PL/pgSQL
Procedural programming language in PostgreSQL

Jeffry Hartanto

jhartanto@comp.nus.edu.sg

28 September 2021

Announcement

- PL/pgSQL won’t be in the Exam but it will be used for the project.
- Recording from CS1010E for Examplify.
- A non-graded quiz will be released by 1 September tentatively.
- Prof. Adi will announce mid-term rules.

Outline

1. Quick Recap on SQL
2. Motivation
3. Host language + SQL
4. PL/pgSQL Part I
5. PL/pgSQL Part II
6. SQL Injection

01 Quick Recap on SQL

01 Quick Recap on SQL

- So far, we have learnt …

SQL 1

DML

ALTER

CREATE

DROP

CONSTRAINT

DDL

INSERT

SELECT

UPDATE

DELETE

SQL 2

LIMIT

ORDER BY

NATURAL JOIN

LEFT/RIGHT JOIN

UNION

FULL JOIN

EXCEPT

INTERSECT

EXISTS

IN

ANY

SCALAR SUBQUERY

SQL 3

GROUP BY

HAVING

MAX

AVG

MIN

CASE

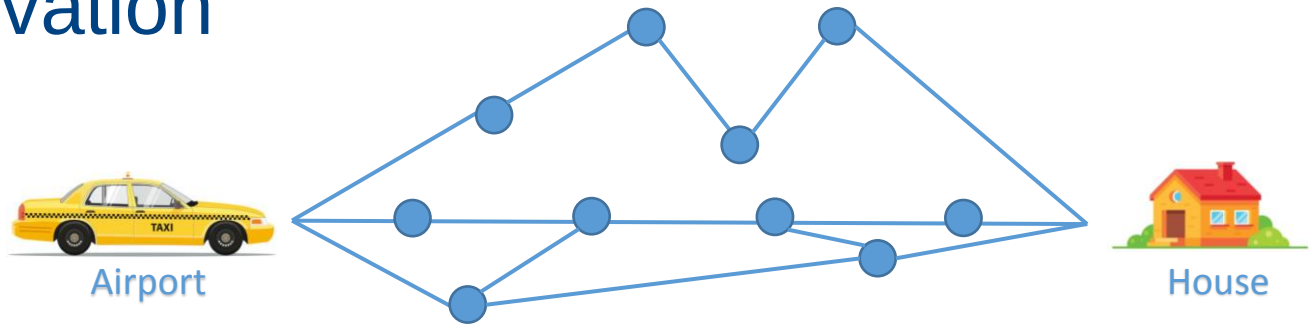
COALESCE

CTE

VIEWS

02 Motivation

02 Motivation



Declarative vs. Procedural

- Declarative specifies the “what”, whereas Procedural specifies the “how”.
- Declarative was traditionally slower than Procedural, but this is changing.
- Declarative tends to require less lines of codes for solving a generic query as compared to Procedural.
- Declarative may require a complex solution for solving a very specific query as compared to Procedural.

02 Motivation

Based on this ranking system of cryptocurrencies,
I want to have daily record of *first three coins* from the TOP 10 cryptocurrencies that are *down by more than 5% in the past 7 days* and are *within 2 ranks apart* from each other.

We will do it!

Rank	Symbol	Changes
1	BTC	-6%
2	ETH	+3%
3	DOGE	-6%
4	ZIL	+10%
5	XMR	-1%
6	SHIB	-8%
7	ADA	+1%
8	LTC	-7%
9	XRP	-7%
10	BNB	-6%

Rank	Symbol	Changes
6	SHIB	-8%
8	LTC	-7%
9	XRP	-7%

Possible to use SQL?

Any easier way?

02 Motivation

Based on this ranking system of cryptocurrencies,
I want to have daily record of *first three coins* from the TOP 10 cryptocurrencies that are *down by more than 5% in the past 7 days* and are *within 2 ranks apart* from each other.

We will do it!

Generally, it is *easier* to use a *procedural language* for problems that require *very specific* traversal of the data.

Two possible solutions:

Host language + SQL
(Java, C, Python, etc.)

PL/pgSQL

03 Host language + SQL

03 Host language + SQL

- Let's use C language as an example.
- There are two types of mixing:

Statement-level Interface

C
+ SQL

Call-level Interface

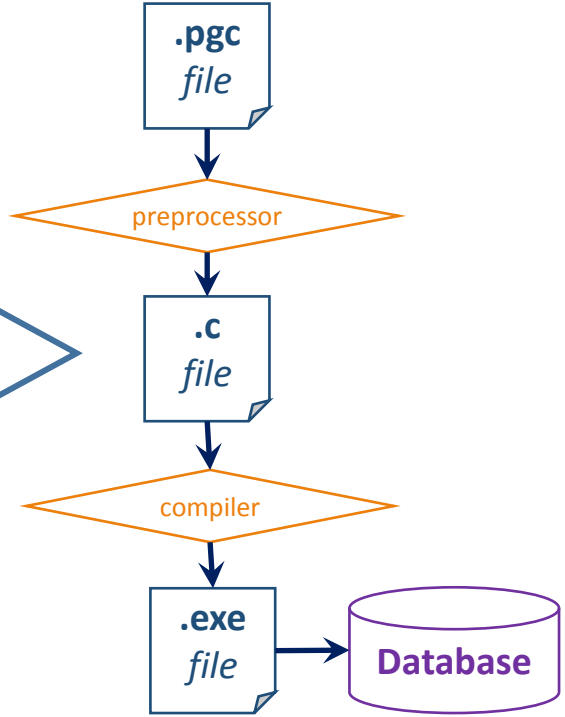
C
only

03 Statement-level Interface

C
+ SQL

Basic idea

1. **Write** a program that mixes host language with SQL.
2. **Preprocess** the program using a preprocessor.
3. **Compile** the program into an executable code.



03 Statement-level Interface

C
+ SQL

"Scores"

Name	Mark
Alice	92
...	...

.pgc
file

```
void main() {
    EXEC SQL BEGIN DECLARE SECTION;
      char name[30]; int mark;
    EXEC SQL END DECLARE SECTION;
    EXEC SQL CONNECT @localhost USER john;
    // some code that assigns values to
    // name and mark.
    EXEC SQL INSERT INTO
      Scores (Name, Mark) VALUES (:name, :mark);
    EXEC SQL DISCONNECT;
}
```

Declaration

Connection

Host language

Query execution

Disconnect

The SQL query above is fixed, i.e., **static SQL**.
Can we generate the SQL query during runtime?
Yes, it is called **Dynamic SQL**.

