[MYSQL DOCUMANTATION

[Document subtitle]

Abstract

[Draw your reader in with an engaging abstract. It is typically a short summary of the document. When you're ready to add your content, just click here and start typing.]

Zia Ul Hassan Chowdhury

Parks_and_recreation DataBase query statement:

Beginner series

SELECT STATEMENT:

1. ***Select statement use to see a specific column or table from database***

```
SELECT*
FROM parks_and_recreation.employee_demographics;


SELECT*
FROM parks_and_recreation.employee_salary;
```

2. *Select first name use to see only **first_name** column from the employee_demographics table:*

```
SELECT first_name
FROM parks_and_recreation.employee_demographics;
Or,
SELECT first_name, last_name, birth_date, age
From employee_demographics;
```

3. ***DISTINCT statement use to show the unique value from the table:***

```
SELECT DISTINCT first_name
FROM parks_and_recreation.employee_demographics;

SELECT distinct gender
FROM parks_and_recreation.employee_demographics;
```

WHERE CLAUSE:

1. *WHERE use to filter……our specific record and SELECT use to filter our table*

```
SELECT*
FROM parks_and_recreation.employee_demographics
WHERE first_name = 'Leslile'
;

SELECT*
FROM parks_and_recreation.employee_demographics
```

```
WHERE first_name = 'Tom';
SELECT*
FROM employee_salary
WHERE salary >= 50000
;

SELECT*
FROM employee_demographics
WHERE gender = ‘Female’
;
```

2. AND OR NOT logical operators

```
SELECT*
FROM employee_demographics
WHERE birth_date > ‘1985-01-01’
AND gender = ‘male’
;
```

```
SELECT*
FROM employee_demographics
WHERE birth_date > '1985-01-01'
OR gender = 'male'
;
```

Either this statement is true or this

```
SELECT*
FROM employee_demographics
WHERE (first_name = 'Tom' AND age ) OR age > 55
;
```

3. LIKE statement

% means anything and _ means specific

```
SELECT*
FROM employee_demographics
WHERE first_name LIKE '%r%' (-- r% means the name start with r in a word will be select and %r% means wherever r in the word select it)
```

;

SELECT*

FROM employee_demographics

WHERE first_name LIKE **'a___%'** (-- a__% means start with a and three character then anything have)

;

GROUP BY and ORDER BY

1. *GROUP BY use to **group together rows** that have the **same value in the specific column** that are grouping on. Once we group together we can run something called aggregate function on those rows.*

SELECT gender, **AVG (age)**

FROM employee_demographics

GROUP BY gender

;

```
SELECT gender, AVG(age), max(age), min(age),
COUNT(age)
FROM employee_demographics
GROUP BY gender
;
```

```
SELECT occupation, salary
FROM employee_salary
GROUP BY occupation, salary
```

2. ORDER BY

Order by use to ASCENDING OR DESCENDING the row

HAVING VS WHERE

*HAVING was specifically created for **filter the specific row from the grouping row.***

```
SELECT gender, AVG(age)
FROM employee_demographics
GROUP BY gender
HAVING AVG (age)>40;
```

```
SELECT occupation, AVG(salary)
FROM employee_salary
WHERE occupation LIKE '%manager%'
GROUP BY occupation
HAVING avg  (salary) > 75000
;
```

LIMIT & ALIASING

1. LIMIT

LIMIT specific the how many row you want in your output

```
SELECT*
FROM employee_demographics
LIMIT 2;
```

2. Aliasing

ALIASING use to change the name of the column.

```
SELECT gender, AVG(age) As avg_age
FROM employee_demographics
GROUP BY gender
```

*[use group by to **group gender** and **avage column**, and having **use to filter the column**, Aliasing **use to change the column name**]*

Having AVG(age)>40;

Intermediate series

[1.10.26]

