```
SHOW DATABASES;         // to display databases

CREATE DATABASE db_name;   // to create db

USE db_name;              // selecting the db

SHOW TABLES;           // to display all tables in the db

DROP DATABASE db_name;    // to delete a db
```

DDL

==========

Numeric Data Types

INT : whole numbers

FLOAT : decimal numbers (approximate)

DECIMAL(M,D) : decimal numbers (precise), for price cols

....

NON Numeric Data Types

CHAR(N) : fixed length character

VARCHAR(N) : varying length character, good for names

ENUM('M','F') : values from a defined list

BOOLEAN : true or false value

....

DATE AND TIME Types

DATE : date (YYYY-MM_DD)

DATETIME : date and the time (YYYY-MM-DD HH:MM:SS)

TIME : time (HHH-MM-SS) can be used to find time bw two events

YEAR : year (YYYY)

==========

PK

PK = UNIQUE + NULL

is a col or a set of cols which uniquely identifies a record in a table.

FK

used to link two tables

is a col whose values mathch the values of another tables PK

```
================================================================================
=======

// ADD AND DELTE COLUMNS INTO A TABLE

SHOW Databases;

USE coffee_store;
SHOW tables;

SELECT * FROM products;

ALTER TABLE products
ADD COLUMN coffee_origin VARCHAR(20);
SELECT * FROM products;

ALTER TABLE products
DROP COLUMN coffee_origin;

================================================================================
=======

// HOW TO DELETE A TABLE


CREATE DATABASE example;

USE example;

CREATE TABLE test(

      id INT AUTO_INCREMENT PRIMARY KEY,
      name VARCHAR(20),
      age INT
)

// SELECT * FROM TEST; displays empty table ..

DROP TABLE test; // deletes the table

================================================================================
======

// To delete all the content in the database...

TRUNCATE TABLE test;
```

```
DELETE FROM test;


exg
SHOW Databases;

USE coffee_store;
SHOW tables;

SELECT * FROM products;

-- INSERT INTO products VALUE(1, "k Badam", 1.50);

-- INSERT INTO products VALUES(2, "k Mango", 5.50);

SET FOREIGN_KEY_CHECKS=0;

TRUNCATE TABLE products;

SET FOREIGN_KEY_CHECKS=1;


================================================================================
======

-- MORE ON ALTER TABLES


CREATE DATABASE test;
SHOW DATABASES;
USE test;

-- firstly creating tables
CREATE TABLE addresses (
      id INT,
    house_number INT,
    city VARCHAR(20),
    postcode VARCHAR(7)
);

CREATE TABLE people (
      id INT,
    first_name VARCHAR(20),
    last_name VARCHAR(20),
    address_id INT
);

CREATE TABLE pets (
      id INT,
    name VARCHAR(20),
    species VARCHAR(20),
    owner_id INT
```

```
);

-- modifying the above created tables.

--  SQL TO ADD A PRIMARY KEY TO A TABLE


{
      Syntax: ALTER TABLE <table_name>
                              ADD PRIMARY KEY (attr_name);

                              ALTER TABLE <table_name>
                              DROP PRIMARY KEY;

                  }

SHOW TABLES;

DESCRIBE addresses;

ALTER TABLE addresses
ADD PRIMARY KEY (id);

ALTER TABLE addresses
DROP PRIMARY KEY;

ALTER TABLE people
ADD PRIMARY KEY(id);

DESCRIBE people;


================================================================================
======


CREATE DATABASE test;
SHOW DATABASES;
USE test;

DESCRIBE people;
-- Syntax for adding FOREIGN KEY into a Table

-- ALTER TABLE <table_name>
-- ADD CONSTRAINT <constraint_name>
-- FOREIGN KEY(column_name) REFERENCES <table_name>(<column_name>)


-- NOTE; the referencing column of referenced table must be a Primary key,
If not it will raise an error.

-- making 'id' in addresses table as PK
ALTER TABLE addresses
```

```
ADD PRIMARY KEY(id);

DESCRIBE addresses;

ALTER TABLE people
ADD CONSTRAINT FK_peopleAddress
FOREIGN KEY (address_id) REFERENCES addresses(id);


-- how to delete the Foreign key from a table..

-- ALTER TABLE <table_name>
-- DROP FOREIGN KEY <constraint_name>;

ALTER TABLE people
DROP FOREIGN KEY FK_peopleAddress;

================================================================================
=====

UNIQUE constraint


CREATE DATABASE test;
SHOW DATABASES;
USE test;

SHOW TABLES;
-- Syntax for adding a UNIQUE CONSTRAINT

DESCRIBE pets;

ALTER TABLE pets
ADD CONSTRAINT u_species UNIQUE(species);

ALTER TABLE pets
DROP INDEX u_species;

================================================================================
======


CREATE DATABASE test;
SHOW DATABASES;
USE test;

SHOW TABLES;
-- Syntax for changing column name and type in a table
-- ALTER TABLE <tablename> CHANGE old_column_name new_column_name
<data_type>;

SELECT * FROM pets;
```

```
UPDATE <tablename> SET name = <newcolumnname>' WHERE name =
'oldcolumnname';

ALTER TABLE pets CHANGE `species` `animal_type` VARCHAR(20);
ALTER TABLE pets CHANGE `ANIMAL_TYPE` `species` VARCHAR(20);

alter table <tablename> rename  <oldcolumnname> to <newcolumnname>;

================================================================================
======


// HOW TO CHANGE THE COLUMN DATA TYPE IN SQL

SYNTAX:

--    ALTER TABLE <tablename> MODIFY <columnname> <datatype>

eg

ALTER TABLE addresses MODIFY city VARCHAR(30);


-- NOTE: datatype can be converted to anyother datatype unless if the
table is empty. If we already have some data then it is not Possible (like
changing from INT to VARCHAR)


================================================================================
======
```