03 Statement-level Interface

C
+ SQL

"Scores"

Name	Mark
Alice	92
...	...

.pgc
file

```
void main() {
```

```
EXEC SQL BEGIN DECLARE SECTION;
  char *query; char name[30]; int mark;
EXEC SQL END DECLARE SECTION;
```

Declaration

```
EXEC SQL CONNECT @localhost USER john;
```

Connection

```
// some code that assigns values to
// name and mark

// assign any SQL statement to the query,
// the query may include name and/or mark.
```

Host language

```
EXEC SQL EXECUTE IMMEDIATE :query;
```

Query execution

```
EXEC SQL DISCONNECT;
```

Disconnect

```
}
```

03 Statement-level Interface

C
+ SQL

"Scores"

Name	Mark
Alice	92
...	...

.pgc
file

```
void main() {
    EXEC SQL BEGIN DECLARE SECTION;
      const char *query = "INSERT INTO Scores
                           VALUES(?, ?);";
      char name[30]; int mark;
    EXEC SQL END DECLARE SECTION;

    EXEC SQL CONNECT @localhost USER john;

    // some code that assigns values to
    // name and mark, or modify query. Then,

    EXEC SQL PREPARE stmt FROM :query;
    EXEC SQL EXECUTE stmt USING :name, :mark;

    EXEC SQL DEALLOCATE PREPARE stmt;
    EXEC SQL DISCONNECT;
}
```

What if we want to use C only?

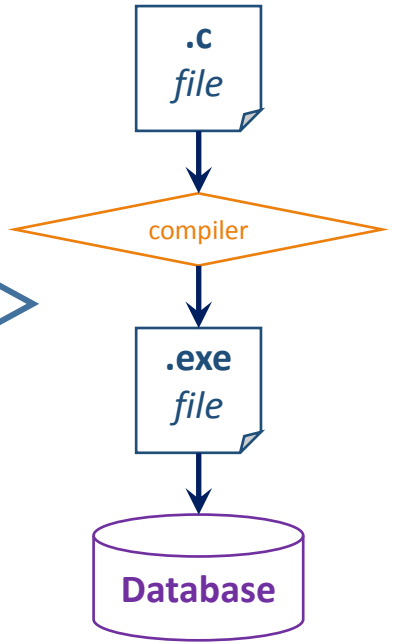
03 Call-level Interface

C
only

Basic idea

1. **Write** in host language only*.
2. **Compile** the program into an executable code.

*Need to load a library that provides APIs to access the DB.
The libraries are libpq, psqlODBC, JDBC, ODBC, etc.



03 Call-level Interface

C *only*

"Scores"

Name	Mark
Alice	92
...	...

.c *file*

```
void main() {
```

```
char *query; char name[30]; int mark;
```

Declaration

```
connection C("dbname = testdb user = postgres \
  password = test hostaddr = 127.0.0.1 \
  port = 5432");
```

Connection

```
// assign any SQL statement to the query,
// the query may include name and/or mark.

work W(C);
W.exec(query);
W.commit();
```

Query execution

```
C.disconnect();
```

Disconnect

```
}
```

Quick Quiz

Is this more like a static or dynamic SQL?

03 Call-level Interface

C *only*

"Scores"

Name	Mark
Alice	92
...	...

.c *file*

```
void main() {
```

```
char *query; char name[30]; int mark;
```

Declaration

```
connection C("dbname = testdb user = postgres \
  password = test hostaddr = 127.0.0.1 \
  port = 5432");
```

Connection

```
// assign any SQL statement to the query,
// the query may include name and/or mark.

work W(C);
W.exec(query);
W.commit();
```

Query execution

```
C.disconnect();
```

Disconnect

```
}
```

Quick Quiz

What's the pros of using this instead of statement-level interface?

03 Summary

- Statement-level Interface

C
+ SQL

  - Code is written in a mix of host language and SQL.
    - Static SQL has fixed queries.
    - Dynamic SQL generates queries at runtime.
  - Code is pre-processed before compiled into an executable program.

- Call-level Interface

C
only

  - Code is written only in host language.
    - Need a library that provides APIs to run the SQL queries.
  - Code is directly compiled into an executable program.

What if we want to use SQL only?

04 PL/pgSQL Part I

04 PL/pgSQL

- SQL-based Procedural Language
  - Server-side Programming
  - ISO standard: SQL/PSM (Persistent Stored Modules).
  - It standardizes syntax and semantics of SQL Procedural Language.
  - Unfortunately, different vendors have different implementations:
    - Oracle `PL/SQL`
    - PostgreSQL `PL/pgSQL`
    - SQL Server `TransactSQL`

Let's learn a new programming language!

04 PL/pgSQL

SQL *only*

- Why do we want to use this?
  - Code reuse.
  - Ease of maintenance.
  - Performance.
  - Security (will be discussed near the end).

SQL *only*

Functions and Procedures

04 Functions

SQL *only* ▶

Converts number marks to letter grades.

Grading scheme:

[70, 100] → A
[60, 70) → B
[50, 60) → C
[0, 50) → F

"Scores"

Name	Mark
Alice	92
Bob	63
Cathy	58
David	47

"Scores"

Name	Grade
Alice	A
Bob	B
Cathy	C
David	F

Quick Quiz

Can we do this with a SQL query?

04 Functions

SQL *only* ▶

"Scores"

Name	Mark
Alice	92
Bob	63
Cathy	58
David	47

[70, 100] → A
[60, 70) → B
[50, 60) → C
[0, 50) → F

Name	Grade
Alice	A
Bob	B
Cathy	C
David	F

```
SELECT Name, CASE
    WHEN Mark >= 70 THEN 'A'
    WHEN Mark >= 60 THEN 'B'
    WHEN Mark >= 50 THEN 'C'
    ELSE 'F'
  END AS Grade
FROM Scores;
```

Can we abstract away the conversion with a function?

04 Functions

SQL *only*

General Syntax

Function name

Function parameters and type, if any

```
CREATE OR REPLACE FUNCTION <name>
  (<param> <type>, <param> <type>, ...)
RETURNS <type> AS $$

  <Code>

$$ LANGUAGE sql;
```

Return type

Enclosed within dollar-quote or '

Main body of the function

The language used by the code

`<type>`: all data types in SQL, a tuple, a set of tuples, custom tuples, triggers.

