

//td[text()='Name']/following-sibling::td  
 custom xpath for Dynamic IDs: <input id="user-12345" />  
 //input[starts-with(@id, 'user-')]

## MySQL

- -- For creating the database
  - create database firstdb;
  - Execute and go to Schemas and then refresh to see the new database
- -- create database firstdb;
  - To use the database, (use the below code) or (Right click on the db and make 'set as Default Schema')
  - use firstdb;
  - It becomes bold in color
- -- create database firstdb;
  - use firstdb;
  - To remove the database
  - drop database firstdb;
- -- create database firstdb;
  - use firstdb;
  - After creating and using database Setting a database to readonly where can't do any operation to it
  - still access the data from it
  - alter database firstdb read only = 1;
- -- create database firstdb;
  - use firstdb;
  - alter database firstdb read only = 1;
  - Like try removing the database using drop after read only mode is enabled, message says it is in read only mode
  - drop database firstdb;
- -- create database firstdb;
  - use firstdb;
  - alter database firstdb read only = 1;
  - To disable the readonly mode
  - alter database firstdb read only = 0;

-- Once it is not read-only mode then you can perform operation in it.

-- drop database first db;

> To create a table:

```
create table employees (  
    employee-id int,  
    first-name varchar(50),  
    last-name varchar(50),  
    hourly-pay decimal(5, 3),  
    hire-date date  
);
```

Then execute and refresh.

> To select a table:

```
select * from employees;
```

> To rename a table:

```
rename table employee to workers;
```

Then refresh the schemas.

> To alter the table:

- clear the query,

- write the below code:

```
alter table employee
```

```
add phone-number varchar(12);
```

- execute and refresh the schema

- or then write the code → ~~select~~ select \* from employee;

> To rename the column:

- clear the query,

- write the below code:

```
Alter table employees
```

```
rename column phone-number to email;
```

- execute then refresh or use → select \* from employees;

> To change the varchar of any column:

- clear query.

- write below code:



alter table employee

modify column email varchar(100);

- Execute and refresh schema then click on columns or [select \* from employees]

➤ To move the column:

- clear query

- write the code:

alter table employees

modify email varchar(100)

after last name;

select \* from employee;

➤ To move to the first:

code:

alter table employee

modify email varchar(100)

first;

select \* from employees;

➤ To remove a column:

code:

alter table employees

drop column email;

select \* from employees;

Insert rows

into a  
table

insert into emp table

values (1, "firstname value", "last name value", "email@value.com", 123.34,  
"2025-05-12");

select \* from emp table;

To add

multiple

rows

➤ clear query

➤ Insert into employees

values (2, "first 1", "last 1", 15.00, "2025-05-11"),

(3, "first 2", "last 2", 15.75, "2025-06-10"),

(4, "first 3", "last 3", 15.25, "2025-06-10");

select \* from employees;

ments with missing values:

insert into employees (employee-id, first-name, last-name):  
values (6, "first 4", "last 4");

select \* from employees;

to select every row and column: select \* from employees

to select first and last name: select first-name, last-name  
from employees;

to select employee with id=1: select \*  
from employees

where employee-id=1;

or you can use != for not equal

to select by hourly-pay: select \*  
from employees

where hourly-pay >=15;

to select by Date: select \*  
from employees

3:34 where hire-date <="2023-01-03";

to select  
by null  
value  
update  
and delete  
data from

select \*

from employees

where hire-date is null; -- or provide 'is not null' for values provided for hire-date

to update a value

update employees

set hourly-pay = 10.25

where employee-id = 6;

select \* from employees;

for not getting any where error

Edit -> Preferences... -> SQL Editor -> uncheck safe

updates -> OK -> close and open again mysql workbench.

to update  
multiple  
values

update employees

set hourly-pay = 10.50,

hire-date = "2023-01-07" or to make null hire-date = null

where employee-id = 6;

select \* from employees;

to provide same value for entire column is

update employees

set hourly-pay = 10.25;

select \* from employees;



