



UNIVERSITAS
INDONESIA

Veritas, Probitas, Justitia

FAKULTAS
ILMU
KOMPUTER

Slide 11

Trigger and Stored Procedure

CSF2600700 - BASIS DATA
SEMESTER GENAP 2019/2020



These slides are a modification to the supplementary slide of “Fundamentals of Database System”, 7th edition, Elmasri/Navathe, 2015: **Chapter 7 More SQL: Complex Queries, Triggers, Views, and Schema Modification**

PostgreSQL Documentation:

<https://www.postgresql.org/docs/12/index.html>



Outline

- Stored Procedure
- Trigger



STORED PROCEDURE

- Sebuah stored procedure adalah kumpulan dari prosedur dan statement SQL yang terdapat pada DBMS.
- Dikenali dengan nama.
- Dieksekusi sebagai sebuah kesatuan (unit).
- Di *postgresql* juga dikenal sebagai function.



Keuntungan Stored Procedure

- Reusable.
- Mengurangi *network traffic* dan meningkatkan performance (mengurangi transmisi individual SQL *statements* pada *network*).



General Form

○ Procedure

```
CREATE [or REPLACE] <procedure name> (<parameters>)  
<local declarations>  
<procedure body>  
LANGUAGE <language>;
```

○ Function

```
CREATE [or REPLACE] <function name> (<parameters>)  
RETURNS <return type>  
<local declarations>  
<function body>  
LANGUAGE <language>;
```



Example (1)

```
CREATE or REPLACE FUNCTION one()  
RETURNS integer AS
```

```
$$
```

```
    SELECT 1 AS result;
```

```
$$
```

```
LANGUAGE SQL;
```

Tanda \$\$ dapat diganti dengan single quote (')

Eksekusi dilakukan dengan memanggil **SELECT one();**



Example (1) – Result

```
postgres=# CREATE or REPLACE FUNCTION one() RETURNS integer AS
postgres-# $$
postgres$#      SELECT 1 AS result;
postgres$# $$
postgres-# LANGUAGE SQL;
CREATE FUNCTION
postgres=# select one();
 one
-----
   1
(1 row)
```



Example (2)

```
CREATE or REPLACE FUNCTION  
add_em(x integer, y integer) RETURNS  
integer AS  
$$  
    SELECT x + y;  
$$  
LANGUAGE SQL;
```

Eksekusi dilakukan dengan memanggil **SELECT add_em(x, y);**



Example (3)

```
CREATE or REPLACE FUNCTION hellow() RETURNS
text AS
$$
DECLARE
    hello text;
BEGIN
    hello := 'Hello World!';
    return hello;
END
$$ LANGUAGE plpgsql;
```

Eksekusi dilakukan denngan memanggil **SELECT hellow();**



Example (3) – Result

```
postgres=# CREATE or REPLACE FUNCTION hellow() RETURNS text AS
postgres-# $$
postgres$# DECLARE
postgres$#     hello text;
postgres$# BEGIN
postgres$#     hello := 'Hello World!';
postgres$#     return hello;
postgres$# END
postgres$# $$ LANGUAGE plpgsql;
CREATE FUNCTION
postgres=# select hellow();
        hellow
-----
Hello World!
```



Example (4)

```
CREATE OR REPLACE FUNCTION sum_salary()  
RETURNS integer AS  
$$  
    DECLARE result integer;  
    BEGIN  
        SELECT sum(salary) into result FROM  
employee;  
        RETURN result;  
    END;  
$$ LANGUAGE plpgsql;
```

Eksekusi dilakukan denngan memanggil **SELECT sum_salary();**



TRIGGER

- Trigger merupakan kode PL/SQL yang secara otomatis dijalankan oleh DBMS jika suatu event database terjadi.
- Event tersebut bisa berupa operasi **INSERT, UPDATE, DELETE**
- Sebuah trigger selalu dijalankan sebelum atau sesudah sebuah data row di-INSERT, di-UPDATE atau di-DELETE.
- Sebuah trigger selalu berasosiasi dengan tabel pada basis data.
- Setiap tabel bisa mempunyai satu atau lebih trigger
- Sebuah trigger dieksekusi sebagai bagian dari transaksi yang men-trigger trigger tersebut.



Keuntungan Trigger

- Trigger dapat digunakan untuk memaksakan constraint yang tidak dapat dilakukan pada perancangan dan implementasi DBMS.
- Trigger dapat secara otomatis memberikan pesan warning jika terjadi gangguan pada IC. Penggunaan trigger yg umum adalah untuk meningkatkan referential IC.
- Trigger dapat digunakan untuk update nilai pada tabel, insert tuple pada tabel, dan memanggil stored procedure yg lain.



Trigger and Stored Procedure

```
CREATE TRIGGER name { BEFORE | AFTER } { event [ OR ... ] }  
ON table_name [ FOR [ EACH ] { ROW | STATEMENT } ]  
[WHEN (condition)]  
EXECUTE {FUNCTION | PROCEDURE} function_name ( arguments );
```

Typical TRIGGER components: E, C, A

○ Event(s)

- These are usually DB updates.
- Specified after the keyword BEFORE (or AFTER).



Trigger and Stored Procedure

```
CREATE TRIGGER name { BEFORE | AFTER } { event [ OR ... ] }  
ON table_name [ FOR [ EACH ] { ROW | STATEMENT } ]
```

→ [WHEN (condition)]

```
EXECUTE {FUNCTION | PROCEDURE} function_name ( arguments );
```

Typical TRIGGER components: E, C, A

○ Condition

- The check whether the rule action should be executed. It is specified in the WHEN clause.
- If no condition, the execution will be executed whenever the event occurs. If there is a condition, the condition is first evaluated, and only if it is true will the rule action be executed.
- This can also be defined in the function or procedure.



Trigger and Stored Procedure

```
CREATE TRIGGER name { BEFORE | AFTER } { event [ OR ... ] }  
ON table_name [ FOR [ EACH ] { ROW | STATEMENT } ]  
[WHEN (condition)]  
EXECUTE {FUNCTION | PROCEDURE} function_name ( arguments );
```

Typical TRIGGER components: E, C, **A**

○ **Action**

- The action can be SQL statements or a stored procedure or function.



COMPANY DB

EMPLOYEE

Fname	Minit	Lname	<u>Ssn</u>	Bdate	Address	Sex	Salary	Super_ssn	Dno
-------	-------	-------	------------	-------	---------	-----	--------	-----------	-----

DEPARTMENT

Dname	<u>Dnumber</u>	Mgr_ssn	Mgr_start_date
-------	----------------	---------	----------------

DEPT_LOCATIONS

<u>Dnumber</u>	<u>Dlocation</u>
----------------	------------------

PROJECT

Pname	<u>Pnumber</u>	Plocation	Dnum
-------	----------------	-----------	------

WORKS_ON

<u>Essn</u>	<u>Pno</u>	Hours
-------------	------------	-------

DEPENDENT

<u>Essn</u>	<u>Dependent_name</u>	Sex	Bdate	Relationship
-------------	-----------------------	-----	-------	--------------



Example (5)

- Suppose we want to check whenever an employee's salary is greater than the salary of his/her direct supervisor in the COMPANY DB

Can you guess what events can trigger this situation?



Example (5)

- Suppose we want to check whenever an employee's salary is greater than the salary of his/her direct supervisor in the COMPANY DB

Can you guess what events can trigger this situation?

- Inserting a new employee record
- Updating an employee's salary
- Updating an employee's supervisor



Example (5) – Trigger

- Suppose we want to check whenever an employee's salary is greater than the salary of his/her direct supervisor in the COMPANY DB

```
CREATE TRIGGER SALARY_VIOLATION  
BEFORE INSERT OR UPDATE OF SALARY ON EMPLOYEE  
FOR EACH ROW  
EXECUTE PROCEDURE INFORM_SUPERVISOR();
```



Example (5) – Function/Stored Procedure

- Suppose we want to check whenever an employee's salary is greater than the salary of his/her direct supervisor in the COMPANY DB

```
CREATE OR REPLACE FUNCTION INFORM_SUPERVISOR() RETURNS
trigger AS $$
BEGIN
    IF (NEW.SALARY > (SELECT SALARY FROM EMPLOYEE WHERE
SSN = NEW.SUPER_SSN))
    THEN
        → RAISE EXCEPTION '% salary cannot greater
than his/her supervisor salary', OLD.FNAME;
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```



Example (5) – PostgreSQL – Result

UPDATE employee SET salary = 45000 where ssn = '123456789';

```
postgres=# select * from employee;
```

fname	minit	lname	ssn	bdate	address	sex	salary	super_ssn	dno
Ramesh		Narayan	666884444			F	38000.00	888665555	5
Alicia	J	Zelaya	999887777	1968-01-19	Castle	F	25000.00		4
Ahmad		Jabbar	987987987			M	25000.00	987654321	4
Joyce		English	453453453			F	25000.00	987654321	5
Jennifer	S	Wallace	987654321	1941-06-19	Bellaire	F	43000.00	333445555	4
James		Borg	888665555			M	55000.00	333445555	1
Franklin	T	Wong	333445555	1955-12-08	Houston	M	40000.00	888665555	5
John	B	Smith	123456789	1965-01-09	Fondren	M	30000.00	333445555	5

```
postgres=# update employee set salary = 45000 where ssn = '123456789';
ERROR:  John salary cannot greater than his/her supervisor salary
CONTEXT:  PL/pgSQL function inform_supervisor() line 5 at RAISE
```



Example (5) – PostgreSQL – Result 2

UPDATE employee SET salary = 25000 where ssn =
'123456789';

```
postgres=# UPDATE employee SET salary = 25000 where ssn = '123456789';
UPDATE 1
postgres=# select * from employee;
```

fname	minit	lname	ssn	bdate	address	sex	salary	super_ssn	dno
Joyce		English	453453453			F	25000.00	987654321	5
Ramesh		Narayan	666884444			F	38000.00	888665555	5
Ahmad		Jabbar	987987987			M	25000.00	987654321	4
Alicia	J	Zelaya	999887777	1968-01-19	Castle	F	25000.00		4
Jennifer	S	Wallace	987654321	1941-06-19	Bellaire	F	43000.00	333445555	4
James		Borg	888665555			M	55000.00	333445555	1
Franklin	T	Wong	333445555	1955-12-08	Houston	M	30000.00	888665555	5
John	B	Smith	123456789	1965-01-09	Fondren	M	25000.00	333445555	5



Example (6)

- Mengecek bahwa SALARY dari EMPLOYEE tidak boleh bernilai negatif.
- Yang menjadi trigger:
 - Insert new employee
 - Update employees' salary



Example (6) – Function/Stored Procedure

```
CREATE OR REPLACE FUNCTION emp_stamp()  
  RETURNS trigger AS  
$$  
  BEGIN  
    IF (NEW.salary < 0) THEN  
      RAISE EXCEPTION '% cannot have  
negative salary', NEW.lname;  
    END IF;  
    RETURN NEW;  
  END;  
$$  
LANGUAGE plpgsql;
```



Example (6) – Trigger

```
CREATE TRIGGER emp_stamp  
  BEFORE INSERT OR UPDATE OF SALARY ON  
  employee  
  FOR EACH ROW  
  EXECUTE PROCEDURE emp_stamp();
```



Example (6) – Result 1

INSERT INTO EMPLOYEE VALUES

('Donald', 'T', 'Duck', '123456788', '1950-11-11',
'Manhattan', 'M', **-100**, NULL, 1);

```
postgres=# select * from employee;
```

fname	minit	lname	ssn	bdate	address	sex	salary	super_ssn	dno
John	B	Smith	123456789	1965-01-09	Fondren	M	30000.00	333445555	5
Franklin	T	Wong	333445555	1955-12-08	Houston	M	40000.00	888665555	5
Joyce		English	453453453			F	25000.00	987654321	5
Ramesh		Narayan	666884444			F	38000.00	888665555	5
James		Borg	888665555			M	55000.00	333445555	1
Jennifer	S	Wallace	987654321	1941-06-19	Bellaire	F	43000.00	333445555	4
Ahmad		Jabbar	987987987			M	25000.00	987654321	4
Alicia	J	Zelaya	999887777	1968-01-19	Castle	F	25000.00		4

(8 rows)

```
postgres=# INSERT INTO EMPLOYEE VALUES('Donald','T','Duck','123456788','1950-11-11','Manhattan','M',  
-100',NULL,1);  
ERROR:  Duck cannot have negative salary  
CONTEXT:  PL/pgSQL function emp_stamp() line 2 at RAISE
```



Example (6) – Result 2

INSERT INTO EMPLOYEE VALUES

('Donald', 'T', 'Duck', '123456788', '1950-11-11', 'Manhattan', 'M', 20000.00, NULL, 1);

```
postgres=# INSERT INTO EMPLOYEE VALUES('Donald','T','Duck','123456788','1950-11-11','Manhattan','M',20000, NULL,1);
INSERT 0 1
```

```
postgres=# select * from employee;
```

fname	minit	lname	ssn	bdate	address	sex	salary	super_ssn	dno
John	B	Smith	123456789	1965-01-09	Fondren	M	30000.00	333445555	5
Franklin	T	Wong	333445555	1955-12-08	Houston	M	40000.00	888665555	5
Joyce		English	453453453			F	25000.00	987654321	5
Ramesh		Narayan	666884444			F	38000.00	888665555	5
James		Borg	888665555			M	55000.00	333445555	1
Jennifer	S	Wallace	987654321	1941-06-19	Bellaire	F	43000.00	333445555	4
Ahmad		Jabbar	987987987			M	25000.00	987654321	4
Alicia	J	Zelaya	999887777	1968-01-19	Castle	F	25000.00		4
Donald	T	Duck	123456788	1950-11-11	Manhattan	M	20000.00		1

(9 rows)



Example (7)

Mengakumulasi jumlah jam kerja EMPLOYEE pada PROJECT.

Misalkan hal ini dipengaruhi oleh penambahan atau penghapusan proyek baru untuk EMPLOYEE.

Catatan:

Relasi EMPLOYEE terlebih dulu di-update dengan melakukan penambahan kolom total_hours_project (dan meng-update isinya jika sebelumnya sudah ada assignment employee terhadap project tertentu).



Example (7) – Function/Stored Procedure

```
CREATE OR REPLACE FUNCTION emp_total_hours_proj()
  RETURNS trigger AS $$
  BEGIN
    IF (TG_OP = 'INSERT') THEN
      UPDATE employee SET total_hours_project =
        total_hours_project + NEW.hours WHERE ssn = NEW.essn;
      RETURN NEW;
    ELSIF (TG_OP = 'DELETE') THEN
      UPDATE employee SET total_hours_project =
        total_hours_project - OLD.hours WHERE ssn = OLD.essn;
      RETURN OLD;
    END IF;
  END;
  $$ LANGUAGE plpgsql;
```



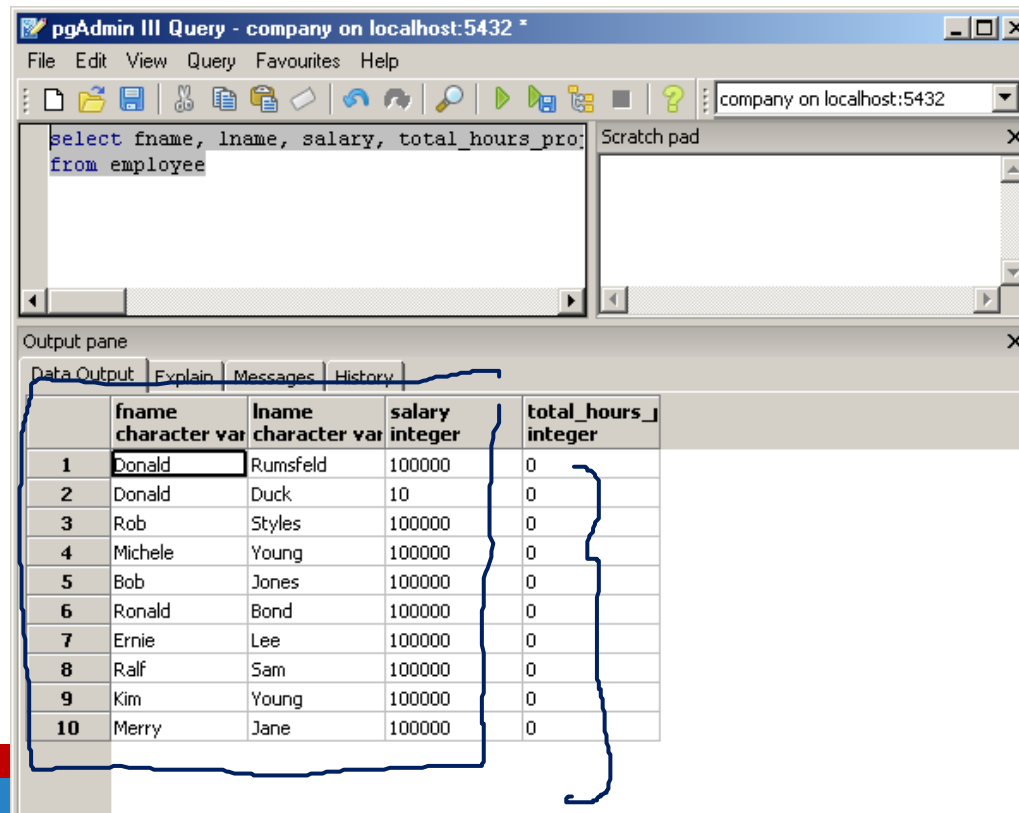
Example (7) - Trigger

```
CREATE TRIGGER emp_total_hours_proj  
  AFTER INSERT OR DELETE  
  ON works_on  
  FOR EACH ROW  
  EXECUTE PROCEDURE emp_total_hours_proj();
```



Example (7) – Implementation

- Sebelum meng-assign WORKS_ON
- SELECT FNAME, LNAME, SALARY, TOTAL_HOURS_PROJECT FROM EMPLOYEE



The screenshot shows the pgAdmin III Query tool interface. The title bar reads "pgAdmin III Query - company on localhost:5432". The menu bar includes File, Edit, View, Query, Favourites, and Help. The toolbar contains various icons for file operations and query execution. The query editor on the left contains the following SQL query:

```
select fname, lname, salary, total_hours_pro  
from employee
```

The output pane at the bottom displays the results of the query in a table format. The table has four columns: fname, lname, salary, and total_hours_project. The data is as follows:

	fname character var	lname character var	salary integer	total_hours_ integer
1	Donald	Rumsfeld	100000	0
2	Donald	Duck	10	0
3	Rob	Styles	100000	0
4	Michele	Young	100000	0
5	Bob	Jones	100000	0
6	Ronald	Bond	100000	0
7	Ernie	Lee	100000	0
8	Ralf	Sam	100000	0
9	Kim	Young	100000	0
10	Merry	Jane	100000	0

A blue hand-drawn bracket on the right side of the table groups rows 1 through 10. Another blue hand-drawn bracket on the left side of the table groups the first two columns (fname and lname) for rows 1 through 10.

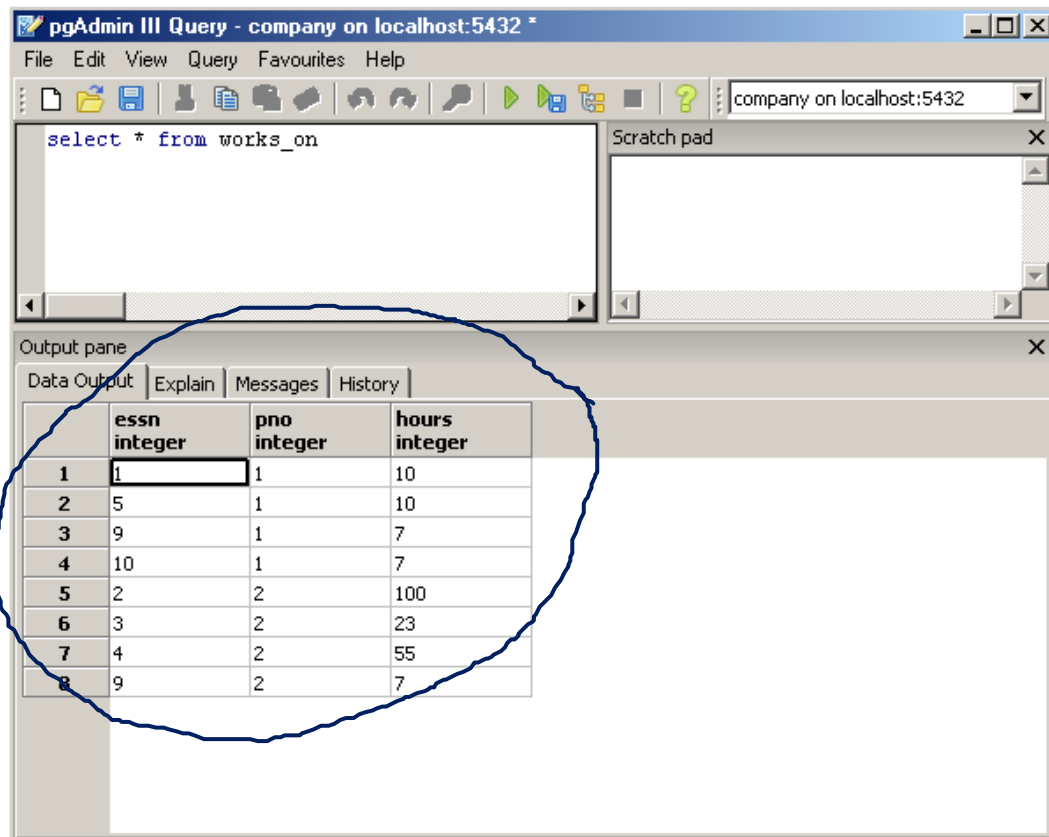
Example (7) – Implementation

- INSERT INTO PROJECT VALUES('1', 'New York', 'NYS Project', '1');
 - INSERT INTO WORKS_ON VALUES('1', '1', '10');
 - INSERT INTO WORKS_ON VALUES('5', '1', '10');
 - INSERT INTO WORKS_ON VALUES('9', '1', '7');
 - INSERT INTO WORKS_ON VALUES('10', '1', '7');
-
- INSERT INTO PROJECT VALUES('2', 'New York', 'NYS2 Project', '1');
 - INSERT INTO WORKS_ON VALUES('2', '2', '100');
 - INSERT INTO WORKS_ON VALUES('3', '2', '23');
 - INSERT INTO WORKS_ON VALUES('4', '2', '55');
 - INSERT INTO WORKS_ON VALUES('9', '2', '7');



Example (7) – Result

- Tabel WORKS_ON



pgAdmin III Query - company on localhost:5432 *

File Edit View Query Favourites Help

company on localhost:5432