Data Manipulation language

1) ADDING DATA INTO THE TABLE

SYNTAX

```
INSERT INTO <table_name> (col1, col2, ...) VALUES (val1, val2, ...);
```

2) UPDATING A VALUE IN THE TABLE

SYNTAX

```
UPDATE <table_name>
SET <col_name> = new_value
WHERE <col_name (mostly primary key)> =  old_value;
```

3) DELETING DATA FROM FROM THE TABLE

SYNTAX

```
// deleting one row

DELETE FROM <table_name>
WHERE name = "";

// we can also delete multiple rows from the table as well.

if WHERE CLAUSE NOT THERE IT WILL DELETE ALL THE ROWS IN THE TABLE.
```

4) SELECTING THE DATA FROM THE TABLE

5) WHERE CLAUSE // used when selecting only certain rows from the table

eg:

```
SELECT * FROM products
WHERE price = 3.00;

SELECT * FROM products
WHERE price = 3.00
AND
coffee_origin = 'Brazil';

SELECT * FROM products
WHERE price = 3.00
OR
coffee_origin = 'Brazil';
```

6) INEQUALITY SYMBOLS

```
>, >=, <, <=     // ALL THESE WILL BE USED IN WHERE CLAUSE
```

7) NULL VALUES

8) IN and NOT IN

eg.

```
SELECT <col_name1, ..> FROM <table_name>
WHERE <col_name> IN (val1, ...);
```

9) BETWEEN

eg.

```
// 1) can be used for date column and also a number column
it will include the range ie, [] open brackets

   SELECT * FROM orders
   WHERE order_time BETWEEN 2017 AND 2018;

   // 2) can also be used in strings
```

eg.

```
SELECT * FROM customers
WHERE last_name BETWEEN 'A' AND 'L';
```

10) lIKE use for pattern match

eg.

```
SELECT * FROM customers
WHERE last_name LIKE "w%";

"w%" start with W and can contain any no of chars after w
"%W%" can contain any no of chars before and after w
"_o_" only one char before and after o.
```

