Principles of Database Systems (CS307)

Lab Session: Trigger

Yuxin Ma

Department of Computer Science and Engineering
Southern University of Science and Technology

- Most contents are from slides made by Stéphane Faroult and the authors of Database System Concepts (7th Edition).
- Their original slides have been modified to adapt to the schedule of CS307 at SUSTech.

Trigger

Trigger - Actions When Changing Tables

A trigger is a specification that the database should automatically execute a particular function whenever a certain type of operation is performed.

-- Chapter 39, PostgreSQL Documentation

A trigger is a statement that the system executes automatically as a side effect of a modification to the database.

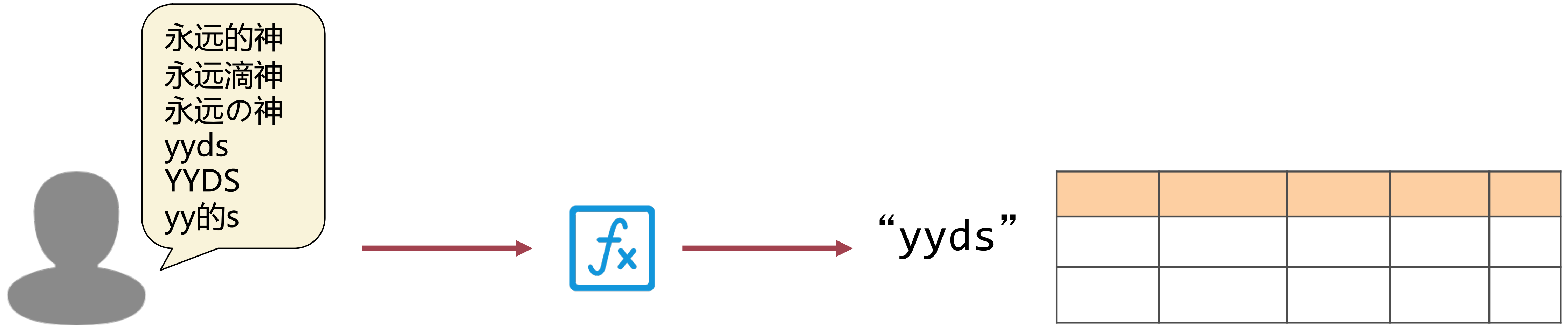
-- Chapter 5.3, Database System Concepts, 7th

- We can attach “actions” to a table
  - They will be executed automatically whenever the data in the table changes
- Purpose of using triggers
  - Validate data
  - Checking complex rules
  - Manage data redundancy

Purpose of Using Triggers

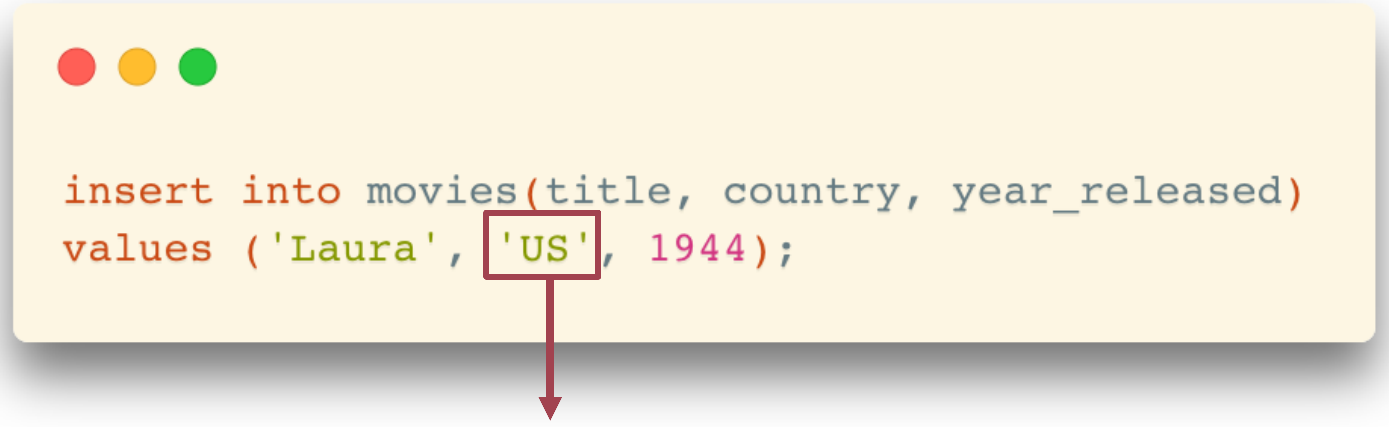
- Validate data
  - Some data are badly processed in programs before sending to the database
  - We need to validate such data before inserting them into the database

  - "On-the-fly" modification
    - Change the input directly when the input arrives



Purpose of Using Triggers

- Validate data
  - Example: insert a row in the movies table
    - In the JDBC program, an `insert` request is written like the following:



```
insert into movies(title, country, year_released)
values ('Laura', 'US', 1944);
```

Need to update it to 'us' before inserting

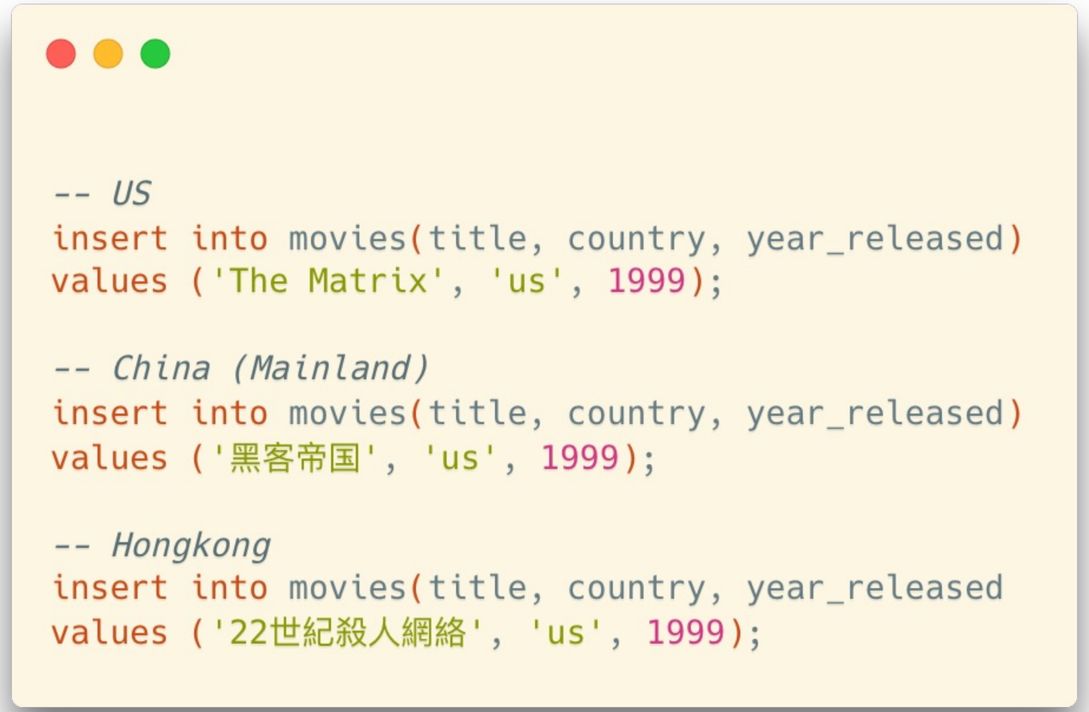
  - Although,
    - Such validation or transformation should be better handled by the application programs