JOINS

JOINS allow you to combine 2 tables if they have a common column

1. ### Inner join: combine 2 tables where they have the same value

```
SELECT *
FROM employee_demographics AS dem
INNER JOIN employee_salary AS sal
    ON dem.employee_id = sal.employee_id;
```

Self join

Outer join

Left join: LF. Join take everything from the left

2. *Right join: RJ. Join take everything from the right, even if there is no match in the join*

```
SELECT*
FROM employee_demographics as dem
RIGHT JOIN employee_salary as sal
        ON dem.employee_id=sal.employee_id
```

3. *SELF JOIN*

it is join that try join self

```
SELECT*
FROM employee_salary as emp
SELF JOIN employee_salary as emp2
ON emp1.employee_id= emp2.employee_id;
```

4. *Joining Multiple Table*

Example: join demographics table to salary table based on their common column, then park_departments table join using common column between the park_departments table and salary table is department id. Then we see three table together.

```
SELECT *
FROM employee_demographics as dem
INNER JOIN employee_salary as sal
ON dem.employee_id = sal.employee_id
INNER JOIN parks_department as pd
ON sal.dept_id = pd.department_id;
```

UNION

UNIONS combines to raw of data from separate table

```
SELECT first_name, last_name
FROM employee_demographics;

SELECT first_name, last_name
FROM employee_salary;
```

If the park company try to cut the old staff how they find the old staff from the data?

SELECT first_name, last_name, ‘old’ as label

FROM **employee_demographics**

WHERE age>50;

Find also who are highly paid?

SELECT first_name ‘Highly paid employee’ As label

WHERE **salary**>70000;

Combine 2 table using UNION?

SELECT first_name, last_name, ‘old’ as label

FROM employee_demographics

WHERE age>50;

UNION

SELECT first_name ‘Highly paid employee’ As label

WHERE salary>70000;

Example2

SELECT first_name, last_name, 'Old Man' as label

FROM employee_demographics

WHERE age > 40 AND gender = 'Male'

```
UNION
SELECT first_name, last_name, 'Old Lady' as label
FROM employee_demographics
WHERE age > 40 AND gender = 'Female'
UNION
SELECT first_name, last_name, 'Highly paid salary' as label
FROM employee_salary
WHERE salary>70000
ORDER BY first_name, last_name;
```

ORDER BY

REPLACE will replace specific characters with a different character that you want

1. *String Function*

Length

```
SELECT first_name, LENGTH(first_name)
FROM employee_demographics;
```

```
SELECT first_name, LENGTH(first_name)
FROM employee_demographics
order by 2;

SELECT first_name, UPPER(first_name)
FROM employee_demographics;
```

2. TRIMS: LEFT TRIMS, RIGHT TRIMS

to take the white space on the front or the end to get rid of it

```
SELECT  RTRIM('   sky   ');


SELECT first_name,
LEFT(first_name, 4),
RIGHT(first_name,4),
SUBSTRING(first_name, 3,2), -- 3,2: 3 position 2 character
birth_date, SUBSTRING(birth_date,1,4) AS birth_year
FROM employee_demographics;
```

3. Replace the replace the character

```
SELECT first_name, REPLACE(first_name, 'A', 'z')
FROM employee_demographics;
```

4. LOCATE

```
SELECT LOCATE('A', 'ZIA');

SELECT first_name, LOCATE('A', first_name)
FROM employee_demographics;
```

5. CONCAT: multiple column in to one single column

```
SELECT first_name, last_name,
CONCAT(first_name,'  ', last_name) AS Full_Name
FROM employee_demographics;
```

-- [numeric function, time function, date function, converting different data types]

Case statements

Case statements allows you to logic in your select statements like if else

```
SELECT first_name, last_name, age,
CASE
   WHEN age <=30 THEN 'Young'
   WHEN age BETWEEN 31 and 50 THEN 'Old'
   WHEN age >=60 THEN'Time to retire'

END as Age_Bracket
FROM employee_demographics;

-- pay increase and bonus
-- < 50000 = 5%
-- > 50000 = 7%
-- Finance = 10%
```

```
SELECT first_name, last_name, salary,
CASE
   WHEN salary < 50000 THEN salary + (salary * 0.05)
   WHEN salary > 50000 THEN salary + (salary * 0.07)
   END as Bonus,
CASE
     WHEN dept_id = 6 THEN salary * .10
END As Bonus
FROM employee_salary;



select *
from parks_departments;
```