To delete all rows: ~~D~~ delete from employees

select \* from employees;

To delete a particular row: delete from employees

where employee-id=6;

select \* from employees;

Autocommit, commit, Rollback:

If all rows are deleted accidentally: By default autocommit set to on,

To set autocommit to off: set AUTOCOMMIT = off;

To commit: commit;

To show all rows: select \* from employees;

now try to delete: delete all rows: delete from employees;

select \* from employees;

To undo the changes: rollback;

Then execute. Then clear the query, then write the below code:

select \* from employees;

To save the changes instead of "to undo the changes":

commit;

Then execute, → clear query → write below code

select \* from employees;

current date() and current time():

→ write below code

create table test (

my-date date,

my-time time,

my-datetime datetime

);

To provide → Insert into test

values

values (current-date(), current-time(), now());

select \* from test;

→ use & replace current-date() to current-date()+1 to increment and -1 to decrement, or yesterday; or replace current-time and now() to NULL

## UNIQUE constraint :

```
> create table products(  
    product-id int,  
    product-name varchar(25) unique,  
    price decimal(4,2)  
);
```

> To add a unique constraint if you have not added to product-name column.  
Alter table products  
add constraint  
unique(product-name);

## NOT NULL constraint :

```
> create table products(  
    product-id int,  
    product-name varchar(25),  
    price decimal(4,2) not null  
);
```

> To add not null constraint to the table already created.  
Alter table products  
modify price decimal(4,2) not null;

## check constraint: is

```
> create table employees(  
    employee-id int,  
    first-name varchar(50),  
    last-name varchar(50),  
    hourly-pay decimal(5,2),  
    hire-date date,  
    check (hourly-pay >= 10.00)  
);
```

or replace check(hourly-pay >= 10.00) to constraint chk-hourly-pay check  
(hourly-pay >= 10.00)

> For adding check constraints to the table already exist:



alter table employees  
add constraint chk\_hourly\_pay check(hourly\_pay >= 10.00);

→ To delete check

alter table employees  
drop check chk\_hourly\_pay;

### Default constraint

→ create table products(  
product\_id int,  
product\_name varchar(25),  
price decimal(4,2) default 0  
);

→ To add a default constraint to the existed table:

alter table products  
alter price set default 0;

→ insert into products (product\_id, product\_name)

values (104, "straw"),  
(105, "napkin"),  
(106, "fork");

→ create table transactions(  
transaction\_id int,  
amount decimal(5,2),  
transaction\_date datetime default now()  
);

→ insert into transactions (transaction\_id, amount)  
values (1, 4.99);

### Primary Keys constraints

→ create table transactions(  
transaction\_id int primary key,  
amount decimal(5,2)  
);

→ To add primary key constraints to the table already exists:

alter table transactions

add constraint

primary key(transaction-id);

Auto-increment

➤ create table transactions (

transaction-id int primary key auto-increment,  
amount decimal (5,2)

);

➤ insert into transactions (amount)

values (4.99);

➤ By default auto-increment starts at 1. But if we want to start from value 1000.

alter table transactions

auto-increment = 1000;

➤ delete from transactions;

select \* from transactions;

➤ insert into transactions (amount)

values (4.99);

Foreign Key constraints: MySQL supports foreign keys. foreign key as a primary key of one table than can be found in different table. Using a foreign key we can create a link between two tables.