04 Functions

SQL *only* ▶

How do we abstract away the conversion with a function?

```
CREATE OR REPLACE FUNCTION convert(Mark INT)
RETURNS CHAR(1) AS $$
  SELECT CASE
      WHEN Mark >= 70 THEN 'A'
      WHEN Mark >= 60 THEN 'B'
      WHEN Mark >= 50 THEN 'C'
      ELSE 'F'
  END;
$$ LANGUAGE sql;
```

```
-- Call the function
SELECT convert(66);
SELECT * FROM convert(66);
```

Quick Quiz

What is the output of this SQL query?

04 Functions

SQL *only*

"Scores"

Name	Mark
Alice	92
Bob	63
Cathy	58
David	47

[70, 100] → A
[60, 70) → B
[50, 60) → C
[0, 50) → F

"Scores"

Name	Grade
Alice	A
Bob	B
Cathy	C
David	F

```
CREATE OR REPLACE FUNCTION convert(Mark INT)
RETURNS CHAR(1) AS $$
  SELECT CASE
      WHEN Mark >= 70 THEN 'A'
      WHEN Mark >= 60 THEN 'B'
      WHEN Mark >= 50 THEN 'C'
      ELSE 'F'
  END;
$$ LANGUAGE sql;
```

```
SELECT Name, … AS Grade FROM Scores;
```

Quick Quiz

Fill in the blank...

04 Functions

SQL *only*

"Scores"

Name	Mark
Alice	92
Bob	63
Cathy	58
David	47

[75, 100] → A
[65, 75) → B
[50, 65) → C
[0, 50) → F

Name	Grade
Alice	A
Bob	B
Cathy	C
David	F

```
CREATE OR REPLACE FUNCTION convert
  (Mark INT)
RETURNS CHAR(1) AS $$
  SELECT CASE
      WHEN Mark >= 70 THEN 'A'
      WHEN Mark >= 60 THEN 'B'
      WHEN Mark >= 50 THEN 'C'
      ELSE 'F'
    END;
$$ LANGUAGE sql;
```

- Code reuse.
- Ease of maintenance.
- Performance.

```
SELECT Name, convert(Mark) FROM Scores;
SELECT Name
FROM Scores WHERE convert(Mark) = 'B';
```

Quick Quiz

What is the output of this SQL query?

04 Functions

SQL *only*

"Scores"

Name	Mark
Alice	92
Bob	63
Cathy	58
David	47

[**75**, 100] → A
[**65**, **75**) → B
[50, **65**) → C
[0, 50) → F

Name	Grade
Alice	A
Bob	B
Cathy	C
David	F

```
CREATE OR REPLACE FUNCTION convert
  (Mark INT)
RETURNS CHAR(1) AS $$
  SELECT CASE
      WHEN Mark >= 75 THEN 'A'
      WHEN Mark >= 65 THEN 'B'
      WHEN Mark >= 50 THEN 'C'
      ELSE 'F'
    END;
$$ LANGUAGE sql;
```

- Code reuse.
- Ease of maintenance.
- Performance.

```
SELECT Name, convert(Mark) FROM Scores;
SELECT Name
FROM Scores WHERE convert(Mark) = 'B';
```

04 Functions

SQL *only*

"Scores"

Name	Mark
Alice	92
Bob	63
Cathy	58
David	47

[75, 100] → A
[65, 75) → B
[50, 65) → C
[0, 50) → F

Name	Grade
Alice	A
Bob	B
Cathy	C
David	F

```
CREATE OR REPLACE FUNCTION convert
  (Mark INT)
RETURNS CHAR(1) AS $$
  SELECT CASE
      WHEN Mark >= 75 THEN 'A'
      WHEN Mark >= 65 THEN 'B'
      WHEN Mark >= 50 THEN 'C'
      ELSE 'F'
    END;
$$ LANGUAGE sql;
```

compiled

- Code reuse.
- Ease of maintenance.
- Performance.

```
SELECT Name, convert(Mark) FROM Scores;
SELECT Name
FROM Scores WHERE convert(Mark) = 'B';
```

How do we return a tuple from a function?

04 Functions

"Scores"

Name	Mark
Alice	92
Bob	63
Cathy	58
David	47

```
CREATE OR REPLACE FUNCTION convert
  (Mark INT)
RETURNS CHAR(1) AS $$
  SELECT CASE
      WHEN Mark >= 75 THEN 'A'
      WHEN Mark >= 65 THEN 'B'
      WHEN Mark >= 50 THEN 'C'
      ELSE 'F'
    END;
$$ LANGUAGE sql;
```

SQL *only* ▶

```
CREATE OR REPLACE FUNCTION GradeStudent
  (Grade CHAR(1))
RETURNS Scores AS $$

  SELECT    *
  FROM      Scores
  WHERE     convert(Mark) = Grade
  LIMIT 1;

$$ LANGUAGE sql;
```

```
SELECT GradeStudent('C');
```

Quick Quiz

What is the output of this SQL query? What if I remove the LIMIT?

04 Functions

"Scores"

Name	Mark
Alice	92
Bob	63
Cathy	58
David	47

```
CREATE OR REPLACE FUNCTION convert
  (Mark INT)
RETURNS CHAR(1) AS $$
  SELECT CASE
      WHEN Mark >= 75 THEN 'A'
      WHEN Mark >= 65 THEN 'B'
      WHEN Mark >= 50 THEN 'C'
      ELSE 'F'
    END;
$$ LANGUAGE sql;
```

SQL *only* ▶

```
CREATE OR REPLACE FUNCTION GradeStudent
  (Grade CHAR(1))
RETURNS Scores AS $$

  SELECT    *
  FROM      Scores
  WHERE     convert(Mark) = Grade
  LIMIT 1;

$$ LANGUAGE sql;
```

```
SELECT GradeStudent('C');
```

How do we return a set of tuples from a function?

04 Functions

"Scores"

Name	Mark
Alice	92
Bob	63
Cathy	58
David	47

```
CREATE OR REPLACE FUNCTION convert
  (Mark INT)
RETURNS CHAR(1) AS $$
  SELECT CASE
      WHEN Mark >= 75 THEN 'A'
      WHEN Mark >= 65 THEN 'B'
      WHEN Mark >= 50 THEN 'C'
      ELSE 'F'
    END;
$$ LANGUAGE sql;
```

SQL *only* ▶

```
CREATE OR REPLACE FUNCTION GradeStudents
  (Grade CHAR(1))
RETURNS SETOF Scores AS $$

  SELECT  *
  FROM    Scores
  WHERE   convert(Mark) = Grade;

$$ LANGUAGE sql;
```

```
SELECT GradeStudents('C');
```

Quick Quiz

What is the output of this SQL query?

