General overview of SQL:- Introduction

↑ Structured Query Language

Let's talk about database:

- What is a database?
  - Database is an organized collection of data stored which can be accessed electronically.
  - Database mostly has one or more tables

Remember:

- SQL is not case-sensitive

Import commands to know before using SQL:

1. SELECT: extracts data
2. UPDATE: updates data
3. DELETE: deletes data
4. CREATE TABLE: creates new table
5. INSERT INTO: inserts new data into database
6. ALTER TABLE: modifies database
7. DROP TABLE: deletes a table

Remember:

- A semi-colon at the end of each SQL statement is a way to separate each statement.
- Multiple SQL queries can be executed at same time.

Query Structure

Table: country

ID	name	pop. (mil)
1	USA	330
2	canada	38
3	Mexico	129

Required Query:

```
SELECT
    field 1 (column 1)
  , field 2 (column 2)
  , field 3 (column 3)
FROM tablename;
```

Selecting fields from the source

← source

To select all records from source / table:

```
SELECT * FROM tablename;
```

Optional Query: Conditions

1. WHERE clause: this is used to filter the data using individual rows

placed after from

```
SELECT
    name
  , population
FROM country
WHERE population > 100;
```

Result:

name	Population
USA	330
Mexico	129

like operator

```
SELECT name
  , population
FROM country
WHERE name like '%xico%';
```

Result:

name	Population
mexico	129

Operator	Description
=	equal
>	Greater than
<	Lesser than
>=	Greater than / equal to
<=	Lesser than / equal to
!=	not equal to
Between	between ranges
Like	pattern matching (eg: '%value%')
In	possible value
IS NULL	check for missing values

Aggregation

2. GROUP BY clause: groups rows that have the same values into a summary

customer_ID	order date	item
1	2021/1/1	book
2	2021/2/1	pen
3	2021/4/1	book

Table: orders

```
SELECT item
    count(customer_ID)
        AS count
FROM orders
GROUP BY item;
```

name the new field count(customer_ID)

Result:

Item	count
Book	2
pen	1

Other aggregate functions that can be used with group by:

- max(): maximum val.
- min(): minimum val.
- sum(): sum
- avg(): mean
- count(): count

Sorting

3. ORDER BY clause: orders the results in either ascending / descending order.

default → (ASC) (DESC)

Customer_ID	order date	item
1	2021/1/1	book
2	2021/2/1	pen
3	2021/4/1	book

Table: orders

```
SELECT *
FROM orders
ORDER BY items;
```

Results:

Customer_ID	Order date	item
1	2021/1/1	book
3	2021/4/1	book
2	2021/2/1	pen

Restricting aggregated value

4. HAVING: Basically a where clause but for aggregation

Customer_ID	order date	item
1	2021/1/1	book
2	2021/2/1	pen
3	2021/4/1	book

Table: orders

```
SELECT item,
  ,count (customer_id) as count
FROM orders
GROUP BY item
HAVING count(customer_id) > 1;
```

placed after group by

Result:

Item	count
Book	2

* Remember: if SQL query has both WHERE & HAVING, WHERE is executed first

Limiting number of rows

5. LIMIT: specifies the number of records we want to return

customer_ID	order date	item
1	2021/1/1	book
2	2021/2/1	pen
3	2021/4/1	book

Table: orders

```
SELECT *
FROM orders
LIMIT 1;
```

Results:

customer_ID	order date	item
1	2021/1/1	book

Get sources from other table (sources)

6. Joins: used to combine rows from 2 or more columns, that column needs to be related to both the tables

Syntax:

```
SELECT *
FROM Table1
JOIN Table2 on Table1.col1 = Table2.col2
```

related columns

Different types of JOINS:

1. Inner Join: returns records that have matching values in both tables

2. Left Join: returns records from the left table (table1), & the matching records from the right table (table2)

- The result is 0 from right side, if there is no match

3. Right Join: returns records from the right table (table 2), & the matching records from the left table (table 1).

- The result is 0 records from the left side, if there is no match.

