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.]

[MySQL DocumANTation

[Document subtitle]

Zia Ul Hassan Chowdhury

|  |  |
| --- | --- |
| *Topic* | *Page* |
|  |  |
|  |  |
| *Window Function* | ***19*** |
|  |  |
|  |  |
|  |  |
|  |  |

***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;

1. *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;

1. ***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’

;

1. *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'

**Either this statement is true or this**

Or

**OR** gender = 'male'

;

SELECT\*

FROM employee\_demographics

WHERE (first\_name = 'Tom' AND age ) **OR** age > 55

;

1. *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

1. *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;

1. *Aliasing*

*ALIASING use to change the name of the column.*

SELECT gender, AVG(age) As avg\_age

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

GROUP BY gender

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*

1. ***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

1. ***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;

1. ***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;

1. *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;

1. *Replace the replace the character*

SELECT first\_name, REPLACE(first\_name, 'A', 'z')

FROM employee\_demographics;

1. *LOCATE*

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

SELECT first\_name, LOCATE('A', first\_name)

FROM employee\_demographics;

1. *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.*

***allows us to perform calculations across a specific set of rows related to the current row. These calculations happen within a defined window of data, and they are particularly useful for aggregates, rankings, and cumulative totals without altering the dataset.***

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;

1. *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;

1. *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;

1. *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 TRIGGER employee\_insart

AFTER INSERT ON employee\_salary -- After Triggers: execute after the specified events occur

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, dept\_id)

VALUES(13, 'Elon', 'Musk', 'Spece X CEO', 10000000, NULL);

SELECT \*

FROM employee\_demographics;

SELECT \*

FROM employee\_salary;

***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;

-- Events

-- retire people who are more than 60

DELIMITER $$

CREATE EVENT delete\_retirees

ON SCHEDULE EVERY 30 second

DO

BEGIN

DELETE

FROM employee\_demographics

WHERE age>=60;

END $$

DELIMITER ;

SELECT \*

FROM employee\_demographics;

SHOW VARIABLES LIKE 'event%';