SQL _Queries

Resource:

https://balanced-quince-db1.notion.site/SQL-7347f5956fe347f887b4132c716cd236#17bc403a1add453db519621da47c1de3

Database queries

CREATE DATABASE LOGICFIRST; -- creates a new database

-- TO DELETE A DATABASE

DROP DATABASE LOGICFIRST;

DROP SCHEMA LOGICFIRST; -- same as above. u can use DATABASE Or SCHEMA

DROP SCHEMA IF EXISTS LOGICFIRST; -- prevents error if db not found

SHOW DATABASES; -- shows all the databases

SHOW SCHEMAS; -- same as above. shows schemas/db

USE SYS; -- uses this database for all further commands

SHOW TABLES;-- shows all tables in the database being used

Table - Create,Delete,Alter

primary key - uniquely identifies a row in a table

//creating a table

```
CREATE TABLE student(
        id INT PRIMARY KEY,
   name VARCHAR(30),
   gpa DECIMAL(3,2)
);
-- ----or-----
CREATE TABLE student(
        id INT,
   name VARCHAR(30),
   gpa DECIMAL(3,2),
```

PRIMARY KEY(id)

);

DROP TABLE student; -- drops the table

DESCRIBE student; -- describes the columns in the table student

ALTER TABLE student ADD department VARCHAR(5); -- Adds a new column department to the student table

ALTER TABLE student DROP COLUMN department; -- drops the department column from student table

-- ---or---

ALTER TABLE student DROP department; -- same as above

Inserting Data

INSERT INTO student VALUES(1,"Aarthi",7.6);

INSERT INTO student VALUES(2,"Anitha",8.5); -- inserts a row. give values in column order

INSERT INTO student VALUES

(3,"Anitha",8.5),

(4,"Arul",8.2),

(5,"Ashwin",7.6); -- inserts more than one row

INSERT INTO student(id,name) VALUES(5,"Balaji"),(6,"Chandru"); -- inserts specific columns.

Select

SELECT * FROM student; -- displays all rows and columns in the student table

SELECT id,name FROM student; -- displays specific columns

Where Clause and Conditions

Query for Employee table (click the initial arrow to expand)

CREATE TABLE employee (

emp_id INT PRIMARY KEY,

ename VARCHAR(30),

job_desc VARCHAR(20),

salary INT);

INSERT INTO employee VALUES(1,'Ram','ADMIN',1000000);

INSERT INTO employee VALUES(2,'Harini','MANAGER',2500000);

INSERT INTO employee VALUES(3,'George','SALES',2000000);

INSERT INTO employee VALUES(4,'Ramya','SALES',1300000);

INSERT INTO employee VALUES(5,'Meena','HR',2000000);

INSERT INTO employee VALUES(6,'Ashok','MANAGER',3000000);

INSERT INTO employee VALUES(7,'Abdul','HR',2000000);

INSERT INTO employee VALUES(8,'Ramya','ENGINEER',1000000);

INSERT INTO employee VALUES(9,'Raghu','CEO',8000000);

INSERT INTO employee VALUES(10,'Arvind','MANAGER',2800000);

INSERT INTO employee VALUES(11,'Akshay','ENGINEER',1000000);

INSERT INTO employee VALUES(12,'John','ADMIN',2200000);

INSERT INTO employee VALUES(13,'Abinaya','ENGINEER',2100000);

where is used to filter the records. The rows are filtered based on conditions.

SELECT column1, column2, ...

FROM table_name

WHERE condition;

Following can be used within the condition.

