



เพิ่มเติม

INSERT

```
INSERT INTO table_name (column1, column2, column3, ...)  
VALUES (value1, value2, value3, ...);
```

```
INSERT INTO table_name  
VALUES (value1, value2, value3, ...);
```

- INSERT INTO account (account_number, branch_name, balance)

VALUES

('1', 'B', 100.00),
(‘2’, ‘A’, 50.00);

เลือกบาง attributes เท่านั้น

- INSERT INTO account

VALUES

('1', 'B', 100.00),
(‘2’, ‘A’, 50.00);

เลือกทุก attributes เลย ใน
กรณี account มีทั้งสิ้น 3
attributes เท่านั้น

DELETE

`DELETE FROM table_name WHERE condition;`

- `DELETE FROM account WHERE account_number='1';`

UPDATE

```
UPDATE table_name
SET column1 = value1, column2 = value2, ...
WHERE condition;
```

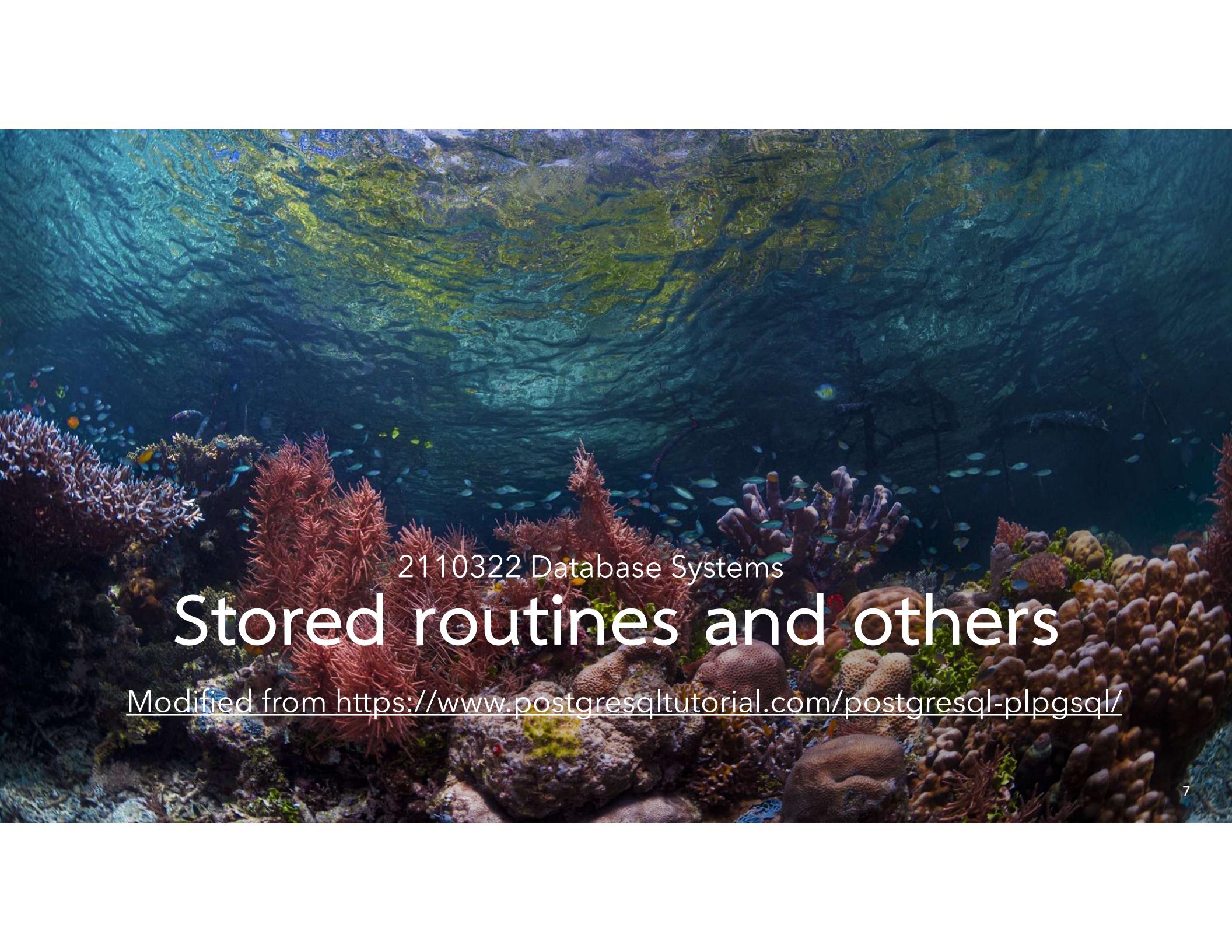
- UPDATE account
SET balance=500.00
WHERE account_number = '1';



ใช้ pgadmin ช่วยออกแบบ DML ได้
บ้าง

อ.วิวัฒน์ สอนอะไรมบ้าง ภายใน 6 ชม.

- Docker containers → ไม่รู้จัก แต่ใช้ตามที่มีให้
- PGAdmin Tool → รู้จักและใช้งานเฉพาะส่วนการทดสอบ SQL editor
- PSQL Tool → ไม่ได้สอน
- **SQL** + stored routines → สอนให้พอใช้งานได้แบบ fast track มีต้นแบบรูปแบบคำสั่งให้ดู SELECT, INSERT, UPDATE, DELETE, ..., FUNCTIONS, PROCEDURES.
- Database administration → ไม่ได้สอน แต่แนะนำเล็กน้อย
- Database performance tuning → ไม่ได้สอน
- ดังนั้นมีอีกหลายอย่างที่ควรรู้ก่อนไปทำงานจริง ซึ่งต้องฝึกฝน ครรصنโน๊กติดต่อขอคำแนะนำ อ. นอกรอบ (ปกติใช้เวลา 30 ชม. ต่อหัวข้อหลัก)

A vibrant underwater photograph of a coral reef. The foreground is filled with various types of coral, including large, branching structures and smaller, rounded forms. Sunlight filters down from the surface in bright rays, creating a dappled light effect on the reef. Numerous small, colorful fish, ranging from yellow to blue and green, swim gracefully through the water. The background is a deep, dark blue, suggesting the depth of the ocean.

2110322 Database Systems

Stored routines and others

Modified from <https://www.postgresqltutorial.com/postgresql-plpgsql/>

หัวข้อที่จะอธิบายประกอบด้วย

- Stored routines
 - Stored functions
 - Stored procedure
 - Triggers

- View
- Index
- EXPLAIN, EXPLAIN ANALYZE
- Transaction
- Database backup and restore via pgAdmin4

ແຄນໄຟ້

Stored routines

SQL command เขียนได้ทีละคำสั่งแล้วจบผลลัพธ์
ออกมานะเป็นตารางเท่านั้น ไม่ต่อเนื่อง อ้างถึงอีกครั้ง ไม่ได้

- Stored routines (SR) เป็น subroutine ที่แอปพลิเคชันสามารถใช้ในการเข้าถึงฐานข้อมูลแบบ relational (RDBMS)/object-relational (ORDBMS) database management system
- Stored routines ใน PostgreSQL เขียนได้หลายภาษาโดยในบทเรียนนี้จะใช้ภาษา PL/pgSQL (procedural language for postgresql)
- PL/pgSQL เป็น block-structured language สามารถใช้ห่อหุ้มชุดคำสั่ง SQL ในรูปแบบ ออบเจกต์เพื่อเก็บไว้ใน database server
- สามารถเรียกใช้งานชุดคำสั่งผ่าน Stored routines ได้โดยไม่ต้องเขียนชุดคำสั่งใหม่ทุกครั้ง

ประโยชน์ของ Stored routines



มีประโยชน์สำหรับแอปพลิเคชันที่ถูกพัฒนาโดยใช้ภาษาหรือแพลตฟอร์มที่ต่างกัน
แต่ต้องการเข้าถึงฐานข้อมูลโดยใช้คำสั่งชุดเดียวกัน



มีประโยชน์ในเรื่อง security อย่างระบบธนาคารเลือกจะใช้ stored procedures และ stored functions แทนการใช้เรียกใช้ชุด operations ที่ใช้ในการเข้าถึงฐานข้อมูลโดยตรง ทำให้
สามารถควบคุมการเข้าถึงข้อมูลได้ดีขึ้น



ลด traffic และ round-trip time ระหว่างแอปพลิเคชันกับ database server
และทำให้ไม่ต้องส่งข้อมูลที่ยังไม่ใช้ข้อมูลสุดท้ายกลับไปมา

ข้อจำกัดของ Stored routines



การพัฒนาซอฟต์แวร์อาจช้าลง เนื่องจากผู้พัฒนาซอฟต์แวร์ส่วนใหญ่ขาดทักษะการใช้ PL/pgSQL



การจัดการเวอร์ชันและการดีบักทำได้ยาก



อาจไม่สามารถนำไปใช้ในระบบจัดการฐานข้อมูลอื่นได้

Stored routines หลักในบทเรียนนี้

มี 3 ประเภทประกอบด้วย

1. Stored Function
2. Stored procedure
3. Trigger

A close-up, underwater photograph of a green sea turtle. The turtle's head and front flippers are visible, moving gracefully through clear blue water. Sunlight filters down from the surface, creating bright highlights on its scaly skin and illuminating the sandy ocean floor below.

PL/pgSQL Overview

Modified from <https://www.postgresqltutorial.com/postgresql-plpgsql/>

Block-structured language

```
[ <<label>> ]  
[ declare  
    declarations ]  
begin  
    statements;  
    ...  
end [ label ];
```

Optional

Must end with semicolon after the END keyword

แต่ละ block ประกอบด้วยสองส่วน

declarations (optional)

body (required)*

Block-structured language (Example)

```
do $$  
  <<first_block>>  
  declare  
    account_count integer := 0;  
  
  begin  
    -- get the number of accounts  
    select count(*)  
      into account_count  
      from account;  
    -- display a message  
    raise notice 'The number of accounts is %', account_count;  
  end first_block $$;
```

branch

| Branch_name | Branch_city | assets |
|-------------|-------------|----------|
| A | Riverside | \$10,000 |
| B | LA | \$20,000 |
| C | Long Beach | \$15,000 |
| D | Irvine | \$12,000 |
| E | Pomona | \$7,000 |
| F | San Jose | \$18,000 |

depositor

| Customer_name | Account_number |
|---------------|----------------|
| Joe | 1 |
| Joe | 2 |
| Mary | 2 |
| Keith | 4 |
| Mike | 5 |
| Keith | 6 |
| Joe | 3 |

account

| Account_number | Branch_name | balance |
|----------------|-------------|---------|
| 1 | B | \$100 |
| 2 | A | \$50 |
| 3 | A | \$30 |
| 4 | F | \$120 |
| 5 | A | \$500 |
| 6 | B | \$324 |

customer

| Customer_name | Customer_street | Customer_only |
|---------------|-----------------|---------------|
| Joe | Joe_street | Y |
| Alan | Mary_street | Y |
| Jason | Jason_street | N |
| Mary | Mary_street | N |
| Mike | Mary_street | Y |
| Keith | Keith_street | N |

borrower

| Customer_name | Loan_number |
|---------------|-------------|
| Joe | 1 |
| Jason | 2 |
| Joe | 3 |
| Keith | 4 |
| Mary | 5 |
| Joe | 6 |

loan

| Loan_number | Branch_name | Amount |
|-------------|-------------|--------|
| 1 | B | \$100 |
| 2 | E | \$27 |
| 3 | F | \$543 |
| 4 | A | \$129 |
| 5 | A | \$26 |
| 6 | B | \$67 |

Block-structured language (Example)

```
do $$  
  <<first_block>>  
  declare  
    account_count integer := 0;
```

```
begin  
  -- get the number of accounts  
  select count(*)  
  into account_count  
  from account;  
  -- display a message  
  raise notice 'The number of accounts is %', account_count;  
end first_block $$;
```

ชื่อตัวแปร ประเภทตัวแปร expression

| account | | |
|----------------|-------------|---------|
| Account_number | Branch_name | balance |
| 1 | B | \$100 |
| 2 | A | \$50 |
| 3 | A | \$30 |
| 4 | F | \$120 |
| 5 | A | \$500 |
| 6 | B | \$324 |

NOTICE: The number of accounts is 7

ไป download ตัวอย่าง **StoredProcedure.txt**
ใน MCV

Block-structured language (Example)

The screenshot shows the pgAdmin Query Tool interface. The title bar indicates the connection is to 'banking/postgres@PostgreSQL 16'. The toolbar has various icons for database management. A pink arrow points to the 'Query History' tab, which is currently selected. Below it, the main query editor window contains the following PL/pgSQL code:

```
1 do $$  
2 <<first_block>>  
3 declare  
4   account_count integer := 0;  
5 begin  
6   -- get the number of accounts  
7   select count(*)  
8     into account_count  
9    from account;  
10  -- display a message  
11  raise notice 'The number of accounts is %', account_count;  
12 end first_block $$;
```

To the right of the code, a yellow box contains the text: 'ลองรัน block นี้ใน pgAdmin ในหน้า Query Tool' (Run this block in pgAdmin in the Query Tool). The bottom pane shows the 'Messages' tab with the output:

NOTICE: The number of accounts is 7
DO
Query returned successfully in 50 msec.

การกำหนดตัวแปรใน PL/pgSQL (row types)

```
do $$  
  <<first_block>>    | ชื่อตัวแปร          | ประเภทตัวแปร  
declare  
  selected_account account%rowtype;  
begin  
  -- get the branch_name and balance of account number 4  
  select *  
  into selected_account  
  from account  
  where account_number = '4';  
  -- display a message  
  raise notice 'The branch_name and balance is %, %',  
    selected_account.branch_name,  
    selected_account.balance;  
end first_block $$;
```

NOTICE: The branch_name and balance is F, 120.00

rowtype หมายถึง data type ของ selected_account
จะเหมือนกับ row ในตาราง account

| account | | |
|----------------|-------------|---------|
| Account_number | Branch_name | balance |
| 1 | B | \$100 |
| 2 | A | \$50 |
| 3 | A | \$30 |
| 4 | F | \$120 |
| 5 | A | \$500 |
| 6 | B | \$324 |

การกำหนดตัวแปรใน PL/pgSQL (record type)

```
do $$  
  <<first_block>>  
declare  
  selected_accounts record;  
begin  
  -- get all accounts with the balance >= 100  
  for selected_accounts in select account_number, branch_name, balance  
    from account  
    where balance >= 100  
    order by balance  
  loop  
    -- display a message  
    raise notice 'The account_number branch_name and balance is %, %, %',  
      selected_accounts.branch_name,  
      selected_accounts.balance,  
      selected_accounts.balance;  
  end loop;  
end first_block $$;
```

ชื่อตัวแปร

ประเภทตัวแปร

record เหมือน place holder เพื่อเก็บผล row หนึ่งๆ ของการ query

เป็นส่วนของการระบุ iteration ของ for loop

การกำหนดตัวแปรใน PL/pgSQL (record type)

```
NOTICE: The account_number branch_name and balance is B, 100.00, 100.00
NOTICE: The account_number branch_name and balance is F, 120.00, 120.00
NOTICE: The account_number branch_name and balance is X, 222.00, 222.00
NOTICE: The account_number branch_name and balance is B, 324.00, 324.00
NOTICE: The account_number branch_name and balance is A, 500.00, 500.00
```

การกำหนดตัวแปรใน PL/pgSQL (constant)

```
do $$  
  <<first_block>>  
declare  
    start_at constant time = now();  
begin  
    -- display a message  
    raise notice 'The current time is %', start_at;  
end first_block $$;
```

กำหนดค่าแล้ว เปลี่ยนค่าไม่ได้ระหว่าง execution

NOTICE: The current time is 21:23:29.844439

การกำหนดตัวแปรใน PL/pgSQL (assert statement)

```
do $$  
  <<first_block>>  
  declare  
    account_count integer := 0;  
  begin  
    -- get the number of accounts  
    select count(*)  
      into account_count  
     from account;  
    -- alert a message when the assert condition is false or null  
    assert account_count > 1000, 'Test assert';  
  end first_block $$;
```

assert statement ใช้ในการ debug เป็นหลัก สำหรับการ report error ให้ใช้ raise

```
ERROR: Test assert  
CONTEXT: PL/pgSQL function inline_code_block line 11 at ASSERT  
SQL state: P0004
```

การตรวจสอบเงื่อนไขใน PL/pgSQL (if-then statement)

```
do $$  
  <<first_block>>  
declare  
  selected_account account%rowtype;  
  input_account_number  account.account_number%type := 0;  
begin  
  -- get the account with specific account number  
  select * from account  
  into selected_account  
  where account_number = input_account_number;  
  
  if not found then  
    raise notice 'The account number % could not be found', input_account_number;  
  end if;  
end first_block $$;
```

Global variable in PL/pgSQL
procedure language, the select
into will update the found variable
to True if record is found.