Subqueries:

is basically a queries in another queries

```
SELECT *
FROM employee_demographics
WHERE employee_id IN
                    (SELECT employee_id
                        FROM employee_salary
                        WHERE dept_id = 1)
;

SELECT first_name, salary,
(SELECT Avg (salary)
FROM employee_salary)
FROM employee_salary;
```

```
SELECT gender, AVG(age), MAX(age), MIN(age),
COUNT(age)
FROM employee_demographics
GROUP BY gender;
```

```
SELECT AVG(max_age)
FROM
(SELECT gender,
AVG(age) AS avg_age,
MAX(age) AS max_age,
MIN(age) AS min_age,
COUNT(age)
FROM employee_demographics
GROUP BY gender) AS Agg_table;
```

Window function:

Like group by except they don't roll everything up into one row when grouping. Window function ***allow us to look at***

a partion *or a group but they each keep their own unique rows in the output.*

```
SELECT dem.first_name, dem.last_name, gender, AVG(salary) as avg_salary
FROM employee_demographics as dem
JOIN employee_salary as sal
  ON dem.employee_id = sal.employee_id
GROUP BY dem.first_name, dem.last_name, gender;
```

1. OVER

```
SELECT dem.first_name, dem.last_name, gender, AVG(salary)
OVER (PARTITION BY gender)
FROM employee_demographics dem
JOIN employee_salary sal
  ON dem.employee_id = sal.employee_id;
```

2. Rolling total

```
SELECT dem.first_name, dem.last_name, gender, salary, AVG(salary)
 OVER (PARTITION BY gender ORDER BY dem.employee_id) AS Rolling_Total
FROM employee_demographics dem
JOIN employee_salary sal
  ON dem.employee_id = sal.employee_id;
```

3. ROW Number

```
SELECT dem.employee_id, dem.first_name, dem.last_name, gender, salary,
ROW_NUMBER () OVER ()
FROM employee_demographics as dem
JOIN employee_salary as sal
  ON dem.employee_id = sal.employee_id;

SELECT dem.first_name, dem.last_name, gender, salary,
ROW_NUMBER () OVER (PARTITION BY gender ORDER BY salary DESC) as
row_num
FROM employee_demographics as dem
JOIN employee_salary as sal
 ON dem.employee_id = sal.employee_id;
```

4. Rank

```
SELECT dem.employee_id, dem.first_name, dem.last_name, gender, salary,
ROW_NUMBER () OVER (PARTITION BY gender ORDER BY salary DESC) AS row_num,
Rank () OVER (PARTITION BY gender ORDER BY salary DESC) as rank_num,
DENSE_RANK () OVER (PARTITION BY gender ORDER BY salary DESC) as dense_num
FROM employee_demographics as dem
JOIN employee_salary as sal
  ON dem.employee_id = sal.employee_id;

SELECT dem.employee_id, dem.first_name, dem.last_name, salary,gender,
ROW_NUMBER () OVER (PARTITION BY gender ORDER BY salary)
```

```
FROM employee_demographics as dem
JOIN employee_salary as sal
  ON dem.employee_id = sal.employee_id;
```

Advance Series

CTES:

common table expression is a temporary, name result set

1. ***Make queries easy to read***
2. ***Allow to divided complex queries into smaller logical building block***
3. ***Improve queries reusability***

```
-- CTEs
-- simple CTE
WITH HighSalaryEmployees AS(
SELECT employee_id, first_name,  salary
FROM employee_salary
WHERE salary > 50000
)
SELECT *
FROM HighSalaryEmployees;
```