=	Equal
>	Greater than
<	Less than
>=	Greater than or equal
<=	Less than or equal
<>	Not equal. Note: In some versions of SQL this operator may be written as !=
BETWEEN	Between a certain range
LIKE	Search for a pattern
IN	To specify multiple possible values for a column
NOT	negation

AND/OR can be used to combine the relational operators.

```
SELECT * FROM employee
WHERE ename = 'Ramya';

SELECT emp_id,ename,salary FROM employee
WHERE salary>2000000;

SELECT emp_id,ename,salary FROM employee
WHERE salary<2600000 AND job_desc = 'MANAGER';

SELECT * FROM employee
WHERE job_desc='ADMIN' OR job_desc='HR'; -- though this works next command is a much better choice

SELECT * FROM employee
WHERE job_desc IN ('ADMIN','HR');

SELECT * FROM employee
WHERE job_desc NOT IN ('MANAGER','CEO');

SELECT * FROM employee
WHERE salary BETWEEN 2000000 AND 3000000
LIMIT 5; --limits the records shown

SELECT * FROM employee
LIMIT 5;  -- Different syntax in oracle/sql server
```

Using Like and wildcards

LIKE is used with WHERE clause for searching a specific pattern in a column. It is used along with the following wild cards

% represents zero or more characters

_ represents exactly one character

SELECT * FROM employee

WHERE ename LIKE 'A%'; -- filters name starting with A

SELECT * FROM employee

WHERE ename LIKE 'R%a'; -- filters name starting WITH R AND ending with a

SELECT * FROM employee

WHERE ename LIKE '%I%'; -- filters name containing I

SELECT * FROM employee

WHERE ename LIKE '__I%'; -- filters name with i as third character

SELECT * FROM employee

WHERE ename LIKE 'R\%'; -- filters name starting with R%. \ is the escape character.

Update and Delete

UPDATE employee

SET job_desc = "Analyst"; -- updates all job_desc of all rows to "Analyst" when safe update not enabled

UPDATE employee

SET job_desc = "Analyst"

WHERE job_desc = "Engineer"; -- changes Engineer to Analyst in all applicable rows

UPDATE employee

SET job_desc = "Analyst"

WHERE emp_id=1;

DELETE FROM employee; -- deletes all rows

DELETE from employee

WHERE emp_id = 12;

Distinct

SELECT DISTINCT job_desc

FROM employee; -- shows only distinct values without duplicates

Order By

SELECT * FROM employee

ORDER BY salary; -- order by salary asc

SELECT * FROM employee

ORDER BY salary ASC; -- order by salary asc

SELECT * FROM employee

ORDER BY salary DESC; -- order by salary desc

SELECT * FROM employee

WHERE job_desc="MANAGER"

ORDER BY salary DESC; -- order the manager salaries in desc order

SELECT * FROM employee

ORDER BY job_desc,ename; -- first sorts by job_desc and then by ename

SELECT * FROM employee

ORDER BY (CASE job_desc -- specific order

WHEN 'CEO' THEN 1

WHEN 'MANAGER' THEN 2

WHEN 'HR' THEN 3

WHEN 'ANALYST' THEN 4

WHEN 'SALES' THEN 5

ELSE 100 END);

Copy Table

INSERT INTO first_table_name [(column1, column2, ... columnN)]

SELECT column1, column2, ...columnN

FROM second_table_name

Functions

- Here is good source for learning all functions https://www.techonthenet.com/mysql/functions/index.php

  aggregate functions https://dev.mysql.com/doc/refman/8.0/en/aggregate-functions.html

SELECT COUNT(*) FROM employee; -- total count of entries in the table

SELECT AVG(salary) FROM employee; -- avg salary of all employees

SELECT AVG(salary)

FROM employee

WHERE job_desc="MANAGER"; -- avg salary of managers

SELECT SUM(salary)

FROM employee

WHERE job_desc="ANALYST"; -- total salary given to all analysts

SELECT * FROM employee

WHERE salary = (SELECT MAX(salary)

FROM employee); -- display the employee with

SELECT MIN(salary) FROM employee;

SELECT UCASE(ename),salary

FROM employee; -- uppercase

SELECT ename,CHAR_LENGTH(ename)

FROM employee;

SELECT ename,CONCAT("Rs.",salary)

FROM employee; -- adds Rs. to the beginning of salary

SELECT ename,CONCAT("Rs.",FORMAT(salary,0))

FROM employee; -- formats the number to add comma. The second argument(0 here) represents digits to round off after decimal

SELECT ename,LEFT(job_desc,4)

FROM employee; -- returns only the first 4 characters of the ename

Using Date

ALTER TABLE employee ADD COLUMN Hire_Date DATE; -- adding hire_date column

UPDATE employee

SET Hire_Date="2012-10-05"; -- updating hire_date

UPDATE employee

SET Hire_Date="2014-10-05"

WHERE job_desc = "ANALYST"; -- updating hire_date

SELECT NOW(); -- Current date and time

SELECT DATE(NOW()); -- current date

SELECT CURDATE(); -- current system date

SELECT DATE_FORMAT(CURDATE(),'%d/%m/%Y'); -- to change the display format. use %d %m and %y or %Y in required format.

