT * FROM employee;

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

/*
RESULT:
------

+--------------+--------+
| manager_name | name   |
+--------------+--------+
|  casey       | chetan |
|  donald      | dhruv  |
|  casey       | donald |
+--------------+--------+
*/

/*
------------------------ 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

and

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

- the last query will give duplicates also.

*/


/*
----------------------- SQL SUB QUERIES ---------------------
                        ---------------------

-> A subquery or inner query or a nested query is a query within another SQL query.
-> It involves 2 select statements.

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

*/

SELECT * FROM student;

/*
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
*/

SELECT AVG(marks) FROM student; -- 87.667

SELECT name
FROM student
WHERE marks > 87.667;

-- But if more students join later then this will change.
-- so for that reason we can use sub queries.

SELECT name, marks
FROM student
WHERE marks > (
    SELECT AVG(marks)  FROM student
);
-- the above sub query will also give us the same result.
-- this subquery is dynamic SQL query.

-- Qs: Find the name of all students with even roll nubmers.
-- Step 1: Find the even roll numbers.
-- Step 2: Find the names of students will even roll no.

SELECT rollNo
FROM student
WHERE rollNo % 2 = 0;

SELECT name
FROM student
WHERE rollNo IN(102, 104, 106);
/*
Result:
------
bhumika
dhruv
farah
*/

-- Now using sub query
SELECT name
FROM student
WHERE rollNo IN (
    SELECT rollNo
    FROM student
    WHERE rollNo % 2 = 0
);

-- this will also give us the same result
/*
Result:
------
bhumika
dhruv
farah
*/


/*

Example using FROM:
------------------

Qs: Find the max marks from the students of Delhi.

Step 1: Find the students of Delhi.
Step 2: Find the max marks using the sublist in step 1

*/

SELECT *
FROM student
WHERE city = "delhi";

SELECT MAX(marks)
FROM (
    SELECT *
    FROM student
    WHERE city = "delhi"
) AS temp;
/**
Result:
------
96
/

-- as Temp is like a temporary table which we get from the sub query
-- and if we are using FROM then we have to write this kind of aliases.

/*
Using sub-queries inside SELECT
*/

SELECT (SELECT MAX(marks) FROM student), name FROM student;
/*
Result:
-------
96      anil
96      bhumika
96      chetan
96      dhruv
96      emanuel
96      farah
*/

----------------------------------------------------
 /*
 -------------------    VIEWS   -------------------
                       ----------

-> A view is a virtual table based on the result-set of an SQL statement.
NOTE:
----
    A view always show us up-to-date data.
    The database engine recreates the view each time a user make a
    query.
*/

CREATE VIEW view1 AS
SELECT rollNo, name
FROM student;

/*
After creating a view we can make multiple queries in this view
as like a table.
*/

SELECT name  FROM view1;

/*
Result:
-------
anil
bhumika
chetan
dhruv
emanuel
farah
*/

----------------------------------------------------
```