This LIKE can be used for number columns also ..,

11) ORDER BY

eg.

```
SELECT * FROM products
ORDER BY PRICE ASC;

DESC
.....................................
// can also be used for strings

SELECT * FROM customers
ORDER BY last_name DESC;
```

```
      NOTE: first null values comes on
      ....................................

      SELECT * FROM orders
      WHERE customer_id = 1
      ORDER BY order_time ASC;



12)  DISTINCT

eg.
      SELECT DISTINCT <col1, ...> FROM <table_name>


13)  LIMIT

eg.
       SELECT * FROM <table_name>
       LIMIT <number of rows to be returned>

       if " LIMIT 5 " // starting 5 rows are returned

       if " LIMIT 5 OFFSET 6 " // starting from id 6, 5 rows are returned


       SELECT * FROM <table_name>
       ORDER BY <col_name>
       LIMIT <number of rows to be returned>

14)  COL NAME ALIAS

      SELECT name AS COFFEE, price, coffee_origin AS COUNTRY FROM
products;




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


SELECT DISTINCT last_name FROM customers
ORDER BY last_name ASC;

SELECT * FROM orders
WHERE customer_id = 1
AND order_time BETWEEN '2017-02-01' AND '2017-02-28'
ORDER BY order_time ASC
LIMIT 3;
```

```
SELECT name, price AS retail_price, coffee_origin FROM products;


-- ----------------------------------------------------------------
SELECT name, price FROM products
WHERE coffee_origin IN ('colombia','indonesia')
-- WHERE coffee_origin = 'colombia' OR coffee_origin = 'indonesia'
ORDER BY name ASC;

SELECT * FROM orders
WHERE order_time BETWEEN '2017-02-01' AND '2017-02-28'
AND customer_id IN (2, 4, 6, 8);

SELECT first_name, phone_number, last_name FROM customers
WHERE last_name LIKE '%ar%';

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

-- Assignment
-- // sol 1
SELECT first_name, phone_number
FROM customers
WHERE gender = 'F'
AND
last_name = 'Bluth';

-- /sol2
SELECT name
FROM products
WHERE price > 3.00
OR
coffee_origin = 'India';

//-- sol3
SELECT count(first_name)
FROM customers
WHERE gender = 'M'
AND
phone_number IS NULL;
```

SELECTING FROM MULTIPLE TABLES BY USING JOINS

JOINS

-- joins allow you to retrieve data from multiple tables in a single SELECT statement.

-- to apply join there must be at least one common/related column between the two tables.

Join Types.

15) INNER JOIN : INTERSECTION OF THE TW0 SETS.

RETRIEVES THE VALUES THAT ARE PRESENT IN BOTH THE TABLES.

16) LEFT JOIN : ALL THE DATA FROM THE LEFT TABLE AND MATCHING DATA FROM THE TABLE2.

17) ROGHT JOIN : VICE-VERSA TO THE LEFT JOIN.

18) FULL JOIN/ OUTER JOIN : RETEIEVES ALL THE DATA FROM TABLE1 AND TABLE2. (NOT PRESENT IN MySQL).

