Day 14

USER-DEFINED FUNCTIONS

Extend functionality

Complex custom calculations

Safe languages

SQL

PL/pgSQL

Extended version of SQL

Unsafe languages

Python

C

USER-DEFINED FUNCTIONS

```
CREATE FUNCTION <function_name> (param1, param2,…)
```

USER-DEFINED FUNCTIONS

```
CREATE OR REPLACE FUNCTION <function_name> (param1,
param2,…)
```

USER-DEFINED FUNCTIONS

```
CREATE FUNCTION <function_name> (param1, param2,…)
```

USER-DEFINED FUNCTIONS

```
CREATE FUNCTION <function_name> (param1, param2,…)
        RETURNS return_datatype
```

USER-DEFINED FUNCTIONS

```
CREATE FUNCTION <function_name> (param1, param2,…)
        RETURNS return_datatype
        LANGUAGE plpgsql [sql|c|…]
```

USER-DEFINED FUNCTIONS

```
CREATE FUNCTION <function_name> (param1, param2,…)
        RETURNS return_datatype
        LANGUAGE plpgsql [sql|c|…]
AS
$$




$$
```

USER-DEFINED FUNCTIONS

```
CREATE FUNCTION <function_name> (param1, param2,…)
        RETURNS return_datatype
        LANGUAGE plpgsql [sql|c|…]
AS
$$
DECLARE
<variable declaration>;



$$
```

USER-DEFINED FUNCTIONS

```
SELECT first_funct(3,4)
```

3+4 + 3

	first_func integer
1	10

```
CREATE FUNCTION <function_name> (param1, param2,…)
        RETURNS return_datatype
        LANGUAGE plpgsql [sql|c|…]
AS
$$
DECLARE
<variable declaration>;
BEGIN
<function_definition>;
END;
$$
```

USER-DEFINED FUNCTIONS

```
CREATE FUNCTION first_funct(param1, param2,…)
        RETURNS return_datatype
        LANGUAGE plpgsql [sql|c|…]
AS
$$
DECLARE
<variable declaration>;
BEGIN
<function_definition>;
END;
$$
```

USER-DEFINED FUNCTIONS

```
CREATE FUNCTION first_funct(c1 INT, c2 INT)
        RETURNS return_datatype
        LANGUAGE plpgsql [sql|c|…]
AS
$$
DECLARE
<variable declaration>;
BEGIN
<function_definition>;
END;
$$
```

USER-DEFINED FUNCTIONS

```
CREATE FUNCTION first_funct(c1 INT, c2 INT)
        RETURNS INT
        LANGUAGE plpgsql [sql|c|…]
AS
$$
DECLARE
<variable declaration>;
BEGIN
<function_definition>;
END;
$$
```

USER-DEFINED FUNCTIONS

```
CREATE FUNCTION first_funct(c1 INT, c2 INT)
        RETURNS INT
        LANGUAGE plpgsql
AS
$$
DECLARE
<variable declaration>;
BEGIN
<function_definition>;
END;
$$
```

USER-DEFINED FUNCTIONS

```
CREATE FUNCTION first_funct(c1 INT, c2 INT)
        RETURNS INT
        LANGUAGE plpgsql
AS
$$
DECLARE
c3 INT;
BEGIN
<function_definition>;
END;
$$
```

USER-DEFINED FUNCTIONS

```
CREATE FUNCTION first_funct(c1 INT, c2 INT)
        RETURNS INT
        LANGUAGE plpgsql
AS
$$
DECLARE
c3 INT;
BEGIN
SELECT c1+c2+3;
END;
$$
```

USER-DEFINED FUNCTIONS

```
CREATE FUNCTION first_funct(c1 INT, c2 INT)
        RETURNS INT
        LANGUAGE plpgsql [sql|c|…]
AS
$$
DECLARE
c3 INT;
BEGIN
SELECT c1+c2+3
INTO c3;
RETURN c3;
END;
$$
```

USER-DEFINED FUNCTIONS

```
CREATE FUNCTION first_funct(c1 INT, c2 INT)
        RETURNS INT
        LANGUAGE plpgsql
AS
$$
DECLARE

BEGIN

RETURN SELECT c1+c2+3;
END;
$$
```

USER-DEFINED FUNCTIONS

```
CREATE FUNCTION first_funct(c1 INT, c2 INT)
        RETURNS INT
        LANGUAGE plpgsql
AS
$$
DECLARE
c3 INT;
BEGIN
SELECT c1+c2+3
INTO c3
FROM table;
RETURN c3;
END;
```

Challenge

Create a function that expects the customer's first and last name and returns the total amount of payments this customer has made.

```
20  SELECT name_search('AMY','LOPEZ')
21
```

Data Output | Explain | Messages | Notifications

	name_search numeric
1	127.71

TRANSACTIONS

Unit of work

One or multiple operations

TRANSACTIONS

Bank transfer

id [PK] integ	first_name text	last_name text	amount numeric (9,2)
1	Tim	Brown	2500.00
2	Sandra	Miller	1600.00

100

One unit of work

TRANSACTIONS

```
BEGIN TRANSACTION;
```

TRANSACTIONS

```
BEGIN WORK;
```

TRANSACTIONS

```
BEGIN;
```

TRANSACTIONS

```
BEGIN;

OPERATION1;
OPERATION2;
```

Not visible in other sessions
(e.g. other users)

TRANSACTIONS

```
BEGIN;

OPERATION1;
OPERATION2;

COMMIT;
```

Not visible in other sessions
(e.g. other users)

TRANSACTIONS

```
BEGIN;

OPERATION1;
OPERATION2;

COMMIT;
```

Challenge

The two employees Miller McQuarter and Christalle McKenny have agreed to swap their positions incl. their salary.

emp_id [PK] integer	first_name text	last_name text	position_title text	salary numeric (8,2)
1	Morrie	Conaboy	CTO	21268.94
2	Miller	McQuarter	Head of BI	14614.00
3	Christalle	McKenny	Head of Sales	12587.00

ROLLBACK

```
BEGIN;

OPERATION1;
OPERATION2;

COMMIT;
```

ROLLBACK

```
BEGIN;

OPERATION1;
OPERATION2;
ROLLBACK;
COMMIT;
```

Undo everything in the current transaction that has not been committed yet!