-- *CTEs for Aggregations*

```
-- Find the average salary per depertment and then select only department with an average salary above 50000

WITH AverageSalary AS (

SELECT employee_id, first_name, salary, AVG(salary) AS AvgSalary

FROM employee_salary

GROUP BY employee_id, first_name, salary

)

SELECT employee_id, first_name, salary

FROM AverageSalary

WHERE salary > 70000;
```

-- CTE : subqueries, refrences with main queris

```
WITH GenderandSalary (Gender, AVG_sal, MAX_sal, MIN_sal, COUNT_sal) AS (

SELECT gender, AVG(salary) avg_sal, MAX(salary) max_sal, MIN(salary) min_sal,COUNT(salary) count_sal

FROM employee_demographics as dem

JOIN employee_salary as sal                    -- join statement use to join two table

  ON dem.employee_id = sal.employee_id                    -- join two table (on) using common id (employee_id)

GROUP BY gender                                    -- grouping only the common column between two table

)

SELECT *

FROM GenderandSalary;
```

-- Multiple CTE: joining multiple quries

```
WITH Demographics AS (
SELECT  employee_id, gender, birth_date
FROM employee_demographics
),
Salary AS (
SELECT employee_id, salary
FROM employee_salary
)
SELECT *
FROM Demographics
JOIN salary
  ON Demographics.employee_id = salary.employee_id;
```

Temporary Tables:

are only visible when they create

Store intermediate result from complex queries

```
CREATE TEMPORARY TABLE temp_table(
first_name varchar(50),                    -- create a temporaray table, which are not add our main database
last_name varchar (50),
favorite_movie varchar (100)
);
```

```
SELECT *
FROM temp_table;

INSERT INTO temp_table
VALUES
('Dane', 'Brug', 'The Dark Night'),
('Bran', 'Alex', 'Dabba');

SELECT*
FROM temp_table;
```

Stored Procedures

Save our sql code execute all the code that you wrote
Reuse code over and over again

It is helpful for store complex queries

-- Stored Procedures: is a reusable set of sql statements that are stored in database

```
CREATE PROCEDURE High_salaries ()
SELECT *
FROM employee_salary
WHERE salary > 50000;

CALL High_salaries;
```

```
DELIMITER $$
CREATE PROCEDURE High_salaries2 ()
BEGIN
	SELECT*
	FROM employee_salary
	WHERE salary>50000;
	SELECT*
	FROM employee_salary
	WHERE salary>10000;
END $$
DELIMITER ;

CALL High_salaries2() ;



DELIMITER $$
CREATE PROCEDURE High_salaries5(p_employee_id INT)
BEGIN
	SELECT salary
	FROM employee_salary
	WHERE employee_id = P_employee_id
;
END $$
DELIMITER ;

CALL High_salaries5(2);
```

TRIGGERS and EVENTS

A Triggers is a stored database object that is automatically executed when certain events occur on a specified table.

1. *Before Triggers: execute before the specified events occurs*
2. *After Triggers: execute after the specified events occur*
3. *INSERT: Triggers fires when a new record is added*
4. *UPDATE: Triggers fires when a record update*
5. *DELETE: Triggers fires when a record is delete*

```
DELIMITER $$
CREATE TRIGGERS employee_insart
        AFTER INSERT ON employee_salary
        FOR EACH ROW
BEGIN
     INSERT INTO  employee_demographics (employee_id, first_name, last_name)
    VALUES(NEW.employee_id, NEW.first_name, NEW.last_name);
END $$
DELIMITER ;
INSERT INTO employee_salary (employee_id, first_name, last_name, occupation, salary)
VALUES (13, 'Elon' , 'Musk',  'space x ceo' 100000);
```

Events:

are scheduled database tasks that are executed at specific times or intervals. They are useful for automating repetitive tasks, such as data cleanup or report generation.

Enable or Disable Event

```
SET GLOBAL events_scheduler = ON;
SET GLOBAL events_scheduler = OFF;
```