04 Functions

"Scores"

Name	Mark
Alice	92
Bob	63
Cathy	58
David	47

```
CREATE OR REPLACE FUNCTION convert
  (Mark INT)
RETURNS CHAR(1) AS $$
  SELECT CASE
      WHEN Mark >= 75 THEN 'A'
      WHEN Mark >= 65 THEN 'B'
      WHEN Mark >= 50 THEN 'C'
      ELSE 'F'
    END;
$$ LANGUAGE sql;
```

SQL *only* ▶

```
CREATE OR REPLACE FUNCTION GradeStudents
  (Grade CHAR(1))
RETURNS SETOF Scores AS $$

  SELECT  *
  FROM    Scores
  WHERE   convert(Mark) = Grade;

$$ LANGUAGE sql;
```

```
SELECT GradeStudents('C');
```

How do we return a custom tuple from a function?

04 Functions

"Scores"

Name	Mark
Alice	92
Bob	63
Cathy	58
David	47

```
CREATE OR REPLACE FUNCTION convert
  (Mark INT)
RETURNS CHAR(1) AS $$
  SELECT CASE
      WHEN Mark >= 75 THEN 'A'
      WHEN Mark >= 65 THEN 'B'
      WHEN Mark >= 50 THEN 'C'
      ELSE 'F'
    END;
$$ LANGUAGE sql;
```

SQL *only* ▶

Default

```
CREATE OR REPLACE FUNCTION CountGradeStudents
  (IN Grade CHAR(1), OUT Mark CHAR(1), OUT Count INT)
RETURNS RECORD AS $$

  SELECT      Mark, COUNT(*)
  FROM        Scores
  WHERE       convert(Mark) = Grade
  GROUP BY    convert(Mark);

$$ LANGUAGE sql;
```

```
SELECT CountGradeStudents('C');
```

Quick Quiz

What is the output of this SQL query?

04 Functions

"Scores"

Name	Mark
Alice	92
Bob	63
Cathy	58
David	47

```
CREATE OR REPLACE FUNCTION convert
  (Mark INT)
RETURNS CHAR(1) AS $$
  SELECT CASE
      WHEN Mark >= 75 THEN 'A'
      WHEN Mark >= 65 THEN 'B'
      WHEN Mark >= 50 THEN 'C'
      ELSE 'F'
    END;
$$ LANGUAGE sql;
```

SQL *only*

```
CREATE OR REPLACE FUNCTION CountGradeStudents
  (IN Grade CHAR(1), OUT Mark CHAR(1), OUT Count INT)
RETURNS RECORD AS $$

  SELECT      convert(Mark) AS Grade, COUNT(*)
  FROM        Scores
  WHERE       convert(Mark) = Grade
  GROUP BY    convert(Mark);

$$ LANGUAGE sql;
```

```
SELECT CountGradeStudents('C');
```

Quick Quiz

What is the output of this SQL query?

04 Functions

"Scores"

Name	Mark
Alice	92
Bob	63
Cathy	58
David	47

```
CREATE OR REPLACE FUNCTION convert
  (Mark INT)
RETURNS CHAR(1) AS $$
  SELECT CASE
      WHEN Mark >= 75 THEN 'A'
      WHEN Mark >= 65 THEN 'B'
      WHEN Mark >= 50 THEN 'C'
      ELSE 'F'
    END;
$$ LANGUAGE sql;
```

SQL *only*

```
CREATE OR REPLACE FUNCTION CountGradeStudents
  (IN Grade CHAR(1), OUT Mark CHAR(1), OUT Count INT)
RETURNS RECORD AS $$

  SELECT      convert(Mark), COUNT(*)
  FROM        Scores
  WHERE       convert(Mark) = Grade;

$$ LANGUAGE sql;
```

```
SELECT CountGradeStudents('C');
```

Quick Quiz

What is the output of this SQL query?

04 Functions

"Scores"

Name	Mark
Alice	92
Bob	63
Cathy	58
David	47

```
CREATE OR REPLACE FUNCTION convert
  (Mark INT)
RETURNS CHAR(1) AS $$
  SELECT CASE
      WHEN Mark >= 75 THEN 'A'
      WHEN Mark >= 65 THEN 'B'
      WHEN Mark >= 50 THEN 'C'
      ELSE 'F'
    END;
$$ LANGUAGE sql;
```

SQL *only*

```
CREATE OR REPLACE FUNCTION CountGradeStudents
  (IN Grade CHAR(1), OUT Mark CHAR(1), OUT Count INT)
RETURNS RECORD AS $$

  SELECT      Grade, COUNT(*)
  FROM        Scores
  WHERE       convert(Mark) = Grade;

$$ LANGUAGE sql;
```

```
SELECT CountGradeStudents('C');
```

How do we return a set of custom tuples from a function?

04 Functions

"Scores"

Name	Mark
Alice	92
Bob	63
Cathy	58
David	47

```
CREATE OR REPLACE FUNCTION convert
  (Mark INT)
RETURNS CHAR(1) AS $$
  SELECT CASE
      WHEN Mark >= 75 THEN 'A'
      WHEN Mark >= 65 THEN 'B'
      WHEN Mark >= 50 THEN 'C'
      ELSE 'F'
    END;
$$ LANGUAGE sql;
```

SQL *only* ▶

```
CREATE OR REPLACE FUNCTION CountGradeStudents
  (OUT Mark CHAR(1), OUT Count INT)
RETURNS SETOF RECORD AS $$

  SELECT      convert(Mark), COUNT(*)
  FROM        Scores
  GROUP BY    convert(Mark);

$$ LANGUAGE sql;
```

```
SELECT CountGradeStudents('C');
```

Quick Quiz

What is the output of this SQL query?