➤ create table customers (

customer-id int primary key auto-increment,

first-name varchar(50),

last-name varchar(50)

);

➤ insert into customers (first-name, last-name)

values ("Fred", "Fish"),

("Larry", "Lobster"),

("bubble", "Bass");

select \* from customers;

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)

);  
select \* from transactions;

→ To drop the foreign key  
alter table transactions from table > foreign keys  
drop foreign key transactions\_ibfk\_1;

To name a foreign key → alter table transactions to apply a foreign key for a table that already exists.  
alter table transactions

add constraint fk-customer-id  
Foreign Key(customer-id) references customers(customer-id);

→ alter table transactions

auto-increment = 1000;

→ insert into transactions(amount, customer-id)  
values (4.99, 3), (2.89, 2), (3.38, 3), (4.99, 1);  
select \* from transactions;

→ Delete from customers  
where customer-id = 3;

Joins (Inner, you can add customer-id to null means not all customer have customer-id left, right) → select \* on left

from transactions inner join customers  
on transactions.customer-id = customers.customer-id;

To select particular columns → select transaction-id, amount, first-name, last-name  
from transactions inner join customers

on transactions.customer-id = customers.customer-id;

left join → select \*

from transactions left join customers

on transactions.customer-id = customers.customer-id;

Right join → select \*

from transactions right join customers ...



Functions in MySQL: To count how many transaction took place on a certain date.

➤ select count(amount)

from transactions;

➤ To name column: select count(amount) as count

from transactions;

or "today's transactions"

➤ To find maximum amount: select max(amount) as maximum  
from transactions;

➤ min() for minimum just as above.

➤ avg() to find average.

➤ sum() to find sum.

➤ For concatenating first and last name that makes full name:

select concat (first-name, last-name) as full-name

from employees; or (first-name, " ", last-name)

logical operators ➤ ... where job = "cook" or job = "cashier";

➤ ... where not job = "manager" and not job = "asst. manager";

➤ ... where hire-date between "2023-01-04" and "2023-01-07";

➤ where job in ("cook", "cashier", "janitor");

wild cards ➤ first name starts with s then

... where first-name like "s%";

➤ last name end with x then

... where last-name like "%x";

➤ underscore wildcard "\_" represents one random character:

... where job like "\_ook"; => returns "cook"

for january: ... where hire-date like "---- -01- --";

➤ no job starts with random character second character is 'a' then any eg: manager

... where select ... where job like "\_a%";

order by ➤ In descending order:

select \* from employees

order by last-name desc;

➤ for ascending order: (by default)

select \* from employees

order by last-name asc;



➤ If the amount of two product is same but want to order by customer-id.  
~~select~~ select... order by amount, customer-id;

select... order by amount asc, customer-id desc;

Limit clause: used to limit the number of records. useful if you are working with a lot of data.  
 can be used to display a large data on pages (pagination).

➤ code to show number of rows in a table or to limit number of rows:

select \* from customers

limit 1;

➤ limit 1; to show first one row; limit 2 to show first 2 rows;

➤ select \* from customers

order by last-name limit 1;

customer id	first name	last name		customer id	first name	last name
1	fred	Fish		3	Bubble	Bass
2	Larry	Lobster	⇒			
3	Bubble	Bass				
4	Penny	puff				

➤ select \* from customers

order by last-name desc limit 1;

customer-id	first name	last name
4	Penny	Puff

➤ 10 customer per page among 100 customer.

➤ select \* from customers

order by limit limit 1,1; first number is offset → 2 Larry Lobster  
 limit the one record after the first

➤ select \* from customers

limit 2; offset of 2

customer-id	first name	last name
3	Bubble	Bass

➤ select \* from customers

limit 3,1;

customer-id	first name	last name
4	penny	puff

➤ among 100 customers and wants to display 10 customers per page then.

select \* from customers

limit 10

union

POCO

SHOT ON POCO F1

➤ already two tables are existed so:

select \* from income; select \* from expenses;

➤ to have a union of both:



select \* from income  
union

select \* from expenses;

> Both the table must have same column

select first-name, last-name from employees  
union

select first-name, last-name from customers;

> union all < Includes any duplicates, union does not allow duplicates like first name and last if same element present in both table

select first-name, last-name from employees  
union all

select first-name, last-name from customers;

self joins > Join another copy of a table to itself used to compare rows of the same table, helps to display a hierarchy of data.