Purpose of Using Triggers

- Check complex rules
  - Sometimes, the business rules are so complex that it cannot be checked via declarative constraints

Purpose of Using Triggers

- Manage data redundancy
  - Some redundancy issues may not be avoided by simply adding constraints
  - For example: We inserted the same movie but in different languages



```
-- US
insert into movies(title, country, year_released)
values ('The Matrix', 'us', 1999);

-- China (Mainland)
insert into movies(title, country, year_released)
values ('黑客帝国', 'us', 1999);

-- Hongkong
insert into movies(title, country, year_released
values ('22世紀殺人網絡', 'us', 1999);
```

It satisfies the constraint of uniqueness on (`title, country, year_released`)

- ... but they represent the same movie

Trigger Activation

- Two key points:
  - When to fire a trigger?
  - What (command) fires a trigger?

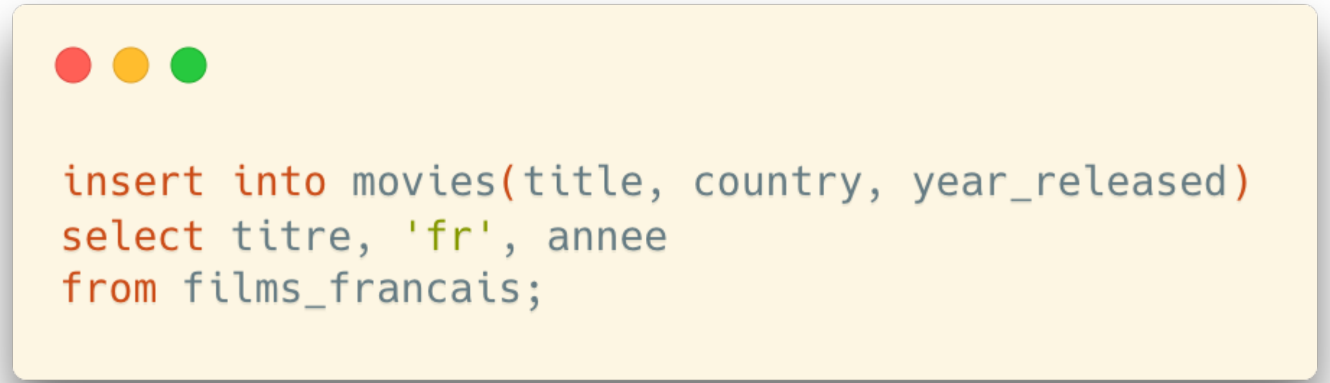
Trigger Activation

- When to fire a trigger?
  - In general: “During the change of data”
    - … but we need a detailed discussion

--- Note: “During the change” means `select` queries won’t fire a trigger.

Trigger Activation: When

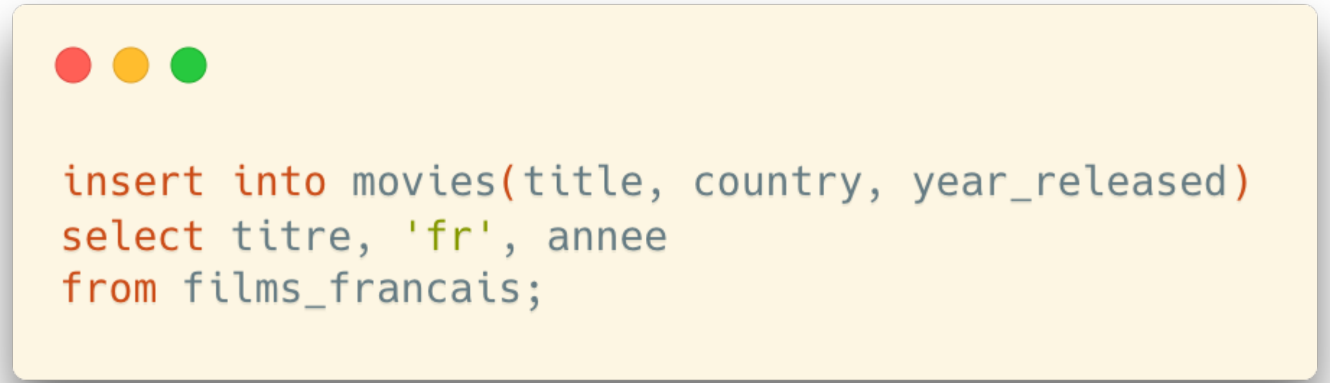
- Example: Insert a set of rows with “`insert into select`”
  - One statement, multiple rows



```sql
insert into movies(title, country, year_released)
select titre, 'fr', annee
from films_francais;
```

Trigger Activation: When

- Example: Insert a set of rows with "`insert into select`"
    - One statement, multiple rows



```
insert into movies(title, country, year_released)
select titre, 'fr', annee
from films_francais;
```

- Option 1: Fire a trigger only once for the statement
    - Before the first row is inserted, or after the last row is inserted
- Option 2: Fire a trigger for each row
    - Before or after the row is inserted

Trigger Activation: When

- Different options between DBMS products

PostgreSQL
ORACLE
IBM DB2

- Before statement
  - Before each row
  - After each row
- After statement

MySQL
SQLite

- ~~Before statement~~
  - Before each row
  - After each row
- ~~After statement~~

Microsoft SQL Server

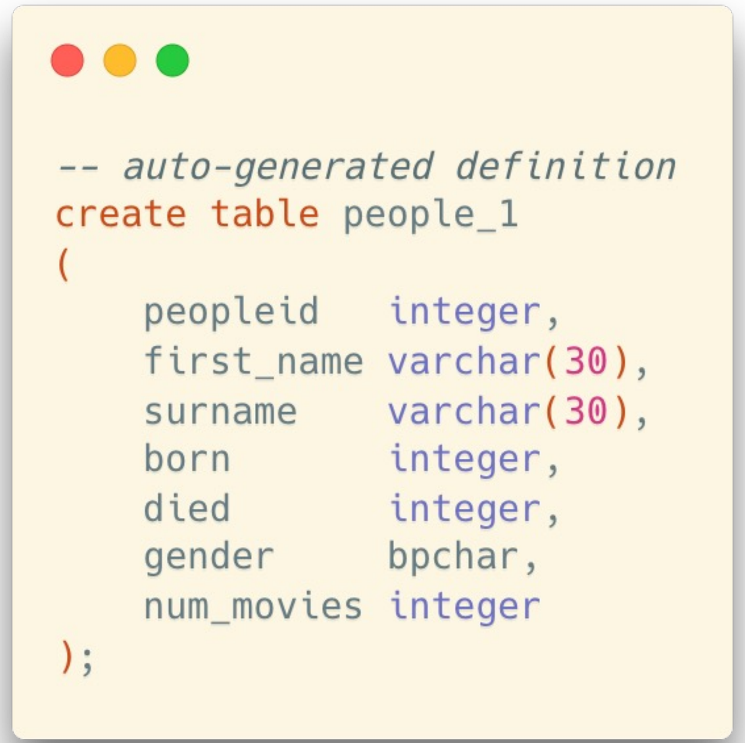
- ~~Before statement~~
  - ~~Before each row~~
  - ~~After each row~~
- After statement