NOTICE: The account number 0 could not be found

การตรวจสอบเงื่อนไขใน PL/pgSQL (if-then-else statement)

```
do $$  
<<first_block>>  
declare  
    selected_account account%rowtype;  
    input_account_number account.account_number%type := 6;  
begin  
    -- get the account with specific account number  
    select * from account  
    into selected_account  
    where account_number = input_account_number;  
  
    if not found then  
        raise notice 'The account number % could not be found',  
            input_account_number;  
    else  
        raise notice 'The branch_name and balance is % and %',  
            selected_account.branch_name,  
            selected_account.balance;  
    end if;  
end first_block $$;
```

NOTICE: The branch_name and balance is B and 324.00

ต่อจากหน้าที่แล้วมี else เข้ามา

```

do $$

<<first_block>>
declare
selected_account account%rowtype;
input_account_number account.account_number%type := 6;
account_level varchar(100);
begin
select * from account
into selected_account
where account_number = input_account_number;
if not found then
    raise notice 'The account number % could not be found',
    input_account_number;
else
    if selected_account.balance > 0 and selected_account.balance <= 200 then
        account_level := 'Standard';
    elsif selected_account.balance > 200 and selected_account.balance <= 400 then
        account_level := 'Silver';
    elsif selected_account.balance > 400 and selected_account.balance <= 600 then
        account_level := 'Gold';
    else
        account_level := 'Platinum';
    end if;
    raise notice 'The account number % is in level %',
    selected_account.branch_name,
    account_level;
end if;
end first_block $$;

```

การตรวจสอบเงื่อนไขใน PL/pgSQL (if-then-elsif statement)

ต่อจากหน้าที่แล้วมีการเช็ค account balance เพื่อกำหนด level ของ account

NOTICE: The account number B is in level Silver

วั่งวน (while loop) ใน PL/pgSQL

```
do $$  
declare  
    counter integer := 0;  
begin  
    while counter < 5 loop  
        raise notice 'Counter %', counter;  
        counter := counter + 1;  
    end loop;  
end$$;
```

```
NOTICE: Counter 0  
NOTICE: Counter 1  
NOTICE: Counter 2  
NOTICE: Counter 3  
NOTICE: Counter 4
```

วัյวน (for loop) ใน PL/pgSQL

```
do $$  
begin  
    for counter in 1..5 loop  
        raise notice 'counter: %', counter;  
    end loop;  
end; $$
```

วัյวน (for loop) ใน PL/pgSQL (จาก slide หน้าแรกๆ)

```
do $$  
  <<first_block>>  
declare  
    selected_accounts record;  
begin  
    -- get all accounts with the balance >= 100  
    for selected_accounts in select account_number, branch_name, balance  
        from account  
        where balance >= 100  
        order by balance  
    loop  
        -- display a message  
        raise notice 'The account_number branch_name and balance is %, %, %',  
            selected_accounts.branch_name,  
            selected_accounts.balance,  
            selected_accounts.balance;  
    end loop;  
end first_block $$;
```

NOTICE: The account_number branch_name and balance is B, 100.00, 100.00
NOTICE: The account_number branch_name and balance is F, 120.00, 120.00
NOTICE: The account_number branch_name and balance is X, 222.00, 222.00
NOTICE: The account_number branch_name and balance is B, 324.00, 324.00
NOTICE: The account_number branch_name and balance is A, 500.00, 500.00

```

do $$

declare
    -- sort by 1: branch_name, 2: balance
    sort_type smallint := 1;
    -- return the number of accounts
    rec_count int := 10;
    -- use to iterate over the account
    rec record;
    -- dynamic query
    query text;

begin
    query := 'select branch_name, balance from account';
    if sort_type = 1 then
        query := query || 'order by branch_name';
    elsif sort_type = 2 then
        query := query || 'order by balance';
    else
        raise 'invalid sort type %s', sort_type;
    end if;

    for rec in execute query using rec_count
    loop
        raise notice '% - %', rec.balance, rec.branch_name;
    end loop;
end; $$
```

วั่งวน (for loop) ใน dynamic query

| | | | |
|---------|--------|---|---|
| NOTICE: | 50.00 | - | A |
| NOTICE: | 30.00 | - | A |
| NOTICE: | 500.00 | - | A |
| NOTICE: | 324.00 | - | B |
| NOTICE: | 100.00 | - | B |
| NOTICE: | 120.00 | - | F |
| NOTICE: | 222.00 | - | X |

```

do $$

declare
    -- sort by 1: branch_name, 2: balance
    sort_type smallint := 2;
    -- return the number of accounts
    rec_count int := 10;
    -- use to iterate over the account
    rec record;
    -- dynamic query
    query text;

begin
    query := 'select branch_name, balance from account';
    if sort_type = 1 then
        query := query || 'order by branch_name';
    elsif sort_type = 2 then
        query := query || 'order by balance';
    else
        raise 'invalid sort type %s', sort_type;
    end if;

    for rec in execute query
    loop
        raise notice '% - %', rec.balance, rec.branch_name;
    end loop;
end; $$
```

วั่งวน (for loop) ใน dynamic query

| | | | |
|---------|--------|---|---|
| NOTICE: | 30.00 | - | A |
| NOTICE: | 50.00 | - | A |
| NOTICE: | 100.00 | - | B |
| NOTICE: | 120.00 | - | F |
| NOTICE: | 222.00 | - | X |
| NOTICE: | 324.00 | - | B |
| NOTICE: | 500.00 | - | A |



branch

| Branch_name | Branch_city | assets |
|-------------|-------------|----------|
| A | Riverside | \$10,000 |
| B | LA | \$20,000 |
| C | Long Beach | \$15,000 |
| D | Irvine | \$12,000 |
| E | Pomona | \$7,000 |
| F | San Jose | \$18,000 |

depositor

| Customer_name | Account_number |
|---------------|----------------|
| Joe | 1 |
| Joe | 2 |
| Mary | 2 |
| Keith | 4 |
| Mike | 5 |
| Keith | 6 |
| Joe | 3 |

account

| Account_number | Branch_name | balance |
|----------------|-------------|---------|
| 1 | B | \$100 |
| 2 | A | \$50 |
| 3 | A | \$30 |
| 4 | F | \$120 |
| 5 | A | \$500 |
| 6 | B | \$324 |

customer

| Customer_name | Customer_street | Customer_only |
|---------------|-----------------|---------------|
| Joe | Joe_street | Y |
| Alan | Mary_street | Y |
| Jason | Jason_street | N |
| Mary | Mary_street | N |
| Mike | Mary_street | Y |
| Keith | Keith_street | N |

borrower

| Customer_name | Loan_number |
|---------------|-------------|
| Joe | 1 |
| Jason | 2 |
| Joe | 3 |
| Keith | 4 |
| Mary | 5 |
| Joe | 6 |

loan

| Loan_number | Branch_name | Amount |
|-------------|-------------|--------|
| 1 | B | \$100 |
| 2 | E | \$27 |
| 3 | F | \$543 |
| 4 | A | \$129 |
| 5 | A | \$26 |
| 6 | B | \$67 |

Stored Function

Modified from <https://www.postgresqltutorial.com/postgresql-plpgsql/>

Stored Functions (SF)

- Function ถูกเรียกใช้เพื่อให้ทำงานแล้วให้ส่งผลลัพธ์กลับมา ส่วน Procedure นักทำงานเสร็จแล้วไม่ต้องส่งผลลัพธ์กลับมา
- จะถูกเรียกใช้งานผ่านการเรียกชื่อ stored function โดยตรง ไม่ผ่านคำสั่ง CALL
- ดังนั้น ชื่อของ stored function ควรจะต้องแตกต่างจาก SQL functions ที่มีอยู่ในระบบอยู่แล้ว เช่น SUM, AVG, MIN, MAX, COUNT เป็นต้น



การใช้งานจะเหมือนการ
เรียกใช้ Aggregated
functions

CREATE FUNCTION syntax

```
create [or replace] function function_name(param_list)
    returns return_type
    language plpgsql
    as
$$
declare
-- variable declaration
begin
-- logic
end;
$$
```

ตัวอย่างโค้ดในสไลด์ก่อนหน้า
สามารถมาใส่ใน \$\$... \$\$
ได้

Modes of param list in PL/pgSQL functions

| IN | OUT | INOUT |
|----------------------------------|---|---|
| The default | Explicitly specified | Explicitly specified |
| Pass a value to function | Return a value from a function | Pass a value to a function and return an updated value. |
| in parameters act like constants | out parameters act like uninitialized variables | inout parameters act like an initialized variables |
| Cannot be assigned a value | Must assign a value | Should be assigned a value |

ใน MySQL สำหรับ stored function ไม่มี OUT/INOUT มี IN เท่านั้น

<https://www.postgresqltutorial.com/postgresql-plpgsql/plpgsql-function-parameters/>

SQL query: តូវាំមី stored functions នៅឯណ៍ប៉ាង

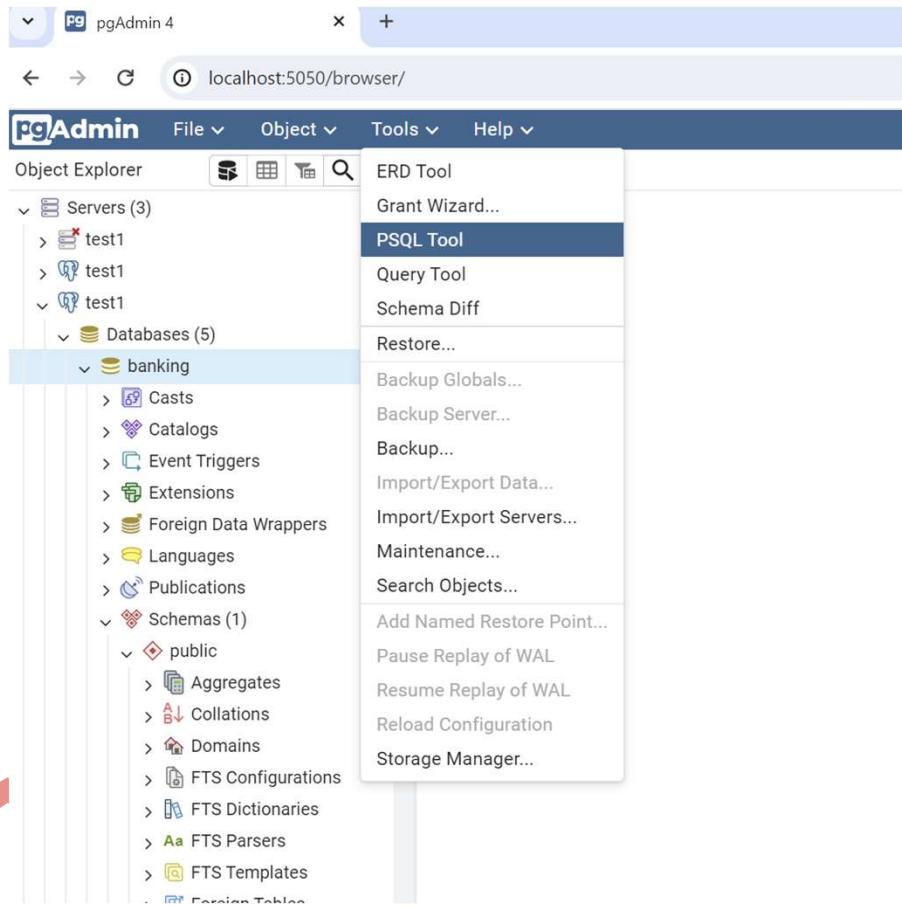
\df

```
banking=# \df
          List of functions
 Schema | Name | Result data type | Argument data types | Type
-----+-----+-----+-----+-----+
(0 rows)
```

MySQL

```
SHOW FUNCTION STATUS;
mysql> SHOW FUNCTION STATUS;
Empty set (0.02 sec)
```

การใช้เครื่องมือ PSQL



- PSQL tool ให้เราใช้คำสั่ง psql เท่านั้น
- psql คือ command line สำหรับ PostgreSQL เท่านั้น
- ต้องจำคำสั่งให้แม่น เช่น \d คือ display all tables

```
banking=# \d
      List of relations
 Schema |    Name     | Type  | Owner
-----+-----+-----+-----
 public | account    | table | root
 public | borrower   | table | root
 public | branch    | table | root
 public | customer  | table | root
 public | depositor | table | root
 public | loan      | table | root
(6 rows)
```

ตัวอย่าง stored function #1

```
CREATE OR REPLACE FUNCTION customer_level(p_moneylevel FLOAT)
```

```
RETURNS VARCHAR(10)
```

```
LANGUAGE plpgsql
```

```
AS
```

```
$$
```

```
DECLARE
```

```
    lvl varchar(10);
```

```
BEGIN
```

```
    IF p_moneylevel > 500 THEN
```

```
        lvl := 'PLATINUM';
```

```
    ELSEIF (p_moneylevel <= 500 AND p_moneylevel >= 100) THEN
```

```
        lvl := 'GOLD';
```

```
    ELSEIF p_moneylevel < 100 THEN
```

```
        lvl := 'SILVER';
```

```
    END IF;
```

```
    RETURN (lvl);
```

```
END;
```

```
$$
```

ฟังก์ชันนี้ส่งออกผลโดยใช้
การ return

IN, OUT or INOUT?

SQL query: ดูว่ามี stored functions อะไรบ้าง อีกที

```
banking=# \df
                                         List of functions
 Schema |      Name       | Result data type | Argument data types
 | Type
-----+-----+-----+-----+
+-----+
 public | customer_level | character varying | p_moneylevel double precision
 | func
(1 row)
```

SQL query: ลองใช้เรียกใช้งานดู

```
SELECT C.customer_name, A.account_number,  
customer_level(A.balance)  
FROM account A, customer C, depositor D  
WHERE A.account_number = D.account_number AND  
C.customer_name = D.customer_name;
```

| customer_name | account_number | customer_level |
|---------------|----------------|----------------|
| Joe | 1 | GOLD |
| Joe | 2 | SILVER |
| Joe | 3 | SILVER |
| Keith | 4 | GOLD |
| Keith | 6 | GOLD |
| Mary | 2 | SILVER |
| Mike | 5 | GOLD |

(7 rows)

ตัวอย่าง stored function #2

```
CREATE OR REPLACE FUNCTION get_account_balance_stat(  
    OUT min_balance NUMERIC,  
    OUT max_balance NUMERIC,  
    OUT avg_balance NUMERIC)  
LANGUAGE plpgsql  
AS  
$$  
BEGIN  
    SELECT MIN(balance)::numeric(6,2), MAX(balance)::numeric(6,2),  
        AVG(balance)::numeric(6,2)  
    INTO min_balance, max_balance, avg_balance  
    FROM account;  
END;  
$$
```

Mode of param list
เป็น OUT ทั้ง 3 ตัวแปร

Function นี้ส่งออกผล
ในตัวแปรที่อยู่ในโหมด
OUT สามตัวแปร

SQL query: ลองใช้เรียกใช้งานดู

```
select get_account_balance_stat();
```

```
get_account_balance_stat
-----
(30.00,500.00,192.29)
(1 row)
```

```
select * from get_account_balance_stat();
```

```
min_balance | max_balance | avg_balance
-----+-----+-----
      30.00 |       500.00 |      192.29
(1 row)
```

ตัวอย่าง stored function #3

```
CREATE OR REPLACE FUNCTION get_customers_with_pattern(  
    p_pattern VARCHAR)  
RETURNS TABLE(  
    cust_name VARCHAR,  
    cust_street VARCHAR  
)  
LANGUAGE plpgsql  
AS  
$$  
BEGIN  
    RETURN query  
    SELECT customer_name, customer_street  
    FROM customer  
    WHERE customer_name LIKE p_pattern;  
END;  
$$
```

Function นี้ returns ผล
ออกมากำในรูปแบบตาราง

SQL query: ลองใช้เรียกใช้งานดู

```
select get_customers_with_pattern('M%');
```

```
get_customers_with_pattern
-----
(Mary,Mary_street)
(Mike,Mary_street)
(2 rows)
```

```
select * from get_customers_with_pattern('M%');
```

```
cust_name | cust_street
-----+-----
Mary      | Mary_street
Mike      | Mary_street
(2 rows)
```

DROP FUNCTION syntax

```
drop function [if exists] function_name(argument_list)  
[cascade | restrict]
```

Exercise # 1

ให้เขียน stored function ชื่อ get_customers_with_sum_balance() เพื่อหาผลรวมของทุกบัญชีเงินฝากของ customer แต่ละคน โดยเมื่อเรียกใช้งานฟังก์ชนจะได้ผลตามตัวอย่างต่อไปนี้ โดยผลเรียงตาม sum_balance จากมากไปน้อย

```
select * from get_customers_with_sum_balance();
```

| customer_name | sum_balance |
|---------------|-------------|
| Mike | 500.00 |
| Keith | 444.00 |
| Joe | 180.00 |
| Mary | 50.00 |
| (4 rows) | |

เราฟัง Lecture ไปก่อน
เดี๋ยวกลับมาทำ EX#1

ເລີດຍ Exercise #1

```
CREATE OR REPLACE FUNCTION get_customers_with_sum_balance()
RETURNS TABLE(
    customer_name VARCHAR,
    sum_balance NUMERIC
)
LANGUAGE plpgsql
AS
$$
BEGIN
    RETURN query
    --
    SELECT C.customer_name, SUM(A.balance) as sum_balance
    FROM customer C, account A, depositor D
    WHERE C.customer_name = D.customer_name AND
        D.account_number = A.account_number
    GROUP BY C.customer_name
    ORDER BY sum_balance DESC;
END;
$$
```

Exercise # 2

ให้เขียน stored function ชื่อ `get_all_customers_with_their_level()` เพิ่มอีก column จากผลใน exercise #1 โดยต้องบอก `customer_level` ด้วย โดยใช้ค่า `sum_balance` เป็นตัวกำหนด `level`

```
select * from get_all_customers_w_their_level();
```

| customer_name | sum_balances | customer_level |
|---------------|--------------|----------------|
| Mike | 500.00 | GOLD |
| Keith | 444.00 | GOLD |
| Joe | 180.00 | GOLD |
| Mary | 50.00 | SILVER |
| (4 rows) | | |

เราฟัง Lecture ไปก่อน
เดี๋ยวกลับมาทำ EX#2

ເຄລຍ Exercise #2

```
CREATE OR REPLACE FUNCTION get_all_customers_w_their_level()
RETURNS TABLE(
    customer_name VARCHAR,
    sum_balances NUMERIC,
    customer_level VARCHAR
)
LANGUAGE plpgsql
AS
$$
DECLARE
    summed_balance NUMERIC;
BEGIN
    RETURN query
    SELECT C.customer_name, SUM(A.balance) as sum_balance, customer_level(SUM(A.balance))
    FROM customer C, account A, depositor D
    WHERE C.customer_name = D.customer_name AND
        D.account_number = A.account_number
    GROUP BY C.customer_name
    ORDER BY sum_balance DESC;
END;
$$
```

Exercise # 3

ให้เขียน stored function ชื่อ get_branches_assets_greater_than(con) เพื่อหาสาขาที่มี assets มากกว่าค่า con ที่กำหนด โดยเมื่อเรียกใช้งานฟังก์ชนจะได้ผลตามตัวอย่างต่อไปนี้

```
select * from get_branches_assets_greater_than(100);
```

| branch_name | branch_city | assets |
|-------------|-------------|-----------|
| A | Riverside | 100000.00 |
| B | LA | 20000.00 |
| C | Long Beach | 15000.00 |
| D | Irvine | 12000.00 |
| E | Pomona | 7000.00 |
| F | San Jose | 18000.00 |

(6 rows)

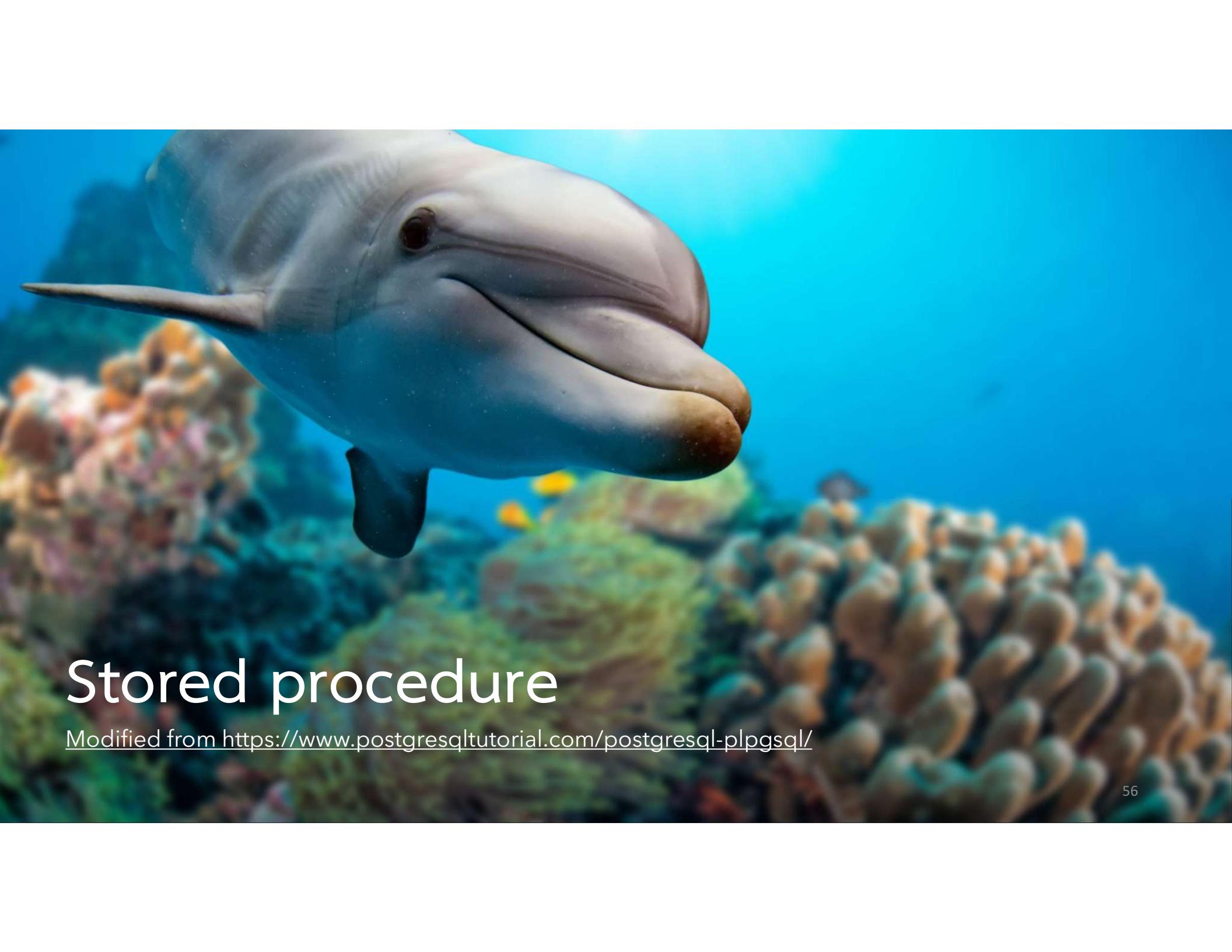
เราฟัง Lecture ไปก่อน
เดี๋ยวกลับมาทำ EX#3
เป็น Assignment

ເຄລຍ Exercise #3

```
CREATE OR REPLACE FUNCTION get_branches_assets_greater_than(con NUMERIC)
RETURNS TABLE(
    branch_name VARCHAR,
    branch_city VARCHAR,
    assets      NUMERIC
)
LANGUAGE plpgsql
AS
$$
BEGIN
    RETURN query
    --
    SELECT B.branch_name, B.branch_city, B.assets
    FROM branch B
    WHERE B.assets > con;
END;
$$
```

อื่นๆ เกี่ยวกับ stored function ใน PostgreSQL

- Function ใน PostgreSQL มี function overloading ด้วย แต่ MySQL ไม่มี
- Function Overloading หมายถึง ชื่อ function เหมือนกัน แต่ parameters หรือ return อาจจะไม่เหมือนกัน ข้อดีคือผู้ใช้งานสามารถเลือกใช้ function ใดก็ได้โดยไม่ต้องระบุ parameters ที่ไม่ต้องการ

A close-up photograph of a dolphin's head and upper body, swimming towards the camera. The dolphin is light grey with a dark grey dorsal fin and a white belly. It is positioned above a vibrant coral reef with various shades of green, yellow, and orange. The background is a clear blue ocean.

Stored procedure

Modified from <https://www.postgresqltutorial.com/postgresql-plpgsql/>

Stored procedure ใน PostgreSQL

- PostgreSQL 11 เป็นต้นมา จึงเพิ่ม Stored procedure เข้ามา เพื่อให้สามารถทำ transaction ใน procedure ได้ (stored function ทำ transaction ไม่ได้)
- โหมดของพารามิเตอร์มีเฉพาะ IN, INOUT ไม่มี OUT
- Stored procedure ไม่มีการ return ค่า แต่ถ้าใช้ return; หมายถึงในอุปกรณ์ทำงานของตัว procedure
- ใช้คำสั่ง call ในการเรียก stored procedure

CREATE PROCEDURE syntax

```
create [or replace] procedure procedure_name(parameter_list)
language plpgsql
as $$  
declare
-- variable declaration
begin
-- stored procedure body
end; $$
```

ตัวอย่าง stored procedure #1

```
CREATE or REPLACE procedure transfer(  
    from_acct VARCHAR(2),  
    to_acct VARCHAR(2),  
    amount NUMERIC  
)  
language plpgsql  
AS $$
```

BEGIN

```
-- subtracting the amount from the sender's account  
UPDATE account  
SET balance = balance - amount  
WHERE account_number = from_acct;
```

โอนเงิน

```
-- adding the amount to the receiver's account  
UPDATE account  
SET balance = balance + amount  
WHERE account_number = to_acct;
```

```
COMMIT;  
END;$$
```

ดูรายการ stored procedure ที่สร้างใช้คำสั่ง \df เมื่อนกัน

```
-----  
banking=# \df  
  
Schema | Name | Argument data types | Result data type | List of functions  
| Type |  
-----+-----+-----+-----+-----+  
public | customer_level | character varying | func | p_moneylevel double  
precision  
public | get_account_balance_stat | record | func | OUT min_balance nume  
ric, OUT max_balance numeric, OUT avg_balance numeric | func  
public | get_customers_with_pattern | TABLE(cust_name character varying, cust_street character varying) | p_pattern character  
varying | func  
public | transfer | | proc | IN from_acct charact  
er varying, IN to_acct character varying, IN amount numeric | proc  
(4 rows)
```

ลองเรียกใช้งาน stored procedure transfer()

| Before | | | After | | |
|----------------|-------------|---------|----------------|-------------|---------|
| account_number | branch_name | balance | account_number | branch_name | balance |
| 1 | B | 100.00 | 2 | A | 50.00 |
| 2 | A | 50.00 | 3 | A | 30.00 |
| 3 | A | 30.00 | 4 | F | 120.00 |
| 4 | F | 120.00 | 6 | B | 324.00 |
| 5 | A | 500.00 | 7 | X | 222.00 |
| 6 | B | 324.00 | 5 | A | 400.00 |
| 7 | X | 222.00 | 1 | B | 200.00 |
| (7 rows) | | | (7 rows) | | |

```
call transfer('5','1',100);
```


branch

| Branch_name | Branch_city | assets |
|-------------|-------------|----------|
| A | Riverside | \$10,000 |
| B | LA | \$20,000 |
| C | Long Beach | \$15,000 |
| D | Irvine | \$12,000 |
| E | Pomona | \$7,000 |
| F | San Jose | \$18,000 |

depositor

| Customer_name | Account_number |
|---------------|----------------|
| Joe | 1 |
| Joe | 2 |
| Mary | 2 |
| Keith | 4 |
| Mike | 5 |
| Keith | 6 |
| Joe | 3 |

account

| Account_number | Branch_name | balance |
|----------------|-------------|---------|
| 1 | B | \$100 |
| 2 | A | \$50 |
| 3 | A | \$30 |
| 4 | F | \$120 |
| 5 | A | \$500 |
| 6 | B | \$324 |

customer

| Customer_name | Customer_street | Customer_only |
|---------------|-----------------|---------------|
| Joe | Joe_street | Y |
| Alan | Mary_street | Y |
| Jason | Jason_street | N |
| Mary | Mary_street | N |
| Mike | Mary_street | Y |
| Keith | Keith_street | N |

borrower

| Customer_name | Loan_number |
|---------------|-------------|
| Joe | 1 |
| Jason | 2 |
| Joe | 3 |
| Keith | 4 |
| Mary | 5 |
| Joe | 6 |

loan

| Loan_number | Branch_name | Amount |
|-------------|-------------|--------|
| 1 | B | \$100 |
| 2 | E | \$27 |
| 3 | F | \$543 |
| 4 | A | \$129 |
| 5 | A | \$26 |
| 6 | B | \$67 |

ตัวอย่าง stored procedure #2

```
CREATE OR REPLACE procedure open_new_saving_account(
```

```
    cust_name VARCHAR,  
    acct_number VARCHAR,  
    branch_name VARCHAR,  
    balance NUMERIC)
```

```
language plpgsql
```

```
AS $$
```

```
BEGIN
```

```
    INSERT INTO customer(customer_name)  
    SELECT cust_name  
    WHERE  
    NOT EXISTS (  
        SELECT C.customer_name FROM customer C WHERE C.customer_name =  
            cust_name);
```

```
    INSERT INTO account VALUES(acct_number, branch_name, balance);  
    INSERT INTO depositor VALUES(cust_name, acct_number);
```

```
END;
```

```
$$
```

เปิดบัญชีเงินฝากใหม่

ตัวอย่าง stored procedure #2

| customer_name | customer_street | customer_only |
|---------------|-----------------|---------------|
| Alan | Mary_street | Y |
| Jason | Jason_street | N |
| Joe | Joe_street | Y |
| Keith | Keith_street | N |
| Mary | Mary_street | N |
| Mike | Mary_street | Y |
| (6 rows) | | |