4. Full Join: returns all records when there is a match in left (table 1) or right (table 2) table records.

Understanding Window Functions

What are window functions?

- computes a metric over groups
- window_functions are allowed only in `select` clause

1. Row numbering: rank each row across specified groups, ordered by specific column

- Rank Function: used to retrieve ranked rows based on the conditions of the ORDER BY clause

When to use RANK?

- when you want to find the name of the highest paying employee at a department.

Table: employee ($K)

id	person_Name	dept	Salary
1	John	HR	$60
2	Mark	Product	$70
3	Judy	SE	$100
4	sandy	HR	$60
5	sam	Payroll	$50

```
SELECT Person_Name
     , dept
     , salary
     , RANK() OVER (ORDER
       BY salary DESC)
       AS Rank
FROM employee;
```

($K)

Person_Name	dept	Salary	Rank
Judy	SE	$100	1
Mark	Product	$70	2
John	HR	$60	3
Sandy	HR	$60	3
Sam	Payroll	$50	5

Result (Analysis)

- Rank ordered by descending order
- If there is a tie between records, RANK will skip the next N-1 position before incrementing the counter.

eg: RANK skipped 4 after 3 and jumped straight to 5

- Therefore, Judy from SE department is the highest paid employee.

RANK() can also be used with PARTITION BY clause.

↗ the rank will be reset for each new partition

Extra:

Syntax:

```
SELECT Person_Name
     , dept
     , salary
     , RANK() OVER (PARTITION BY dept
       ORDER BY salary desc) as RANK
FROM employee;
```

- partition the results by dept column
- each company, the rank will be set as 1.

Table: employee

id	personName	dept	Salary ($K)
1	John	HR	$60
2	Mark	Product	$70
3	Judy	SE	$100
4	sandy	HR	$90
5	sam	Payroll	$50

Person_Name	dept	salary ($K)	Rank
John	HR	$60	1
Sandy	HR	$90	2
Sam	Payroll	$50	1
Mark	Product	$70	1
Judy	SE	$100	1

Result

- Dense_Rank function: similar to RANK() except it doesnt skip any ranks if there is a tie between the ranks of the preceeding records.

Table: employee

id	personName	dept	Salary ($K)
1	John	HR	$60
2	Mark	Product	$70
3	Judy	SE	$100
4	sandy	HR	$90
5	sam	Payroll	$50

Person_Name	dept	salary ($K)	Rank
John	HR	$60	1
Sandy	HR	$90	2
Sam	Payroll	$50	1
Mark	Product	$70	1
Judy	SE	$100	1

Result

```
SELECT Person_Name
      ,dept
      ,salary
      ,DENSE_RANK() OVER(PARTITION BY
                         dept ORDER BY salary
                                   DESC)
                         as RANK
FROM employee;
```

- **Row_Number function:** this function only returns the row number of the sorted records from 1 onwards.

Table: employee

id	person_Name	dept	Salary ($K)
1	John	HR	$60
2	Mark	Product	$70
3	Judy	SE	$100
4	Sandy	HR	$90
5	Sam	Payroll	$50

```
SELECT Person_Name
     , dept
     , salary
     , ROW_NUMBER OVER (ORDER BY
                        Salary desc) as
                                  Rank
FROM employee;
```

Result ↙

Person_Name	dept	Salary ($K)	Rank
Judy	SE	100	1
sandy	HR	90	2
Mark	Product	70	3
John	HR	60	4
Sam	Payroll	50	5

- Partition can also be applied to row_number()

(2) Values: window functions along with types of values

- **FIRST_VALUE (column name):** its a window function that returns the first value in ordered set of values.