Trigger Activation: What

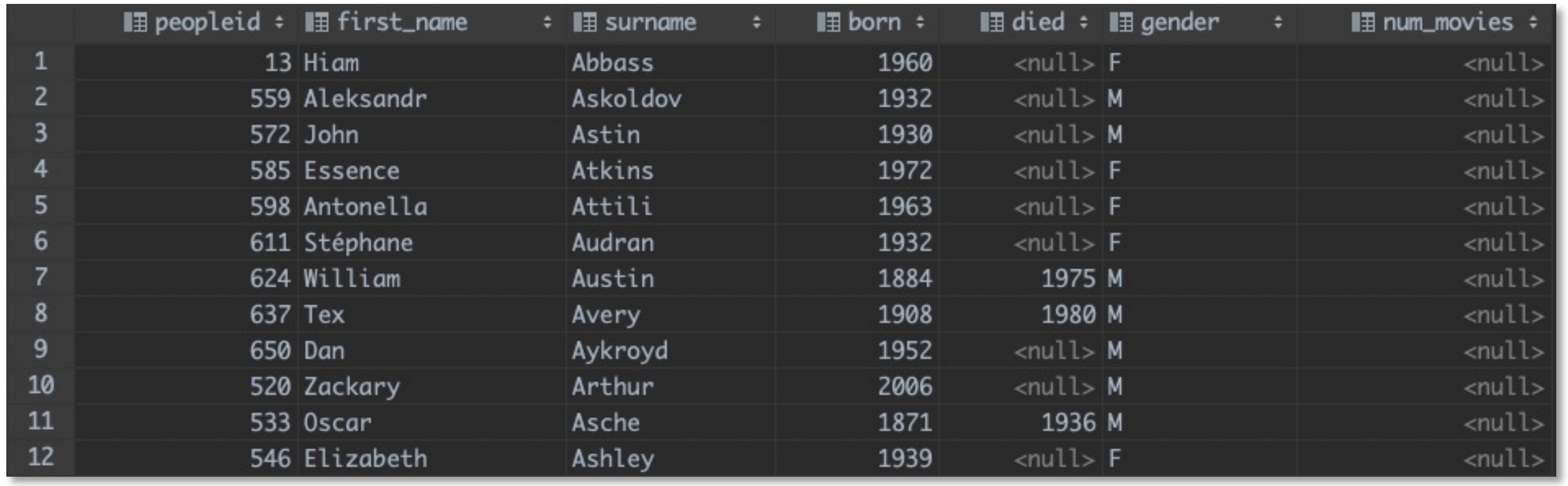
- What (command) fires a trigger?
  - `insert`
  - `update`
  - `delete`

Example of Triggers

- A (Toy) Example
  - For the following `people_1` table, count the number of movies when updating a person and save the result in the `num_movies` column



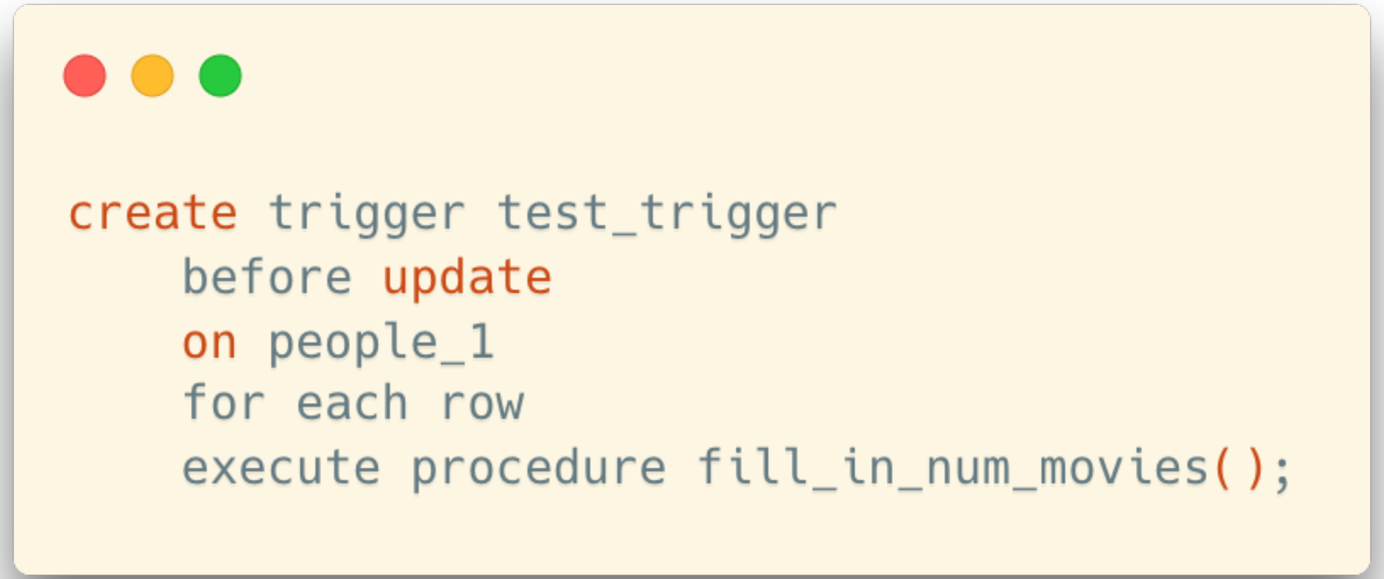
```
-- auto-generated definition
create table people_1
(
    peopleid   integer,
    first_name varchar(30),
    surname    varchar(30),
    born       integer,
    died       integer,
    gender     bpchar,
    num_movies integer
);
```



	peopleid	first_name	surname	born	died	gender	num_movies
1	13	Hiam	Abbass	1960	<null>	F	<null>
2	559	Aleksandr	Askoldov	1932	<null>	M	<null>
3	572	John	Astin	1930	<null>	M	<null>
4	585	Essence	Atkins	1972	<null>	F	<null>
5	598	Antonella	Attili	1963	<null>	F	<null>
6	611	Stéphane	Audran	1932	<null>	F	<null>
7	624	William	Austin	1884	1975	M	<null>
8	637	Tex	Avery	1908	1980	M	<null>
9	650	Dan	Aykroyd	1952	<null>	M	<null>
10	520	Zackary	Arthur	2006	<null>	M	<null>
11	533	Oscar	Asche	1871	1936	M	<null>
12	546	Elizabeth	Ashley	1939	<null>	F	<null>