> Alter table customers

add referral-id int;

select \* from customers;

> update customers

set referral-id=1

where customer-id=2;

select \* from customers;

> select \*

from customers as a

inner join customers as b

on a.referral-id = b.customer-id;

customer-id	first-name	last-name	referral-id	customer-id	first-name	last-name	referral-id
2	Larry	Lofster	1	1	Fred	Fish	NULL
3	Bubble	Bass	2	2	Larry	Lofster	1
4	Perry	puff	2	2	Larry	Lofster	1

> select a.customer-id, a.first-name, a.last-name, b.first-name, b.last-name  
from customers as a

Inner join customers as b

on a.referral-id = b.customer-id;



> select a.customer-id, a.first-name, a.last-name,  
concat(b.first-name, " ", b.last-name) as "referred-by"  
from customers as a  
inner join customers as b - (a)

on a.referral-id = b.customer-id;

customer-id	first-name	last-name	referred-by
2	Larry	Lobster	Fred Fish
3	Bubble	Bass	Larry Lobster
4	Pappy	Puff	Larry Lobster

line (a) can also be written as: left join customers as b

> alter table employees

add supervisor-id int;

> others reports to asst. manager and asst. manager to manager use the  
following command to set values

update employees

set supervisor-id = 1

where employee-id = 5; -- then final table is as follows

employee-id	firstname	lastname	hourly-pay	job	hire date	supervisor-id
1	Eugene	Krabs	25.50	manager	2023-01-02	null
2	Squidward	Tentacles	15.00	cashier	2023-01-03	5
3	Shengetof	Squarepants	12.50	cook	2023-01-04	5
4	Patrick	Star	12.50	cook	2023-01-05	5
5	Sandy	Cheeks	17.25	asst. manager	2023-01-06	1
6	Sheldon	Plankton	10.00	janitor	2023-01-07	5

> select \*

from employees as a

inner join employees as b

on a.supervisor-id = b.employee-id;

> delete a.first-name, a.last-name,

concat(b.first-name, " ", b.last-name) as "reports to",  
from employees as a

inner join employees as b - (b)



on a.supervisor\_id = b.employee\_id;  
then tables will be:

first_name	last_name	Reports-to
Squidward	Tentacles	Sandy cheeks
Spongebob	Squarepants	"
Patrick	Star	"
Sandy	cheeks	Eilgene Krabs
Sheldon	Plankton	Sandy cheeks

Line (6) can be written as: left join employees as b

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.

→ create view employee-attendance as  
select first\_name, last\_name  
from employees;

-- In Schema, refresh, under Views we have employee-attendance

→ select \* from employee-attendance;

→ select \* from employee-attendance  
order by last\_name asc;

→ To drop a view: drop view employee-attendance;

→ Adding a column:

alter table customers

add column email varchar(50);

→ update customers

set email = "Lobster@gmail.com"

where customer\_id = 2;

→ create view customers-emails as select email  
from customers;

→ select \*

from customer-emails;

→ If you update values in table it also updates the views.



## Indexes

INDEX (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 customers;

...	key-name	...	column-name	...
...	PRIMARY	...	customer-id	...

➤ to create index:

create index last-name-idx  
on customers(last-name);

➤ show indexes from customers;

...	key-name	column-name	...
	last-name	last-name	...
	-idx		

➤ to create multi-column index

create index last-name-first-name-idx  
on customers(last-name, first-name);

➤ execute → clear then

show indexes from customers;

➤ to drop index:

alter table customers

drop index last-name-idx;

show indexes from customers;

➤ select \* from customers

where last-name = "Puff" and first-name = "Penny";  
this is first in the sequence.

seq-in-index	column-name
1	last-name
2	first-name

➤ for the first name we cannot utilize the index

select \* from customers

where first-name = "Penny";

Subquery ➤ for comparing hourly-pay of each with its average.

