



# Advanced SQL Topics



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



# Timestamps and Extract

PART ONE

DISPLAYING CURRENT TIME INFORMATION



- 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.



- 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
  - **TIMESTAMPTZ** - Contains date,time, and timezone



- 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!



- 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





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



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



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



- 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



- TO\_CHAR()
  - General function to convert data types to text
  - Useful for timestamp formatting
  - Usage
    - TO\_CHAR(date\_col, 'mm-dd-yyyy')



- All of these functions are best understood through example, so let's jump to pgadmin and work with these functions!



# Timestamps and Extract

CHALLENGE TASKS



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





- Expected Result

1	MARCH
2	MAY
3	FEBRUARY
4	APRIL



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



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



- 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!*



- Expected Result
  - 2948



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



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



# Mathematical Functions





- Let's quickly explore some mathematical operations we can perform with SQL!
- This is best shown through examples and the documentation, so we'll jump straight to pgAdmin.



# String Functions and Operations



- PostgreSQL also provides a variety of string functions and operators that allow us to edit, combine, and alter text data columns.
- Let's explore the documentation to see what is available for us!



# String Functions and Operations

QUICK CHALLENGE TASK



# SubQuery



- In this lecture we will we discuss how to perform a subquery as well as the EXISTS function.



- 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.



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





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



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



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



- *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`



- *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)`



- *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)`



- *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)`



- *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)`





- 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.



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



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



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



- 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.



- Typical Syntax

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



- Subqueries and EXISTS are best learned through example, so let's jump to pgAdmin!



# Self-Join





- 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.



- 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.



- 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.



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



- Syntax
  - SELECT tableA.col, tableB.col  
FROM table AS tableA  
JOIN table AS tableB ON  
tableA.some\_col = tableB.other\_col



- Syntax
  - SELECT tableA.col, tableB.col  
FROM table AS tableA  
JOIN table AS tableB ON  
tableA.some\_col = tableB.other\_col



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



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





- 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



- 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

- 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



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



- Syntax
  - SELECT tableA.col, tableB.col  
FROM table AS tableA  
JOIN table AS tableB ON  
tableA.some\_col = tableB.other\_col



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



- Syntax
  - `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
...	...	...



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





- Syntax
  - SELECT **tableA.col**, tableB.col  
FROM employees AS **tableA**  
JOIN employees AS tableB ON  
**tableA.some\_col** = tableB.other\_col



- Syntax
  - 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
...	...	...



- 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`



- Syntax
  - 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
...	...	...



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



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



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





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



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



- Syntax
  - **SELECT** emp.name, report.name  
**FROM** employees **AS** emp  
**JOIN** employees **AS** report **ON**  
emp.emp\_id = report.report\_id



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



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

name	rep
Andrew	Charlie
Bob	Charlie
Charlie	David
David	Andrew



- Let's explore an example on our dvdrental database in pgAdmin!