Example of Triggers

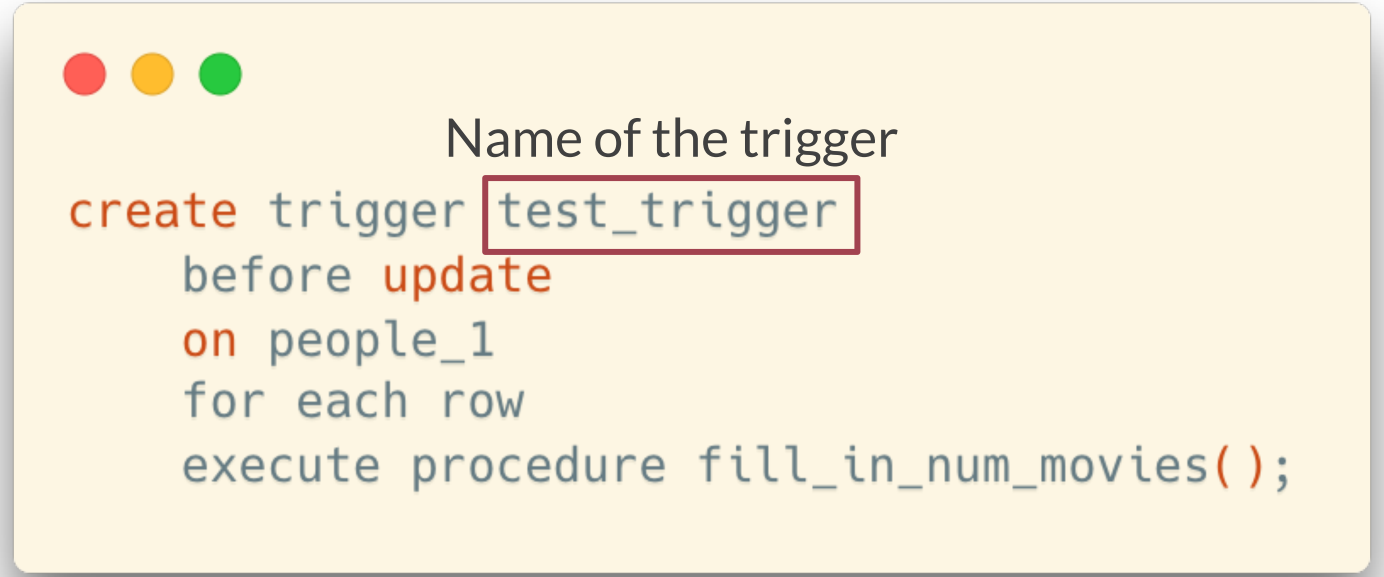
- Create a trigger



```
create trigger test_trigger
    before update
    on people_1
    for each row
    execute procedure fill_in_num_movies();
```

Example of Triggers

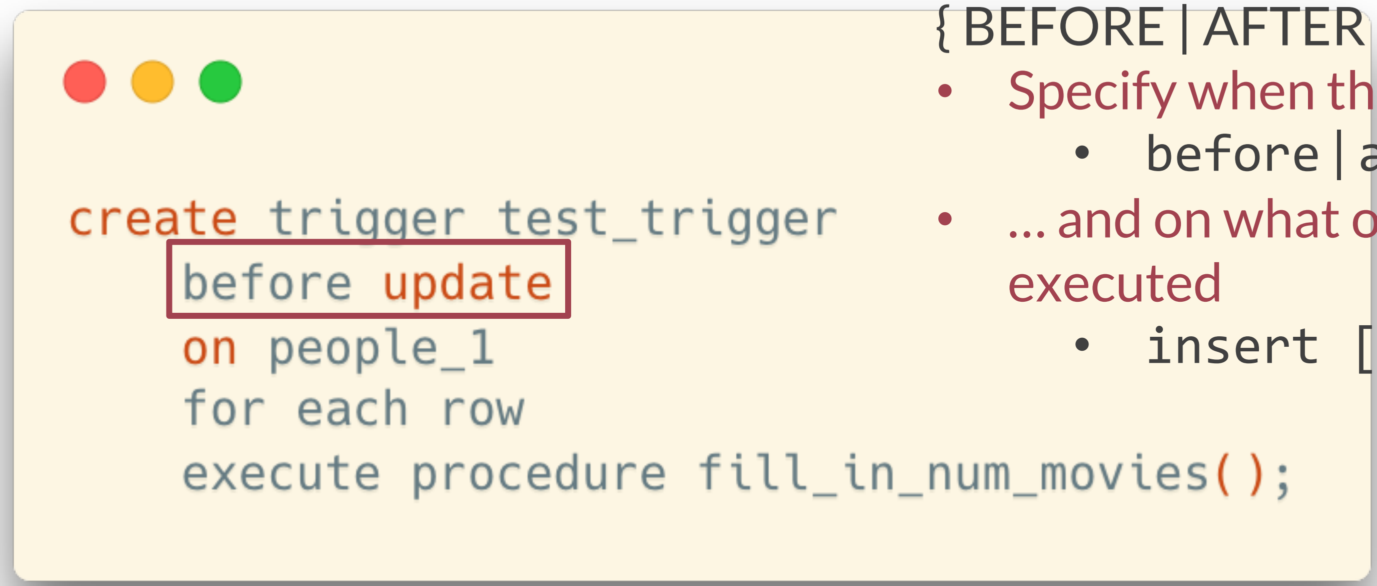
- Create a trigger



```
create trigger test_trigger
    before update
    on people_1
    for each row
    execute procedure fill_in_num_movies();
```

Example of Triggers

- Create a trigger



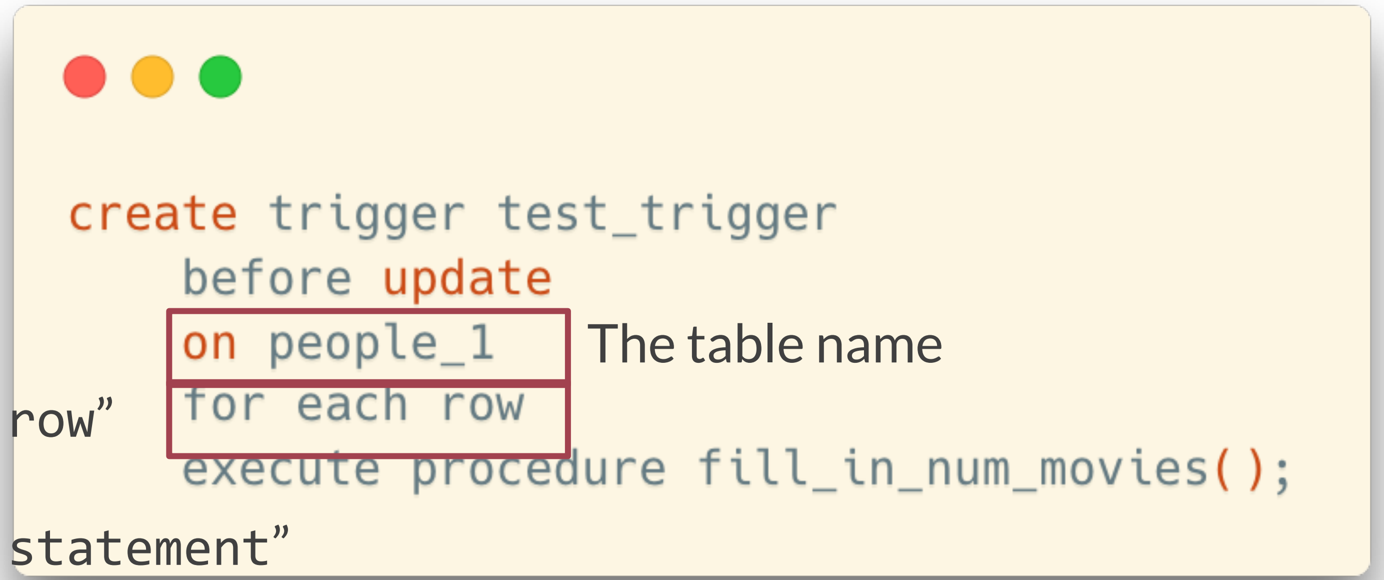
```
create trigger test_trigger
    before update
    on people_1
    for each row
    execute procedure fill_in_num_movies();
```

{ BEFORE | AFTER | INSTEAD OF } { event [OR ...] }

- Specify when the trigger will be executed
    - `before | after`
- ... and on what operations the trigger will be executed
    - `insert [or update [or delete]]`

Example of Triggers

- Create a trigger



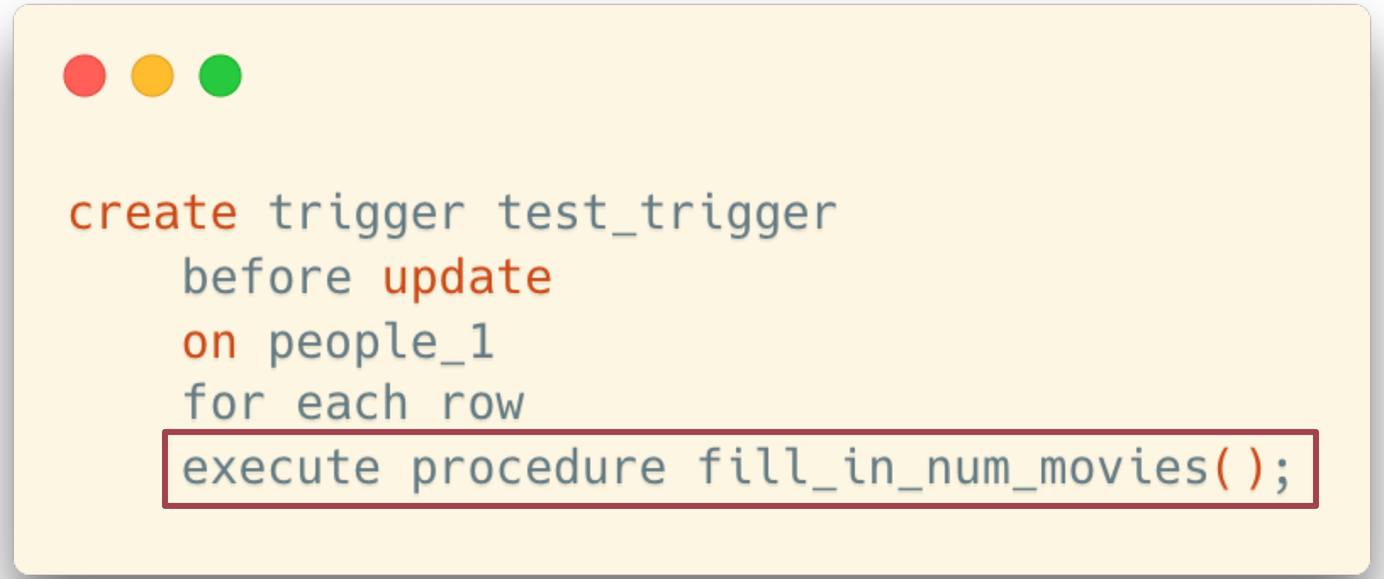
```
create trigger test_trigger
    before update
    on people_1
    for each row
    execute procedure fill_in_num_movies();
```

The table name

"`for each row`"
or
"`for each statement`"
(default)

Example of Triggers

- Create a trigger

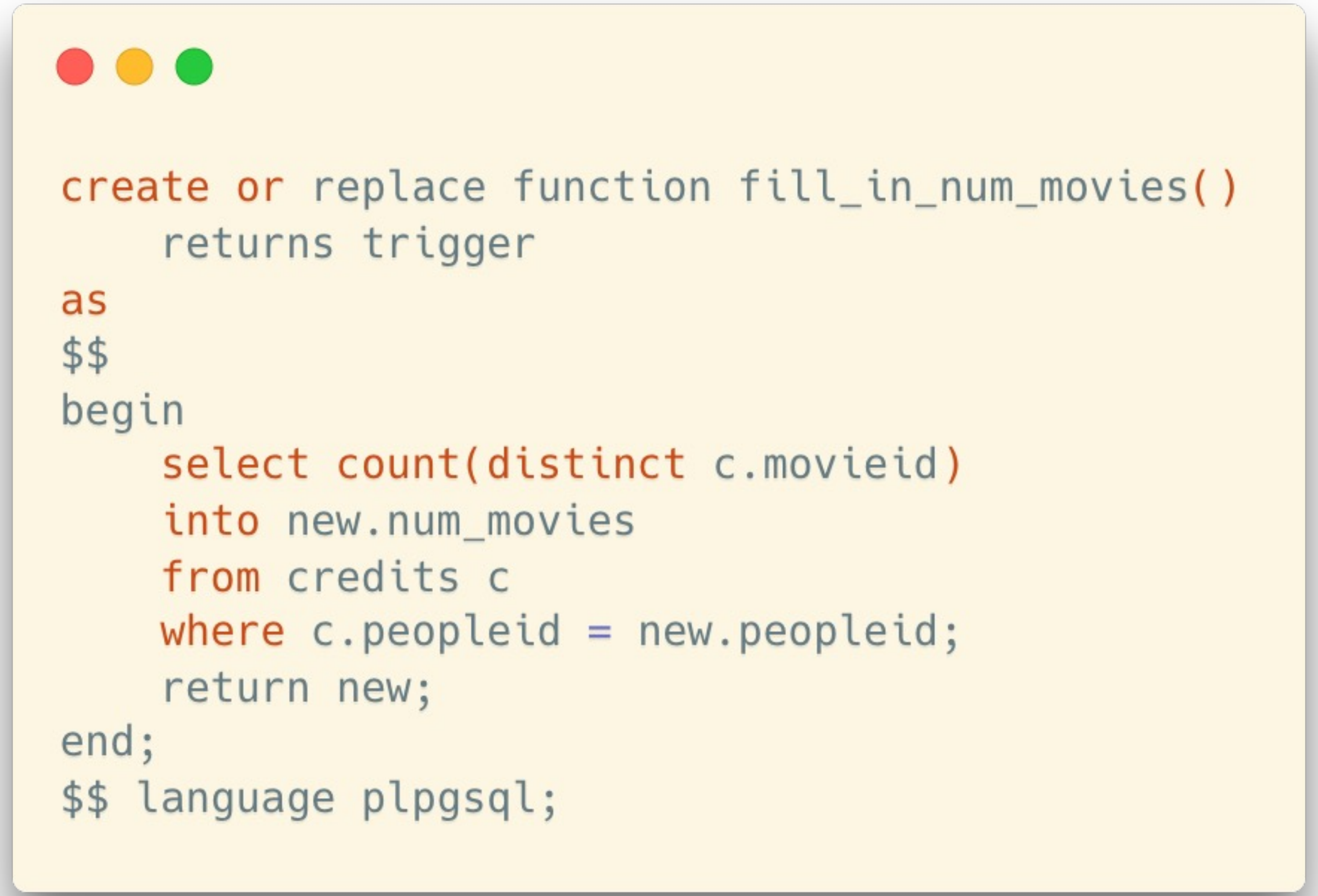


```
create trigger test_trigger
    before update
    on people_1
    for each row
    execute procedure fill_in_num_movies();
```