ROLLBACK

```
BEGIN;

OPERATION1;
OPERATION2;
OPERATION3;
OPERATION4;

ROLLBACK;
COMMIT;
```

ROLLBACK

```
BEGIN;

OPERATION1;
OPERATION2;
SAVEPOINT op2;
OPERATION3;
OPERATION4;

ROLLBACK TO SAVEPOINT op2;
COMMIT;
```

ROLLBACK

```
BEGIN;

OPERATION1;
OPERATION2;
SAVEPOINT op2;
OPERATION3;
SAVEPOINT op3;
OPERATION4;

ROLLBACK TO SAVEPOINT op3;
COMMIT;
```

Savepoints work only within a current transaction

ROLLBACK

```
BEGIN;

OPERATION1;
OPERATION2;
SAVEPOINT op2;
OPERATION3;
SAVEPOINT op3;
OPERATION4;

RELEASE SAVEPOINT op3;
COMMIT;
```

ROLLBACK;
ends transaction

ROLLBACK TO SAVEPOINT;
does not end transaction

Deleting a savepoint

STORED PROCEDURES

User-defined function:
`CREATE FUNCTION`

Downside:
They cannot execute transactions

`BEGIN;` `COMMIT;` `ROLLBACK;`

✓ Stored procedures:
They support transactions

STORED PROCEDURES

```
CREATE PROCEDURE <procedure_name> (param1, param2,…)
```

STORED PROCEDURES

```
CREATE OR REPLACE PROCEDURE <procedure_name> (param1, param2,…)
```

STORED PROCEDURES

```
CREATE PROCEDURE <procedure_name> (param1, param2,…)
```

STORED PROCEDURES

```
CREATE PROCEDURE <procedure_name> (param1, param2,…)
        LANGUAGE plpgsql [sql|c|…]
AS
$$
DECLARE
<variable declaration>
BEGIN
<procedure_definition>
END;
$$
```

STORED PROCEDURES

```
CREATE FUNCTION <function_name> (param1, param2,…)
        RETURNS INT
        LANGUAGE plpgsql [sql|c|…]
AS
$$
DECLARE
<variable declaration>
BEGIN
<function_definition>
RETURN expression;
END;
$$
```

Not possible in a stored procedure!

STORED PROCEDURES

```
CREATE PROCEDURE <procedure_name> (param1, param2,…)
        LANGUAGE plpgsql [sql|c|…]
AS
$$
DECLARE
<variable declaration>
BEGIN
<procedure_definition>
END;
$$
```

Not possible in a stored procedure!

STORED PROCEDURES

```
CREATE PROCEDURE <procedure_name> (param1, param2,…)
        LANGUAGE plpgsql [sql|c|…]
AS
$$
DECLARE
<variable declaration>
BEGIN
<procedure_definition>
RETURN;
END;
$$
```

Not possible in a stored procedure!

TRANSACTIONS

Bank transfer

id [PK] integ	first_name text	last_name text	amount numeric (9,2)
1	Tim	Brown	2500.00
2	Sandra	Miller	1600.00

100

One unit of work

STORED PROCEDURES

```
CREATE PROCEDURE sp_transfer (tr_amount INT, sender INT,recipient INT)
          LANGUAGE plpgsql
AS
$$
```

STORED PROCEDURES

```
CREATE PROCEDURE sp_transfer (tr_amount INT, sender INT,recipient INT)
          LANGUAGE plpgsql
AS
$$
BEGIN
-- subtract from sender's balance
UPDATE acc_balance
SET amount = amount - tr_amount
WHERE id = sender;
```

STORED PROCEDURES

```
CREATE PROCEDURE sp_transfer (tr_amount INT, sender INT,recipient INT)
          LANGUAGE plpgsql
AS
$$
BEGIN
-- subtract from sender's balance
UPDATE acc_balance
SET amount = amount - tr_amount
WHERE id = sender;

-- add to recipient's balance
UPDATE acc_balance
SET amount = amount - tr_amount
WHERE id = recipient;
```

STORED PROCEDURES

```
CREATE PROCEDURE sp_transfer (tr_amount INT, sender INT,recipient INT)
          LANGUAGE plpgsql
AS
$$
BEGIN
-- subtract from sender's balance
UPDATE acc_balance
SET amount = amount - tr_amount
WHERE id = sender;

-- add to recipient's balance
UPDATE acc_balance
SET amount = amount + tr_amount
WHERE id = recipient;

COMMIT;
END;
$$
```

STORED PROCEDURES

```
CALL <store_procedure_name> (param1, param2,…);
```

STORED PROCEDURES

```
CALL <store_procedure_name> (param1, param2,…);
```

`sp_transfer (tr_amount, sender,recipient)`

```
CALL sp_transfer (100, 1,2);
```

Challenge

Create a stored procedure called emp_swap that accepts two parameters emp1 and emp2 as input and swaps the two employees' position and salary.

Test the stored procedure with emp_id 2 and 3.

emp_id [PK] integer	first_name text	last_name text	position_title text	salary numeric (8,2)
1	Morrie	Conaboy	CTO	21268.94
2	Miller	McQuarter	Head of BI	14614.00
3	Christalle	McKenny	Head of Sales	12587.00