code

```
 -- INNER JOIN
SELECT products.name, orders.order_time
FROM orders INNER JOIN products
ON orders.product_id = products.id;

-- equuvalent short hand version

SELECT p.name, p.price, o.order_time
FROM orders o JOIN products p
ON o.product_id = p.id
WHERE p.id = 5
ORDER BY o.order_time;


-- SELF practice

SELECT c.first_name, o.product_id
FROM customers c JOIN orders o
ON c.id = o.customer_id
ORDER BY o.order_time;
```

```
SELECT c.first_name, p.name
FROM orders o JOIN products p
ON o.product_id = p.id
JOIN customers c
ON c.id = o.customer_id
ORDER BY o.order_time;

DESC orders;


================================================================================
======


-- left join

-- changing the data for the demonstration of left join
UPDATE orders
SET customer_id = NULL -- ORIGINAL DATA orders.customer_id = 1
WHERE id = 1;

-- applying left join
-- orders left join customers

SELECT o.id, c.first_name, c.phone_number, o.order_time
FROM orders o LEFT JOIN customers c
ON o.customer_id = c.id
ORDER BY o.order_time
LIMIT 5;

-- customers left join orders
-- NOTE :  OBSERVE THE CHANGES IN THE OUTPUT
SELECT o.id, c.first_name, c.phone_number, o.order_time
FROM customers c LEFT JOIN orders o
ON c.id = o.customer_id
ORDER BY o.order_time
LIMIT 5;


================================================================================
========

Right join

-- right join

-- changing the data for the demonstration of left join
UPDATE orders
SET customer_id = NULL -- ORIGINAL DATA orders.customer_id = 1
WHERE id = 1;

-- applying RIGHT join
-- orders right join customers
```

```
SELECT o.id, c.first_name, c.phone_number, o.order_time
FROM orders o RIGHT JOIN customers c
ON o.customer_id = c.id
ORDER BY o.order_time
LIMIT 5;

-- customers right join orders
-- NOTE :  OBSERVE THE CHANGES IN THE OUTPUT
SELECT o.id, c.first_name, c.phone_number, o.order_time
FROM customers c RIGHT JOIN orders o
ON c.id = o.customer_id
ORDER BY o.order_time
LIMIT 5;

================================================================================
=======

-- retrieving data from 3 separate tables using one SELECT statement

SELECT p.name, p.price, c.first_name, c.last_name, o.order_time
FROM products p JOIN orders o -- customers and products table do not have
common col
ON p.id = o.product_id
JOIN customers c
ON c.id = o.customer_id
ORDER BY o.order_time;



SELECT p.name, p.price, c.first_name, c.last_name, o.order_time
FROM products p JOIN orders o -- customers and products table do not have
common col
ON p.id = o.product_id
JOIN customers c
ON c.id = o.customer_id
WHERE c.last_name = 'martin'
ORDER BY o.order_time;

================================================================================
=====


-- exercise

SELECT o.product_id, o.id, c.phone_number
FROM orders o JOIN customers c
ON o.customer_id = c.id
WHERE o.product_id = 4;

SELECT p.name, o.order_time
FROM orders o JOIN products p
ON o.product_id = p.id
```

```
WHERE p.name = 'filter'
AND o.order_time BETWEEN '2017-01-15' AND '2017-02-14';

SELECT p.name, p.price, c.first_name, c.gender, o.order_time
FROM products p JOIN orders o
ON p.id = o.product_id
JOIN customers c
ON c.id = o.customer_id
WHERE c.gender = 'F'
AND o.order_time BETWEEN '2017-01-01' AND '2017-01-31';
```

DATABASE DESIGN

NORMALIZATION
RELATIONS
CONSTRAINTS

19) DATA NORMALIZATION

PROCESS OF EFFICIENTLY ORGANIiNG THE DATA IN A DB.

BECOZ

TO ELIMINATE REDUNDANT DATA

TO ONLY STORE RELATED DATA IN A TABLE

BENEFITS

REDUCE STORAGE SPACE

REDUCE INSERT, UPDATE, AND DELETION ANOMALIES

IMPROVE QUERY PPRFORMANCE

20) FIRST NORMAL FORM

TABLES ARE IN 1 NF IF:

NO REPEATED ROWS OF DATA

COLS ONLY CONTAIN A SINGLE VALUE

THE TABLE HAS A PK

21) SECOND NORMAL FORM

FIRST IT SHOULD BE IN 1-NF

EVERY COLUMN THAT IS NOT A PK OF THE TABLE IS DEPENDENT ON THE WHOLE OF THE PK. i,e. removing unrelated data from the table

22) 3rd N-f

FIRST IT SHOULD BE IN 2-CF

EVERY COLUMN THAT IS NOT A PRIMARY KEY IS ONLY DEPENDENT ON THE WHOLE OF THE PRIMARY KEY

==========
==========

23) RELATIONSHIPS the links between our tables.

TABLES ARE RELATED THROUGH PK's AND FK's

ONE TO ONE, ONE TO MANY, MANY TO MANY

24) ONE TO ONE REL.

EG. PERSON AND AADHAR ID

25) ONE TO MANY REL.

EG. PRODUCT TO ORDERS

EG. CUSTOMER AND ORDERS

26) MANY TO MANY

==========
==========
27)

NOT NULL : col cant contain any null val.

UNIQUE : every row of a col should be unique.

PK : col that uniquely identifies a each row of data.

Foreign Key : col that references the PK of another table.

CHECK : controls the values that can be inserted into a column.
eg : CHECK(rating BETWEEN 0 AND 100);

DEFAULT : if no val is inserted into a col, i can set a def val.
eg : email DEFAULT 'No DATA';

```
Show databases;
-- CREATE DATABASE Cinema_tickets_booking;
use Cinema_tickets_booking;
SHOW TABLES;

DESCRIBE reserved_seats;
SELECT * FROM bookings;
SELECT * FROM customers;
SELECT * FROM films;
SELECT * FROM reserved_seats;
SELECT * FROM rooms;
SELECT * FROM screenings;
SELECT * FROM seats;


================================================================================
====


-- Aggregate functions

-- perform a calculation on data within a column and returns one result
row.
-- can use GROUP BY clauses to group the results by one (or more) ce
results by oneolumns.
-- can use a HAVING clause in a similar way to a WHERE clause in a SELECT
statement to filter the results set.

-- COUNT
-- NOTE: doesn't count the null values when counting on the specific
column
-- we can add WHERE clause
SELECT COUNT(*) FROM customers;

-- SUM
-- NOTE: can be applied to a specific numeric columns only (and not *),
SELECT * FROM rooms;
SELECT SUM(no_seats) FROM rooms;
-- we can add WHERE clause

-- MIN and MAX

SELECT * FROM films;

SELECT MAX(length_min) FROM films;
SELECT MIN(length_min) FROM films;

-- Average
-- we can add WHERE clause
SELECT * FROM films;

SELECT AVG(length_min) FROM films;
```

```
SELECT AVG(length_min) FROM films
WHERE length_min > 120;

-- EXERCISE
-- my solutions
select count(b.customer_id)
from bookings b JOIN screenings s
on b.screening_id = s.id
where b.customer_id = 10;
-- (OR)
select count(*) from bookings
where customer_id = 10;

select count(*)
from screenings s join films f
on s.film_id = f.id
where f.name = 'Blade Runner 2049';

select count(distinct (c.id))
from customers c join bookings b
on c.id = b.customer_id;

select count(distinct(customer_id)) from bookings;

-- c.last_name will not work as many guys can have same last_name... =D
select count(distinct(c.last_name))
from customers c join bookings b
on c.id = b.customer_id;
```

GROUP BY

Eg: SELECT customer_id, COUNT(id) FROM bookings; // ERROR

in the above query customer_id is not an aggregate function so we must put that column in a Group By clause.

```
SELECT customer_id, COUNT(id) FROM bookings
Group By customer_id;
```

HAVING

Acts like a WHERE clause and a SELECT statemunt, but for a GROUP BY statement.

```
Eg: SELECT customer_id, screening_id, COUNT(id) FROM bookings
GROUP BY customer_id, screening_id
HAVING customer_id = 10;

                            // NOTE: here we cant use WHERE clause as
GROUP BY is done.
```

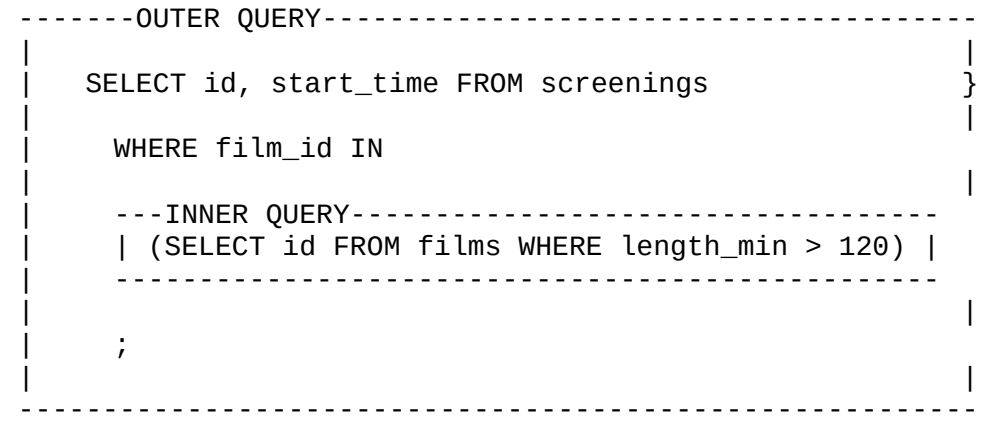
```
 -- ========Exercise=============

select b.customer_id, COUNT(rs.id)
FROM bookings b JOIN reserved_seats rs
ON b.id = rs.booking_id
GROUP BY b.customer_id;

select f.name, f.length_min, COUNT(s.id)
FROM films f JOIN screenings s
ON f.id = s.film_id
GROUP BY f.name, f.length_min
HAVING f.length_min >= 120;
```