SELECT DATEDIFF(CURDATE(),'2020-01-01') DAYS; -- to calculate date difference

```
SELECT CURDATE() 'start date',
 DATE_ADD(CURDATE(),INTERVAL 1 DAY) 'one day later',
 DATE_ADD(CURDATE(),INTERVAL 1 WEEK) 'one week later',
 DATE_ADD(CURDATE(),INTERVAL 1 MONTH) 'one month later',x
 DATE_ADD(CURDATE(),INTERVAL 1 YEAR) 'one year later';
```

start date	one day later	one week later	one month later	one year later
2022-02-12	2022-02-13	2022-02-19	2022-03-12	2023-02-12

2022-02-12 2022-02-13 2022-02-19

Group By and Having

Group by is used to group the table based on conditions and analyze values within the group using aggregate functions.

Where is used to filter the rows before grouping. Having is used to filter the groups.

```
SELECT job_desc,FORMAT(AVG(salary),0) avg_sal
FROM employee
GROUP BY job_desc; -- shows avg salary of each category within job_desc


SELECT job_desc,COUNT(emp_id) count
FROM employee
GROUP BY job_desc; -- displays number of employees count in each job_desc category


SELECT job_desc,COUNT(emp_id) AS count -- using as for aliasing
FROM employee
GROUP BY job_desc
HAVING COUNT(emp_id)>1; --  displays number of employees count in each job_desc category only when count is greater than 1.


SELECT job_desc,COUNT(emp_id) AS count
FROM employee
GROUP BY job_desc
HAVING COUNT(emp_id)>1
```

ORDER BY job_desc; -- same as above ordered by job_desc asc

SELECT job_desc,COUNT(emp_id) AS count

FROM employee

GROUP BY job_desc

HAVING COUNT(emp_id)>1

ORDER BY COUNT(emp_id) DESC -- same but ordered by Desc order of COUNT in each group

SELECT job_desc,COUNT(emp_id) AS count

FROM employee

WHERE salary>1500000

GROUP BY job_desc

HAVING COUNT(emp_id)>1

ORDER BY COUNT(emp_id) DESC; -- with additional filtering of salary> 15L. Only those with sal more than 15L is considered for grouping

job_desc	avg_sal
ADMIN	1,000,000
ANALYST	1,366,667
CEO	8,000,000
HR	2,000,000
MANAGER	2,766,667
SALES	1,650,000

job_desc	count
ADMIN	1
MANAGER	3
SALES	2
HR	2
ANALYST	3
CEO	1

Constraints

NOT NULL, AUTO_INCREMENT, DEFAULT, CHECK, UNIQUE

```
CREATE TABLE employee (
emp_id INT PRIMARY KEY AUTO_INCREMENT, -- id will be auto incremented for new rows
ename VARCHAR(30) NOT NULL, -- null value cannot be inserted for the column
job_desc VARCHAR(20) DEFAULT 'unassigned', -- sets default when not mentioned
salary INT,
pan VARCHAR(10) UNIQUE,-- cannot contain duplicates
CHECK (salary>100000));

INSERT INTO employee(ename,salary) VALUES ('Ramya',1000000);
INSERT INTO employee(ename,salary) VALUES ('Riya',10000); -- erros because of violation of salary check constraint
SELECT * FROM employee;
```