➤ select avg(hourly-pay) from employees;

➤ select first-name, last-name, hourly-pay,



(select avg(hourly-pay) from employees) as avg-pay  
from employees;

➤ select first-name, last-name, hourly-pay  
from employees

where hourly-pay > (select avg(hourly-pay) from employees);

➤ select ~~customer-id~~ distinct customer-id  
from transactions

← to remove duplicates

where customer-id is not null;

➤ Above code behaves as subquery

select first-name, last-name  
from customers

where customer-id in -①

(select distinct customer-id  
from transactions

where customer-id is not null);

line ① can also be written as:

where customer-id not in

or

where customer-id in (1,2,3);

Group By: aggregate all rows by a specific column often used with aggregate functions  
eg: sum(), max(), min(), Avg(), count()

➤ Sum of amount per day:

select sum(amount), order-date - ①

from transactions

group by order-date;

➤ To find maximum amount per of each date use following command in -①:

select max(amount), order-date

➤ use count() to check number of transactions.

➤ select sum(amount), customer-id

from transactions

group by customer-id;

➤ when group by is used instead of using where, use Having.



## Rollup

- select count(amount), customer-id  
from transactions  
group by customer-id  
having count(amount) > 1; and customer-id is not null;  
Extension of the Group by clause, produces another row and shows the grand total (super-aggregate value).
- select sum(amount), order-date - ①  
from transactions  
group by order-date with rollup;
- line ① can also be written as  
select count(transaction-id), order-date ...
- select count(transaction-id) as "# of orders", customer-id  
from transactions  
group by customer-id with rollup;
- select sum(hourly-pay) as "hourly pay", employee-id  
from employees  
group by employee-id with rollup;

## On delete

- on delete set null = when a Foreign key is deleted, replace FK with null
- on delete cascade = when a Foreign key is deleted, delete row
- delete \* from transactions;
- There may be a foreign key while deleting the row: cause error:  
delete from customers  
where customer-id = 4;  
Delete \* from customers;   
customer-id column is a foreign key.  
foreign key constraint fail
- set foreign-key-checks = 0; ← use this to remove without error.  
delete from customers  
where customer-id = 4;
- set foreign-key-checks = 1; ← set back to one
- In transactions

Transaction-id	amount	customer-id	order-date
1005	2.49	4	

- Adding back to the Customers: 4 ← can replace by null or delete row



insert into customers

values (4, "peppy", "Puff", 2, "PP...");

➤ During creating a table (new)

create table transactions (

transaction-id int primary key,

amount decimal (5, 2),

customer-id int,

order-date date,

Foreign key (customer-id) references customers (customer-id)

on delete set null

);

2:53

➤ Deleting a foreign key that already exist.

alter table transactions drop foreign key fk-customer-id;

➤ adding foreign-key constraint to transaction table having clause on delete set null.

alter table transactions

add constraint fk-customer-id

foreign key (customer-id) references customers (customer-id)

on delete set null;

➤ delete from customers

where customer-id=4;

select \* from transactions;

➤ add peppy to customers:

insert into customers

values (4, "peppy", "Puff", 2, "PP...");

➤ alter table transactions

drop foreign key fk-customer-id;

5:26

➤ For adding cascade to on delete cascade clause to a table that already exist

alter table transactions

add constraint fk-transactions-id

foreign key (customer-id) references customers (customer-id)

on delete cascade;

update transactions



```
set customer-id=4
where transaction-id=1005;
Delete from customers
where customer id=4;
```

Select \* from transactions; // then entire row will be removed.  
 \* Because it was referencing the customer with an id=4. we set the foreign key to delete the entire row when foreign key is deleted, that i.e., on delete cascade.

Stored procedure

➤ Is prepared sql code that you can save, great if there's a query that you write often.

```
➤ create procedure get-customers()
begin
  select * from customer; end;
```

end;

➤ delimiter \$\$ ← or //

```
create procedure get-customers()
begin
  select * from customers;
end $$
delimiter;
```

➤ then refresh navigator, under stored-procedure we have get-customers to invoke  
 call get-customers(); # clear the query

➤ Does same as select \* from customers.