ก่อนเปิดบัญชีเงินฝาก
ใหม่

| customer_name | account_number | account_number | branch_name | balance |
|---------------|----------------|----------------|-------------|---------|
| Joe | 1 | 2 | A | 50.00 |
| Joe | 2 | 3 | A | 30.00 |
| Joe | 3 | 4 | F | 120.00 |
| Keith | 4 | 6 | B | 324.00 |
| Keith | 6 | 7 | X | 222.00 |
| Mary | 2 | 1 | B | 100.00 |
| Mike | 5 | 5 | A | 500.00 |
| (7 rows) | | (7 rows) | | |

```
call open_new_saving_account('Babara', '10', 'B', 800.0);
```

ตัวอย่าง stored procedure #2

| customer_name | customer_street | customer_only |
|---------------|-----------------|---------------|
| Alan | Mary_street | Y |
| Jason | Jason_street | N |
| Joe | Joe_street | Y |
| Keith | Keith_street | N |
| Mary | Mary_street | N |
| Mike | Mary_street | Y |
| Babara | | |

(7 rows)



หลังเปิดบัญชีเงินฝาก
ใหม่

| customer_name | account_number |
|---------------|----------------|
| Joe | 1 |
| Joe | 2 |
| Joe | 3 |
| Keith | 4 |
| Keith | 6 |
| Mary | 2 |
| Mike | 5 |
| Babara | 10 |

(8 rows)

| account_number | branch_name | balance |
|----------------|-------------|---------|
| 2 | A | 50.00 |
| 3 | A | 30.00 |
| 4 | F | 120.00 |
| 6 | B | 324.00 |
| 7 | X | 222.00 |
| 1 | B | 100.00 |
| 5 | A | 500.00 |
| 10 | B | 800.00 |

(8 rows)

ตัวอย่าง stored procedure #3

```
CREATE OR REPLACE procedure close_saving_account(  
    cust_name VARCHAR,  
    acct_number VARCHAR)  
language plpgsql  
AS $$  
BEGIN  
    DELETE FROM account A  
    WHERE account_number = acct_number;  
    DELETE FROM depositor D  
    WHERE account_number = acct_number;  
END;  
$$
```

ปิดบัญชีเงินฝาก
แต่ไม่ได้ลบลูกค้า

ตัวอย่าง stored procedure #2

| customer_name | customer_street | customer_only |
|---------------|-----------------|---------------|
| Alan | Mary_street | Y |
| Jason | Jason_street | N |
| Joe | Joe_street | Y |
| Keith | Keith_street | N |
| Mary | Mary_street | N |
| Mike | Mary_street | Y |
| Babara | | |

(7 rows)

ก่อนปิดบัญชีเงินฝาก
เลขที่ '10'

| customer_name | account_number |
|---------------|----------------|
| Joe | 1 |
| Joe | 2 |
| Joe | 3 |
| Keith | 4 |
| Keith | 6 |
| Mary | 2 |
| Mike | 5 |
| Babara | 10 |

(8 rows)

| account_number | branch_name | balance |
|----------------|-------------|---------|
| 2 | A | 50.00 |
| 3 | A | 30.00 |
| 4 | F | 120.00 |
| 6 | B | 324.00 |
| 7 | X | 222.00 |
| 1 | B | 100.00 |
| 5 | A | 500.00 |
| 10 | B | 800.00 |

(8 rows)

ตัวอย่าง stored procedure #3

| customer_name | customer_street | customer_only |
|---------------|-----------------|---------------|
| Alan | Mary_street | Y |
| Jason | Jason_street | N |
| Joe | Joe_street | Y |
| Keith | Keith_street | N |
| Mary | Mary_street | N |
| Mike | Mary_street | Y |
| Babara | | |

(7 rows)

หลังปิดบัญชีเงินฝากเลขที่ '10'
แต่ Babara ยังคง

call close_saving_account('Babara', '10');

| customer_name | account_number |
|---------------|----------------|
| Joe | 1 |
| Joe | 2 |
| Joe | 3 |
| Keith | 4 |
| Keith | 6 |
| Mary | 2 |
| Mike | 5 |

(7 rows)

| account_number | branch_name | balance |
|----------------|-------------|---------|
| 2 | A | 50.00 |
| 3 | A | 30.00 |
| 4 | F | 120.00 |
| 6 | B | 324.00 |
| 7 | X | 222.00 |
| 5 | A | 400.00 |
| 1 | B | 200.00 |

(7 rows)

DROP PROCEDURE syntax

```
drop procedure [if exists] procedure_name (argument_list)  
[cascade | restrict]
```

Trigger



Triggers

- Trigger เป็น user-defined function แบบนึง โดยเราต้อง สร้าง function ขึ้นมาก่อน แล้วเอา function นั้นมาผูก (bind) กับตารางหนึ่งๆ
- Trigger ต่างจาก function ทั่วไปตรงที่จะถูกเรียกใช้งานโดยอัตโนมัติเมื่อมี event ที่เกี่ยวข้อง เช่น INSERT, UPDATE, DELETE, TRUNCATE เกิดขึ้น

Triggers

- **ข้อดี** ใช้ตรวจสอบ integrity ของข้อมูลได้ ใช้จัดการ errors ในระดับฐานข้อมูลได้ ใช้เป็นตัวรันงานตามเวลา (scheduled jobs) ได้ ใช้ทำ auditing ได้ เช่นครมาร์ทอะไรกับฐานข้อมูล ก็ให้เขียนการเปลี่ยนแปลงทั้งหมดไว้ได้
- **ข้อเสีย** ดีบักได้ยาก เพราะ trigger จะถูกรันอัตโนมัติโดยระบบ และเพิ่ม overhead ของตัว DBMS บาง รวมทั้งต้องรู้ว่ามี triggers อะไรอยู่บ้าง และแต่ละตัวที่ logic การทำงานอย่างไร

Triggers ใน PostgreSQL มี 2 ระดับหลักๆ ก็อ

1. Row-level triggers
2. Statement-level triggers

โดยอยู่ในระดับไหนขึ้นอยู่กับ จำนวนครั้งที่ถูกเรียกใช้งาน และถูกเรียกใช้งานที่เวลาใด เช่น คำสั่ง UPDATE ที่ต้องแก้ไขข้อมูล 20 รายการ ถ้าเป็น Row-level ตัว trigger จะถูกเรียกใช้ 20 ครั้ง ถ้าเป็น Statement-level ตัว trigger เดียวกันจะถูกเรียกใช้ 1 ครั้ง

ความแตกต่างระหว่าง PostgreSQL triggers และ SQL triggers ที่เป็นมาตรฐาน

- PostgreSQL รองรับ event **TRUNCATE** ด้วยนอกเหนือจาก INSERT, DELETE, UPDATE
- PostgreSQL อนุญาตให้กำหนด statement-level trigger สำหรับวิว (views) ได้
- PostgreSQL บังคับให้ใช้เฉพาะ stored function เพื่อกำหนด action ของ trigger ในขณะที่ SQL มาตรฐาน จะใช้ SQL commands ได้ก็ได้ ไม่ต้องเป็น stored function

CREATE TRIGGER syntax

1

```
CREATE FUNCTION trigger_function()
RETURNS TRIGGER
LANGUAGE PLPGSQL
AS $$
BEGIN
    -- trigger logic
END;
$$:
```

2

```
CREATE TRIGGER trigger_name
{BEFORE | AFTER} { event }
ON table_name
[FOR [EACH] { ROW | STATEMENT }]
EXECUTE PROCEDURE trigger_function
```

SQL query: สร้าง Trigger สักอัน

ตัวอย่างนี้ สร้างตารางเพื่อกีบ log ก่อน

```
CREATE TABLE TriggerTime(exec_time timestamp NOT NULL);
```

สร้าง stored function ที่ทำการ INSERT current time ไปยัง ตาราง TriggerTime

```
CREATE OR REPLACE FUNCTION log_adding_new_account()
RETURNS TRIGGER
language plpgsql
AS $$

BEGIN
    INSERT INTO TriggerTime VALUES(now());
END;$$
```

SQL query: สร้าง Trigger สักอัน

สร้าง stored function ที่ทำการ INSERT current time ไปยัง ตาราง TriggerTime

```
CREATE OR REPLACE FUNCTION log_adding_new_account()
RETURNS TRIGGER
language plpgsql
AS $$

BEGIN
    INSERT INTO TriggerTime VALUES(now());
END;$$
```

ผูก (bind) stored function log_adding_new_account() เข้ากับ Trigger

```
CREATE TRIGGER adding_new_account
BEFORE INSERT
ON account
FOR EACH ROW
EXECUTE PROCEDURE log_adding_new_account();
```

SQL query: สร้าง Trigger สักอัน

```
banking=# select * from triggertime;
exec_time
-----
(0 rows)
```

```
call open_new_saving_account('Babara', '11', 'B', 800.0);
```

```
banking=# select * from triggertime;
exec_time
-----
2023-10-14 15:54:14.553786
(1 row)
```

SQL query: តូវាំង Triggers នៅទីណា

\dS [table name]

```
banking=# \dS account;
                                         Table "public.account"
   Column      |      Type       | Collation | Nullable |      Default
-----+-----+-----+-----+-----+
account_number | character varying(2) |           | not null |
branch_name    | character varying(45) |           |           | NULL::character varying
balance        | numeric(10,2)     |           |           | NULL::numeric
Indexes:
  "account_pkey" PRIMARY KEY, btree (account_number)
Triggers:
  adding_new_account BEFORE INSERT ON account FOR EACH ROW EXECUTE FUNCTION log_adding_new_account()
```

MySQL

SHOW TRIGGERS;

Exercise 4

ให้เขียน Trigger ชื่อ address_audit เพื่อทำการเก็บข้อมูลที่อยู่เดิมของลูกค้าก่อนมีการเปลี่ยนข้อมูลที่อยู่

- สร้างตารางชื่อ address_audit_log เพื่อเก็บประวัติข้อมูลที่อยู่เดิม และวันที่มีการเปลี่ยนแปลงข้อมูลที่อยู่
- สร้าง stored function ชื่อ log_address_history เพื่อทำการ INSERT ข้อมูลที่อยู่เดิม พร้อมวันที่มีการเปลี่ยนแปลงข้อมูลที่อยู่นี้ (ใช้ now()) ได้
- สร้าง Trigger ชื่อ updating_new_address และผูก Trigger นี้กับ function ในข้อที่ 2
- แสดงผลก่อนและหลังการ UPDATE ที่อยู่ของลูกค้า

เราฟัง Lecture ไปก่อน
เดียวกลับมาทำ EX#4
เป็น Assignment

ເລີດຍ Exercise #4

```
CREATE TABLE address_audit_log(customer_name VARCHAR, previous_addr VARCHAR,  
exec_time timestamp NOT NULL);
```

```
CREATE OR REPLACE FUNCTION log_address_history()  
RETURNS TRIGGER  
language plpgsql  
AS $$  
BEGIN  
    INSERT INTO address_audit_log VALUES(OLD.customer_name, OLD.customer_street, now());  
    RETURN NEW;  
END;  
$$
```

```
CREATE TRIGGER updating_new_address  
BEFORE UPDATE  
ON customer  
FOR EACH ROW  
EXECUTE PROCEDURE log_address_history();
```

ເນລຍ Exercise #4

ລອງເປີ່ມທີ່ອຸໍ່ຂອງລູກຄ້າ

```
UPDATE customer SET customer_street = 'Green Street, Oregon'  
WHERE customer_name = 'Alan';
```

| customer_name | customer_street | customer_only |
|---------------|----------------------|---------------|
| Jason | Jason_street | N |
| Joe | Joe_street | Y |
| Keith | Keith_street | N |
| Mary | Mary_street | N |
| Mike | Mary_street | Y |
| Babara | | |
| Alan | Green Street, Oregon | Y |

(7 rows)

| banking=# select * from address_audit_log; | | |
|--|---------------|---------------------------|
| customer_name | previous_addr | exec_time |
| Alan | Mary_street | 2023-10-14 17:36:25.48262 |
| (1 row) | | |

DROP/DISABLE/ENABLE TRIGGER syntax

```
DROP TRIGGER [IF EXISTS] trigger_name  
ON table_name [ CASCADE | RESTRICT ];
```

```
ALTER TABLE table_name  
DISABLE TRIGGER trigger_name | ALL
```

```
ALTER TABLE table_name  
ENABLE TRIGGER trigger_name | ALL;
```



จน Stored Routines

...
เริ่ม View

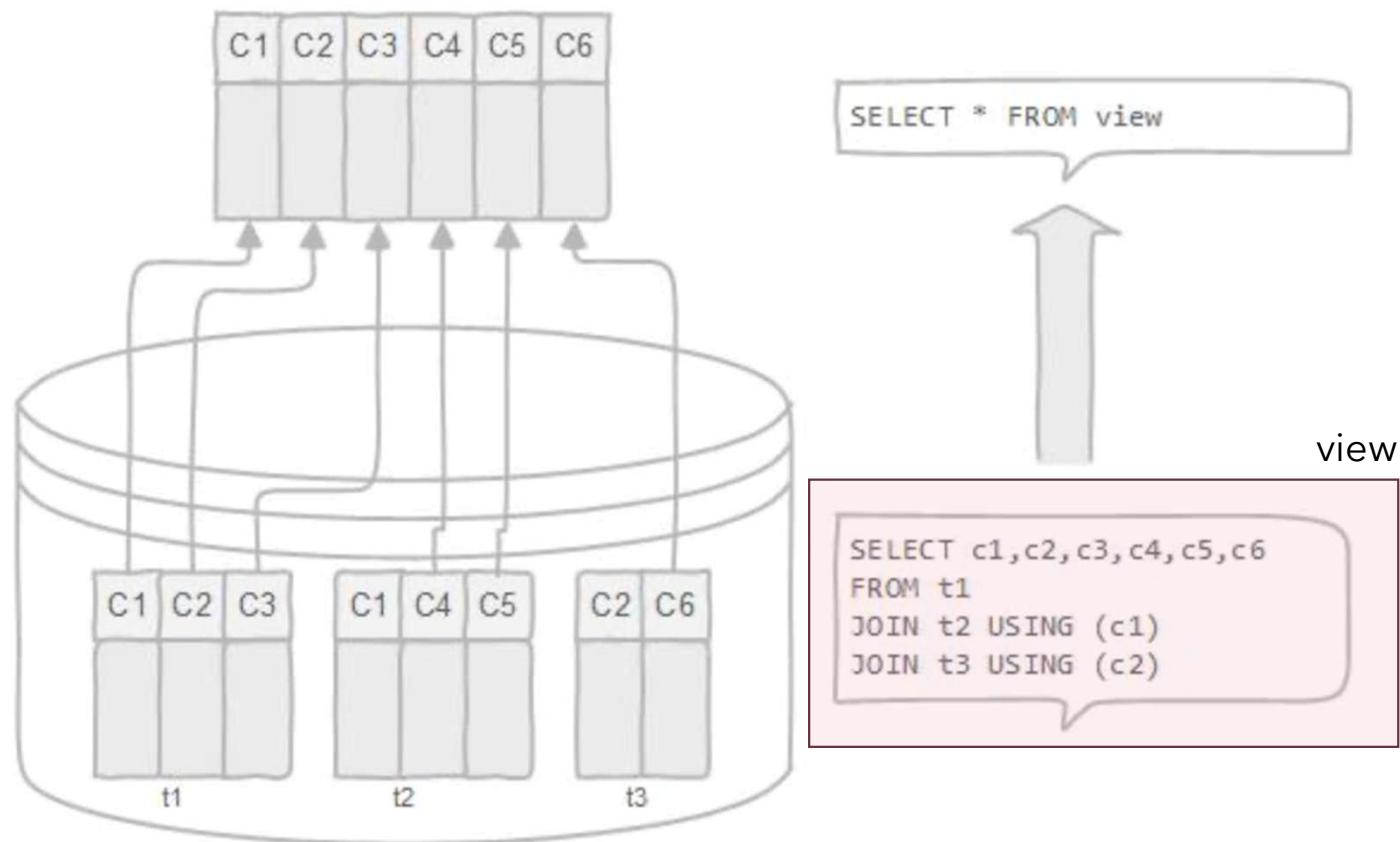


View

<https://www.postgresqltutorial.com/postgresql-views/managing-postgresql-views/>

Views

- วิว เป็นตารางเสมือน (Virtual table) โดยมีชุดของแอทริบิวต์และรายการข้อมูลตามผลของการ query หนึ่งๆ อีกนัยหนึ่ง คือ การตั้งชื่อให้กับ query เพื่อนำมาใช้ซ้ำได้โดยง่าย
- ชุดของแอทริบิวต์ในวิว มาจากตารางต่างๆ ที่ระบุใน query
- เขียนวิวไว้ ก็สามารถใช้ร่วมกับ query อื่นได้ ทำให้ query อื่นๆ นั้นกระชับมากขึ้น หรือเข้าใจได้ง่ายขึ้น แต่ query cost สูง เพราะต้องทำ query ใหม่ทุกรอบที่เรียกใช้วิว (ยกเว้นสร้างเป็น Materialized views)
- เขียนวิวไว้ และกำหนดให้เข้าถึงข้อมูลคอมมอนเฉพาะของตารางหนึ่งๆ ผ่าน วิว เท่านั้น หรือกำหนดสิทธิในการเข้าถึง view สำหรับผู้ใช้ ช่วยปกป้องความปลอดภัยของข้อมูลได้
- เพราะวิวถูกรันใหม่ทุกรอบ ถ้าตารางจริงมีข้อมูลที่เปลี่ยนไป ข้อมูลวิว ก็เปลี่ยนด้วย

