



Advanced SQL Commands Lecture

- 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.

- ในส่วนที่หนึ่ง เราจะพูดถึงคำสั่งสองสามคำสั่งที่รายงานข้อมูลเวลาและวันที่ย้อนกลับ
- สิ่งเหล่านี้จะมีประโยชน์มากกว่าเมื่อสร้างตารางและฐานข้อมูลของเราเอง แทนที่จะใช้ 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

- เราได้เห็นแล้วว่า PostgreSQL สามารถเก็บข้อมูลวันที่และเวลาได้:
 - TIME - มีเวลาเท่านั้น
 - DATE - มีเฉพาะวันที่เท่านั้น
 - TIMESTAMP - ประกอบด้วยวันที่และเวลา
 - TIMESTAMPTZ - ประกอบด้วยวันที่ เวลา และเขตเวลา

- 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!
- ควรพิจารณาอย่างรอบคอบเมื่อออกแบบตารางและฐานข้อมูลและเลือก time data type
 - คุณอาจต้องการระดับ TIMESTAMPTZ แบบเต็มหรือไม่ก็ได้ ทั้งนี้ขึ้นอยู่กับสถานการณ์
 - โปรดจำไว้ว่า คุณสามารถลบข้อมูลประวัติได้เสมอ แต่คุณไม่สามารถเพิ่มได้!
- Let's explore functions and operations related to these specific data types:
 - TIMEZONE
 - NOW
 - TIMEOFDAY
 - CURRENT_TIME
 - CURRENT_DATE

```
SHOW TIMEZONE --> แสดงตำแหน่ง
SELECT NOW() --> แสดงเวลา ตามมาตรฐาน GMT (Greenwich Mean Time)
SELECT TIMEOFDAY() --> บอกวันเดือนปีเวลา
SELECT CURRENT_TIME --> เวลาปัจจุบัน
SELECT CURRENT_DATE --> วันเดือนปีปัจจุบัน
```

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')`

```
#EXTRACT ดึงข้อมูล
SELECT EXTRACT(YEAR FROM payment_date) AS year
FROM payment; --> MONTH , DAY , WEEK , QUARTER

#AGE การบอกอายุของข้อมูลจนถึงปัจจุบัน
SELECT AGE(payment_date)
FROM payment;
```

```
#TO_CHAR
SELECT TO_CHAR(payment_date, 'MONTH-YYYY')
FROM payment;

SELECT TO_CHAR(payment_date, 'MON/dd/YYYY')
FROM payment;
```

- เพิ่มเติม `TO_CHAR`

9.8. Data Type Formatting Functions

9.8. Data Type Formatting Functions The PostgreSQL formatting functions provide a powerful set of tools for converting various data types (date/time, ...

 <https://www.postgresql.org/docs/12/functions-formatting.html>



Timestamps and Extract

CHALLENGE TASKS

CHALLENGE

- การชำระเงินเกิดขึ้นในช่วงเดือนใด
- จัดรูปแบบคำตอบของคุณเพื่อส่งคืนชื่อเต็มเดือน

```
SELECT distinct(TO_CHAR(payment_date, 'MONTH')) AS MONTH
FROM payment;
```

Data Output		Messages	Notifications
	month text		
1	MARCH		
2	MAY		
3	FEBRUARY		
4	APRIL		

- จำนวนการชำระเงินที่เกิดขึ้นในวันจันทร์?
- หมายเหตุ: เราไม่ได้แสดงวิธีการทำเช่นนี้ให้คุณเห็นอย่างชัดเจน แต่ใช้เอกสารประกอบหรือ Google เพื่อทำความเข้าใจ!

```
#วิธีผม
SELECT COUNT(*)
FROM payment
WHERE TO_CHAR(payment_date, 'day') LIKE 'm%';
--> 2948

#solution
SELECT COUNT(*)
FROM payment
WHERE EXTRACT(dow FROM payment_date) = 1
--> 2948
```

Mathematical Functions


- มาสำรวจการดำเนินการทางคณิตศาสตร์ที่เราสามารถทำได้ด้วย SQL กันอย่างรวดเร็ว!

Table 9-2. Mathematical Operators

Operator	Description	Example	Result
+	addition	2 + 3	5
-	subtraction	2 - 3	-1
*	multiplication	2 * 3	6
/	division (integer division truncates the result)	4 / 2	2
%	modulo (remainder)	5 % 4	1
^	exponentiation (associates left to right)	2.0 ^ 3.0	8
/	square root	/ 25.0	5
/	cube root	/ 27.0	3
!	factorial (deprecated, use factorial() instead)	5 !	120
!!	factorial as a prefix operator (deprecated, use factorial() instead)	!! 5	120
@	absolute value	@ -5.0	5
&	bitwise AND	91 & 15	11
	bitwise OR	32 3	35
#	bitwise XOR	17 # 5	20
~	bitwise NOT	~1	-2
<<	bitwise shift left	1 << 4	16
>>	bitwise shift right	8 >> 2	2

- เพิ่มเติมน Mathematical Functions and Operators

Mathematical Functions and Operators

 <https://www.postgresql.org/docs/9.5/functions-math.html>



```
SELECT ROUND(rental_rate/replacement_cost,2)*100 FROM film;
```


String Functions and Operations

- นอกจากนี้ PostgreSQL ยังมีฟังก์ชันสตริงและโอเปอเรเตอร์ที่หลากหลายซึ่งช่วยให้เราแก้ไขรวม และแก้ไขคอลัมน์ข้อมูลข้อความได้