```
select * from works_on
```

Scratch pad

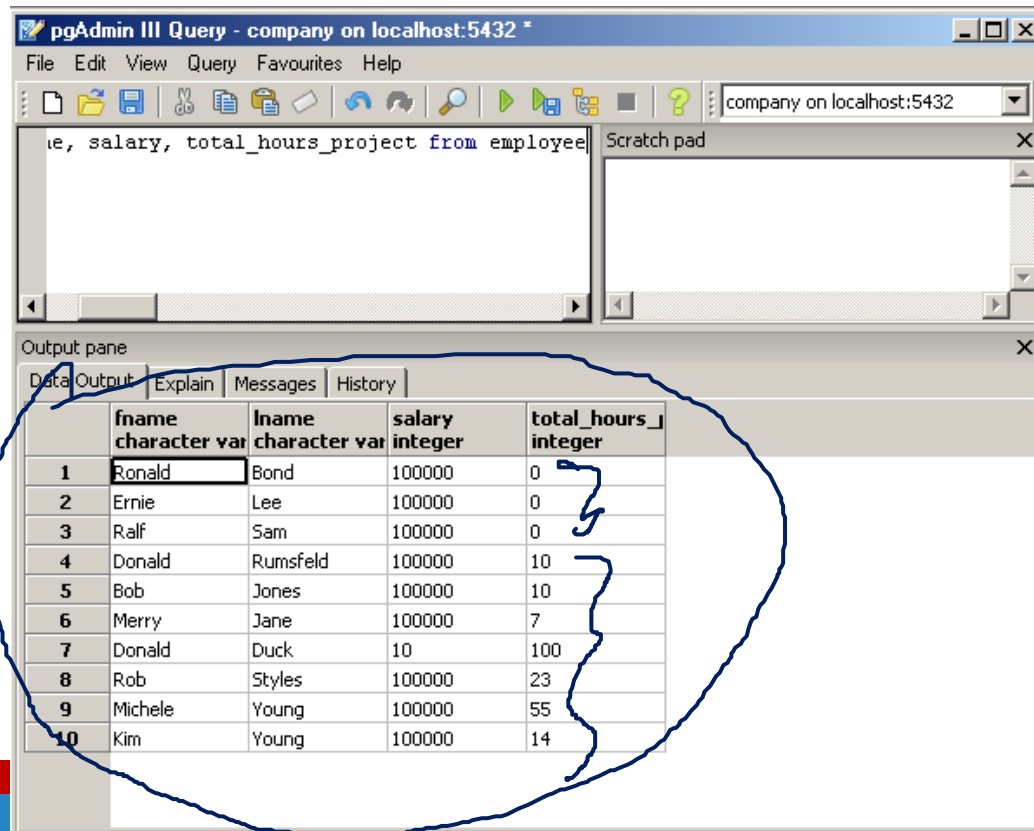
Output pane

Data Output Explain Messages History

	essn integer	pno integer	hours integer
1	1	1	10
2	5	1	10
3	9	1	7
4	10	1	7
5	2	2	100
6	3	2	23
7	4	2	55
8	9	2	7

Example (7) – Result

- Setelah meng-assign WORKS_ON
- **SELECT FNAME, LNAME, SALARY, TOTAL_HOURS_PROJECT FROM EMPLOYEE**



pgAdmin III Query - company on localhost:5432 *

File Edit View Query Favourites Help

company on localhost:5432

ie, salary, total_hours_project from employee

Scratch pad

Output pane