The actual procedure for the trigger

Example of Triggers

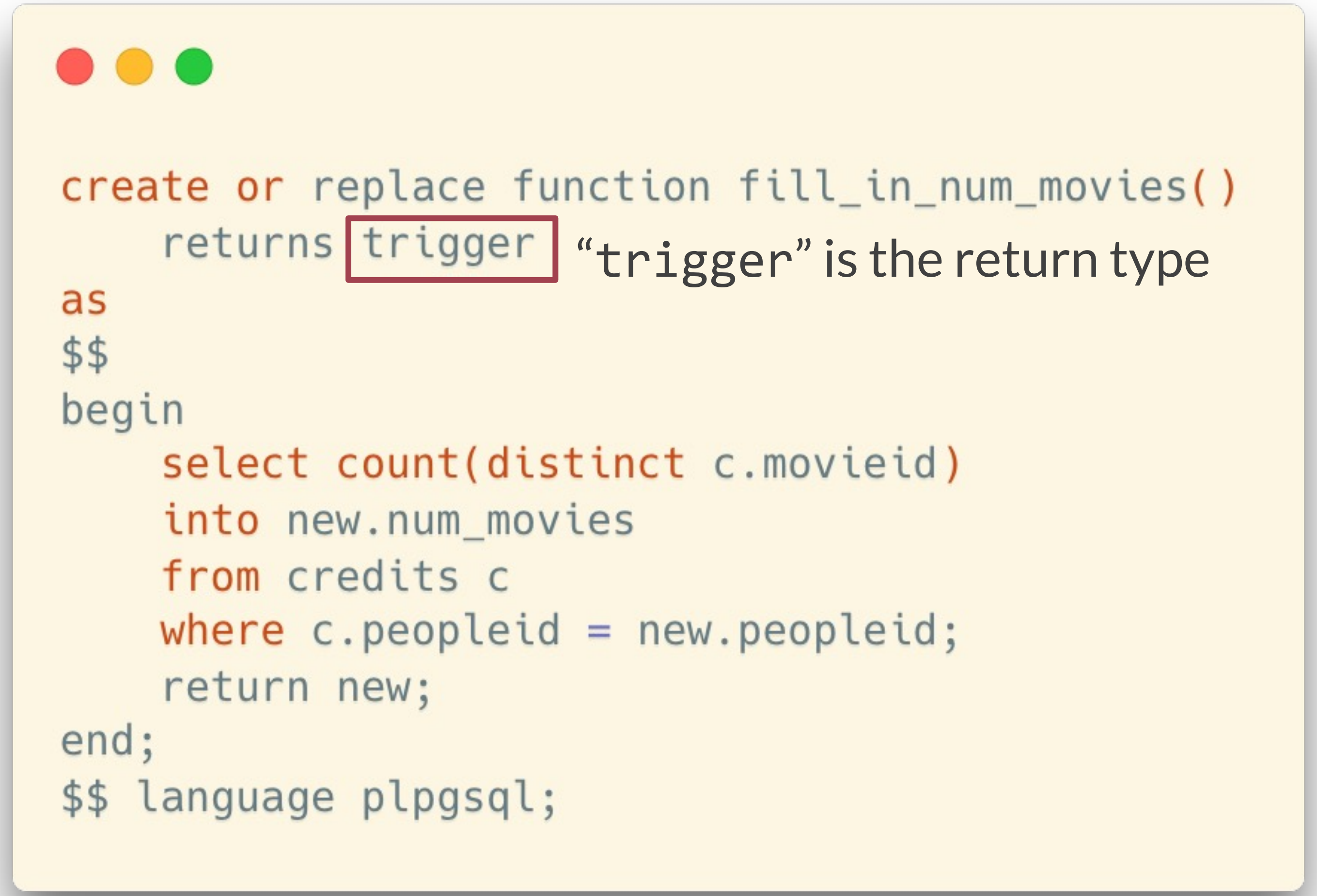
- Create a trigger
  - Besides, a corresponding procedure should be created as well



```
create or replace function fill_in_num_movies()
    returns trigger
as
$$
begin
    select count(distinct c.movieid)
    into new.num_movies
    from credits c
    where c.peopleid = new.peopleid;
    return new;
end;
$$ language plpgsql;
```

Example of Triggers

- Create a trigger
  - Besides, a corresponding procedure should be created as well