Syntax :-

```
SELECT
    person_name
  , dept
  , salary
  First_value (person_name) OVER (
          ORDER BY Salary ) lowest_salary
FROM employees;
```

(Table below)

↓

Table: employee ($K)

id	personName	dept	Salary
1	John	HR	$60
2	Mark	Product	$70
3	Judy	SE	$100
4	Sandy	HR	$90
5	Sam	Payroll	$50

Result:

personName	dept	Salary	lowest_salary
Sam	Payroll	$50	Sam
John	HR	$60	Sam
Mark	Product	$70	Sam
Sandy	HR	$90	Sam
Judy	SE	$100	Sam

```
SELECT
    person_name
  , dept
  , salary
First_value (person_name) OVER( PARTITION BY dept
        ORDER BY Salary ) lowest_salary
FROM employees;
```

Table: employee ($K)

id	personName	dept	Salary
1	John	HR	$60
2	Mark	Product	$70
3	Judy	SE	$100
4	Sandy	HR	$90
5	Sam	Payroll	$50

Result:

personName	dept	Salary ($K)	lowest_salary
John	HR	60	John
Sandy	HR	90	John
Sam	Payroll	50	Sam
Mark	Product	70	Mark
Judy	SE	100	Judy

- LAST_VALUE (column value): its a window function that returns the last value in an ordered set of values

```
SELECT
    person_name
  , dept
  , salary
last_value (person_name) OVER( PARTITION BY dept
        ORDER BY Salary ) lowest_salary
FROM employees;
```

Table: employee

id	personName	dept	Salary ($K)
1	John	HR	$60
2	Mark	Product	$70
3	Judy	SE	$100
4	sandy	HR	$90
5	sam	Payroll	$50

Result:

personName	dept	Salary ($K)	lowest_salary
John	HR	60	sandy
Sandy	HR	90	sandy
sam	Payroll	50	sam
Mark	Product	70	Mark
Judy	SE	100	Judy

- LAG (column name): its a window function that provides access to a row at a specified physical offset which comes before the current row.

Useful for calculating the difference between the current row & previous row.

Table: Salary

id	year	salary
100	2017	3100
101	2017	2600
102	2018	2592
103	2018	6060
104	2018	9540
101	2018	3000
102	2019	5000

```
SELECT id
    ,year
    ,salary
    ,LAG(Salary) OVER(
    PARTITION BY id
    ORDER BY year) previous_
                   salary
FROM salary;
```

id	year	salary	Previous_salary
100	2017	3100	NULL
101	2017	2600	NULL
101	2018	3000	2600
102	2018	2592	NULL
102	2019	5000	2592
103	2018	6060	NULL
104	2018	9540	NULL

- Lead (column_name): its a window function that provides access to a row at a specified physical offset which comes after the current row.

Useful for calculating the difference between the current row & following row.

Table: Hire

Name	Hire_date
John	2007-06-17
Mark	2007-09-17
Judy	2007-09-21
Sandy	2008-01-03
Sam	2009-01-13

```
SELECT name
    , Hire_date
    , LEAD (hire_date, 1)
        OVER (ORDER BY
            hire_date)
        As next_hire
            date
FROM Hire;
```

Result:

Table: Hire

Name	Hire_date	next_hire date
John	2007-06-17	2007-09-17
Mark	2007-09-17	2007-09-21
Judy	2007-09-21	2008-01-03
Sandy	2008-01-03	2009-01-13
Sam	2009-01-13	NULL

- we can do the same with partition, to see the hire dates for each department.

- N^{th} value: window function that allows to get a value from the N^{th} row in an ordered set of rows.
  - can use to find the highest/second/third highest salary.

Table: employee ($K)

id	person_Name	dept	salary
1	John	HR	$60
2	Mark	Product	$70
3	Judy	SE	$100
4	Sandy	HR	$90
5	Sam	Payroll	$50

```
SELECT
    Person_name
  , Salary
  , NTH_VALUE(Person_name,2)
    OVER (ORDER BY Salary desc
    ) as second_highest
FROM employee
```

person_Name	salary	second_highest
Sam	$50	NULL
John	$60	John
Mark	$70	John
Sandy	$90	John
Judy	$100	John

Understanding date functions

1: CURRENT_DATE: returns the current date

2: CURRENT_TIME: returns the current time

3: CURRENT_TIMESTAMP: returns the current time & date

4. convert date to string: CAST() or TO_CHAR() functions converts date to string

5. convert string to date: CAST() or TO_DATE functions to convert a string to a date

6 DATEADD: Add an interval to a date

eg: