

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.
- A non-graded quiz will be released by 1 October, tentatively.
- Refer to the recording from CS1010E for Exemplify.
- Prof. Adi will announce mid-term rules.



Outline

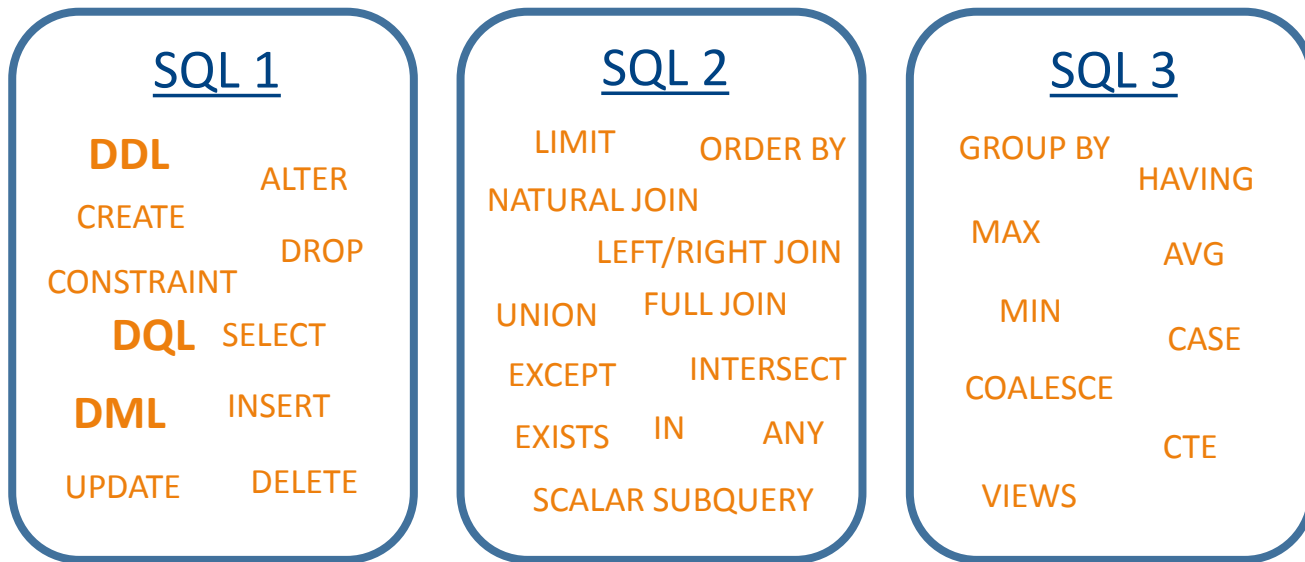
1. Quick Recap on SQL
2. Motivation
3. Host language + SQL
4. PL/pgSQL Part I
5. PL/pgSQL Part II
6. SQL Injection

01

Quick Recap on SQL

Quick Recap on SQL

- So far, we have learnt ...



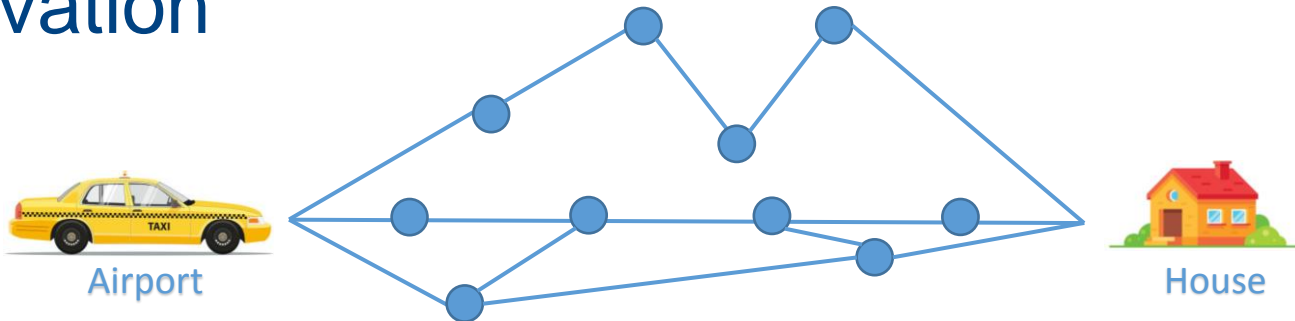
Takeaway: you guys have learnt a lot of SQL constructs, keywords, and statements. But there is still a need to use a procedural language for solving some problems such that PostgreSQL provides PL/pgSQL.

