ANCHOR by Panasonic

Date : / /

SQL

Structured Query language

- create - update - retrieve - delete

DBMs - database management system

Database :- Think of it as a folder
Tables :- files found within folders

To create database.
- CREATE DATABASE myDB;
- USE myDB;

To delete database
- DROP DATABASE myDB;

To make a read only Database
Here you can not make any changes but we can access it →

Date : / /

SQL

- ALTER DATABASE myDB READ ONLY=1;
  (1 → you can only read; 0 → you can modify)
- CREATE TABLE employees(
  employee_id INT,
  first_name VARCHAR(50),
  last_name VARCHAR(50),
  hourly_pay DECIMAL(5,2),
  hire_date DATE

When we want to select all elements in a database.

- SELECT * from employees;

To rename a table
- rename table employees to workers;
  (employees → old table; workers → rename we want to do)

To add new col in table
ALTER TABLE employees
ADD phone_number VARCHAR(15);

To alter and rename
ALTER TABLE employees
RENAME COLUMN phone_number to email;

To modify size

ALTER TABLE employees
modify column email varchar(100);

To change position

ALTER TABLE employees
modify email varchar(100)
after last_name;

To insert value inside Table
INSERT INTO employees
VALUES (1, "Eugene", "Krabs", 25.50, "2023-01-02");
(year - month - day)
To add multiple elements =>
insert into employees values
(), (), ();

Where clause.
- select * from employees where hourly_pay <= 35;

!=
- select * from employees where employee_id != 1;

- select * from employees where hire_date IS NULL;

To updat table.

```
update employees
set hourly_pay = 10.25
where employee_id = 106;
```

To set same value throughout columns

```
update employees set hourly_pay = 10.25;
```

- `delete from employees;`
  ⇓ It will delete all the tables always give where clause with it.

```
delete from employees
where employee_id = 6;
```

When we not want to commit changes automatically then use.

- `set autocommit = off;`

```
commit;
```

⇓ after this if you delete some thing automatically then you can use.

- `Rollback;`

⇒ And if you want to save changes then use

- `commit;`

To use current date, time and datetime.

- Create table test (mydate date, mytime time, mydatetime datetime);
- insert into test values (current_date(), current_time(), now());

It will set the current time and date

current_date() + 1 ⇒ next day
current_date() - 1 ⇒ previous day

- Unique constraint

- Create table products (
  product_id INT,
  product_name varchar(25) UNIQUE,
  price DECIMAL(4,2)
  );

⇓ No two values should be same

If we forget to add the unique key then we can alter and add constraint.

- 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
  );

⇓ It should never be zero

- If you are in safe mode, you can disable by going to: Edit ⇒ Preferences ⇒ SQL Editor ⇓ uncheck safe updates ⇐ (restart)

ANCHOR by Panasonic

Date: / /

If we want to already existing table then we can use alter.

- alter table products modify price decimal(4,2) NOT NULL;

- check constraint.

```
alter table employees add
constraint chk_hourly_pay
check (hourly_pay >= 10.00);
```

to drop the check

```
alter table employees drop
check chk_hourly_pay;
```

DEFAULT constraint.

If we not specify value then it will be set to default

```
alter table product alter
price ~~decimal~~ set default 0;
```

- Primary key ⇒ It should be unique + not null
- A table can have only one primary key constraint
- No duplicate and no null value

```
Create table transactions (
transaction_id INT PRIMARY KEY,
amount DECIMAL(5,2)
);
```

If already exist then use

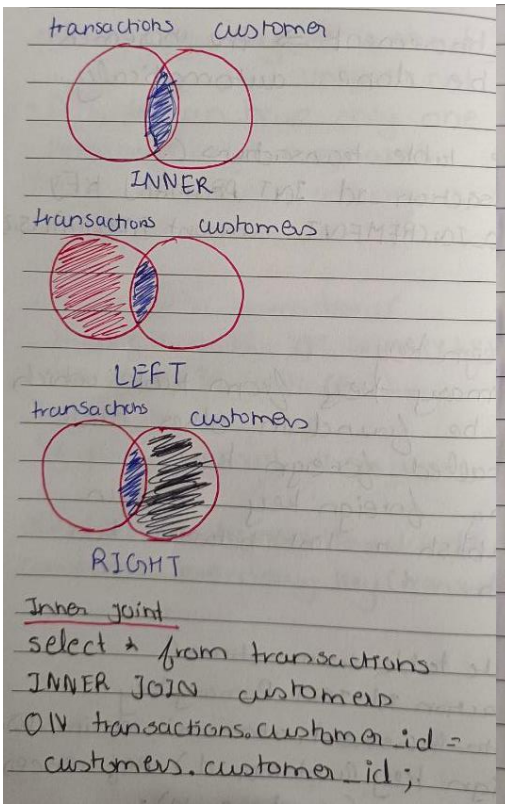
```
alter table transactions add
constraint primary key (transaction_id);
```

- Auto increment ⇒ the increment will be done automatically.

```
Create table transactions (
transaction_id INT PRIMARY KEY
AUTO_INCREMENT, amount DECIMAL(5,2)
);
```

- foreign key
- a primary key from table which can be found in other table is called foreign key.
- using foreign key we can establish a link between two tables.

```
Create table transactions (
transaction_id INT Primary key
Auto-increment, amount decimal(5,2)
foreign key (customer_id) References
customers (customer_id);
```