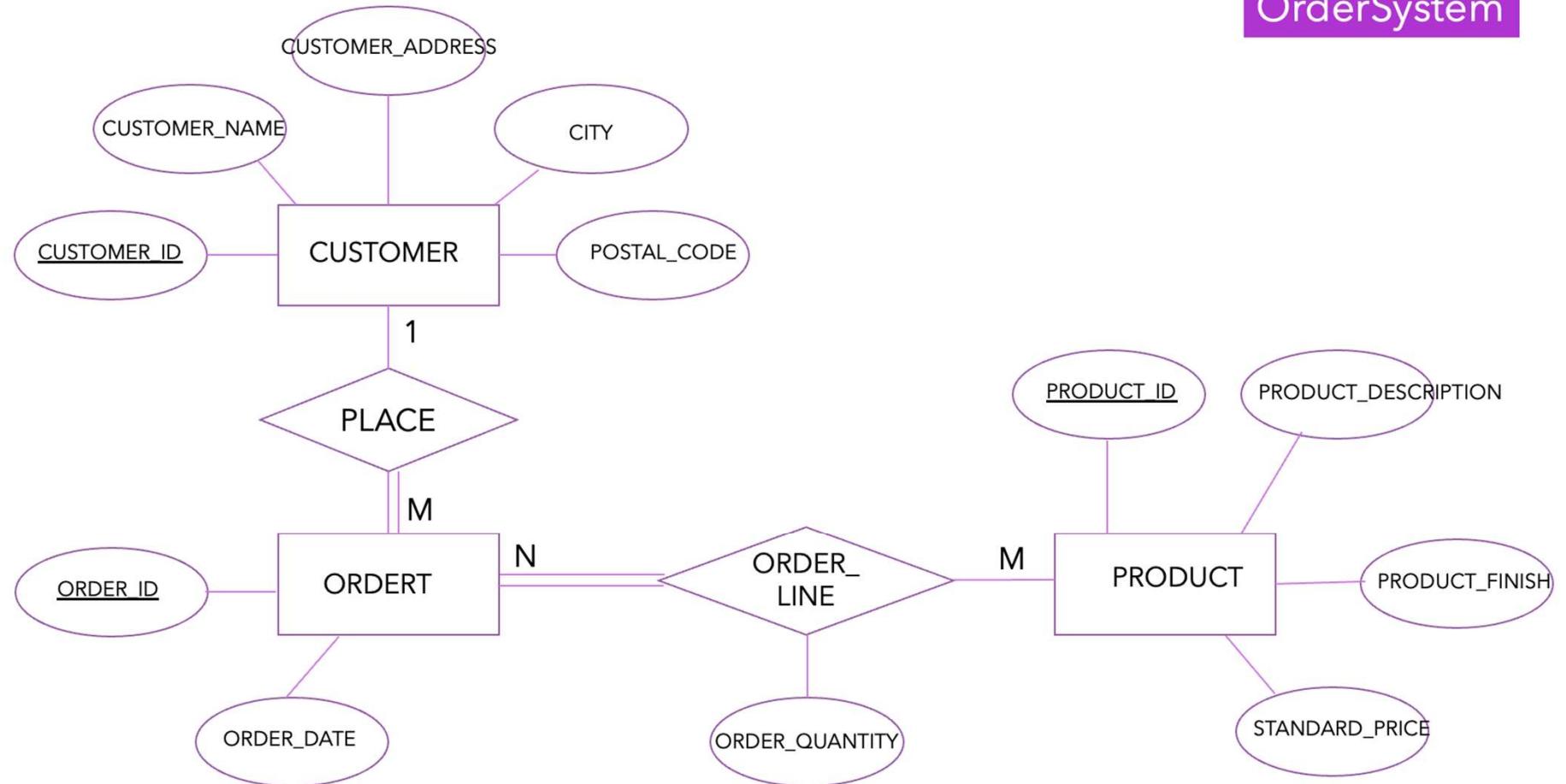


<https://www.postgresqltutorial.com/postgresql-views/managing-postgresql-views/>
[Online access on Oct 14, 2023]

CREATE VIEW

```
CREATE VIEW view_name AS query;
```

OrderSystem



แสดงรหัสสินค้าและคำบรรยายสินค้าที่มีรายการสั่งซื้อมากสุด

ordert

| order_id | order_date | customer_id |
|----------|---------------------|-------------|
| 2 | 2020-01-08 12:00:00 | 10009 |
| 3 | 2020-01-10 23:00:00 | 10001 |
| 4 | 2020-01-11 23:00:00 | 10001 |
| 5 | 2020-01-12 23:00:00 | 10001 |
| 6 | 2020-01-13 23:00:00 | 10001 |
| 7 | 2020-01-14 23:00:00 | 10001 |
| 8 | 2020-01-15 23:00:00 | 10001 |
| 9 | 2020-01-16 23:00:00 | 10001 |
| 10 | 2020-01-17 23:00:00 | 10001 |
| 11 | 2020-01-18 23:00:00 | 10001 |
| 12 | 2020-01-19 23:00:00 | 10001 |
| 14 | 2020-01-20 23:00:00 | 10001 |
| 15 | 2020-01-21 23:00:00 | 10001 |
| 16 | 2020-01-10 23:00:00 | 10002 |
| 17 | 2020-01-11 23:00:00 | 10002 |
| 18 | 2020-01-12 23:00:00 | 10002 |
| 19 | 2020-01-13 23:00:00 | 10002 |
| 20 | 2020-01-14 23:00:00 | 10002 |
| 21 | 2020-01-15 23:00:00 | 10002 |
| 22 | 2020-01-16 23:00:00 | 10002 |
| 23 | 2020-01-17 23:00:00 | 10002 |
| 24 | 2020-01-18 23:00:00 | 10002 |
| 25 | 2020-01-19 23:00:00 | 10002 |
| 26 | 2020-01-20 23:00:00 | 10002 |
| 27 | 2020-01-21 23:00:00 | 10002 |
| 28 | 2020-01-22 23:00:00 | 10002 |
| 29 | 2020-01-23 23:00:00 | 10002 |
| 30 | 2020-01-24 23:00:00 | 10002 |
| 31 | 2020-01-25 23:00:00 | 10002 |
| 32 | 2020-01-26 23:00:00 | 10002 |
| 33 | 2020-01-27 23:00:00 | 10002 |
| 34 | 2020-01-28 23:00:00 | 10002 |
| 35 | 2020-01-29 23:00:00 | 10002 |

order_line

| order_id | product_id | ordered_quantity |
|----------|------------|------------------|
| 2 | 2 | 3 |
| 2 | 3 | 10 |
| 2 | 4 | 1 |
| 2 | 5 | 10 |
| 3 | 1 | 10 |
| 3 | 3 | 11 |
| 3 | 5 | 1 |
| 4 | 6 | 10 |
| 5 | 1 | 1 |
| 5 | 2 | 5 |
| 5 | 3 | 5 |
| 5 | 4 | 5 |
| 5 | 5 | 5 |
| 5 | 6 | 5 |

product

| product_id | product_description | product_finish | standard_price |
|------------|---------------------|----------------|----------------|
| 1 | Stool bar | Red Oak | 250.00 |
| 2 | Arm chair | Walnut | 1200.00 |
| 3 | Cabinet | White Ash | 500.00 |
| 4 | Triple Cabinet | Natural Ash | 1500.00 |
| 6 | Lazy Boy | White Ash | 2000.00 |
| 7 | kitchen cabinet | Cherry | 1500.00 |
| 8 | table | Red Oak | 550.00 |
| 5 | Sofabed | Cherry | 5400.00 |

ตัวอย่างการสร้างและเรียกใช้งานวิว

```
CREATE VIEW ordered_products AS  
SELECT product_id, count(order_id) AS sum_products  
FROM order_line  
GROUP BY product_id;
```

View name

```
CREATE VIEW top_product AS  
SELECT product_id  
FROM ordered_products  
WHERE sum_products IN (SELECT MAX(sum_products) FROM ordered_products);
```

```
SELECT product_id, product_description  
FROM PRODUCT  
WHERE product_id IN (SELECT * FROM top_product);
```

จากโจทย์ใน
Exercise 4 ที่ผ่านมา

แสดงรหัสสินค้าและคำบรรยายสินค้าที่มีรายการสั่งซื้อมากสุด

| product_id | product_description |
|------------|---------------------|
| 3 | Cabinet |
| 5 | Sofabed |

(2 rows)

ดูรายละเอียดของวิวที่สร้างไว้

\d+ [view name]

```
ordersystem=# \d+ ordered_products
                                         View "public.ordered_products"
   Column      |  Type   | Collation | Nullable | Default | Storage | Description
-----+-----+-----+-----+-----+-----+-----+
product_id    | integer |           |          |          | plain   |
sum_products  | bigint  |           |          |          | plain   |
View definition:
SELECT product_id,
       count(order_id) AS sum_products
  FROM order_line
 GROUP BY product_id;
```

ในฐานข้อมูล ordersystem มี view อะไรบ้างแล้วนะ

```
select table_name from INFORMATION_SCHEMA.views  
WHERE table_schema = ANY (current_schemas(false));
```

| table_name |
|--------------------------|
| ----- |
| ordered_products |
| top_product |
| customer_orders |
| ordered_product_quantity |
| top_product_quantity |
| (5 rows) |

เพิ่มรายการในตารางจริงมีผลต่อวิวทันที

ลองเพิ่มรายการใน order_line order_id | product_id

INSERT INTO order_line VALUES(11,3,10);

จากนั้นลอง select ข้อมูลจาก views อีกที

SELECT * FROM ordered_products;

| product_id | sum_products |
|------------|--------------|
| 3 | 4 |
| 5 | 3 |
| 4 | 2 |
| 6 | 2 |
| 2 | 2 |
| 1 | 2 |

(6 rows)

SELECT * FROM top_product;

| product_id |
|------------|
| 3 |
| (1 row) |

| product_id | product_description |
|------------|---------------------|
| 3 | Cabinet |
| (1 row) | |

SELECT product_id, product_description
FROM PRODUCT
WHERE product_id IN (SELECT * FROM top_product);

DROP VIEW syntax

```
DROP VIEW [IF EXISTS] view_name  
[CASCADE | RESTRICT]
```

Views เพิ่มเติม

- เราตั้งให้ view เป็นแบบ read only เท่านั้นและถ้าต้องการ insert ข้อมูลก็ต้องไป insert ที่ตารางต้นฉบับ
- การใช้ view ทำให้ใช้งานง่ายสำหรับผู้ที่ไม่คุ้นเคยกับภาษา SQL แต่มี query cost สูง เพราะ virtual table จะถูกสร้างขึ้นใหม่ทุกครั้งที่ view นั้นๆ ถูกเรียกใช้งาน

ตัวอย่างการ Insert into ... View ...

1 **CREATE VIEW v1 AS**

```
SELECT * FROM boat NATURAL JOIN reserve;
```

2 **SELECT * FROM v1;**

3 **INSERT INTO v1 (bid,bname,color) VALUES**
(110,'Myboat','Black');

4 **SELECT * FROM v1;** => ได้อะไรออกมาบ้าง ต่างจาก 2 ง่วงไร เพราะอะไร

ตัวอย่างการ Insert into ... View ...

```
SELECT * FROM boat B LEFT OUTER JOIN reserve R  
ON B.bid=R.bid;
```

```
SELECT * FROM boat B RIGHT OUTER JOIN reserve R  
ON B.bid=R.bid;
```

ผลการ SELECT ข้างต้น
ได้จำนวนบรรทัดต่างกัน
อย่างไร

A vibrant yellow and blue angelfish is the central focus, swimming diagonally from the bottom right towards the top left against a dark blue, textured background.

Index

Index

- Indexing คือ การนำ attribute ที่กำหนดเป็น indexing attributes มาช่วยในการเข้าถึงข้อมูลจากตาราง
- โดยปกติ attribute ที่เป็น Primary key และ Foreign key จะได้รับการทำ index โดยอัตโนมัติ
- เราควรทำ Index ให้กับ attribute ที่เราใช้สืบค้นหรือเรียงลำดับเป็นประจำ เช่น เงื่อนไขใน where หรือ order by เป็นต้น
- เราสามารถทำ Index ให้กับ attribute เหล่านี้โดยใช้คำสั่ง **create index**
- อย่างไรก็ตามการมี Index เป็น overhead ในกรณี insert, delete ข้อมูล เช่นกัน และถ้าจำนวนข้อมูลมีน้อยก็อาจจะไม่คุ้มในการทำ index รวมทั้งต้องเสียพื้นที่ในการเก็บ index ด้วย

CREATE INDEX syntax

btree,hash, ...

```
CREATE INDEX index_name ON table_name [USING method]
(
    column_name [ASC | DESC] [NULLS {FIRST | LAST}],
    ...
);
```

สร้างฐานข้อมูลใหม่ชื่อ companion

1. สร้างตารางใหม่ชื่อ customer

```
CREATE TABLE customer(  
    idx INT,  
    cid VARCHAR(40),  
    first_name VARCHAR(50),  
    last_name VARCHAR(50),  
    company VARCHAR(100),  
    city VARCHAR(100),  
    country VARCHAR(80),  
    phone_1 VARCHAR(20),  
    phone_2 VARCHAR(20),  
    email VARCHAR(50),  
    subscription_date DATE,  
    website VARCHAR(150))
```

สร้างฐานข้อมูลใหม่ชื่อ companion

2. นำข้อมูลเข้าตาราง

```
\copy customer FROM '/Users/duangdaowichadakul/Desktop/2110322_DB SYS_2566_1/  
customers-500000.csv' DELIMITER ',' CSV HEADER;
```

3. ลองดูข้อมูลในตาราง

```
SELECT * FROM customer;
```

4. ลองดู schema ของตารางผ่านคำสั่ง \d customer

```
EXPLAIN ANALYZE SELECT * FROM customer WHERE country = 'Thailand';
```

QUERY PLAN

```
Gather (cost=1000.00..15918.87 rows=2027 width=159) (actual time=0.462..39.752 rows=2025 loops=1)
  Workers Planned: 2
  Workers Launched: 2
    -> Parallel Seq Scan on customer (cost=0.00..14716.17 rows=845 width=159) (actual time=0.233..32.443 rows=675 loops=3)
      Filter: ((country)::text = 'Thailand'::text)
      Rows Removed by Filter: 165992
Planning Time: 0.132 ms
Execution Time: 39.861 ms
```

Before indexing country column

```
CREATE INDEX country_idx ON customer(country);
```

```
companion=# EXPLAIN ANALYZE SELECT * FROM customer WHERE country = 'Thailand';
          QUERY PLAN
```

```
Bitmap Heap Scan on customer (cost=24.13..5327.38 rows=2027 width=159) (actual time=4.674..109.584 rows=2025 loops=1)
  Recheck Cond: ((country)::text = 'Thailand'::text)
  Heap Blocks: exact=1881
    -> Bitmap Index Scan on country_idx (cost=0.00..23.62 rows=2027 width=0) (actual time=3.753..3.754 rows=2025 loops=1)
      Index Cond: ((country)::text = 'Thailand'::text)
Planning Time: 1.251 ms
Execution Time: 110.265 ms
(7 rows)
```

After indexing country column

```
companion=# EXPLAIN ANALYZE SELECT * FROM customer WHERE country = 'Thailand';
          QUERY PLAN
```

```
Bitmap Heap Scan on customer (cost=24.13..5327.38 rows=2027 width=159) (actual time=0.691..4.520 rows=2025 loops=1)
  Recheck Cond: ((country)::text = 'Thailand'::text)
  Heap Blocks: exact=1881
    -> Bitmap Index Scan on country_idx (cost=0.00..23.62 rows=2027 width=0) (actual time=0.345..0.346 rows=2025 loops=1)
      Index Cond: ((country)::text = 'Thailand'::text)
Planning Time: 0.156 ms
Execution Time: 4.717 ms
(7 rows)
```

```
EXPLAIN ANALYZE SELECT first_name, last_name, phone_1 FROM customer WHERE phone_1  
LIKE '(449)%';
```

QUERY PLAN

```
Gather  (cost=1000.00..15721.17 rows=50 width=30) (actual time=0.256..39.879 rows=85 loops=1)  
Workers Planned: 2  
Workers Launched: 2  
-> Parallel Seq Scan on customer  (cost=0.00..14716.17 rows=21 width=30) (actual time=0.443..35.410 rows=28 loops=3)  
    Filter: ((phone_1)::text ~~ '(449)%'::text)  
    Rows Removed by Filter: 166638  
Planning Time: 0.122 ms  
Execution Time: 39.915 ms
```

Before indexing phone_1 column

```
CREATE INDEX phone1_idx ON customer(phone_1);
```

QUERY PLAN

```
Index Scan using phone1_idx on customer  (cost=0.42..8.44 rows=50 width=30) (actual time=0.822..6.584 rows=85 loops=1)  
  Index Cond: (((phone_1)::text >= '(449)'::text) AND ((phone_1)::text < '(449*'::text))  
  Filter: ((phone_1)::text ~~ '(449)%'::text)  
Planning Time: 1.297 ms  
Execution Time: 6.633 ms  
(5 rows)
```

After indexing phone_1 column

```
companion=# EXPLAIN ANALYZE SELECT first_name, last_name, phone_1 FROM customer WHERE phone_1 LIKE '(449)%';  
QUERY PLAN
```

```
Index Scan using phone1_idx on customer  (cost=0.42..8.44 rows=50 width=30) (actual time=0.044..0.220 rows=85 loops=1)  
  Index Cond: (((phone_1)::text >= '(449)'::text) AND ((phone_1)::text < '(449*'::text))  
  Filter: ((phone_1)::text ~~ '(449)%'::text)  
Planning Time: 0.171 ms  
Execution Time: 0.251 ms  
(5 rows)
```

ลองดู schema ของตารางผ่านคำสั่ง \d customer อีกครั้ง

```
companion=# \d customer;
Table "public.customer"
 Column          |      Type       | Collation | Nullable | Default
-----+-----+-----+-----+-----+
 idx      | integer
 cid      | character varying(40)
 first_name | character varying(50)
 last_name | character varying(50)
 company   | character varying(100)
 city      | character varying(100)
 country    | character varying(80)
 phone_1    | character varying(40)
 phone_2    | character varying(40)
 email      | character varying(50)
 subscription_date | date
 website    | character varying(150)
Indexes:
 "country_idx" btree (country)
 "phonel_idx" btree (phone_1)
```

PostgreSQL เลือก index แบบใด และการ query ลักษณะไหนได้
ประโยชน์จากการทำ index แบบต่างๆ

<https://www.postgresqltutorial.com/postgresql-indexes/postgresql-index-types/>

\d student_w_grade;

| Table "public.student_w_grade" | | | | |
|--------------------------------|-----------------------|-----------|----------|---------|
| Column | Type | Collation | Nullable | Default |
| sid | character varying(10) | | not null | |
| q1 | numeric(10,2) | | | |
| q2 | numeric(10,2) | | | |
| q3 | numeric(10,2) | | | |
| q4 | numeric(10,2) | | | |
| q5 | numeric(10,2) | | | |
| total | numeric(10,2) | | | |
| grade | character varying(2) | | | |

Indexes:

"student_w_grade_pkey" PRIMARY KEY, btree (sid)

How long it will take to fetch the first row

How long it will take to return all the rows

EXPLAIN SELECT * FROM student_w_grade WHERE grade = 'B';

Before create index

QUERY PLAN

```
Seq Scan on student_w_grade  (cost=0.00..22.50 rows=200 width=41)
  Filter: ((grade)::text = 'B'::text)
(2 rows)
```

CREATE INDEX grade_index ON student_w_grade(grade);

EXPLAIN SELECT * FROM student_w_grade WHERE grade = 'B';

After create index

QUERY PLAN

```
Bitmap Heap Scan on student_w_grade  (cost=5.70..18.20 rows=200 width=41)
  Recheck Cond: ((grade)::text = 'B'::text)
    -> Bitmap Index Scan on grade_index  (cost=0.00..5.65 rows=200 width=0)
      Index Cond: ((grade)::text = 'B'::text)
(4 rows)
```

DROP INDEX grade_index;

EXPLAIN ANALYZE SELECT * FROM student_w_grade WHERE grade = 'B';

Before create index

QUERY PLAN

```
Seq Scan on student_w_grade  (cost=0.00..22.50 rows=200 width=41) (actual time=0.028..0.177 rows=200 loops=1)
  Filter: ((grade)::text = 'B'::text)
  Rows Removed by Filter: 800
Planning Time: 0.128 ms
Execution Time: 0.206 ms
```

CREATE INDEX grade_index ON student_w_grade(grade);

EXPLAIN ANALYZE SELECT * FROM student_w_grade WHERE grade = 'B';

After create index

QUERY PLAN

```
Bitmap Heap Scan on student_w_grade  (cost=5.70..18.20 rows=200 width=41) (actual time=0.093..0.163 rows=200 loops=1)
  Recheck Cond: ((grade)::text = 'B'::text)
  Heap Blocks: exact=10
-> Bitmap Index Scan on grade_index  (cost=0.00..5.65 rows=200 width=0) (actual time=0.056..0.057 rows=200 loops=1)
  Index Cond: ((grade)::text = 'B'::text)
Planning Time: 0.169 ms
Execution Time: 0.221 ms
```

```
DROP INDEX grade_index;
```

```
EXPLAIN ANALYZE SELECT grade, count(*) FROM student_w_grade GROUP BY grade;
```

QUERY PLAN

```
-----  
HashAggregate  (cost=25.00..25.05 rows=5 width=10) (actual time=0.373..0.375 rows=5 loops=1)  
  Group Key: grade  
  Batches: 1  Memory Usage: 24kB  
-> Seq Scan on student_w_grade  (cost=0.00..20.00 rows=1000 width=2) (actual time=0.021..0.122 rows=1000 loops=1)  
Planning Time: 0.139 ms  
Execution Time: 0.424 ms
```

| grade | count |
|----------|-------|
| C+ | 35 |
| B+ | 325 |
| A | 434 |
| C | 6 |
| B | 200 |
| (5 rows) | |

```
CREATE INDEX grade_index ON student_w_grade(grade);
```

```
EXPLAIN ANALYZE SELECT grade, count(*) FROM student_w_grade GROUP BY grade;
```

QUERY PLAN

```
-----  
HashAggregate  (cost=25.00..25.05 rows=5 width=10) (actual time=0.405..0.408 rows=5 loops=1)  
  Group Key: grade  
  Batches: 1  Memory Usage: 24kB  
-> Seq Scan on student_w_grade  (cost=0.00..20.00 rows=1000 width=2) (actual time=0.020..0.127 rows=1000 loops=1)  
Planning Time: 0.159 ms  
Execution Time: 0.459 ms
```

DROP INDEX grade_index;

EXPLAIN ANALYZE SELECT grade, count(*) FROM student_w_grade GROUP BY grade ORDER BY GRADE ASC;

QUERY PLAN

```
Sort  (cost=25.11..25.12 rows=5 width=10) (actual time=0.870..0.872 rows=5 loops=1)
  Sort Key: grade
  Sort Method: quicksort  Memory: 25kB
->  HashAggregate  (cost=25.00..25.05 rows=5 width=10) (actual time=0.835..0.851 rows=5 loops=1)
    Group Key: grade
    Batches: 1  Memory Usage: 24kB
      ->  Seq Scan on student_w_grade  (cost=0.00..20.00 rows=1000 width=2) (actual time=0.028..0.245 rows=1000 loops=1)
Planning Time: 0.189 ms
Execution Time: 0.942 ms
(9 rows)
```

CREATE INDEX grade_index ON student_w_grade(grade);

EXPLAIN ANALYZE SELECT grade, count(*) FROM student_w_grade GROUP BY grade ORDER BY GRADE ASC;

```
EXPLAIN SELECT product_id, product_description  
FROM PRODUCT  
WHERE product_id IN (SELECT * FROM top_product);
```

QUERY PLAN

```
Nested Loop (cost=90.41..90.68 rows=1 width=122)  
  -> HashAggregate (cost=90.26..90.27 rows=1 width=4)  
    Group Key: order_line.product_id  
    -> Hash Join (cost=85.72..90.26 rows=1 width=4)  
      Hash Cond: ((count(order_line.order_id)) = (max((count(order_line_1.order_id)))))  
      -> HashAggregate (cost=40.60..42.60 rows=200 width=12)  
        Group Key: order_line.product_id  
        -> Seq Scan on order_line (cost=0.00..30.40 rows=2040 width=8)  
      -> Hash (cost=45.11..45.11 rows=1 width=8)  
        -> Aggregate (cost=45.10..45.11 rows=1 width=8)  
          -> HashAggregate (cost=40.60..42.60 rows=200 width=12)  
            Group Key: order_line_1.product_id  
            -> Seq Scan on order_line order_line_1 (cost=0.00..30.40 rows=2040 width=8)  
  -> Index Scan using product_pk on product (cost=0.15..0.41 rows=1 width=122)  
    Index Cond: (product_id = order_line.product_id)  
(15 rows)
```

Index

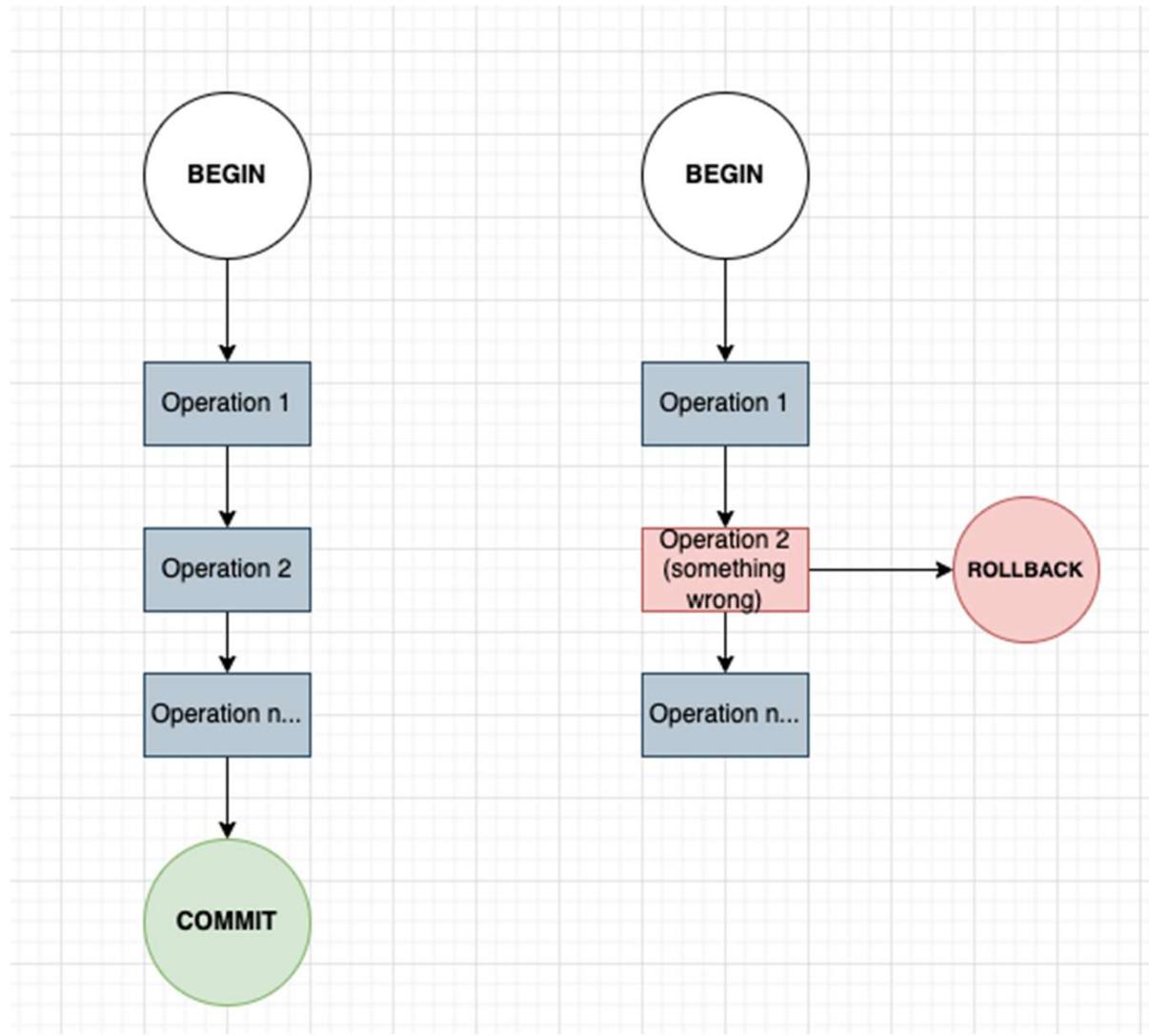
- ใช้คอลัมน์ต่างๆ ของตารางมาทำเป็นดัชนีเพิ่มเติมจาก คีย์หลักหรือ primary key
- MySQL ใช้ index ในส่วนของ WHERE clause ถ้ามีการใช้คอลัมน์ที่ทำ index ไว้ ทำให้เข้าถึงข้อมูลที่ต้องการได้เร็วขึ้น
- การสร้างมี trade-off เพราะ DBMS ก็ต้องเสียพื้นที่ในการเก็บ index เพิ่มด้วย เช่นกัน ก็ต้องประเมินว่า index ไหนจำเป็น ไม่จำเป็น

Exercise สั้นๆ 3 นาที

- ลอง explain query เพื่อดึงข้อมูล sum(balance) จากตาราง account แยกตามสาขา
- ลองทำดู ให้สร้าง index ชื่อ b1 โดยใช้ branch_name ในตาราง account
- ลอง explain query ดูอีกครั้ง



Transaction, Commit, Rollback



What is a Database Transaction?