Foreign Key

Foreign key is a field in one table referring to the primary key of another table.

```
-- drop previously created tables and create a branch table
CREATE TABLE branch (
branch_id INT PRIMARY KEY AUTO_INCREMENT,
br_name VARCHAR(30) NOT NULL,
addr VARCHAR(200) );

-- create employee table with branch_id as foreign key. It refers to the branch_id of branch table.
CREATE TABLE employee (
emp_id INT PRIMARY KEY,
ename VARCHAR(30),
job_desc VARCHAR(20),
salary INT,
branch_id INT,
```

CONSTRAINT FK_branchId FOREIGN KEY(branch_id) REFERENCES branch(branch_id));

-- droping FK

ALTER TABLE employee

DROP FOREIGN KEY FK_branchId;

Index

Index are used for fast lookups. Speeds up select query but delays insert/update. Also take up more memory.

SHOW INDEX FROM employee; -- show current indices

CREATE INDEX name_index ON employee(ename); -- creates a new index

ALTER TABLE employee

DROP INDEX name_index; -- drop index

ALTER TABLE employee

ADD INDEX(ename); -- create index using alter command

On Delete

CREATE TABLE employee (

emp_id INT PRIMARY KEY AUTO_INCREMENT,

ename VARCHAR(30) NOT NULL,

job_desc VARCHAR(20),

salary INT,

branch_id INT,

CONSTRAINT FK_branchId FOREIGN KEY(branch_id) REFERENCES branch(branch_id)

ON DELETE CASCADE -- on deleting a row in branch table, the corresponding entries in employee table will be deleted

);

CREATE TABLE employee (

emp_id INT PRIMARY KEY AUTO_INCREMENT,

ename VARCHAR(30) NOT NULL,

job_desc VARCHAR(20),

salary INT,

branch_id INT,

CONSTRAINT FK_branchId FOREIGN KEY(branch_id) REFERENCES branch(branch_id)

ON DELETE SET NULL -- on deleting a row in branch table, the branchid corresponding entries in employee table will be made null

);

Joins

Joins are used to join columns from two tables

DROP TABLE employee; -- drop and freshly create

CREATE TABLE branch (

branch_id INT PRIMARY KEY AUTO_INCREMENT,

br_name VARCHAR(30) NOT NULL,

addr VARCHAR(200));

CREATE TABLE employee (

emp_id INT PRIMARY KEY AUTO_INCREMENT,

ename VARCHAR(30) NOT NULL,

job_desc VARCHAR(20),

salary INT,

branch_id INT,

CONSTRAINT FK_branchId FOREIGN KEY(branch_id) REFERENCES branch(branch_id)

);

INSERT INTO branch VALUES(1,"Chennai","16 ABC Road");

INSERT INTO branch VALUES(2,"Coimbatore","120 15th Block");

