MySQL

database // stores the tables

create database database_name; //to create database

use database_name; // to use create database or already existed database

drop database database_name; // to drop database

alter database database_name read only = 1; // to read only mode we cant do modification to database
to remove read only mode use 0 instead of 1 (read only = 0).

----------table----------

create table table_name (column_name datatype, column_name datatype) ;

```
create table employees (
employee_id int,
    first_name varchar(50),
    last_name varchar(50),
    hourly_pay decimal(5,2),
    hire_date date
);
```

datatype - int , varchar(size), decimal(num,precision), date, datetime

select ----------
select * from table_name;   // select all the columns(*) from the table

select * from employees;

rename----------

rename table existing_name to new_name;

rename table employees to workers;

drop----------
drop table table_name;

drop table employees;

alter----------
syntax:
alter table table_name add column_name datatype; //to add new column in a table
ex:
alter table employees
add phone_number varchar(15);

syntax:
alter table table_name rename column column_name to new_name; // to rename column name to new name
ex:
alter table employees
rename column phone_number to email;

syntax:
alter table table_name modify column column_name datatype(100); // to add a new datatype or modify to new datatype

ex:
alter table employees
modify column email varchar(100);

syntax:
alter table table_name modify column_name datatype after/first column_name; // to alter the position of the column, after new to come after the given column and first use to present as first column in table

ex:
alter table employees
modify email varchar(100)
after last_name;

alter table fM
modify salary int first; //first

syntax:
alter table table_name drop column column_name; // to drop a column from a table.

ex:
alter table employees
drop column email;

---insert---

---

syntax:
insert into table_name(column_names) values (values);

ex:
insert into fM(name,age,phone_number) values ("harish",22,"9848479");

for mutiple insertion
syntax:
insert into table_name(column_names) values (column_values), (column_values), (column_values), (column_values);

ex:
insert into fM(name,age,phone_number) values ("pachu",17,"3454780"), ("dhanu",48,"9544084");

where clause // to select specify row by giving condition to match the row. where clause can use comparison operator like =, >, <. >=, <= , != (not equal) or <>

syntax:
select name,age from fM where age=22;

to select a null value row use null keyword
select * from employees where hire_date is null;

opposite ie is not null
select * from employees where hire_date is not null;

----------update----------

update table_name
set column_name = value
where condition;

//if you dont specify where condition then the value is applied to whole column.

ex:
update employees
set hourly_pay = 10.25
where employee_id = 4;

----------delete----------

delete from table_name
where condition;

//if you dont specify where condition then the whole rows will be deleted.

delete from employees
where employee_id = 6;

----------autocommit----------

by default mysql on autocommit (autocommit means all the transaction will be save, if you want to go back to previous transaction then you have to set it to off by
syntax : set autocommit = off;
----------commit----------
commit is used to create savepoint.
----------rollback----------
rollback;
// used to rollback or go back to previous transaction that is saved

----------currentDate()----------
current_date() //returns current date
current_time() //returns current time
now() //returns both date and time

----------unique----------
```
create table products()
product_id int,
product_name varchar(25) unique,
price decimal(4,2)
);
```
unique -- no duplications of values

to add unique contraint using alter command

syntax:
```
alter table products
add constraint
unique(product_name);
```

----------not null----------
not null -- value cant be null

```
create table products()
```

product_id int,
product_name varchar(25),
price decimal(4,2) not null
);

to add not null contraint using alter command

syntax:
alter table products
modify price decimal(4,2) not null;

----------check----------
-
check -- value should be valid to all rows in table
create table products()
product_id int,
product_name varchar(25),
price decimal(4,2),
contraint contraint_name check(price > 100)
);

to add check contraint using alter command

sytax:
alter table products
add contraint contraint_name check(price > 100);

to drop check contraint

alter table products
drop check contraint_name;

----------default----------
---
if we provide a default contraint that value is inerted in row if we dont specify value while inserting the row value.

```
create table products()
product_id int,
product_name varchar(25),
price decimal(4,2) default 0
);

alter table products
alter price set default 0;
```

----------primary----------

primary = unique + not null

```
create table transactions(
transaction_id int primary key,
amount decimal(5,2)
);

alter table transactions
add contraint primary key(transaction_id);
```

----------autoincrement----------

autoincrement starts from 1 and it works only on keys.

```
create table transactions(
transaction_id int primary key auto_increment,
amount decimal(5,2)
);

alter table transactions
auto_increment = 100;
```

----------foreign key----------

using foreign key we can make relation / link with two table.
primary => parent
foreign => child

```
create table transactions(
transaction_id int primary key auto_increment,
amount decimal(5,2),
customer_id int,
foreign key(customer_id) references customers(customer_id)
);
```

to drop foreign key

```
alter table transactions
drop foreign key foreign_key_name;
```

to add constraint using alter command

```
alter table transactions
add constraint fk_customer_id
foreign key(customer_id) references customers(customer_id);
```