02

Motivation

Tells the taxi driver the *address of the house*, then he/she can figure out the way to the house.

Tells the taxi driver the *step-by-step directions* to the house.



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**.

In the context of SQL, it was slower when the query is complex, i.e., has many joins and/or subqueries.

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.



Highlighted rows are those with < -5% changes.

Since the manager asks for the *first three coins*, then 8, 9, 10 is **NOT** the answer.

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%



Expected output

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



We will do it!

Possible to use SQL?

Possible, but can be very complex.

Any easier way?

Yes, traverse the top 10 coins to find the first three desired coins.

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.)

Just briefly covered.

PL/pgSQL

Main topic of this lecture

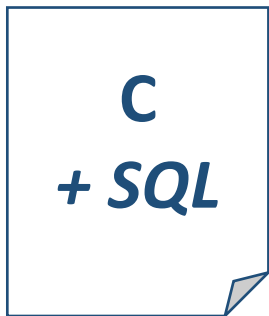
03

Host language + SQL

Host language + SQL

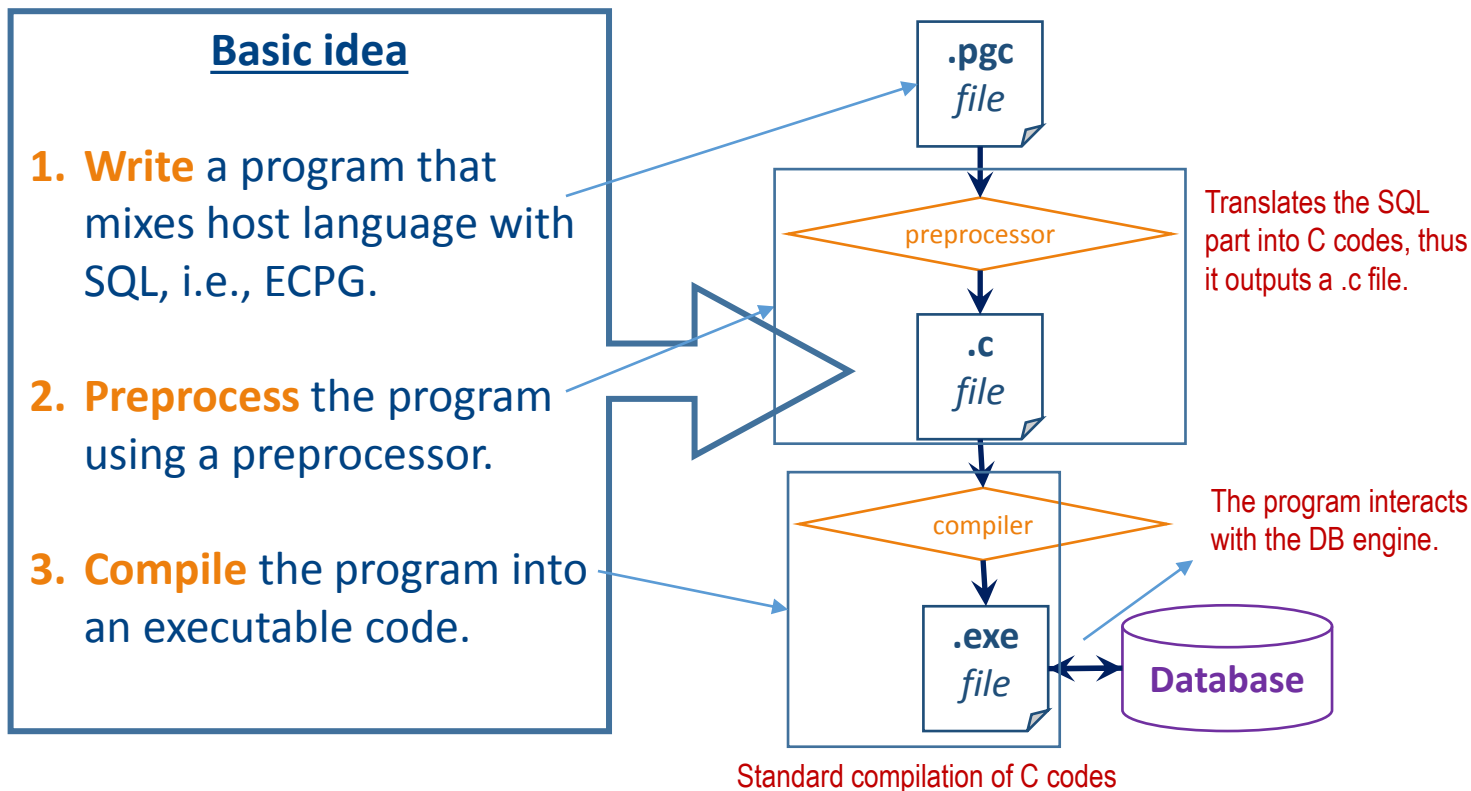
- Let's use **C language** as an example for **host language**.
- There are **two** types of mixing:

Statement-level
Interface



Call-level
Interface





Statement-level Interface

.pgc
file

void main() {

Array of characters of size 30.

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 SELECT ... INTO :name FROM ...;

EXEC SQL DISCONNECT;

}

"Scores"

Name	Mark
Alice	92
...	...

Declaration

declares variables that are shared between host & SQL.

Connection

 setup DB connection with localhost
and username John

Host language

perform operations using host language

Query execution

perform SQL queries/statements.

Disconnect

Disconnect from DB to release resources

Defines a section that involves SQL.

Defines a line that involves SQL.

SELECT ... INTO ...

Assigns the selected value into the
shared variable.Note that the number of
values/variables should be the same.

Use shared variables in SQL by adding a colon : as prefix

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

Yes, it is called Dynamic SQL.

Referring to the INSERT and
SELECT statements.
What if we want the user to
generate the query? Not possible
using static SQL.

Statement-level Interface

.pgc
file

```
void main() {
```

Use a variable with type "string"
instead to store the SQL statements.

```
EXEC SQL BEGIN DECLARE SECTION;  
char *query; 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
```

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

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

```
EXEC SQL DISCONNECT;
```

```
}
```

"Scores"

Name	Mark
Alice	92
...	...

Declaration

Connection

Host language

Query execution

Disconnect

Run the SQL statement in the query variable.

Quick Quiz

Can this be used to retrieve query results?

No, this approach is usually used for DML and DDL, e.g., CREATE, INSERT, UPDATE, DELETE.

03

Statement-level Interface

.pgc
file

"Scores"

Name	Mark
Alice	92
...	...

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 [INTO ...] USING :name, :mark;

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

Another example of a dynamic SQL that can retrieve query results.

"partially" compile the SQL statements in variable query, and "save" it in stmt.

Execute the query in stmt using some parameters, if any.
One prepared stmt can be executed multiple times with different parameters.

Use INTO ... to assign retrieved values to variables, if needed.

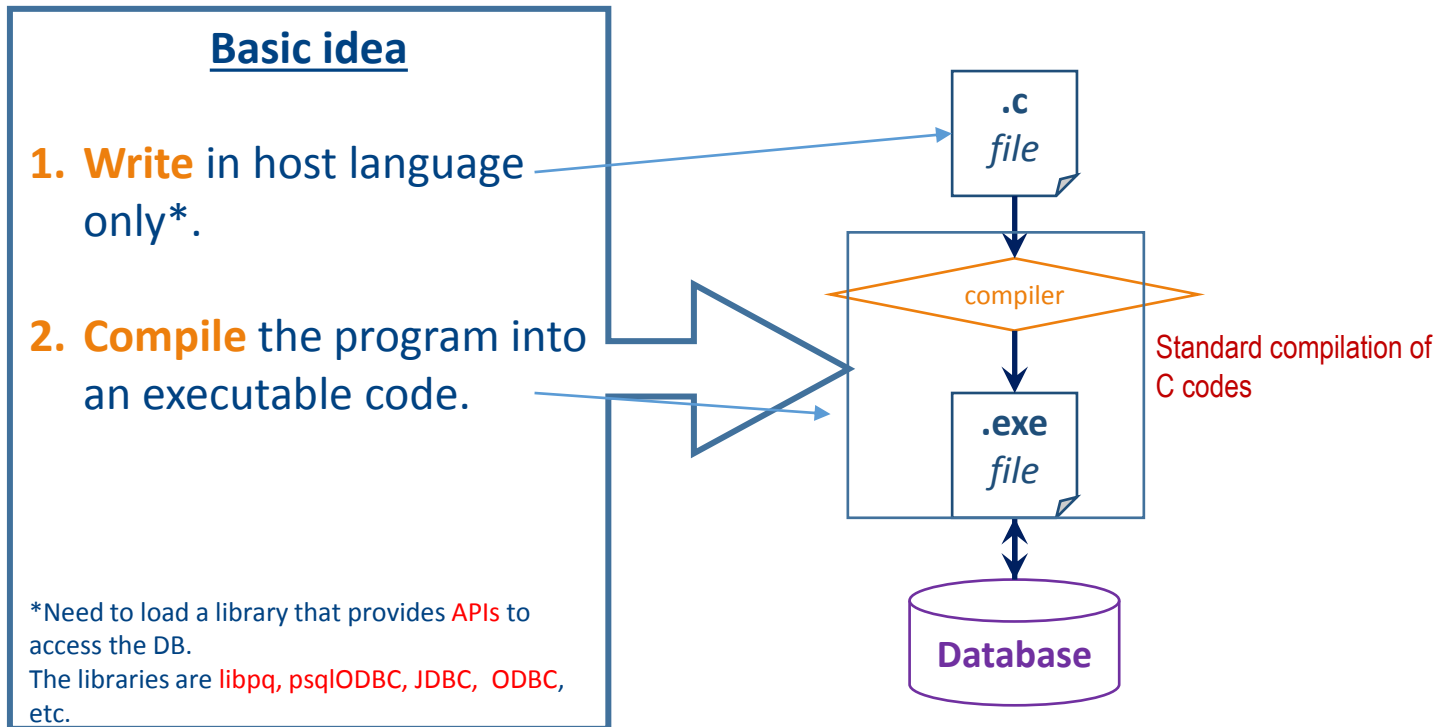
Variables that are passed as parameters, i.e., "replacing" the ? signs in query variable.

Release resources on stmt.

What if we want to use C only?

03 Call-level Interface

C
only



03 Call-level Interface

C
only

.c
file

```
void main() {
```

Everything is in host language.
With some objects, C and W, that
belongs to the class defined by the
library.

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

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

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

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

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

```
}
```

"Scores"

Name	Mark
Alice	92
...	...

Declaration

Connection

Query execution

Disconnect

Quick Quiz

Is this more like a static or dynamic SQL? **Dynamic**, due to usage of string variable to store the SQL statements.

Quick Quiz

What's the pros of using this instead of statement-level interface?
Some libraries provide functions, i.e., insert, that is compatible for many DB engines.

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

Functions and Procedures

- 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!

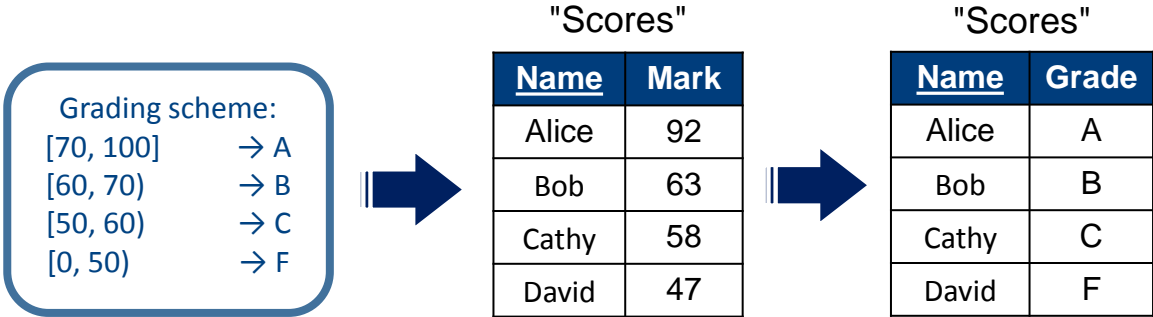
PL/pgSQL

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

Click this whenever there is a play button

SQL only

Converts number marks to letter grades.

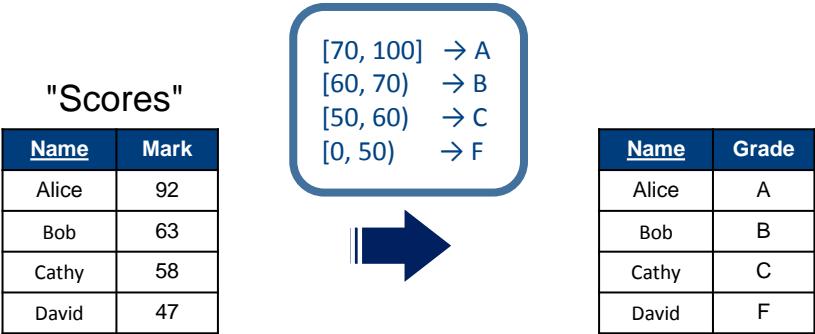


Quick Quiz

Can we do this with a SQL query? Yes, using CASE

Click this →

SQL only ▶



Example of using SQL only, i.e., without using SQL function.

←

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;

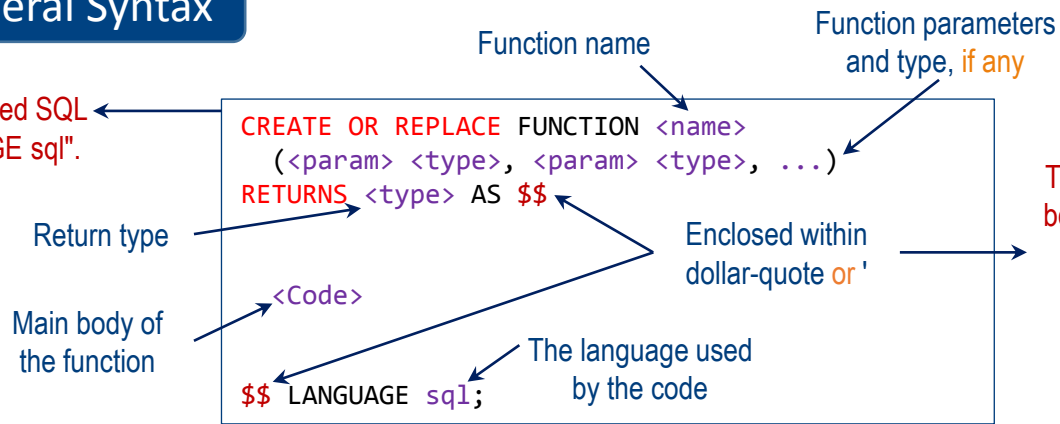
Can we abstract away the conversion with a function? Yes.

04

Functions

General Syntax

Strictly speaking, this is called SQL function due to "LANGUAGE sql".



These are used to enclosed the main body of the function. If you use ', then to define a string need two single quotes "A", Check the link on the next slide.

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

Will be covered in the next lecture



04

Functions

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;
```

The difference is more obvious when
the function returns a tuple. Try it out.

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

This returns value of composite type

This returns value in "table" format

Quick Quiz
What is the
output of this
SQL query?
Click the link
above to reveal
the answer.

04

Functions

"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... The blank is convert(mark).
Thus, the select query gives the output table.

04

Functions

"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.

Convert(mark) can be re-used for different queries as shown below.

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

Quick Quiz

What is the output of this SQL query?

The second select query returns the name of the student where the letter grade is B, i.e., Bob.

04

Functions

"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.

→ If I need to change the grading scheme, then just need to change the range in the function.

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

04

Functions

"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.**

The function is compiled, thus when the function is called multiple times the DB engine won't keep checking the validity of the query/function. This is more efficient than say CTE and Views.

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

How do we return a **tuple** from a function?

"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');
```

When the tuple consists of attributes of a particular table, i.e., Scores table

If there are multiple SELECT statements, then only the result of last SELECT statement will be returned. In other words, the result of the statements are not cached.

Quick Quiz

What is the output of this SQL query? Click the link above to reveal the answer.

What if I remove the LIMIT? It's okay, only the first tuple is shown.

Try running `SELECT * FROM GradeStudent('C')`, and notice the different output format.

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

"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

Add SETOF to return
more than one tuples.

```
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? Click the link above to reveal the answer.

How do we return a custom tuple from a function?

"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

OUT indicates output parameter.

IN indicates input parameter,
which is the default.

Return one custom tuple according
to the OUT parameters

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

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

$$ LANGUAGE sql;
```

Parameter and attribute name can
differ, but not the order and type.

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

Quick Quiz

What is the output of this SQL
query? Click the link above to
reveal the answer.

It will throw an error because Mark
appears in select statement but
not in GROUP BY.

To fix it, replace Mark with either
convert(Mark) or Grade.

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

"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;
```



Add SETOF to return more than one custom tuples.

```
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();
```

Can we **simplify** the params for custom tuples? **Yes!**

"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

Another way to return more
than one custom tuples.

```
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();
```

Can the function return "nothing"?

"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;
```

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

To make a function returns "nothing".

Having a SELECT statement in a "void" function will not throw any error.

Throws an error because Grade is unidentified at this step. Thus, need to remove this part.

Can't we use **procedure** for this? **Yes!**

"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

General Syntax

The only differences as
compared to a function

No RETURN

Main body of
the function

Procedure name

Procedure parameters
and type, if any

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

```
AS $$
```

```
<Code>
```

Enclosed within
dollar-quote or '

```
$$ LANGUAGE sql;
```

The language used
by the code

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

"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;
```



```
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;
```

Use CALL to execute a procedure, instead of SELECT.

```
← CALL UpdateMark(1);
```

Any question?

Summary

- SQL Functions

- Returns a value
 - SQL data types, Set of existing tuples, Set of custom tuples, Etc..
- CREATE OR REPLACE FUNCTION <function_name>(...)
- SELECT <func_name>(...) or SELECT ... FROM <func_name>(...)

- SQL Procedures

- No return value
- CREATE OR REPLACE PROCEDURE <procedure_name>(...)
- CALL <procedure_name>(...)

05

PL/pgSQL Part II

Variables and Control Structure

PL/pgSQL Part II

- Previous **SQL 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
 - LOOP ... END LOOP
 - EXIT WHEN ...
 - WHILE ... LOOP ... END LOOP
 - FOR ... IN ... LOOP ... END LOOP

05

Functions

General Syntax

Strictly speaking, this is called PL/pgSQL function due to "LANGUAGE plpgsql".

DECLARE is needed to declare one or more variables.

BEGIN ... END; is mandatory

```
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?

Change FUNCTION to PROCEDURE, and remove RETURNS <type>.

05

Variables

Use `:=` to assign value to a variable. Variables can be initialized when declared.

This is optional. When executed, it will return the output params, namely mark1 and mark2, then **exit** the function. Otherwise, the function will end naturally and return the output params.

```
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;
```

Assign selected marks into output parameters.

Input parameters.

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

"Scores"

Name	Mark
Alice	92
Bob	63
Cathy	58
David	47

SQL only



Quick Quiz

What is the output of this SQL query?
Click the link above to reveal the answer.

Why the result is not (77, 77)?
Because the returned values are mark1 and mark2, which is (92, 63).

How to return multiple tuples?

05

Variables

One way to return a multiple tuples.

```
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 or RETURN QUERY does not exit the function.

These returns will be cached in the output "table". Thus, this function will output two tuples.

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

"Scores"

Name	Mark
Alice	92
Bob	63
Cathy	58
David	47

SQL only



Quick Quiz

What is the output of this SQL query? Click the link above to reveal the answer.

Must be followed by a SELECT statement, as it expects a query.

Returns the output parameters.

```
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;
```

Just to show
the syntax.

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

"Scores"

Name	Mark
Alice	92
Bob	63
Cathy	58
David	47

SQL
only

Quick Quiz

What is the output of
this SQL query?
Modify the function
in the previous link.

It works the same way as the typical "while loop" in procedural language.

```
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');
```

"Scores"

Name	Mark
Alice	92
Bob	63
Cathy	58
David	47

SQL
only

Quick Quiz

What is the output of this SQL query?
Modify the function in the previous link.

Repetition

```
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;
```

LOOP
 EXIT WHEN temp < 30;
 temp := temp / 2;
 END LOOP;

in Procedural Language ...

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

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

"Scores"

Name	Mark
Alice	92
Bob	63
Cathy	58
David	47

SQL only

Quick Quiz

What is the output of this SQL query?
 Modify the function in the previous link.

Name	Mark
Alice	92
Bob	63
Cathy	58
David	47

```

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;

```

Looping through
each element in an
array

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


05

Is that all?

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.



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%

We will do it!

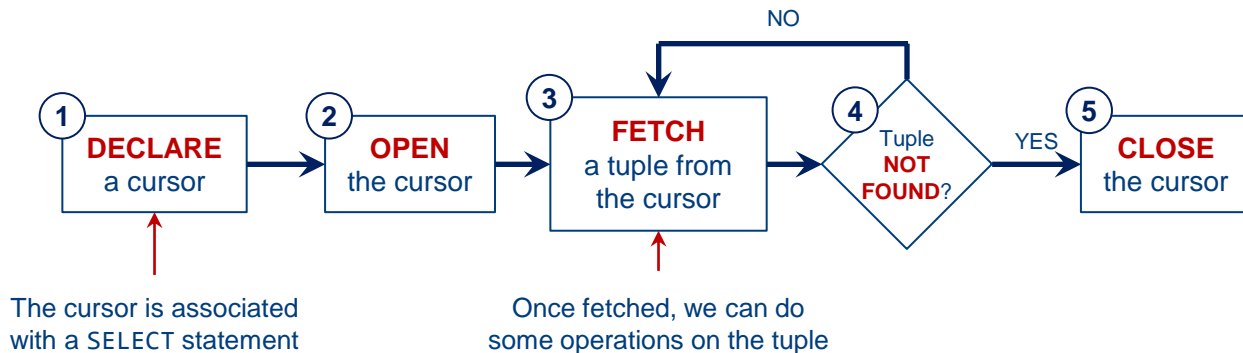


How do we **traverse** a query's result?

05

Cursor

- 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.

Cursor

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*.

Let's simplify the original problem.



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%

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) using a cursor**.

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

Cursor

```
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;
```

Cursor declaration

```
SELECT consCryptosDown(3);
```

curs →

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

Fetch BTC at first iteration.

i.e., cursor reaches the end of the "table".

Fetch SHIB at first iteration.

```
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;
```

Fetch and return the three consecutive coins.

```
  CLOSE curs;
  RETURN;
```

Close cursor to release resources, then exit the function.

```
END IF;
```

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

Relative to the current position of the cursor.
Move cursor to the bottom border of BTC at first iteration, thus the next FETCH curs INTO r1 will return DOGE.

Main idea:
For each tuple in the "table", check if the rank of tuple after its next tuple is two ranks higher than itself.

true

false

Rank	Symbol	Changes
1	BTC	-6%
3	DOGE	-6%
6	SHIB	-8%
8	LTC	-7%
9	XRP	-7%
10	BNB	-6%



05

Cursor

• 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 BTC
- **FETCH LAST FROM curs INTO r;** → Fetch BNB
- **FETCH ABSOLUTE 3 FROM curs INTO r;**
 - Fetch the 3rd tuple, i.e., SHIB.
- **FETCH RELATIVE -2 FROM curs INTO r;**
- **MOVE ... FROM curs;**
- **UPDATE/DELETE ... WHERE CURRENT OF curs;** →

Perform update or delete statement on the tuple at the current position of the cursor.

curs →

Rank	Symbol	Changes
1	BTC	-6%
3	DOGE	-6%
6	SHIB	-8%
8	LTC	-7%
9	XRP	-7%
10	BNB	-6%

Summary

- 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

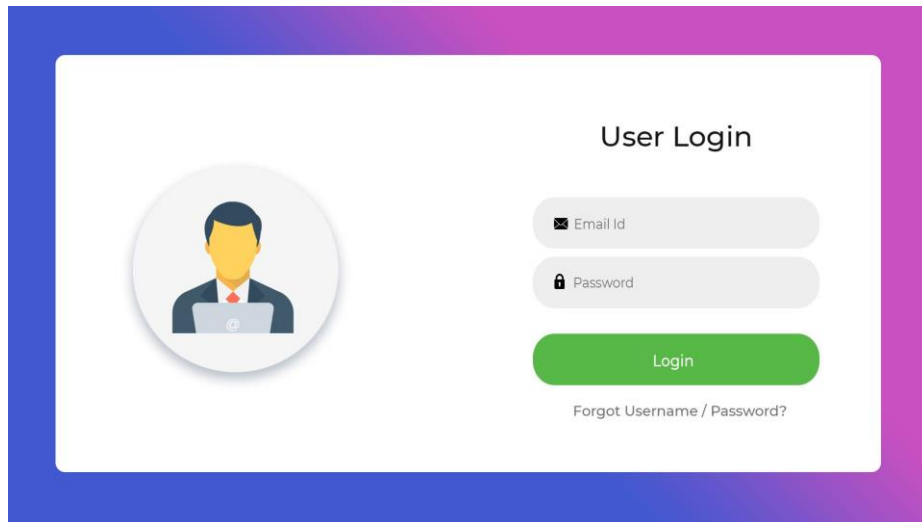
SQL Injection

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

SQL Injection

- **What is it?**

- A class of attacks on **dynamic SQL**.



SQL Injection

• What is it?

- A class of attacks on **dynamic SQL**.

• Expected case

- email = aa@bb.com
- password = abcd
- if (count > 0) { ... }

Take user inputs

↓
IF user with those email and password exists, then count(*) will be > 0. Note that there can be multiple way to check if log in successful, this is just one example.

```
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';
```

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) { ... }

Take user inputs

IF user with those email and password exists, then count(*) will be > 0. Note that there can be multiple way to check if log in successful, this is just one example.

```
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 --;
```

Because of this, the count(*) == number of tuples in the Users table, no matter what the email or password are.

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 ... --;
```

Even worse, malicious user can drop a table, say User table. ←

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;
```

```
}
```

Generated Query

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

```
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 email[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 --';
```

The count(*) is no longer always equal to the number of tuples in the Users table

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

anonymous feedback:

<https://forms.gle/FesPvnAPiABKxx5U6>