SUBQUERIES :

Subqueries are queries nested within other queries.

alsa referred to as NESTED QUERIES

Eg.



-- Subqueries can be used in SELECT, INSERT, UPDATE or DELETE query.

-- The nested query can be in WHERE clause or the FROM.

-- Two types of subquery:

28) Non-Correlated : Inner query runs INDEPENDENTLY of the OUTER QUERY.

here the inner query runs ONLY ONCE, and PRODUCES A SET OF RESULTS
which are then used by the outer query.

Eg. above eg.

29) Correlated : Inner Query runs DEPENDENTLY of the OUTER QUERY.

-- inner query can't independently run of the outer query.

inner q. uses data from Outer qu.

-- inner query runs for EVERY ROW in the OUTER QUERY.

Eg.
```
SELECT screening_id, customer_id,
(SELECT COUNT(seat_id)
FROM reserved_seats WHERE booking_id = b.id)
FROM bookings b;
```

-- non correlated queries coming from 'WHERE'

-- NON CORRELATED SUBQUERIES

```
select id, start_time from screenings
where film_id in
(select id from films where length_min > 120);

-- refer to Lv- 92

select first_name, last_name, email from customers
where id in (select customer_id from bookings where screening_id = 1);

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

-- non correlated queries coming from 'FROM'

-- NON CORRELATED SUBQUERIES (from FROM)
select max(no_seats) from
(select booking_id, COUNT(seat_id) as no_seats from reserved_seats
group by booking_id) b;
-- every derived table must have it own alias.
-- the inner query is creatind a DERIVED TABLE and we call it as b
select booking_id, COUNT(seat_id) as no_seats from reserved_seats
group by booking_id;

select avg(no_seats), max(no_seats) from
(select booking_id, COUNT(seat_id) as no_seats from reserved_seats
group by booking_id) b;

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

-- exercise----------

select avg(length_min) as avglength from films;

select name, length_min from films
where length_min > (select avg(length_min) from films);

 select max(sid), min(sid) from
 (select f.name, count(s.id) as sid
 from films f join screenings s
 on f.id = s.film_id
 group by f.name) it;

-- alternative
select max(id), min(id) from
(select film_id, count(id) as id from screenings
group by film_id) a;

select f.name,
(select count(id) from screenings where film_id = f.id)
from films f;
```

MYSQL FUNCTIONS

-- are stored programs which can be passed parameters and return a value.
-- eg, aggregate functions
max(), min() etc

30) String functions

-- funcions that take in 'string' paremeters.. such as char of varchar datatypes

1.0 concatenation

syntax: SELECT CONCAT(col1, col2) AS new_col FROM table;

Eg.

```
select CONCAT(first_name,' ' ,last_name) as full_name from customers;

select CONCAT(first_name,' ' ,last_name, ' ', email) as full_name_and_email from customers;

select CONCAT('This is ', first_name,' ' ,last_name, ' and email ', email)    as data_ from customers;
```

31) Substring

```
syntax: SELECT SUBSTRING("Example", 3, 3) AS Sub;
-- length is optional.

select substring("Example", 3, 3) as Sub; -- amp

select substring("Example", 3) as Sub; -- ample

select substring("Example", -4) as Sub; -- mple

select substring("Example", -4, 2) as Sub; -- mp

select name from films;

select substring(name, 1, 3) as short_name from films;

select substring(name, 5, 4) as short_name from films;

-- start from ending (2nd last ie -2) of length 2
select substring(name, -2, 2) as short_name from films;

select substring(name, -5, 4) as short_name from films;
```