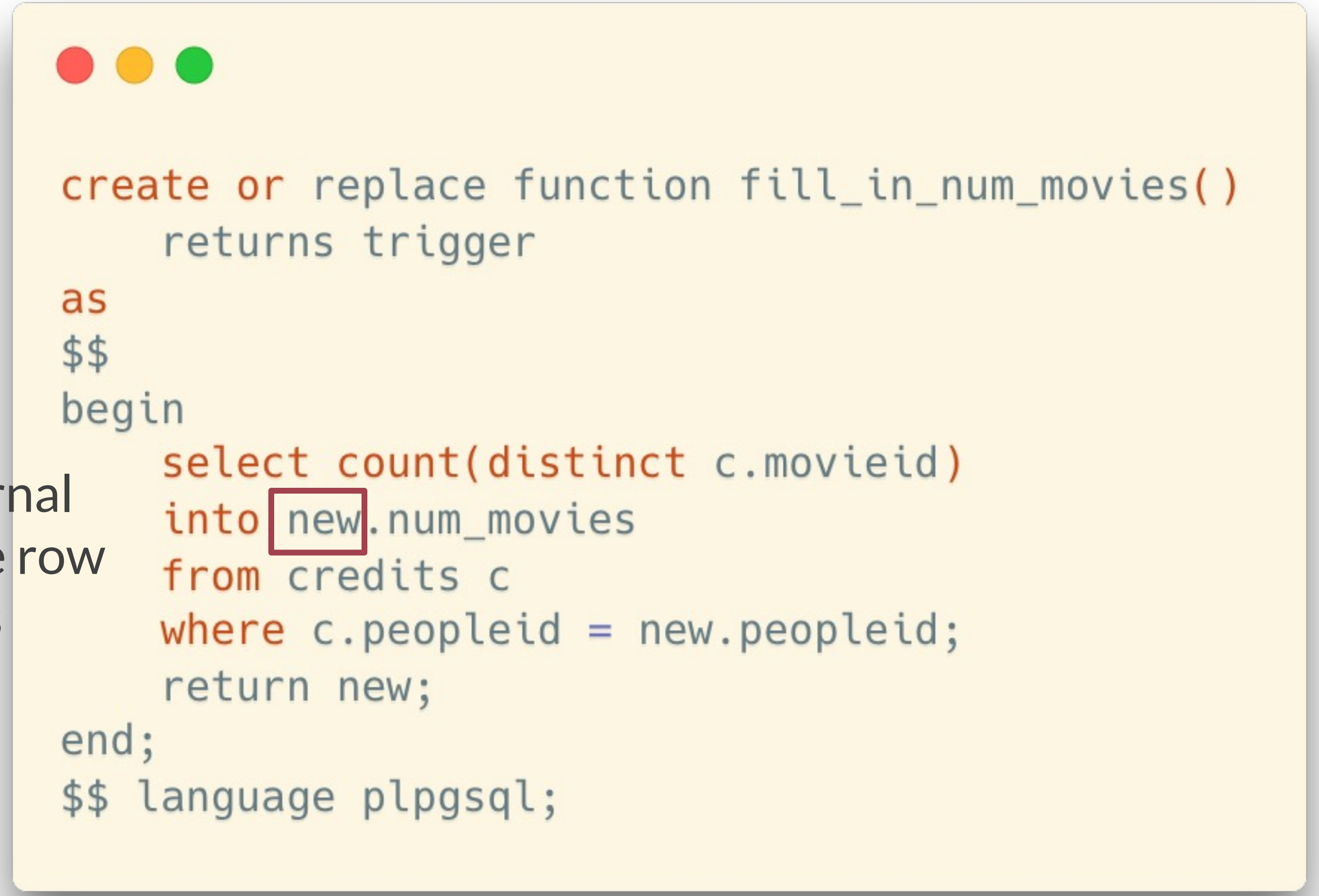


```
create or replace function fill_in_num_movies()
    returns trigger
as
$$
begin
    select count(distinct c.movieid)
    into new.num_movies
    from credits c
    where c.peopleid = new.peopleid;
    return new;
end;
$$ language plpgsql;
```

"trigger" is the return type

Example of Triggers

- Create a trigger
  - Besides, a corresponding procedure should be created as well

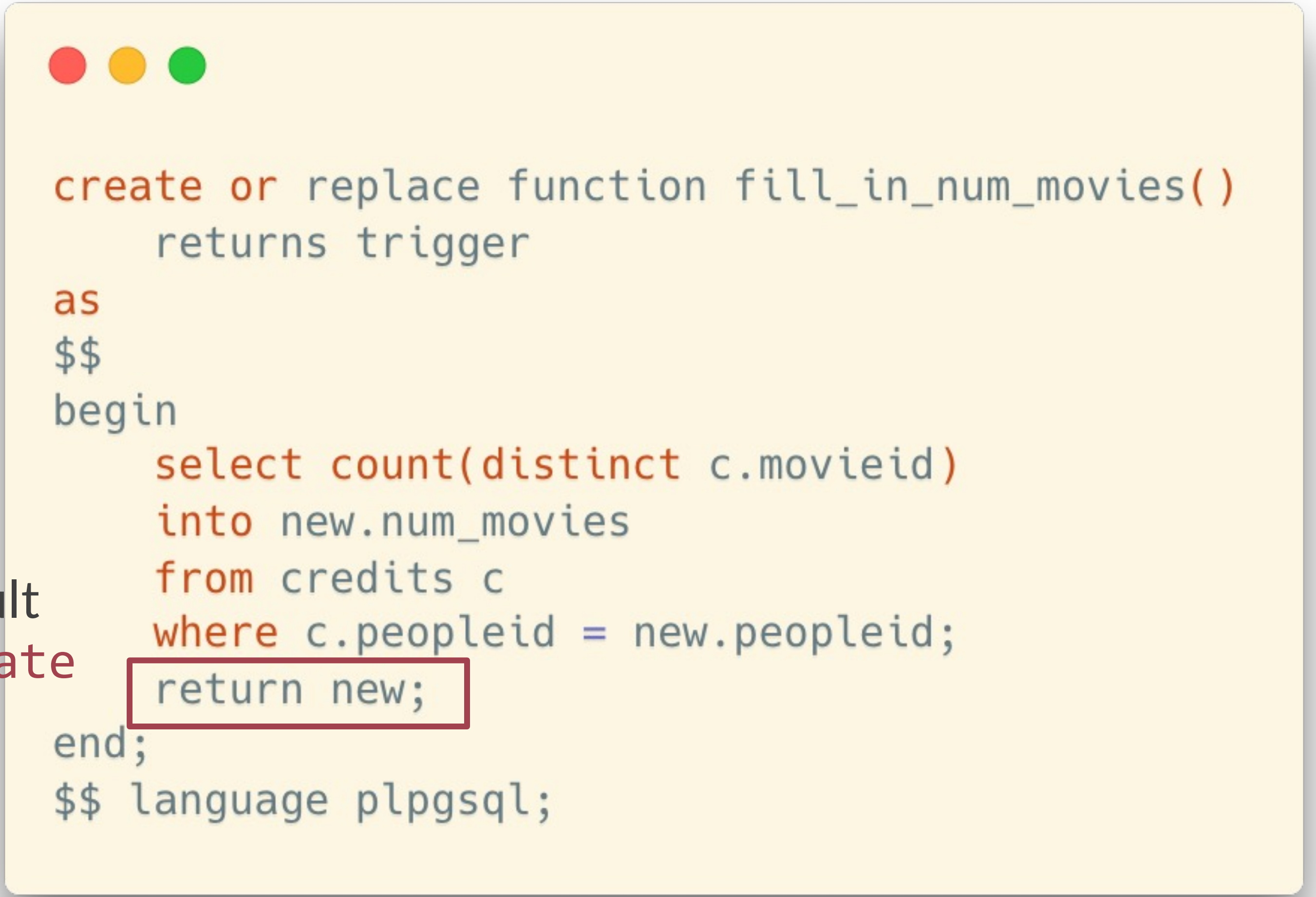


```
create or replace function fill_in_num_movies()
    returns trigger
as
$$
begin
    select count(distinct c.movieid)
    into new.num_movies
    from credits c
    where c.peopleid = new.peopleid;
    return new;
end;
$$ language plpgsql;
```

”new” and “old” are two internal variables that represents the row before and after the changes