INSERT INTO branch VALUES(3,"Mumbai","25 XYZ Road");

```
INSERT INTO branch VALUES(4,"Hydrabad","32 10th Street");

INSERT INTO employee VALUES(1,'Ram','ADMIN',1000000,2);
INSERT INTO employee VALUES(2,'Harini','MANAGER',2500000,2);
INSERT INTO employee VALUES(3,'George','SALES',2000000,1);
INSERT INTO employee VALUES(4,'Ramya','SALES',1300000,2);
INSERT INTO employee VALUES(5,'Meena','HR',2000000,3);
INSERT INTO employee VALUES(6,'Ashok','MANAGER',3000000,1);
INSERT INTO employee VALUES(7,'Abdul','HR',2000000,1);
INSERT INTO employee VALUES(8,'Ramya','ENGINEER',1000000,2);
INSERT INTO employee VALUES(9,'Raghu','CEO',8000000,3);
INSERT INTO employee VALUES(10,'Arvind','MANAGER',2800000,3);
INSERT INTO employee VALUES(11,'Akshay','ENGINEER',1000000,1);
INSERT INTO employee VALUES(12,'John','ADMIN',2200000,1);
INSERT INTO employee VALUES(13,'Abinaya','ENGINEER',2100000,2);
INSERT INTO employee VALUES(14,'Vidya','ADMIN',2200000,NULL);
INSERT INTO employee VALUES(15,'Ranjani','ENGINEER',2100000,NULL);

SELECT * FROM employee;
SELECT * FROM branch;

-- inner join: only matching rows
SELECT employee.emp_id,employee.ename,employee.job_desc,branch.br_name
FROM employee
INNER JOIN branch
ON employee.branch_id=branch.branch_id
ORDER BY emp_id;

-- below query gives same result without using join keyword
SELECT employee.emp_id,employee.ename,employee.job_desc,branch.br_name
FROM employee,branch
```

```
WHERE employee.branch_id=branch.branch_id
ORDER BY emp_id;

-- using table name alias
SELECT e.emp_id,e.ename,e.job_desc,b.br_name
FROM employee AS e
INNER JOIN branch AS b
ON e.branch_id=b.branch_id
ORDER BY e.emp_id;

-- Right join is matched rows + all other rows in right table
SELECT employee.emp_id,employee.ename,employee.job_desc,branch.br_name
FROM employee
RIGHT JOIN branch
ON employee.branch_id=branch.branch_id
ORDER BY emp_id;

-- Left join is matched rows with all other rows in left table
SELECT employee.emp_id,employee.ename,employee.job_desc,branch.br_name
FROM employee
LEFT JOIN branch
ON employee.branch_id=branch.branch_id
ORDER BY emp_id;

-- Cross join joins each row of first table with every other row of second table
SELECT employee.emp_id,employee.ename,employee.job_desc,branch.br_name
FROM employee
CROSS JOIN branch;

-- displays the employee count in each branch
SELECT b.br_name,COUNT(e.emp_id)
```

```
FROM branch as b
JOIN employee as e
ON b.branch_id = e.branch_id
GROUP BY e.branch_id;
```

Union

union combines two table having equal number of columns and matching datatypes

```
-- create client table similar to branch table
CREATE TABLE clients (
client_id INT PRIMARY KEY AUTO_INCREMENT,
location VARCHAR(30) NOT NULL,
addr VARCHAR(200) );

INSERT INTO clients VALUES(1,"NewYork","25 10th Block");
INSERT INTO clients VALUES(2,"Coimbatore","120 15th Block");
INSERT INTO clients VALUES(3,"London","21 ABC Road");

-- combines the two tables removing duplicates
SELECT * FROM branch
UNION
SELECT * FROM clients;

-- combines the two tables without removing duplicates
SELECT * FROM branch
UNION ALL
SELECT * FROM clients;
```

Subqueries, Exists, Any, All

Subqueries combine more than 2 queries.

```
-- Displays employee list in Chennai Branch
SELECT * FROM employee
```

```
WHERE branch_id = (SELECT branch_id
FROM branch
WHERE br_name="Chennai");

-- Displays the employees with min salary
SELECT * FROM employee
WHERE salary = (SELECT MIN(salary)
FROM employee);

        - displays the branches containing atleast one admin
SELECT branch_id,br_name
FROM branch
WHERE EXISTS
( SELECT * FROM employee
WHERE job_desc="ADMIN" AND branch.branch_id = employee.branch_id);

-- displays the branch info in which any employee gets more than 25L
SELECT branch_id,br_name
FROM branch
WHERE branch_id = ANY
(SELECT branch_id FROM employee
WHERE salary>2500000);

-- displays employees not working in chennai or coimbatore

SELECT * FROM employee
WHERE branch_id <> ALL ( SELECT branch_id FROM branch
WHERE br_name IN ("Chennai","Coimbatore"));
```

Views

```
CREATE VIEW emp_br
```

```
AS
SELECT employee.emp_id,employee.ename,employee.job_desc,branch.br_name
FROM employee
INNER JOIN branch
ON employee.branch_id=branch.branch_id
ORDER BY emp_id;

SELECT * FROM emp_br; -- selecting all rows from view

DROP VIEW emp_br; -- delete view

CREATE OR REPLACE VIEW emp_br  -- modify view
AS
SELECT employee.emp_id,employee.ename,employee.job_desc,branch.br_name
FROM employee
INNER JOIN branch
ON employee.branch_id=branch.branch_id;
```