04 Functions

"Scores"

Name	Mark
Alice	92
Bob	63
Cathy	58
David	47

```
CREATE OR REPLACE FUNCTION convert
  (Mark INT)
RETURNS CHAR(1) AS $$
  SELECT CASE
      WHEN Mark >= 75 THEN 'A'
      WHEN Mark >= 65 THEN 'B'
      WHEN Mark >= 50 THEN 'C'
      ELSE 'F'
    END;
$$ LANGUAGE sql;
```

SQL *only*

```
CREATE OR REPLACE FUNCTION CountGradeStudents
  (OUT Mark CHAR(1), OUT Count INT)
RETURNS SETOF RECORD AS $$

  SELECT      convert(Mark), COUNT(*)
  FROM        Scores
  GROUP BY    convert(Mark);

$$ LANGUAGE sql;
```

```
SELECT CountGradeStudents('C');
```

Can we simplify the params for custom tuples? Yes!

04 Functions

"Scores"

Name	Mark
Alice	92
Bob	63
Cathy	58
David	47

```
CREATE OR REPLACE FUNCTION convert
  (Mark INT)
RETURNS CHAR(1) AS $$
  SELECT CASE
      WHEN Mark >= 75 THEN 'A'
      WHEN Mark >= 65 THEN 'B'
      WHEN Mark >= 50 THEN 'C'
      ELSE 'F'
    END;
$$ LANGUAGE sql;
```

SQL *only*

```
CREATE OR REPLACE FUNCTION CountGradeStudents()
RETURNS TABLE(MARK CHAR(1), COUNT INT) AS $$

  SELECT      convert(Mark), COUNT(*)
  FROM        Scores
  GROUP BY    convert(Mark);

$$ LANGUAGE sql;
```

```
SELECT CountGradeStudents('C');
```

Can the function return "nothing"?

04 Functions

"Scores"

Name	Mark
Alice	92
Bob	63
Cathy	58
David	47

```
CREATE OR REPLACE FUNCTION convert
  (Mark INT)
RETURNS CHAR(1) AS $$
  SELECT CASE
      WHEN Mark >= 75 THEN 'A'
      WHEN Mark >= 65 THEN 'B'
      WHEN Mark >= 50 THEN 'C'
      ELSE 'F'
    END;
$$ LANGUAGE sql;
```

SQL *only* ▶

```
CREATE OR REPLACE FUNCTION UpdateMark
  (IN amount INT)
RETURNS VOID AS $$

  UPDATE Scores SET Mark = Mark + amount;
  ALTER TABLE Scores ADD COLUMN IF NOT EXISTS
    Grade CHAR(1) DEFAULT NULL;
  UPDATE Scores SET Grade = convert(Mark);
  SELECT * FROM Scores;

$$ LANGUAGE sql;
```

Throws an error because Grade is unidentified.

```
SELECT UpdateMark(1);
```

Quick Quiz

Can this function be created?

04 Functions

"Scores"

Name	Mark
Alice	92
Bob	63
Cathy	58
David	47

```
CREATE OR REPLACE FUNCTION convert
  (Mark INT)
RETURNS CHAR(1) AS $$
  SELECT CASE
      WHEN Mark >= 75 THEN 'A'
      WHEN Mark >= 65 THEN 'B'
      WHEN Mark >= 50 THEN 'C'
      ELSE 'F'
    END;
$$ LANGUAGE sql;
```

SQL *only*

```
CREATE OR REPLACE FUNCTION UpdateMark
  (IN amount INT)
RETURNS VOID AS $$

  UPDATE Scores SET Mark = Mark + amount;
  ALTER TABLE Scores ADD COLUMN IF NOT EXISTS
    Grade CHAR(1) DEFAULT NULL;
  SELECT * FROM Scores;

$$ LANGUAGE sql;
```

```
SELECT UpdateMark(1);
```

Can't we use procedure for this? Yes!

04 Procedures

SQL *only*

General Syntax

Procedure name

Procedure parameters and type, if any

```
CREATE OR REPLACE PROCEDURE <name>
  (<param> <type>, <param> <type>, ...)
AS $$

  <Code>

$$ LANGUAGE sql;
```

Enclosed within dollar-quote or '

Main body of the function

The language used by the code

`<type>`: all data types in SQL, a tuple, a set of tuples, custom tuples, triggers.

04 Procedures

"Scores"

Name	Mark
Alice	92
Bob	63
Cathy	58
David	47

```
CREATE OR REPLACE FUNCTION convert
  (Mark INT)
RETURNS CHAR(1) AS $$
  SELECT CASE
      WHEN Mark >= 75 THEN 'A'
      WHEN Mark >= 65 THEN 'B'
      WHEN Mark >= 50 THEN 'C'
      ELSE 'F'
    END;
$$ LANGUAGE sql;
```

SQL *only* ▶

```
CREATE OR REPLACE PROCEDURE UpdateMark
  (IN amount INT)
AS $$

  UPDATE Scores SET Mark = Mark + amount;
  ALTER TABLE Scores ADD COLUMN IF NOT EXISTS
    Grade CHAR(1) DEFAULT NULL;
  SELECT * FROM Scores;

$$ LANGUAGE sql;
```

```
CALL UpdateMark(1);
```

Any question?

04

Summary

SQL *only*

- SQL Functions
  - Returns a value
    - SQL data types
    - Set of existing tuples
    - Set of new tuples
    - Etc..
  - `CREATE OR REPLACE FUNCTION <function_name>(...)`
  - `SELECT <function_name>(...)`
- SQL Procedures
  - No return value
  - `CREATE OR REPLACE PROCEDURE <function_name>(...)`
  - `CALL <function_name>(...)`

05 PL/pgSQL Part II

SQL *only*

Variables and Control Structure

05

PL/pgSQL Part II

SQL *only*

- Previous functions or procedures are limited to executing one or more SQL queries sequentially.
- PL/pgSQL is more powerful than that as it has variables and control structure.
- List of control structure:
  - `IF ... END IF`
  - `IF ... ELSIF ... THEN ... ELSE ... END IF`
  - `EXIT ... WHEN ...`
  - `LOOP ... END LOOP`
  - `WHILE ... LOOP ... END LOOP`
  - `FOR ... IN ... LOOP ... END LOOP`

05 Functions

SQL *only*

General Syntax

```
CREATE OR REPLACE FUNCTION <name>
  (<param> <type>, <param> <type>, ...)
RETURNS <type> AS $$
DECLARE
    … variables …
BEGIN

  <Code>

END;
$$ LANGUAGE plpgsql;
```

The language used by the code

Quick Quiz

How about a procedure?