Data Output Explain Messages History

	fname character var	lname character var	salary integer	total_hours_ integer
1	Ronald	Bond	100000	0
2	Ernie	Lee	100000	0
3	Ralf	Sam	100000	0
4	Donald	Rumsfeld	100000	10
5	Bob	Jones	100000	10
6	Merry	Jane	100000	7
7	Donald	Duck	10	100
8	Rob	Styles	100000	23
9	Michele	Young	100000	55
10	Kim	Young	100000	14



Example (8)

Menghitung gaji total EMPLOYEE yang merupakan akumulasi SALARY dengan tunjangan yang bergantung pada jam kerja pada PROJECT.



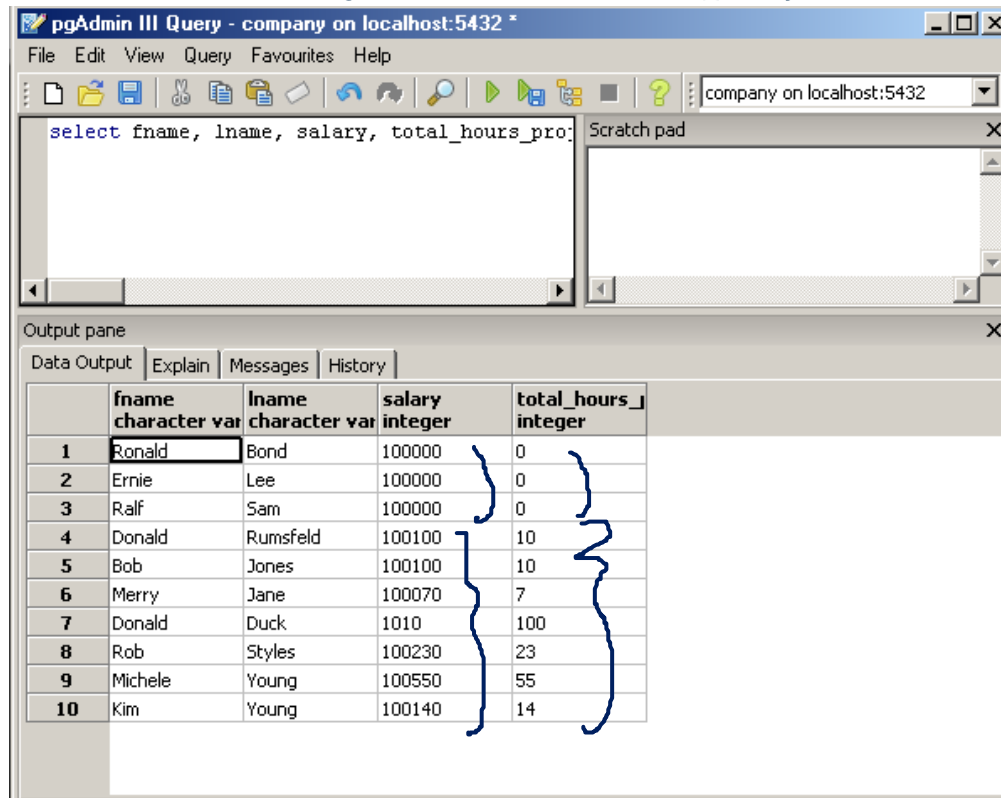
Example (8) – Function/Stored Procedure