Table 9-6. SQL String Functions and Operators

Function	Return Type	Description	Example	Result
string string	text	String concatenation	'Post' 'greSQL'	PostgreSQL
string non-string or non-string string	text	String concatenation with one non-string input	'Value: ' 42	Value: 42
bit_length(string)	int	Number of bits in string	bit_length('jose')	32
char_length(string) or character_length(string)	int	Number of characters in string	char_length('jose')	4
lower(string)	text	Convert string to lower case	lower('TOM')	tom
octet_length(string)	int	Number of bytes in string	octet_length('jose')	4
overlay(string placing string from int [(for int)])	text	Replace substring	overlay('Txxxxx' placing 'hom' from 2 for 4)	Thomas
position(substring in string)	int	Location of specified substring	position('om' in 'Thomas')	3
substring(string [(from int) [(for int)])]	text	Extract substring	substring('Thomas' from 2 for 3)	hom
substring(string from <i>pattern</i>)	text	Extract substring matching POSIX regular expression. See Section 9.7 for more information on pattern matching.	substring('Thomas' from '...\$')	mas
substring(string from <i>pattern</i> for <i>escape</i>)	text	Extract substring matching SQL regular expression. See Section 9.7 for more information on pattern matching.	substring('Thomas' from '%#*o_a#*' for '#')	oma
trim([leading trailing both] [characters] from string)	text	Remove the longest string containing only the characters (a space by default) from the start/end/both ends of the string	trim(both 'x' from 'xTOMxx')	Tom
upper(string)	text	Convert string to upper case	upper('tom')	TOM

- เพิ่มเติม String Functions and Operations

String Functions and Operators

 <https://www.postgresql.org/docs/9.1/functions-string.html>



```
#LENGTH หาความยาวของ str
SELECT LENGTH(first_name) FROM customer;

# || เชื่อม concat
SELECT upper(first_name) || ' ' || upper(last_name) AS full_name FROM customer;
```

SubQuery

- 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.

- subQuery ช่วยให้คุณสร้าง complex queries โดยทำแบบสอบถามจากผลลัพธ์ของข้อความค้นหาอื่นเป็นหลัก
- ไวยากรณ์นั้นตรงไปตรงมาและเกี่ยวข้องกับคำสั่ง SELECT สองคำสั่ง
- ลองนึกภาพตารางที่มีชื่อนักเรียนและคะแนนสอบ

- Standard Query
 - SELECT student,grade
FROM test_scores

- Standard Query
 - `SELECT student, grade`
`FROM test_scores`
- Standard Query to return average grade
 - `SELECT AVG(grade)`
`FROM test_scores`
- เราจะเอารายชื่อนักเรียนที่ทำคะแนนได้ดีกว่าเกรดเฉลี่ยได้อย่างไร?
- ดูเหมือนว่าเราต้องการสองขั้นตอน ก่อนอื่นให้หาเกรดเฉลี่ย จากนั้นเปรียบเทียบส่วนที่เหลือของตารางกับขั้นตอนนั้น
- *How can we get a list of students who scored better than the average 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)`

- subquery จะดำเนินการก่อนเนื่องจากอยู่ในวงเล็บ
- นอกจากนี้ เรายังสามารถใช้ตัวดำเนินการ IN ร่วมกับเคียวรีย่อยเพื่อตรวจสอบกับผลลัพธ์หลายรายการที่ส่งคืน

```
#SubQuery
SELECT title, rental_rate
FROM film
WHERE rental_rate >
(SELECT AVG(rental_rate) FROM film);

SELECT film_id, title
FROM film
WHERE film_id IN
(SELECT inventory.film_id
FROM rental
INNER JOIN inventory ON inventory.inventory_id = rental.inventory_id
WHERE return_date BETWEEN '2005-05-29' AND '2005-05-30')
ORDER BY film_id
;
```

- *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)
- ตัวดำเนินการ EXISTS ใช้เพื่อทดสอบการมีอยู่ของแถวใน subquery

- โดยทั่วไป Subquery จะถูกส่งผ่านในฟังก์ชัน EXISTS() เพื่อตรวจสอบว่าแถวใดถูกส่งกลับมาพร้อมกับ subquery

- Typical Syntax

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

```
SELECT first_name, last_name  
FROM customer AS c  
WHERE EXISTS  
(SELECT * FROM payment as p  
WHERE p.customer_id = c.customer_id  
AND amount > 11)
```

```
SELECT first_name, last_name  
FROM customer AS c  
WHERE NOT EXISTS  
(SELECT * FROM payment as p  
WHERE p.customer_id = c.customer_id  
AND amount > 11)
```

Self-Join

- self-join เป็นการ join กับตัวเอง
- มีประโยชน์สำหรับการเปรียบเทียบค่าในคอลัมน์ของแถวภายในตารางเดียวกัน
- self-join สามารถใช้ได้เป็นการรวมสองสำเนาของตารางเดียวกัน
- ตารางไม่ได้ถูกคัดลอกจริง ๆ แต่ SQL ดำเนินการคำสั่งเหมือนจริง
- ไม่มีคีย์เวิร์ดพิเศษสำหรับ self-join , JOIN syntax มาตรฐานที่เรียบง่ายพร้อมตารางเดียวกันในทั้งสองส่วน
- อย่างไรก็ตาม เมื่อใช้การรวมตัวเอง จำเป็นต้องใช้นามแฝง alias สำหรับตาราง มิฉะนั้น ชื่อตารางจะคลุมเครือ

● 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.

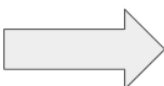
EMPLOYEES		
emp_id	name	report
1	Andrew	3
2	Bob	3
3	Charlie	4
4	David	1

- 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

PIERIAN  DATA

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

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

PIERIAN  DATA

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

- 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

```
SELECT f1.title, f2.title, f1.length
FROM film AS f1
INNER JOIN film AS f2 ON
f1.film_id != f2.film_id
AND f1.length = f2.length;
```