05 Variables

"Scores"

Name	Mark
Alice	92
Bob	63
Cathy	58
David	47

SQL *only* ▶

```
CREATE OR REPLACE FUNCTION splitMarks
  (IN name1 VARCHAR(20), IN name2 VARCHAR(20),
  OUT mark1 INT, OUT mark2 INT)
RETURNS RECORD AS $$
DECLARE
  temp INT := 0;
BEGIN
  SELECT mark INTO mark1 FROM Scores WHERE name = name1;
  SELECT mark INTO mark2 FROM Scores WHERE name = name2;

  temp := (mark1 + mark2) / 2;

  UPDATE Scores SET mark = temp WHERE name = name1 OR name = name2;
  RETURN; --optional
END;
$$ LANGUAGE plpgsql;
```

```
SELECT splitMarks('Alice', 'Bob');
```

Quick Quiz

What is the output of this query?

05 Variables

"Scores"

Name	Mark
Alice	92
Bob	63
Cathy	58
David	47

SQL *only* ▶

```
CREATE OR REPLACE FUNCTION splitMarks
  (IN name1 VARCHAR(20), IN name2 VARCHAR(20))
RETURNS TABLE(Mark1 INT, Mark2 INT) AS $$
DECLARE
  temp INT := 0;
BEGIN
  SELECT mark INTO mark1 FROM Scores WHERE name = name1;
  SELECT mark INTO mark2 FROM Scores WHERE name = name2;

  temp := (mark1 + mark2) / 2;

  UPDATE Scores SET mark = temp WHERE name = name1 OR name = name2;
  RETURN QUERY SELECT mark1, mark2;
  RETURN NEXT;
END;
$$ LANGUAGE plpgsql;
```

! RETURN NEXT/QUERY does **not** exit the function

```
SELECT splitMarks('Alice', 'Bob');
```

Quick Quiz

What is the output of this query?

05 Selection

"Scores"

Name	Mark
Alice	92
Bob	63
Cathy	58
David	47

SQL *only*

```
CREATE OR REPLACE FUNCTION splitMarks
  (IN name1 VARCHAR(20), IN name2 VARCHAR(20))
RETURNS TABLE(Mark1 INT, Mark2 INT) AS $$
DECLARE
  temp INT := 0;
BEGIN
  -- SELECT statements are omitted.
  temp := (mark1 + mark2) / 2;
  IF temp > 60 THEN     temp := temp / 2;
  ELSIF temp > 50 THEN  temp := temp - 20;
  ELSE                  temp := temp - 10;
  END IF;
  -- UPDATE statement is omitted.
  RETURN QUERY SELECT mark1, mark2;
END;
$$ LANGUAGE plpgsql;
```

```
SELECT splitMarks('Alice', 'Bob');
```

05 Repetition

"Scores"

Name	Mark
Alice	92
Bob	63
Cathy	58
David	47

SQL *only*

```
CREATE OR REPLACE FUNCTION splitMarks
  (IN name1 VARCHAR(20), IN name2 VARCHAR(20))
RETURNS TABLE(Mark1 INT, Mark2 INT) AS $$
DECLARE
  temp INT := 0;
BEGIN
  -- SELECT statements are omitted.
  temp := (mark1 + mark2) / 2;
  WHILE temp > 30 LOOP
    temp := temp / 2;
  END LOOP;
  -- UPDATE statement is omitted.
  RETURN QUERY SELECT mark1, mark2;
END;
$$ LANGUAGE plpgsql;
```

```
SELECT splitMarks('Alice', 'Bob');
```

05 Repetition

"Scores"

Name	Mark
Alice	92
Bob	63
Cathy	58
David	47

SQL *only*

```
CREATE OR REPLACE FUNCTION splitMarks
  (IN name1 VARCHAR(20), IN name2 VARCHAR(20))
RETURNS TABLE(Mark1 INT, Mark2 INT) AS $$
DECLARE
  temp INT := 0;
BEGIN
  -- SELECT statements are omitted.
  temp := (mark1 + mark2) / 2;
  LOOP
    EXIT WHEN temp < 30;
    temp := temp / 2;
  END LOOP;
  -- UPDATE statement is omitted.
  RETURN QUERY SELECT mark1, mark2;
END;
$$ LANGUAGE plpgsql;
```

in Imperative Language ...

```
while (true) {
  if (temp < 30)
    break;
}
```

```
SELECT splitMarks('Alice', 'Bob');
```

05 Repetition

"Scores"

Name	Mark
Alice	92
Bob	63
Cathy	58
David	47

SQL *only*

```
CREATE OR REPLACE FUNCTION splitMarks
  (IN name1 VARCHAR(20), IN name2 VARCHAR(20))
RETURNS TABLE(Mark1 INT, Mark2 INT) AS $$
DECLARE
  temp INT := 0; d INT; denoms INT[] := ARRAY[1, 2, 3];
BEGIN
  -- SELECT statements are omitted.
  temp := (mark1 + mark2) / 2;
  FOREACH d IN ARRAY denoms LOOP
    temp := temp / d;
  END LOOP;
  -- UPDATE statement is omitted.
  RETURN QUERY SELECT mark1, mark2;
END;
$$ LANGUAGE plpgsql;
```

```
SELECT splitMarks('Alice', 'Bob');
```

05 Is that all?

SQL *only*

Based on this ranking system of cryptocurrencies,
I want to have daily record of *first three coins* from the TOP 10 cryptocurrencies that are *down by more than 5% in the past 7 days* and are *within 2 ranks apart* from each other.

We will do it!

Rank	Symbol	Changes
1	BTC	-6%
...	...	...
...	...	...
8	LTC	-7%
9	XRP	-7%
10	BNB	-6%

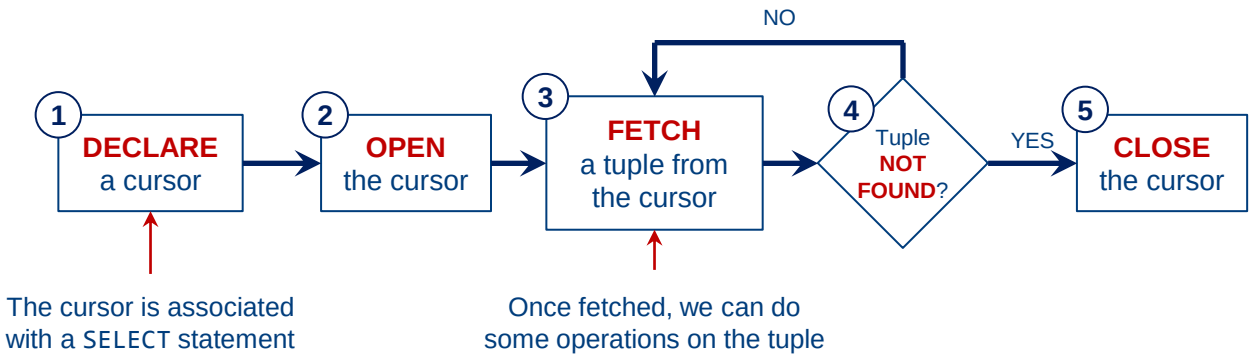
Rank	Symbol	Changes
6	SHIB	-8%
8	LTC	-7%
9	XRP	-7%

How do we traverse a query's result?