32) UPPER AND LOWER CASE

syntax:

```
SELECT UPPER(col) AS new_col_name FROM table1;
SELECT LOWER(col) AS new_col_name FROM table1;
```

Eg.

```
select upper(name) as films from films;
select lower(name) as rooms from rooms;
```

```
-- exercise --------------------------------

select concat(name,' : ' ,length_min, 'mins') as film_len from films;

select substring(email, 5) as email_domain from customers;

select lower(first_name) as first_name, upper(last_name) as last_name
from customers
where last_name = 'Smith';

select substring(name, -3) from films;

select concat(substring(first_name, 1, 3), last_name) as res from
customers;
```

```
DATE

SELECT DATE("2018-06-05");

select date('2018-06-05: 07:45:32');

select date(start_time) from screenings;

select * from screenings
where date(start_time) = '2017-10-02';

select * from screenings
where date(start_time) between '2017-10-02' and '2017-10-05';

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

MONTH

select month('2018-06-05'); -- 6

select month('2018-06-05 07:45:32'); -- 6

select month(start_time) from screenings; -- 10

select * from screenings where month(start_time) = 10;

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

YEAR

select year('2018-06-05'); -- 2018

select year('2018-06-05 07:45:32'); -- 3018

select year(start_time) from screenings;

select * from screenings where month(start_time) = 10 and year(start_time)
= 2017;

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

-- exercise ----

select film_id, start_time
from screenings
where date(start_time) = '2017-10-20';

select * from screenings
where date(start_time) between '2017-10-6' and '2017-10-13';
```

```
select * from screenings
where month(start_time) = '10' and year(start_time) = '2017';
```

```
Show databases;
-- CREATE DATABASE Cinema_tickets_booking;
use Cinema_tickets_booking;
SHOW TABLES;

DESCRIBE reserved_seats;
SELECT * FROM bookings;
SELECT * FROM customers;
SELECT * FROM films;
SELECT * FROM reserved_seats;
SELECT * FROM rooms;
SELECT * FROM screenings;
SELECT * FROM seats;

=======================ALL CHALLENGE
QUESTIONS==================================

-- 1)
select name, length_min  from films
where length_min >= 120;

-- 2)
select f.id, f.name from films f
join
(select film_id, count(film_id) from screenings
group by film_id
order by count(film_id) desc
limit 1) b
on f.id = b.film_id;
-- or

select f.name, count(s.film_id) as showings from
screenings s join films f
on f.id = s.film_id
group by film_id
order by showings desc
limit 1;

-- 3

select f.name, count(b.screening_id) as no_screenings from
bookings b join screenings s
on b.screening_id = s.id
join films f
on s.film_id = f.id
where f.name = 'Jigsaw';

select count(*) from bookings as no_bookings where screening_id
in (select id from screenings
where film_id = 5);

-- 4
```

```
select concat(c.first_name, ' ', c.last_name) as full_name, b.freq from
customers c join
(select customer_id, count(customer_id) as freq from bookings
 group by customer_id
 order by freq desc
 limit 5) b
 where c.id = b.customer_id;

 -- or

select c.first_name, c.last_name, count(b.id) as no_bookings
from bookings b join customers c
on c.id = b.customer_id
group by c.first_name, c.last_name
order by no_bookings desc limit 5;

-- 5
select f.name, f.id, count(s.room_id) as no_screenings
from films f join screenings s
on f.id = s.film_id
where s.room_id = (select id from rooms where name = 'Chaplin')
and start_time between date('2017-10-01') and date('2017-10-31')
group by f.name, f.id
order by no_screenings desc;

select film_id, count(room_id) as freq from screenings where room_id =
(select id from rooms where name = 'Chaplin')
group by film_id
order by freq desc;

-- 6

select distinct customer_id from bookings;

select count(distinct customer_id) from bookings;

select count(*) from
(select c.first_name, c.last_name, count(b.customer_id)
from customers c join bookings b
on c.id = b.customer_id
group by c.first_name, c.last_name) b;
```