- A transaction is an action or series of actions that are being performed by a single user or application program, which reads or updates the contents of the database.
- A **DATABASE TRANSACTION** is a logical unit of processing in a DBMS which accesses and possibly modifies the contents of a database. In a nutshell, database transactions represent real-world events of any enterprise.
- Example :

X's Account :

Open_Account(X)

Old_Balance = X.balance

New_Balance = Old_Balance - 800

X.balance = New_Balance

Close_Account(X)

Y's Account :

Open_Account(Y)

Old_Balance = Y.balance

New_Balance = Old_Balance + 800

Y.balance = New_Balance

Close_Account(Y)

Transaction

- ทรานแซคชัน ทำให้สามารถควบคุมการเพิ่มลดเปลี่ยนแปลงข้อมูลในฐานข้อมูลโดยมีคุณสมบัติ Atomicity เช่น **ทรานแซคชันโอนเงิน** จะต้อง
 - มีการ update ข้อมูลเพื่อลดจำนวนเงินของบัญชีต้นทาง
 - มีการ update ข้อมูลเพื่อเพิ่มจำนวนเงินของบัญชีปลายทาง
 - ** ต้องทำทั้ง 2 updates ถูกต้องเรียบร้อย ถึงถือว่า transaction นี้ทำงานสำเร็จ แล้วค่อยเขียน (commit) ลงฐานข้อมูลจริงๆ ถ้าเสร็จแค่บางส่วนแล้วเกิด error เสียก่อน ต้อง rollback ซุดคำสั่งของทรานแซคชันนั้นที่ทำไปแล้วทั้งหมด

การกำหนด transaction ใน PostgreSQL

1. BEGIN TRANSCATION; หรือ
2. BEGIN WORK; หรือ
3. BEGIN; ก็ได้

ลองชุดคำสั่งต่อไปนี้ Session 1

```
START TRANSACTION;  
INSERT INTO boat VALUES(112,'AAA','Black');  
INSERT INTO boat VALUES(113,'BBB','Red');  
SELECT * FROM boat;
```

| bid | bname | color |
|----------|-----------|-------|
| 101 | Interlake | Blue |
| 102 | Interlake | Red |
| 103 | Clipper | Green |
| 104 | Marine | Red |
| 112 | AAA | Black |
| 113 | BBB | Red |
| (6 rows) | | |

ลองคำสั่งต่อไปนี้ Session 2

SELECT * FROM boat;

| bid | bname | color |
|----------|-----------|-------|
| 101 | Interlake | Blue |
| 102 | Interlake | Red |
| 103 | Clipper | Green |
| 104 | Marine | Red |
| (4 rows) | | |

Session 2 นี้ยังไม่เห็นข้อมูล
ที่เพิ่มจาก Session 1

กลับมา Session 1 แล้วลอง COMMIT

COMMIT;

กลับมา Session 2

SELECT * FROM boat;

| bid | bname | color |
|------------|-----------|-------|
| 101 | Interlake | Blue |
| 102 | Interlake | Red |
| 103 | Clipper | Green |
| 104 | Marine | Red |
| 112 | AAA | Black |
| 113 | BBB | Red |
| (6 rows) | | |

Session 2 เห็นข้อมูลที่เพิ่ม
จาก Session 1 แล้ว

ถ้า Session 1 เรียกคำสั่ง
ROLLBACK แทน COMMIT
จะเกิดอะไรขึ้น

ตัวอย่างของ Transaction ใน Stored procedure

- <https://gist.github.com/anilahir/d54f89d8f4edbc8b7e99ef2557371339>

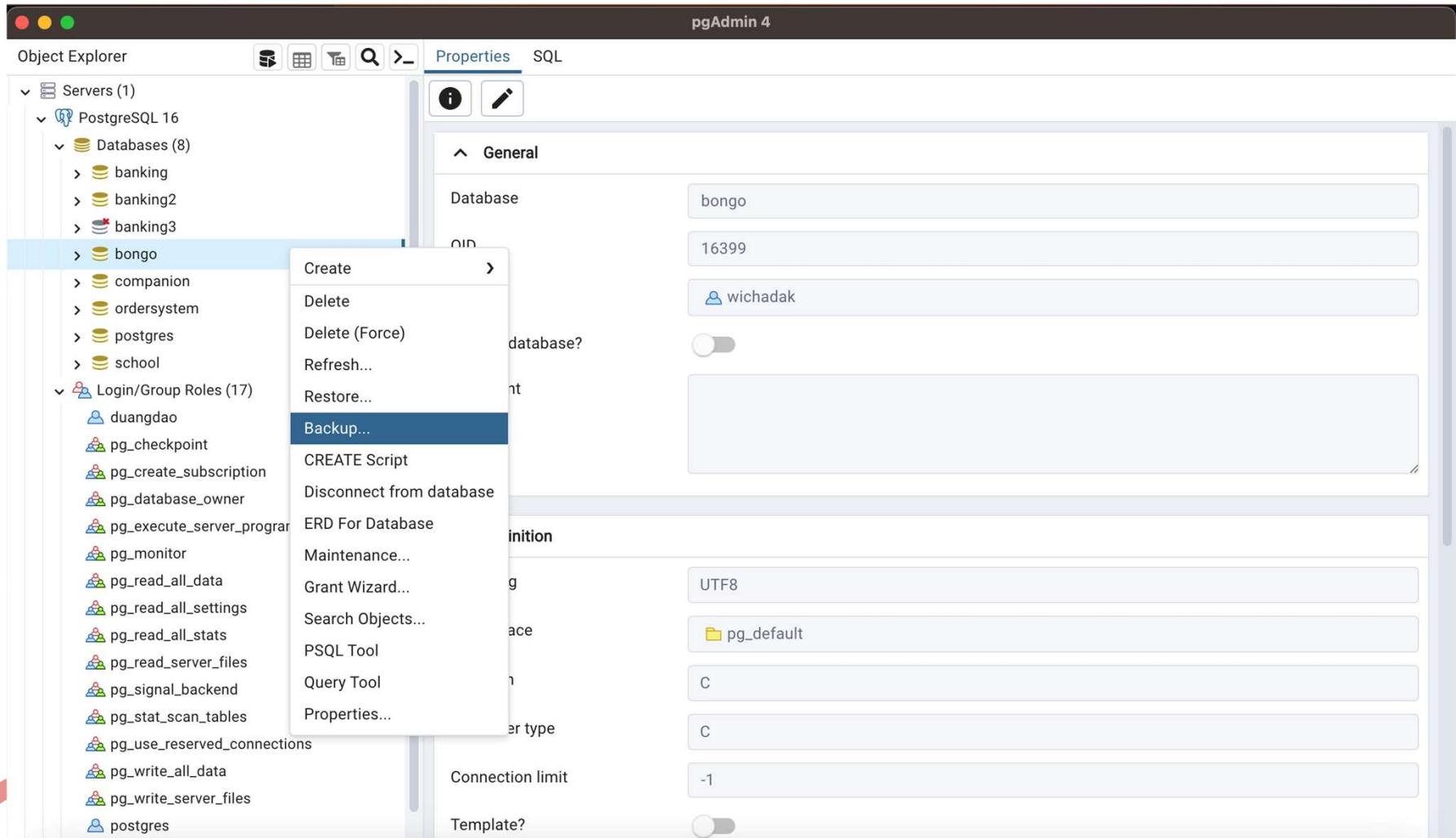
Database maintenance



A wide-angle photograph of an underwater ecosystem. A massive school of small, silvery fish swims through a dense forest of tall, yellowish-green kelp. The water is a deep, translucent blue. In the lower right foreground, a large, textured rock or piece of driftwood rests on the ocean floor, partially covered in green moss or algae. The scene is bathed in natural sunlight filtering down from the surface.

Database backup/restore

Database backup



Database backup

The screenshot shows the pgAdmin 4 interface with the title bar "pgAdmin 4". The left sidebar is titled "Object Explorer" and lists "Servers (1)", "PostgreSQL 16", "Databases (9)", "bongo" (selected), "bongo2", and "Login/Group Roles (17)". The main area is titled "Backup (Database: bongo)" and contains tabs for "General", "Data Options", "Query Options", "Table Options", and "Options". The "General" tab is selected, showing fields for "Filename" (set to "/Users/duangdaowichadakul/Desktop/2110322_DBSYS_2566_1/bongo_v"), "Format" (set to "Custom"), "Compression ratio", "Encoding" (set to "Select an item..."), "Number of jobs", and "Role name" (set to "wichadak"). To the right of the main window is a sidebar titled "Start Time" with a list of backup start times from October 15, 2023, at 8:33:26 to 8:21:25.

| Start Time |
|------------------------|
| 10/15/2023, 8:33:26... |
| 10/15/2023, 8:30:18... |
| 10/15/2023, 8:26:42... |
| 10/15/2023, 8:23:16... |
| 10/15/2023, 8:22:19... |
| 10/15/2023, 8:21:25... |

Database backup

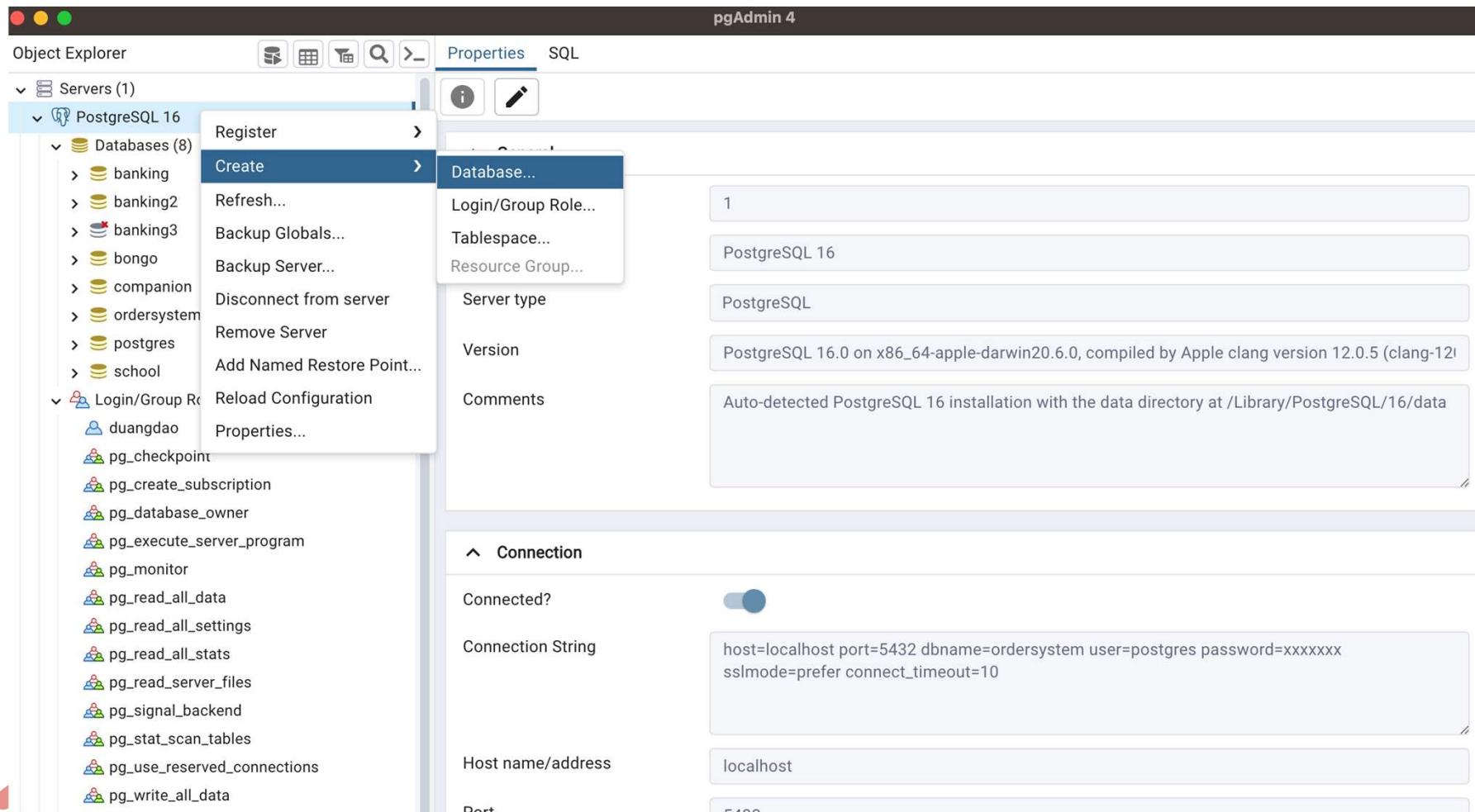
The screenshot shows the pgAdmin 4 interface. In the Object Explorer, under 'Servers (1) > PostgreSQL 16 > Databases (8)', the 'bongo' database is selected. The Properties tab is active, displaying the following details:

- General**: Database: bongo, OID: 16399, Owner: wichadak, System database?: Off, Comment: (empty)
- Definition**: Encoding: UTF8, Tablespace: pg, Collation: C, Character type: C, Connection limit: -1, Template?: (empty)

Two notifications are visible at the bottom right:

- Process completed**: Backing up an object on the server 'PostgreSQL 16 (localhost:5432)' from database 'bongo'. [View Processes](#)
- Process started**: Backing up an object on the server 'PostgreSQL 16 (localhost:5432)' from database 'bongo'. [View Processes](#)

Database restore



Database restore

The screenshot shows the pgAdmin 4 interface. In the Object Explorer, a database named 'bongo2' is selected under the 'Databases' section of the 'PostgreSQL 16' server. A context menu is open over this database, with the 'Restore...' option highlighted in blue. The 'Properties' tab is active in the top navigation bar. The main panel displays the 'General' properties for the database, including its name ('bongo2'), OID ('24981'), owner ('wichadak'), and connection limit ('-1').

Object Explorer

pgAdmin 4

Servers (1)

PostgreSQL 16

Databases (9)

- banking
- banking2
- banking3
- bongo
- bongo2**
- Casts
- Catalogs
- Event Triggers
- Extensions
- Foreign Data Wrappers
- Languages
- Publications
- Schemas
- Subscriptions

companion

ordersystem

postgres

school

Login/Group Roles (17)

- duangdao
- pg_checkpoint
- pg_create_subscription
- pg_database_owner
- pg_execute_server_program

Properties

SQL

General

Database: bongo2

OID: 24981

Owner: wichadak

Connection limit: -1

Create

- Delete
- Delete (Force)
- Refresh...
- Restore...**
- Backup...
- CREATE Script
- Disconnect from database
- ERD For Database
- Maintenance...
- Grant Wizard...
- Search Objects...
- PSQL Tool
- Query Tool
- Properties...