Cursor

SQL *only*

- A cursor enables us to access each individual row returned by a SELECT statement
- Workflow:



- Can use other statements at step 3 such as `MOVE, UPDATE, DELETE, etc`.

05 Cursor

SQL *only*

Based on this ranking system of cryptocurrencies,
I want to have daily record of first three consecutive coins from the TOP 10 cryptocurrencies that are *down by more than 5% in the past 7 days.*

Rank	Symbol	Changes
1	BTC	-6%
2	ETH	+3%
3	DOGE	-6%
4	ZIL	+10%
5	XMR	-1%
6	SHIB	-8%
7	ADA	+1%
8	LTC	-7%
9	XRP	-7%
10	BNB	-6%

Idea?

One possible solution:

1. Query the cryptos that are *down by more than 5% in the past 7 days* from the top 10 of the given ranking system.
2. Find the three consecutive coins by traversing (1).

Note that (1) is declarative and (2) is procedural.

05 Cursor

SQL *only* ▶

```
CREATE OR REPLACE FUNCTION consCryptosDown
  (IN num INT)
RETURNS TABLE(rank INT, sym CHAR(4)) AS $$
DECLARE
  curs  CURSOR FOR (SELECT * FROM cryptosRank
                    WHERE changes < -5);
  r1     RECORD;
  r2     RECORD;
BEGIN
  OPEN curs;

  LOOP


    …


  END LOOP;

  CLOSE curs;
END;
$$ LANGUAGE plpgsql;
```

```
FETCH curs INTO r1;
EXIT WHEN NOT FOUND;
FETCH RELATIVE (num-1)
  FROM curs INTO r2;
EXIT WHEN NOT FOUND;

IF r2.rank - r1.rank = 2 THEN
  MOVE RELATIVE -(num) FROM curs;
  FOR c IN 1..num LOOP
    FETCH curs INTO r1;
    rank := r1.rank;
    sym := r1.symbol;
    RETURN NEXT;
  END LOOP;
  CLOSE curs;
  RETURN;
END IF;

MOVE RELATIVE -(num - 1) FROM curs;
```

curs →

Rank	Symbol	Changes
1	BTC	-6%
2	ETH	+3%
3	DOGE	-6%
4	ZIL	+10%
5	XMR	-1%
6	SHIB	-8%
7	ADA	+1%
8	LTC	-7%
9	XRP	-7%
10	BNB	-6%

Cursor

SQL *only*

- Cursor movement
  - `FETCH curs INTO r;`
  - `FETCH NEXT FROM curs INTO r;`
- Other variants
  - `FETCH PRIOR FROM curs INTO r;`
    - Fetch from previous row
  - `FETCH FIRST FROM curs INTO r;`
  - `FETCH LAST FROM curs INTO r;`
  - `FETCH ABSOLUTE 3 FROM curs INTO r;`
    - Fetch the 3rd tuple
  - `FETCH RELATIVE -2 FROM curs INTO r;`
  - `MOVE LAST FROM curs;`
  - `UPDATE/DELETE <table> … WHERE CURRENT OF curs;`

curs →

Rank	Symbol	Changes
1	BTC	-6%
2	ETH	+3%
3	DOGE	-6%
4	ZIL	+10%
5	XMR	-1%
6	SHIB	-8%
7	ADA	+1%
8	LTC	-7%
9	XRP	-7%
10	BNB	-6%

05

Summary

SQL *only*

- `plpgsql` Control Structures
  - Declare `DECLARE <var> <type> BEGIN`
  - Assignment `<var> := ...`
  - Selection `IF ... THEN ... ELSIF ... THEN ... ELSE ... END IF`
  - Repetition `LOOP ... END LOOP`
    `WHILE ... LOOP ... END LOOP`
    - Break `EXIT WHEN ...`
- Cursor
  - Declare → Open → Fetch → Check *(repeat)* → Close
  - `FETCH [PRIOR | FIRST | LAST | ABSOLUTE n | RELATIVE n] [FROM] <cursor> INTO <var>`
  - `MOVE [PRIOR | FIRST | LAST | ABSOLUTE n | RELATIVE n] [FROM] <cursor>;`
  - `[UPDATE | DELETE] … WHERE CURRENT OF <cursor>;`

05 PL/pgSQL - Practice

Based on this ranking system of cryptocurrencies,
I want to have daily record of *first three coins* from the TOP 10 cryptocurrencies that are *down by more than 5% in the past 7 days* and are *within 2 ranks apart* from each other.

We will do it!

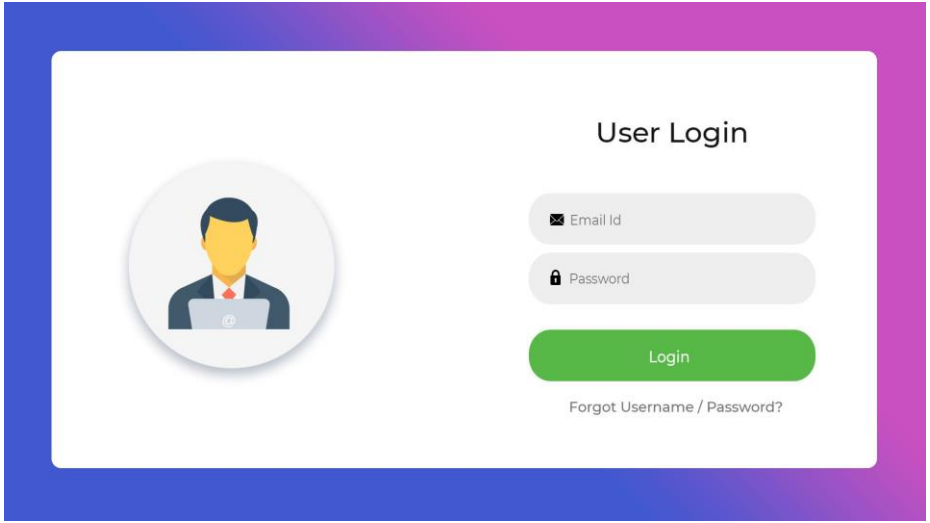
Homework. Any question?

