



Advanced SQL Topics

By
Vijaya Nandini M



SQL

- Section Overview
 - Timestamps and EXTRACT
 - Math Functions
 - String Functions
 - Sub-query
 - Self-Join



Timestamps and Extract

PART ONE
DISPLAYING CURRENT TIME
INFORMATION



SQL

- In Part One, we will go over a few commands that report back time and date information.
- These will be more useful when creating our own tables and databases, rather than when querying a database.



SQL

- We've already seen that PostgreSQL can hold date and time information:
 - **TIME** - Contains only time
 - **DATE** - Contains only date
 - **TIMESTAMP** - Contains date and time
 - **TIMESTAMPZ** - Contains date, time, and timezone



SQL

- Careful considerations should be made when designing a table and database and choosing a time data type.
- Depending on the situation you may or may not need the full level of `TIMESTAMPTZ`
- Remember, you can always remove historical information, but you can't add it!



SQL

- Let's explore functions and operations related to these specific data types:
 - TIMEZONE
 - NOW
 - TIMEOFDAY
 - CURRENT_TIME
 - CURRENT_DATE



Timestamps and Extract

PART TWO
EXTRACTING TIME AND DATE
INFORMATION



SQL

- Let's explore extracting information from a time based data type using:
 - `EXTRACT()`
 - `AGE()`
 - `TO_CHAR()`



SQL

- `EXTRACT()`
 - Allows you to “extract” or obtain a sub-component of a date value
 - `YEAR`
 - `MONTH`
 - `DAY`
 - `WEEK`
 - `QUARTER`



SQL

- `EXTRACT()`
 - Allows you to “extract” or obtain a sub-component of a date value
 - `EXTRACT(YEAR FROM date_col)`



SQL

- AGE()
 - Calculates and returns the current age given a timestamp
 - Usage:
 - AGE(date_col)
 - Returns
 - 13 years 1 mon 5 days 01:34:13.003423



SQL

- TO_CHAR()
 - General function to convert data types to text
 - Useful for timestamp formatting
 - Usage
 - TO_CHAR(date_col, 'mm-dd-yyyy')



Timestamps and Extract

CHALLENGE TASKS



SQL

- Challenge
 - During which months did payments occur?
 - Format your answer to return back the full month name.



SQL

- Expected Result

1	MARCH
2	MAY
3	FEBRUARY
4	APRIL



SQL

- Hints
 - You do not need to use EXTRACT for this query.



SQL

- Solution
 - SELECT
DISTINCT(TO_CHAR(payment_date,'MONTH'))
FROM payment



SQL

- Challenge
 - How many payments occurred on a Monday?

NOTE: We didn't show you exactly how to do this, but use the documentation or Google to figure this out!



SQL

- Expected Result
 - 2948



SQL

- Hints
 - Use EXTRACT
 - Review the **dow** keyword
 - PostgreSQL considers Sunday the start of a week (indexed at 0)



SQL

- Solution
 - `SELECT COUNT(*)
FROM payment
WHERE EXTRACT(dow FROM payment_date) =
1`



SubQuery



SQL

- A sub query allows you to construct complex queries, essentially performing a query on the results of another query.
- The syntax is straightforward and involves two SELECT statements.



SQL

- Let's imagine a table consisting of student names and their test scores



SQL

- Standard Query
 - `SELECT student,grade`
`FROM test_scores`



SQL

- Standard Query to return average grade
 - `SELECT AVG(grade)`
`FROM test_scores`



SQL

- *How can we get a list of students who scored better than the average grade?*
 - `SELECT AVG(grade)`
`FROM test_scores`



SQL

- *It looks like we need two steps, first get the average grade, then compare the rest of the table against it.*
 - `SELECT AVG(grade)`
`FROM test_scores`



SQL

- *This is where a subquery can help us get the result in a “single” query request*
 - `SELECT student, grade
FROM test_scores
WHERE grade > (SELECT AVG(grade)
FROM test_scores)`



SQL

- *This is where a subquery can help us get the result in a “single” query request*
 - `SELECT student, grade
FROM test_scores
WHERE grade > (SELECT AVG(grade)
FROM test_scores)`



SQL

- *This is where a subquery can help us get the result in a “single” query request*
 - `SELECT student, grade`
`FROM test_scores`
`WHERE grade > (70)`



SQL

- *This is where a subquery can help us get the result in a “single” query request*
 - `SELECT student,grade
FROM test_scores
WHERE grade > (SELECT AVG(grade)
FROM test_scores)`



SQL

- The subquery is performed first since it is inside the parenthesis.
- We can also use the IN operator in conjunction with a subquery to check against multiple results returned.



SQL

- *A subquery can operate on a separate table:*
 - `SELECT student, grade
FROM test_scores
WHERE student IN
(SELECT student
FROM honor_roll_table)`



SQL

- *A subquery can operate on a separate table:*
 - `SELECT student, grade
FROM test_scores
WHERE student IN
(('Zach' , 'Chris' , 'Karissa'))`



SQL

- *A subquery can operate on a separate table:*
 - `SELECT student, grade
FROM test_scores
WHERE student IN
(SELECT student
FROM honor_roll_table)`



SQL

- The EXISTS operator is used to test for existence of rows in a subquery.
- Typically a subquery is passed in the EXISTS() function to check if any rows are returned with the subquery.