```
CREATE OR REPLACE FUNCTION  
  procedure_salary() RETURNS void  
AS $$  
BEGIN  
  UPDATE employee SET salary = salary +  
  total_hours_project * 10 WHERE  
  total_hours_project > 0;  
END;  
$$ LANGUAGE plpgsql;
```



Example (8) – Result

- Untuk meng-execute dapat digunakan query sebagai berikut:
- **SELECT * FROM PROCEDURE_SALARY();**
- **SELECT fname, lname, salary, total_hours_proj FROM EMPLOYEE;**

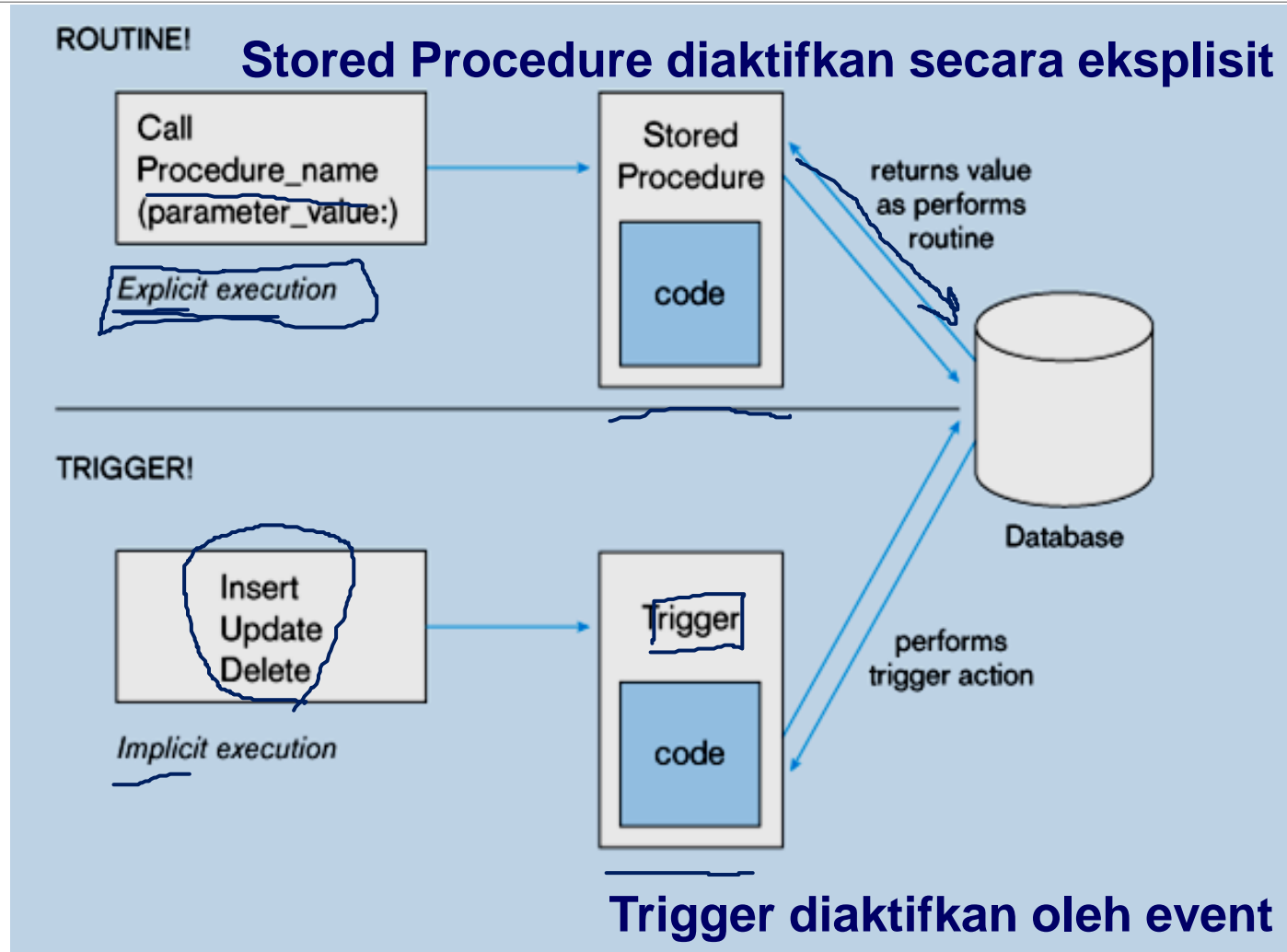


The screenshot shows the pgAdmin III Query tool interface. The title bar reads "pgAdmin III Query - company on localhost:5432 *". The menu bar includes File, Edit, View, Query, Favourites, and Help. The toolbar contains various icons for file operations and query execution. The query editor contains the SQL query: `select fname, lname, salary, total_hours_proj`. The output pane shows the results of the query in a table format. The table has four columns: **fname** (character var), **lname** (character var), **salary** (integer), and **total_hours_proj** (integer). The results are listed in 10 rows, numbered 1 to 10. Handwritten blue brackets are drawn on the right side of the table, grouping the rows by salary and total hours. The first three rows (1-3) have a salary of 100000 and total hours of 0. The next four rows (4-7) have salaries of 100100, 100100, 100070, and 1010, with total hours of 10, 10, 7, and 100 respectively. The last three rows (8-10) have salaries of 100230, 100550, and 100140, with total hours of 23, 55, and 14 respectively.

	fname character var	lname character var	salary integer	total_hours_ integer
1	Ronald	Bond	100000	0
2	Ernie	Lee	100000	0
3	Ralf	Sam	100000	0
4	Donald	Rumsfeld	100100	10
5	Bob	Jones	100100	10
6	Merry	Jane	100070	7
7	Donald	Duck	1010	100
8	Rob	Styles	100230	23
9	Michele	Young	100550	55
10	Kim	Young	100140	14



Trigger vs Stored Procedure



- 
-
- **EKSPLORASI & LATIHAN**
 - **Latihan Trigger – Stored Procedure**