Inner joint

```
select * from transactions
INNER JOIN customers
ON transactions.customer_id =
customers.customer_id;
```

Left joint

```
select * from transactions
LEFT JOIN customers ON
transactions.customer_id =
customers.customer_id;
```

Right joint

```
select * from transactions
RIGHT JOIN customers ON
transactions.customer_id =
customers.customer_id;
```

- If you want to count the no. of transactions

```
select count(amount) AS
"today's transactions" from
transactions;
```

• To find maximum

• select MAX (amount) AS maximum
FROM transactions;

• select MIN (amount)
FROM transactions;

• select sum (amount) as sum
from transactions;

ALIASE → It will concatinate two strings

Select CONCAT (first_name, last_name) AS Full_name
FROM employees;

AND
select * from employees
where hire_date < "2023-01-5"
AND "job = "cook";

OR
select * from employees
where job = "cook" OR job = "cashier";

NOT
select * from employees
where NOT job = "manager";

BETWEEN
select * from employees
where hire_date BETWEEN "2023-01-04"
AND "2023-01-07";

IN
select * from employees
where job IN ("cook", "cashier", "janitor");

• Wild card characters % _
• used to substitude one or more characters in a string

LIKE ⇒ It is use in where clause to search for any pattern.

when we want to return the name starting with S at that time we will be using S along with % also the LIKE operator will be used

Select * from employees
where first_name LIKE "s%";

% r
↳ name will end with r

* Pattern Matching
WHERE Name LIKE 'a%'
WHERE Name LIKE '%a'
WHERE Name LIKE '%a%'
WHERE Name LIKE '_a%'
WHERE CustomerName LIKE 'a_%'
WHERE ContactName LIKE 'a%y'

_ ⇒ It represent one random letter

* Order By clause ⇒ arranged in ascending order

• select * from employees
ORDER BY last_name;

• select * from employees
ORDER BY last_name DESC;
descending order

- Limit clause is used to limit the number of records.
- Useful if you're working with a lot of data.
- can be used to display a large data on pages (paginations).

- select * from customers
limit 3;

we can find the highest or lowest when we combine it with order by clause

- select * from customers
order by last_name, limit 1;
desc
⇓
It will limit the number of elements

using offset is very useful when working with large data set

select * from customers
limit 3, 1;
offset

- Union combines the results of two or more SELECT statements
(It will not include duplicate values)
⇓
It should have same no. of columns)

select * from income
union
select * from expenses;

- If we want to perform the union of two table and they are not equal in size then we can not perform union operation.

union all - It will include all elemement even the duplicate one.

- 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 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
ON a.referral_id = b.customer_id;

Imp

* 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;

- select * from employee_attendance;

- 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

- create index last_name_idx ON customers (last_name);
- show indexes from customers;
- select * from customers where last_name = "Puff";

- create index last_name_first_name_idx ON customers (last_name, first_name);
- show indexes from customers;

* 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, hourly_pay from employees where hourly_pay > (select avg(hourly_pay) from employees);

- select first_name, last_name from customers where customer_id IN (select distinct customer_id from transactions where customer_id is not null);

- GroupBy = aggregate all rows by a specific column often used with aggregate functions ex. SUM(), MAX(), MIN(), AVG(), COUNT();

- select sum(amount), order_date from transactions group by order_date;

- select count(amount), customer_id from transactions group by customer_id;

after using groupby clause we will use having clause it will act same as group by clause

Date: / /

by Panasonic

add additional row = with grand total

Rollup extension of the GROUP By clause
- produces another row and shows the GRAND TOTAL (super-aggregate value).

- 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 SET NULL = when a FK is deleted, replace FK with NULL
- ON DELETE CASCADE = when a fk is deleted, delete row

```
Alter table transactions
Add constraint fk_customer_id
Foreign key (customer_id) References
customers (customer_id) ON Delete
set null;
```

```
alter table transactions
add constraint FK_transactions_id
Foreign key (customer_id) References
customers (customer_id)
ON delete cascade;
```

- Stored procedure = is prepared SQL code that you can save great if there's a query that you write often.

```
- DELIMITER $$
Create procedure get_customers()
BEGIN
    Select * from customers;
END $$
DELIMITER ;

CALL get_customers();
```

```
DELIMITER $$
Create procedure find_customer
(IN id INT)
BEGIN
    select *
    from customers
    where customer_id = id;
END $$
DELIMITER;
//

- CALL find_customer (1);
- 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", "Labstar");
```

Stored procedure = is prepared SQL code that you can save great if theres query that you write often.
- reduces network traffic
- Increases performance
- secure, admin can grant permission to use
- Increases memory usage of every connection.

- Trigger - when an event happens, do something
- ex. (INSERT, UPDATE, DELETE)
- checks data, handles errors, auditing tables

```
UPDATE employees
set salary = hourly_pay * 2080;
Select * from employees;
```

- create trigger before_hourly_pay_update
before update on employees
for each row
set NEW.salary = (NEW.hourly_pay * 2080);

- show trigger;

- update employees
set hourly_pay = 50
where employee_id = 1;
select * from employees;

It will change the details of salary when hourly_pay is changed as trigger is called, in case of update operation

- update employees
set hourly_pay = hourly_pay + 1;
select * from employees;