----------joins----------

inner joins -- common in two tables

```
select * from table1 inner join table2
on table1.column_name = table2.column_name;
```

left joins -- all left table rows and common in two tables

```
select * from table1 left join table2
on table1.column_name = table2.column_name;
```

right joins -- all right table rows and common in two tables

```
select * from table1 right join table2
on table1.column_name = table2.column_name;

----------------------------------------------------functions----------------------------------
-----------------
select count(amount) as count from transactions;
max(column_name)
min(column_name)
avg(column_name
sum(column_name)
concat(first_name , " ", last_name) as full name

-----------------------------------------------------and, or , not---------------------------------
----------------
and, or, not -- (logical operators)
and--
select * from employees
where job="cook" and hire_date <"2023-01-5";

or--
select * from employees
where job="cook" or job="cashier";

not --
select * from employees
where not job = "manager";

between -- used with only one  column
select * from employees
where hire_date between "2023-01-5" and "2023-05-5";

in -- to check the value is present in the table
select * from employees
where job in ("cook","cashier","janitor");
```

----------wild card characters----------

% , _
if you want to work with wild character than use -- like keyword
% match any numbers of character
_ match only one numbers of character

```
select * from employees
where job like "_a%";
```

----------order by----------

```
select * from employees
order by last_name desc;
```

----------limit----------

limit -- used to limit the number of records

```
select * from employees
limit 2;
```

```
select * from table_name
limit offset_value, limit_value
```

----------unions----------

union combines the results of two or more select statements.
union works with two table must have a same numbers of columns.
union dont allow duplicates
union all -- allow duplicates

```
select * from income;
union
select * from expenses;
```

----------self-joins----------

self join
-- join another copy of a table to itself
-- used to compare rows of the same table
-- helps to display a heirarchy of data

select * from customers as A inner joins customers as B
on A.column_name = B.column_name; // here alias A is refering to original table and B is refering to duplicate/ copy of a table

----------views----------

views
-- a virtual table based on the result-set of an sql statement
-- the fields in a view are fields from one or more real tables in the database
-- they're not real tables, but can be interacted with as if they were

select * from employees;

create view workingOnProject as
select first_name , last_name
from employees;

to drop view:
drop view view_name;

----------indexes----------

-- Indexx (BTree data structure)
-- indexes are used to find values within a specific column more quickly
-- mysql normally searches sequentially through a column
-- the longer the column, the more expensive the operation is
-- update takes more time, select takes less time

```
show indexes from table_name;

create index index_name
on table_name(column_name);

multi-column index:

create index index_name
on table_name(column_name, column_name);

to drop index:

alter table table_name
drop index index_name;

----------subqueires----------
--subquery
-- a query within another query
-- query(subquery)

select first_name, last_name, hourly_pay, (select avg(hourly_pay) from employees) as
avg_pay
from employees;

select first_name, last_name
from employees
where hourly_pay > (select avg(hourly_pay) from employees);

----------group by----------
--group by = aggregate all rows by a specific column
often used with aggregate functions
ex. sum() , max() , min() , count(), avg()

select sum(amount) from transactions group by order_date;
```

using where clause flowed by group by will not work
so we use having clause

----------roll up----------

rollup -- extension of the group by clause
produces another row and shows the grand total (super-aggregate value)

```
select sum(amount) from transactions group by order_date with rollup;
```

----------on delete ----------

on delete set null = when a Foreign key is deleted, replace Foreign key with null
on delete cascade = when a Foreign key is deleted, delete row

```
create table transactions(
transaction_id int primary key auto_increment,
amount decimal(5,2),
customer_id int,
foreign key(customer_id) references customers(customer_id)
on delete set null
);
```

```
create table transactions(
transaction_id int primary key auto_increment,
amount decimal(5,2),
customer_id int,
foreign key(customer_id) references customers(customer_id)
on delete cascade
);
```

----------stored procedure----------

stored procedure = is prepared sql code that you can save
great if there's a query that you write often

```
DELIMITER $$
create procedure procedure_name()
begin
select * from customers;
end $$
DELIMITER ;
```

to call stored procedure:

```
call procedure_name();
```

to drop stored procedure:

```
drop procedure procedure_name;
```

----------trigger----------

Trigger = when an event happens, do something
ex. insert , delete , update
checks data, handles errors, auditing tables

```
create trigger trigger_name
before update on table_name
for each row
set column_name = condition / expression
```

```
create trigger trigger_name
before insert on table_name
for each row
set new.salary = (new.hourly_pay * 2080);
```

```
create trigger trigger_name
after delete on table_name
for each row
update table_name
set column_name = column_name - OLD.value
```

```
where column_name = value;

show triggers;

drop trigger trigger_name;
```