06 SQL Injection

- Code reuse.
- Ease of maintenance.
- Performance.
- Security.

06 SQL Injection

- **What is it?**
  - A class of attacks on dynamic SQL.



06 SQL Injection

- **What is it?**
  - A class of attacks on dynamic SQL.
  - **Expected case**
    - `email = aa@bb.com`
    - `password = abcd`
    - `if (count > 0) { … }`

```
void main() {
    EXEC SQL BEGIN DECLARE SECTION;
      char *query;
    EXEC SQL END DECLARE SECTION;

    EXEC SQL CONNECT TO @localhost USER john;

    char email[100];
    scanf("%s", email);
    char password[100];
    scanf("%s", password);

    //query = "SELECT COUNT(*) FROM Users" +
    //      "WHERE email = '" + name + "'" +
    //      "AND password = '" + password + "';";

    EXEC SQL EXECUTE IMMEDIATE :query;

    EXEC SQL DISCONNECT;
}
```

Generated Query

```
SELECT COUNT(*)
FROM Users
WHERE email  = 'aa@bb.com'
AND password = 'abcd';
```

06 SQL Injection

- **What is it?**
  - A class of attacks on dynamic SQL.
  - **Malicious case**
    - `email = aa@bb.com`
    - `password = 'OR 1 = 1 --`
    - `if (count > 0) { … }`

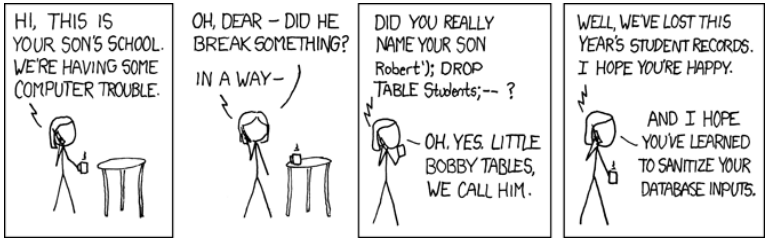
```
void main() {
  EXEC SQL BEGIN DECLARE SECTION;
    char *query;
  EXEC SQL END DECLARE SECTION;

  EXEC SQL CONNECT TO @localhost USER john;

  char email[100];
  scanf("%s", email);
  char password[100];
  scanf("%s", password);

  //query = "SELECT COUNT(*) FROM Users" +
  //      "WHERE email = '" + name + "'" +
  //      "AND password = '" + password + "';";

  EXEC SQL EXECUTE IMMEDIATE :query;

  EXEC SQL DISCONNECT;
}
```

Generated Query

```
SELECT COUNT(*)
FROM Users
WHERE email  = 'aa@bb.com'
AND password = ''
OR 1 = 1 --;
```

06 SQL Injection

- **What is it?**
  - A class of attacks on dynamic SQL.
  - **Malicious case**
    - `email = aa@bb.com`
    - `password = '; DROP TABLE … --`




```
void main() {
EXEC SQL BEGIN DECLARE SECTION;
  char *query;
EXEC SQL END DECLARE SECTION;

EXEC SQL CONNECT TO @localhost USER john;

char email[100];
scanf("%s", email);
char password[100];
scanf("%s", password);

//query = "SELECT COUNT(*) FROM Users" +
//      "WHERE email = '" + name + "'" +
//      "AND password = '" + password + "';";

EXEC SQL EXECUTE IMMEDIATE :query;

EXEC SQL DISCONNECT;
}
```

Generated Query

```
SELECT COUNT(*)
FROM Users
WHERE email  = 'aa@bb.com'
AND password = '';
DROP TABLE … --;
```

06 SQL Injection

- **How to Protect?**
  - Use a function or procedure
  - **Why?**
    - SQL function or procedure is compiled and stored in DB
    - At runtime, anything in `email and password` are treated as strings.

```
void main() {
    EXEC SQL BEGIN DECLARE SECTION;
      char *query;
    EXEC SQL END DECLARE SECTION;

    EXEC SQL CONNECT TO @localhost USER john;

    char email[100];
    scanf("%s", email);
    char password[100];
    scanf("%s", password);

    //query = "SELECT * FROM verifyUser" +
    //      "(" + name + "," + password + ");";

    EXEC SQL EXECUTE IMMEDIATE :query;

    EXEC SQL DISCONNECT;
}
```

```
CREATE OR REPLACE FUNCTION verifyUser
  (IN email_param TEXT, IN password_param TEXT)
RETURNS INT AS $$
  SELECT COUNT(*) FROM Users
  WHERE email  = email_param
  AND password = password_param;
$$ LANGUAGE sql;
```

SQL Injection

- **How to Protect?**
  - Use prepares statements
  - **Why?**
    - SQL query is compiled when it is prepared.
    - At runtime, anything in `email and password` are treated as strings.

```
void main() {
    EXEC SQL BEGIN DECLARE SECTION;
      const char *query = "SELECT COUNT(*)
                           FROM Users
                           WHERE email = ?
                           AND password = ?;";
       char name[100], password[100];
    EXEC SQL END DECLARE SECTION;

    EXEC SQL CONNECT TO @localhost USER john;

    scanf("%s", email);
    scanf("%s", password);

    EXEC SQL PREPARE stmt FROM :query;
    EXEC SQL EXECUTE stmt USING :email, :password;
    EXEC SQL DEALLOCATE PREPARE stmt;
    EXEC SQL DISCONNECT;
}
```

Generated Query

```
SELECT COUNT(*)
FROM Users
WHERE email  = 'aa@bb.com'
AND password = '\' OR 1 = 1 --';
```

Summary

1. Quick Recap on SQL
   - "Generic" queries may be easier to be solved using SQL.
2. Motivation
   - "Specific" queries may be easier to be solved using a procedural language.
3. Host language + SQL
   - Use host procedural language to interact with the database.
4. PL/pgSQL Part I
   - Use SQL procedural language, e.g., function and procedure.
5. PL/pgSQL Part II
   - Use SQL procedural language, e.g., variables, cursor, and control structure.
6. SQL Injection
   - Sanitize user inputs to avoid injection of malicious query.

THANK YOU