➤ To drop a procedure

```
drop procedure get-customers;
```

➤ Takes some id using procedure:

```
create procedure find-customer(IN id INT)
begin
```

```
  select *
  from customers
  where customer-id=id;
```

delimiter \$\$

```
create procedure find-customer(in id int)
```

Type of data  
 ↓



begin

```
select *  
from customers  
where customer_id = id;
```

end \$\$

Delimiter;

➤ To invoke customer id with 1

call find\_customer(1);

➤ Drop procedure find\_customer;

➤ Using first name and last name

delimiter \$\$

create procedure find\_customer (In f-name varchar(50), In l-name varchar(50))

begin

```
select * from customers
```

```
where first-name = f-name and last-name = l-name;
```

end \$ \$

delimiter;

➤ call find\_customer ("Larry", "Lobster");

➤ reduces network traffic; increases performance;

➤ secure, admin can grant permission to use;

➤ Increases memory usage of every connection;

Triggers: ➤ when an event happens, do something, eg: (Insert, update, delete)

checks data, handles errors, audition tables.

➤ alter table employees

add column salary decimal(10,2) after hourly\_pay;

select \* from employees;

➤ update employees

set salary = hourly\_pay \* 2080;

select \* from employees;

➤ whenever we want to update employee hourly pay, we would also like to update salary automatically, no need to calculate manually each time.



➤ Before or after, what  $\Rightarrow$  Before update; "for each row" ~~not~~ for more than one row  
set new.salary = (new.hourly-pay \* 2080); new is used for newly updated value.  
not considering only old value of salary or hourly-pay.

➤ Triggers available at: tables  $\rightarrow$  employees  $\rightarrow$  Triggers

➤ To show all the triggers:

Show triggers;

➤ update employees

Set hourly-pay = 50

where employee-id = 1;

Select \* from employees; # new automatically updates salary.

➤ update employees

set hourly-pay = hourly-pay + 1; # incrementing everybody's by one

Select \* from employees; # now see everybody's salary gets changed.

➤ create trigger before-hourly-pay-insert

before insert on employees

for each row

set new.salary = (new.hourly-pay \* 2080);

➤ insert into employees

values (6, "Sheldon", "Plankton", 10, Null, "Janitor", "2023-01-07", 5);

Select \* from employees;

(done automatically)

# salary automatically generated.

➤ create table expenses

expense-id int primary key;

expense-name varchar(50),

expense-total decimal(10,2)

); Select \* from expenses;

➤ insert into expenses

values (1, "salaries", 0),

(2, "supplies", 0),

(3, "taxes", 0);

Select \* from expenses;

➤ update expenses



set expense-total = (select sum(salary) from employees)  
where expense-name = "salaries";

select \* from expenses;

➤ when ever we delete a employee, we would update value from other table.

➤ create trigger after salary-delete

after delete on employees

for each row

update expenses

set expense-total = expense-total - old.salary

where expense-name = "salaries";

➤ ~~delete~~ delete from employees

where employee-id = 6;

select \* from expenses; # where expense-total gets changed after delete employee.

➤ Trigger that updates salaries when inserting a new employee.

create trigger after-salary-insert

after insert on employees

for each row [table]

update expenses

set expense-total = expense-total + new.salary

where expense-name = "salaries";

➤ insert into employees

salaries automatically by trigger

values (6, "Sheldon", "Plankton", 10, Null, "Janitor", "2023-01-07", 5);

select \* from expenses; # when inserting a new employee salary gets changed.

➤ now when we want to update salary then expense-total gets changed.

create trigger after salary-update

after update on employees

for each row

update expenses

set expense-total = expense-total + (new.salary - old.salary)

where expense-name = "salaries";

➤ update employees

set hourly-pay = 100



Page No.			
Date			

where employee\_id = 1; # expense-total gets changed when updating salary.