Database restore

The screenshot shows the pgAdmin 4 interface with the title bar "pgAdmin 4". The left sidebar is titled "Object Explorer" and lists "Servers (1)", "PostgreSQL 16", "Databases (9)", "bongo2", and "Login/Group Roles (17)". The "bongo2" node is selected and highlighted with a blue background. A context menu is open over the "bongo2" node, showing options like "Restore", "Copy", "Delete", and "Properties". The main pane displays the "Restore (Database: bongo2)" dialog. The "General" tab is selected, showing the following fields:

- Format: Custom or tar
- Filename: /Users/duangdaowichadakul/Desktop/2110322_DBSYS_2566_1/bongo_v
- Number of jobs: (empty)
- Role name: wichadak

Below the dialog are three buttons: "Close", "Reset", and a large blue "Restore" button. To the right of the dialog, a table lists recent restore operations:

| Start Time |
|------------------------|
| 10/15/2023, 8:38:25... |
| 10/15/2023, 8:35:17... |
| 10/15/2023, 8:33:26... |
| 10/15/2023, 8:30:18... |
| 10/15/2023, 8:26:42... |
| 10/15/2023, 8:23:16... |
| 10/15/2023, 8:22:19... |
| 10/15/2023, 8:21:25... |

Hot backup and cold backup

- **Hot backups** สามารถ backup ในระหว่างที่ระบบทำงานอยู่ ไม่ต้องหยุดฐานข้อมูลก่อน ในการนี้ของ MySQL จะทำได้ถ้าเป็น ***MySQL Enterprise Backup***
- **Cold backups** จะต้องทำการ shutdown เซิร์ฟเวอร์ของ MySQL ก่อน โดยแบ่งออกเป็น อีก 2 ประเภทคือ
 - Physical backup และ
 - Logical backup

Physical and Logical backups

- **Physical backups** ประกอบด้วย raw copies ของไดเรคทอรีและไฟล์ที่เก็บข้อมูลในฐานข้อมูล
 - มีประโยชน์ในการนิ่งของการสำรองฐานข้อมูลที่มีขนาดใหญ่ที่จำเป็นต้องสามารถกู้คืนได้อย่างรวดเร็ว
- **Logical backups** จะเก็บข้อมูลในลักษณะของชุดคำสั่งที่แสดงโครงสร้างและตารางในฐานข้อมูล เช่นคำสั่ง CREATE DATABASE, CREATE TABLE รวมทั้งคำสั่ง INSERT เป็นต้น
 - การสำรองข้อมูลลักษณะนี้ เหมาะกับฐานข้อมูลที่มีข้อมูลไม่มาก ทำให้เราสามารถปรับแก้ค่าในตาราง หรือโครงสร้างของตารางในฐานข้อมูลได้ง่าย หรือเมื่อเราต้องการสร้างฐานข้อมูลเดียวกันในเครื่องอื่นๆ

ขั้นตอนการทำ Physical backup

1. ทำการ slow shutdown ตัวเซิร์ฟเวอร์ของ MySQL โดยต้องตรวจสอบว่าไม่มี errors เกิดขึ้น
2. Copy ไฟล์ข้อมูล InnoDB -> ไฟล์ ibdata และ ไฟล์ .ibd ทั้งหลาย ไปยังที่ปลอดภัย
3. Copy ไฟล์ล็อกของ InnoDB -> ไฟล์ ib_logfile ทั้งหลาย ไปยังที่ปลอดภัย
4. Copy ไฟล์คอนฟิก my.conf ไปยังที่ปลอดภัย

สำหรับการทำ Logical backup สามารถทำได้ผ่านคำสั่ง mysqldump

<https://dev.mysql.com/doc/mysql-backup-excerpt/8.0/en/innodb-backup.html>

MySQLWorkbench

localhost

SQL File 9* × banking ×

Tables Columns Indexes Triggers Views Stored Procedures Functions Events

| Name | Engine | Version | Row Format | Rows | Avg Row Length | Data Length | Max Data Length | Index Length |
|-------------|--------|---------|------------|------|----------------|-------------|-----------------|--------------|
| TriggerTime | InnoDB | 10 | Compact | 1 | 16384 | 16384 | 0 | 0 |
| account | InnoDB | 10 | Compact | 7 | 2340 | 16384 | 0 | 0 |
| borrower | InnoDB | 10 | Compact | 6 | 2730 | 16384 | 0 | 0 |
| branch | InnoDB | 10 | Compact | 6 | 2730 | 16384 | 0 | 0 |
| customer | InnoDB | 10 | Compact | 6 | 2730 | 16384 | 0 | 0 |
| depositor | InnoDB | 10 | Compact | 7 | 2340 | 16384 | 0 | 0 |
| loan | InnoDB | 10 | Compact | 6 | 2730 | 16384 | 0 | 0 |

MANAGEMENT

- Server Status
- Client Connections
- Users and Privileges
- Status and System Variables
- Data Export
- Data Import/Restore

INSTANCE

- Startup / Shutdown
- Server Logs
- Options File

SCHEMAS

- Filter Objects
- amramudb
- banking** (selected)
- banking_1
- boatalrental
- bongo
- ONLINESHOI
- ordersystem
- testdb

Schema Inspector

- Set as Default Schema
- Filter to This Schema
- Copy to Clipboard
- Send to SQL Editor
- Create Schema...
- Alter Schema...
- Drop Schema...
- Search Table Data...
- Refresh All

Maintenance > Refresh

Action Response Duration / Fetch Time

| | | |
|-----------------------|-------------------|-----------------------|
| SHOW PROCEDURE STATUS | 5 row(s) returned | 0.002 sec / 0.000 sec |
| SHOW FUNCTION STATUS | 1 row(s) returned | 0.002 sec / 0.000 sec |

MySQL Workbench

localhost

File Edit View Query Database Server Tools Scripting Help

MySQL Workbench

MANAGEMENT

- Server Status
- Client Connections
- Users and Privileges
- Status and System Variables
- Data Export
- Data Import/Restore

INSTANCE

- Startup / Shutdown
- Server Logs
- Options File

SCHEMAS

- amramudb
- banking**
- banking_1
- boatrental
- bongo
- ONLINESHOPPING
- ordersystem
- testdb

SQL File 10 x Administration - Data Import/Restore # banking x

Tables Columns Indexes Triggers Views Stored Procedures Functions Events

| Name | Engine |
|----------------|--------|
| TriggerTime | InnoDB |
| account | InnoDB |
| borrower | InnoDB |
| branch | InnoDB |
| customer | InnoDB |
| depositor | InnoDB |
| loan | InnoDB |

Table Maintenance Operations

Select tables and click the operation you want to perform.
NOTE: Some commands may require locking tables until completion, which may take a long time for large tables.

Analyze Table

Analyzes and stores the key distribution for a table.
During the analysis, the table is locked with a read lock for InnoDB and MyISAM.

Don't write to BINLOG (local)

Analyze Table

Optimize Table

Reorganizes the physical storage of table data and associated index data, to reduce storage space and improve I/O efficiency when accessing the table.

Optimize FULLTEXT only Number of words to optimize per run: 2000

Don't write to BINLOG (local)

Optimize Table

Check Table

CHECK TABLE checks a table or tables for errors.
For MyISAM tables, the key statistics are updated as well.

Fast Changed

Check Table

Checksum Table

CHECKSUM TABLE reports a checksum for the contents of a table.

Quick (if supported)

Checksum Table

< Summary List

Server / Data Export

The screenshot shows the MySQL Workbench interface with a yellow header bar containing the text "Server / Data Export". The main window is titled "Data Export" and displays a tree view of database objects under "Select Database Objects to Export". The "Export Schema" tab is selected, showing checkboxes for various schema objects. Several checkboxes are checked, including "TriggerTime", "account", "borrower", "branch", "customer", "depositor", and "loan". Below the tree view, there are buttons for "Refresh" and "7 tables selected". To the right, there are buttons for "Select Tables" and "Unselect All". The "Options" section contains two radio buttons: "Export to Dump Project Folder" (selected) and "Export to Self-Contained File". The "Export to Dump Project Folder" option has a sub-section explaining it allows selective restore but may be slower, with a field set to "/Users/wichadak/dumps/Dump20201003". The "Export to Self-Contained File" option has a field set to "/Users/wichadak/dumps/Dump20201003.sql". Other options in the "Options" section include "Create Dump in a Single Transaction (self-contained file only)" (unchecked), "Dump Stored Routines (Procedures and Functions)" (checked), "Dump Events" (unchecked), and "Skip table data (no-data)" (unchecked). At the bottom, a message says "Press [Start Export] to start..." and a "Start Export" button is visible. The left sidebar shows the MySQL Workbench navigation pane with sections like MANAGEMENT, INSTANCE, and SCHEMAS.

Server / Data Export

Data Export

Select Database Objects to Export

Export Schema

- ONLINESHOPPING
- amramudb
- banking
- banking_1
- boatalental
- bongo
- mysql
- ordersystem
- performance_schema
- testdb

Refresh 7 tables selected Select Tables Unselect All

Options

Export to Dump Project Folder /Users/wichadak/dumps/Dump20201003 ...

Each table will be exported into a separate file. This allows a selective restore, but may be slower.

Export to Self-Contained File /Users/wichadak/dumps/Dump20201003.sql ...

All selected database objects will be exported into a single, self-contained file.

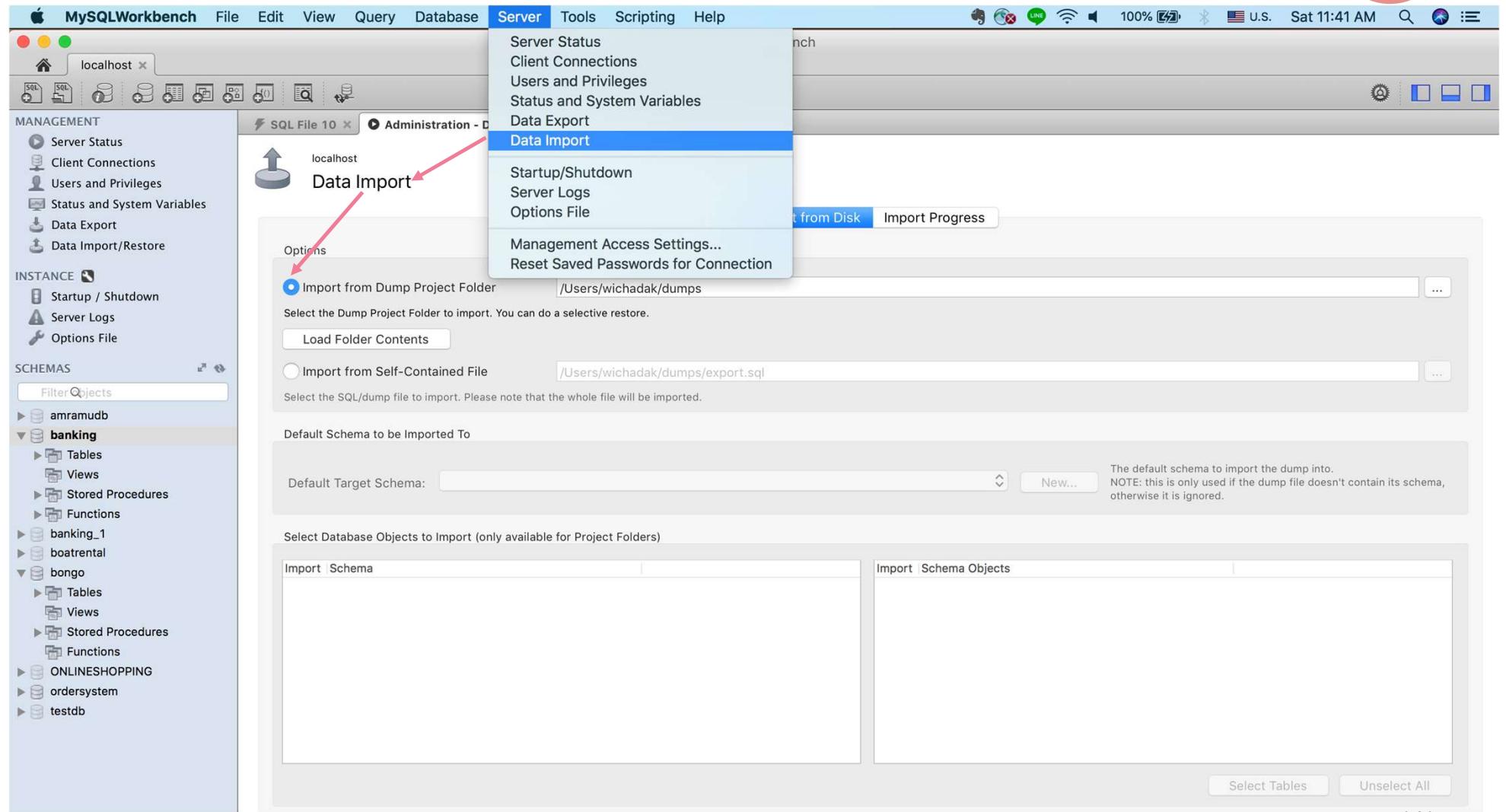
Create Dump in a Single Transaction (self-contained file only)

Dump Stored Routines (Procedures and Functions)

Dump Events

Skip table data (no-data)

Press [Start Export] to start... Start Export



Conclusion

SQL DDL (Data Definition Language)

SQL DML (Data Manipulation Language)

SPL Stored Routines

- Stored Procedure
- Stored Function
- Trigger

ทดลอง explain analyze <<sql command>>

The screenshot shows the pgAdmin 4 interface. On the left, the Object Browser displays the database structure under 'Databases (3)'. The 'banking' database is selected. The 'ordersystem' database contains the following objects: Casts, Catalogs, Event Triggers, Extensions, Foreign Data Wrappers, Languages, Publications, Schemas, and Subscriptions. The 'postres' database contains Login/Group Roles and Tablespaces.

In the main window, the 'Query' tab is active, showing the following SQL code:

```
1 explain select * from account;
2
3 explain analyze select * from account;
```

The 'Data Output' tab is selected, displaying the execution plan and timing information:

| | QUERY PLAN |
|---|--|
| 1 | Seq Scan on account (cost=0.00..14.90 rows=490 width=136) (actual time=7.915..7.921 rows=6 loops=... |
| 2 | Planning Time: 0.068 ms |
| 3 | Execution Time: 7.941 ms |



ย้อนกลับไปทำ **Exercise 1, 2**

และ

MCV Assignment Exercise#5

MySQL Storage Engine

- MyISAM : tables are optimized for compression and speed
- InnoDB : tables fully support ACID-compliant and transactions
- MERGE : virtual table w/ multiple MyISAM tables that have a similar structure
- Memory: tables stored in memory w/ hash indexes
- Archive: the engine will compress and decompress when INSERT and SELECT and w/o indexes
- CSV: stores data in comma-separated values file format

<https://www.mysqltutorial.org/understand-mysql-table-types-innodb-myisam.aspx/>

<https://www.mysqltutorial.org/wp-content/uploads/2018/03/MySQL-Storage-Engines-Feature-Summary.pdf> 145

SQL query: SELECT INTO

- จากตัวอย่างคำสั่ง SQL ก่อนหน้ามีการใช้ SELECT INTO เพื่อใส่ค่าเข้าตัวแปร
- นอกจากนี้ เราสามารถใช้ SELECT ... INTO ในการส่งผลของ SELECT ออกไฟล์ได้ เช่น

```
SELECT * INTO OUTFILE 'test_select_into.txt'  
FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY ''"  
LINES TERMINATED BY '\n'  
FROM customer;
```

Oops!! Access denied!!!

```
mysql> SELECT * INTO OUTFILE 'test_select_into.txt'  
-> FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY ''"  
-> LINES TERMINATED BY '\n'  
-> FROM customer;  
ERROR 1045 (28000): Access denied for user 'wichadak'@'localhost' (using password: YES)
```

SQL query: ไปเป็น root เพื่อเพิ่มเติมสิทธิให้ user wichadak เขียนลงไฟล์ system ได้ด้วย

คำสั่งนี้ไม่รวมสิทธิในการ
เขียนไฟล์ system

```
mysql> GRANT ALL PRIVILEGES ON bongo.* TO 'wichadak'@'localhost';  
mysql> FLUSH PRIVILEGES;
```

ต้องให้สิทธิเพิ่มเติมและมีแต่
แบบ global เท่านั้น

```
mysql> GRANT FILE ON *.* TO 'wichadak'@'localhost';
```

Logout และกลับมาเป็น user wichadak ลอง SELECT INTO ใหม่อีกที

```
mysql> SELECT * INTO OUTFILE '/tmp/test_select_into.csv'  
      -> FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '\"'  
      -> LINES TERMINATED BY '\n'  
      -> FROM customer;  
Query OK, 6 rows affected (0.01 sec)
```

ใจแล้วจ้า :D

SQL query: test_select_into.txt หน้าตาเป็นยังไง

```
"Alan","Mary_street","Y"  
"Jason","Jason_street","N"  
"Joe","Joe_street","Y"for help.  
"Keith","Keith_street","N"  
"Mary","Mary_street","N"  
"Mike","Mary_street","Y"  
mysql> use banking;
```

Big data source

- <https://github.com/datablist/sample-csv-files>