SQL

- Typical Syntax

```
SELECT column_name  
FROM table_name  
WHERE EXISTS  
(SELECT column_name FROM  
table_name WHERE condition);
```



Self-Join



SQL

- A self-join is a query in which a table is joined to itself.
- Self-joins are useful for comparing values in a column of rows within the same table.



SQL

- The self join can be viewed as a join of two copies of the same table.
- The table is not actually copied, but SQL performs the command as though it were.
- There is no special keyword for a self join, its simply standard JOIN syntax with the same table in both parts.



SQL

- However, when using a self join it is necessary to use an alias for the table, otherwise the table names would be ambiguous.
- Let's see a syntax example of this.



SQL

- Syntax
 - `SELECT tableA.col, tableB.col`
`FROM table AS tableA`
`JOIN table AS tableB ON`
`tableA.some_col = tableB.other_col`



SQL

- Syntax
 - SELECT tableA.col, tableB.col
FROM table AS tableA
JOIN table AS tableB ON
tableA.some_col = tableB.other_col



SQL

- Syntax
 - `SELECT tableA.col, tableB.col`
`FROM table AS tableA`
`JOIN table AS tableB ON`
`tableA.some_col = tableB.other_col`



SQL

- Syntax
 - `SELECT tableA.col, tableB.col`
`FROM table AS tableA`
`JOIN table AS tableB ON`
`tableA.some_col = tableB.other_col`



SQL

- Let's explore a more realistic situation of when you would use this.



SQL

- Let's explore a more realistic situation of when you would use this.

EMPLOYEES		
emp_id	name	report
1	Andrew	3
2	Bob	3
3	Charlie	4
4	David	1



SQL

- Each employee sends reports to another employee.

EMPLOYEES		
emp_id	name	report_id
1	Andrew	3
2	Bob	3
3	Charlie	4
4	David	1



SQL

- We want results showing the employee name and their reports recipient name

EMPLOYEES		
emp_id	name	report_id
1	Andrew	3
2	Bob	3
3	Charlie	4
4	David	1



name	rep
Andrew	Charlie
Bob	Charlie
Charlie	David
David	Andrew



SQL

- Syntax
 - `SELECT tableA.col, tableB.col
FROM table AS tableA
JOIN table AS tableB ON
tableA.some_col = tableB.other_col`



SQL

- Syntax
 - `SELECT tableA.col, tableB.col`
`FROM table AS tableA`
`JOIN table AS tableB ON`
`tableA.some_col = tableB.other_col`



SQL

- Syntax
 - `SELECT tableA.col, tableB.col`
`FROM table AS tableA`
`JOIN table AS tableB ON`
`tableA.some_col = tableB.other_col`



SQL

- SELECT tableA.col, tableB.col
FROM table AS tableA
JOIN table AS tableB ON
tableA.some_col = tableB.other_col

EMPLOYEES		
emp_id	name	report_id
1	Andrew	3
...
3	Charlie	4
4	David	1



SQL

- Syntax
 - `SELECT tableA.col, tableB.col
FROM employees AS tableA
JOIN employees AS tableB ON
tableA.some_col = tableB.other_col`



SQL

- Syntax
 - SELECT **tableA.col**, tableB.col
FROM employees AS **tableA**
JOIN employees AS tableB ON
tableA.some_col = tableB.other_col



SQL

- SELECT **tableA.col**, tableB.col
FROM employees AS **tableA**
JOIN employees AS tableB ON
tableA.some_col = tableB.other_col

EMPLOYEES		
emp_id	name	report_id
1	Andrew	3
...
3	Charlie	4
4	David	1



SQL

- Syntax
 - SELECT **emp**.col, tableB.col
FROM employees AS **emp**
JOIN employees AS tableB ON
emp.some_col = tableB.other_col



SQL

- Syntax
 - `SELECT emp.col, tableB.col`
`FROM employees AS emp`
`JOIN employees AS tableB ON`
`emp.some_col = tableB.other_col`



SQL

- `SELECT emp.col, tableB.col`
`FROM employees AS emp`
`JOIN employees AS tableB ON`
`emp.some_col = tableB.other_col`

EMPLOYEES		
emp_id	name	report_id
1	Andrew	3
...
3	Charlie	4
4	David	1



SQL

- Syntax
 - `SELECT emp.col, report.col`
`FROM employees AS emp`
`JOIN employees AS report ON`
`emp.some_col = report.other_col`



SQL

- Syntax
 - `SELECT emp.col, report.col`
`FROM employees AS emp`
`JOIN employees AS report ON`
`emp.some_col = report.other_col`



SQL

- Syntax
 - `SELECT emp.col, report.col
FROM employees AS emp
JOIN employees AS report ON
emp.emp_id = report.report_id`



SQL

- Syntax
 - `SELECT emp.col, report.col`
`FROM employees AS emp`
`JOIN employees AS report ON`
`emp.emp_id = report.report_id`



SQL

- Syntax
 - `SELECT emp.name, report.name`
`FROM employees AS emp`
`JOIN employees AS report ON`
`emp.emp_id = report.report_id`



SQL

- Syntax
 - **SELECT** emp.name, report.name
FROM employees **AS** emp
JOIN employees **AS** report **ON**
emp.emp_id = report.report_id



SQL

- Syntax
 - **SELECT** emp.name, report.name **AS** rep
FROM employees **AS** emp
JOIN employees **AS** report **ON**
emp.emp_id = report.report_id



SQL

- We want results showing the employee name and their reports recipient name

name	rep
Andrew	Charlie
Bob	Charlie
Charlie	David
David	Andrew