Example of Triggers

- Create a trigger
  - Besides, a corresponding procedure should be created as well



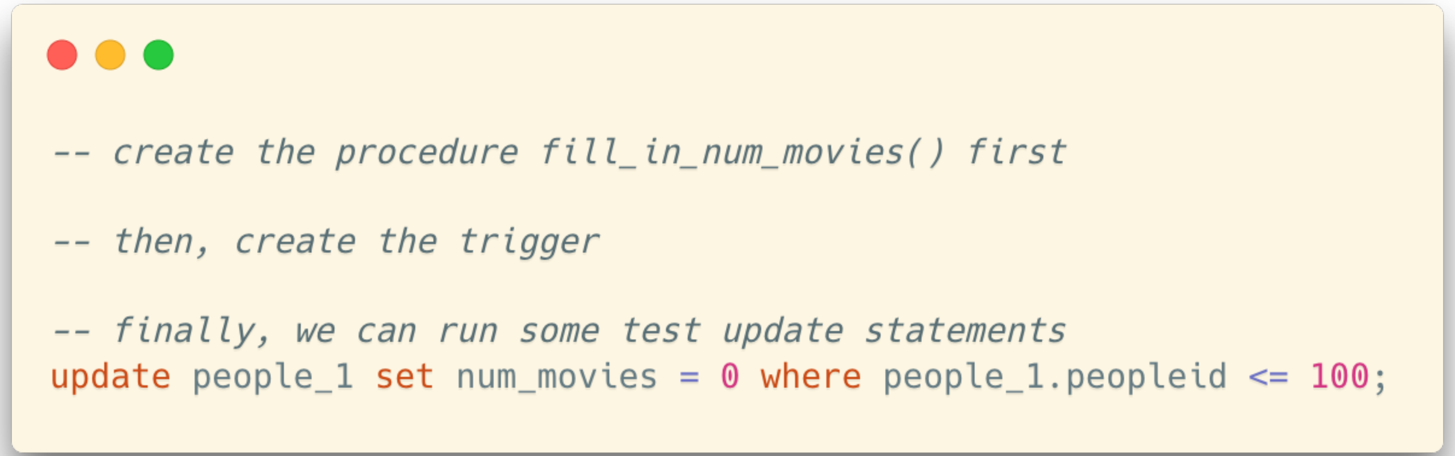
```
create or replace function fill_in_num_movies()
    returns trigger
as
$$
begin
    select count(distinct c.movieid)
    into new.num_movies
    from credits c
    where c.peopleid = new.peopleid;
    return new;
end;
$$ language plpgsql;
```

Remember to return the result which will be used in the `update` statement

Example of Triggers

- Create a trigger
  - Besides, a corresponding procedure should be created as well
    - Remember to create the procedure before creating the trigger

- Run test updates



```sql
-- create the procedure fill_in_num_movies() first

-- then, create the trigger

-- finally, we can run some test update statements
update people_1 set num_movies = 0 where people_1.peopleid <= 100;
```

Before and After Triggers

- Differences between before and after triggers
    - “Before” and “after” the operation is done (`insert`, `update`, `delete`)
    - If we want to update the incoming values in an `update` statement, the “before trigger” should be used since the incoming values have not been written to the table yet

Before and After Triggers

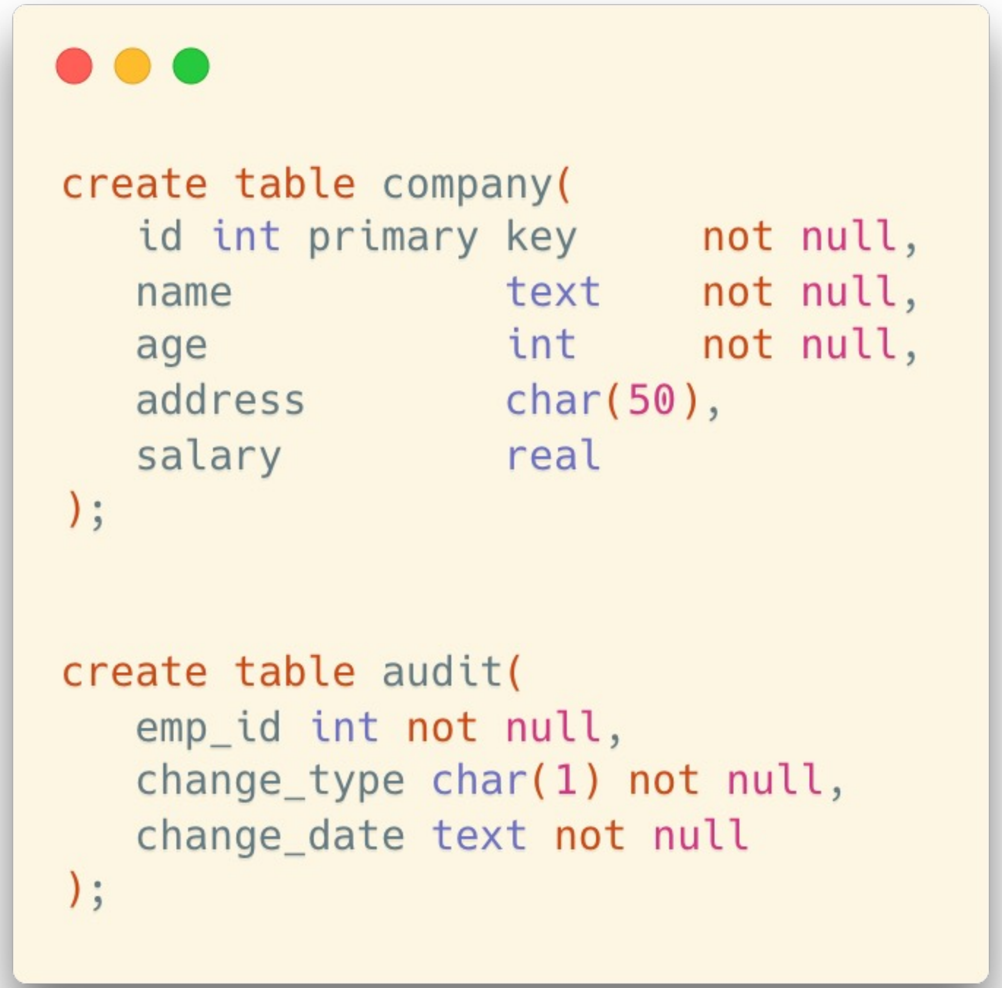
- Typical usage scenarios for trigger settings
  - Modify input on the fly
    - before insert / update
    - for each row
  - Check complex rules
    - before insert / update / delete
    - for each row
  - Manage data redundancy
    - after insert / update / delete
    - for each row

Example: Auditing

- One good example of managing some data redundancy is keeping an audit trail
  - It won't do anything for people who steal data
    - (remember that `select` cannot fire a trigger – although with the big products you can trace all queries)
  - … but it may be useful for checking people <u>who modify data</u> that they aren't supposed to modify

Example: Auditing

- Trace the insertions and updates to employees in a company

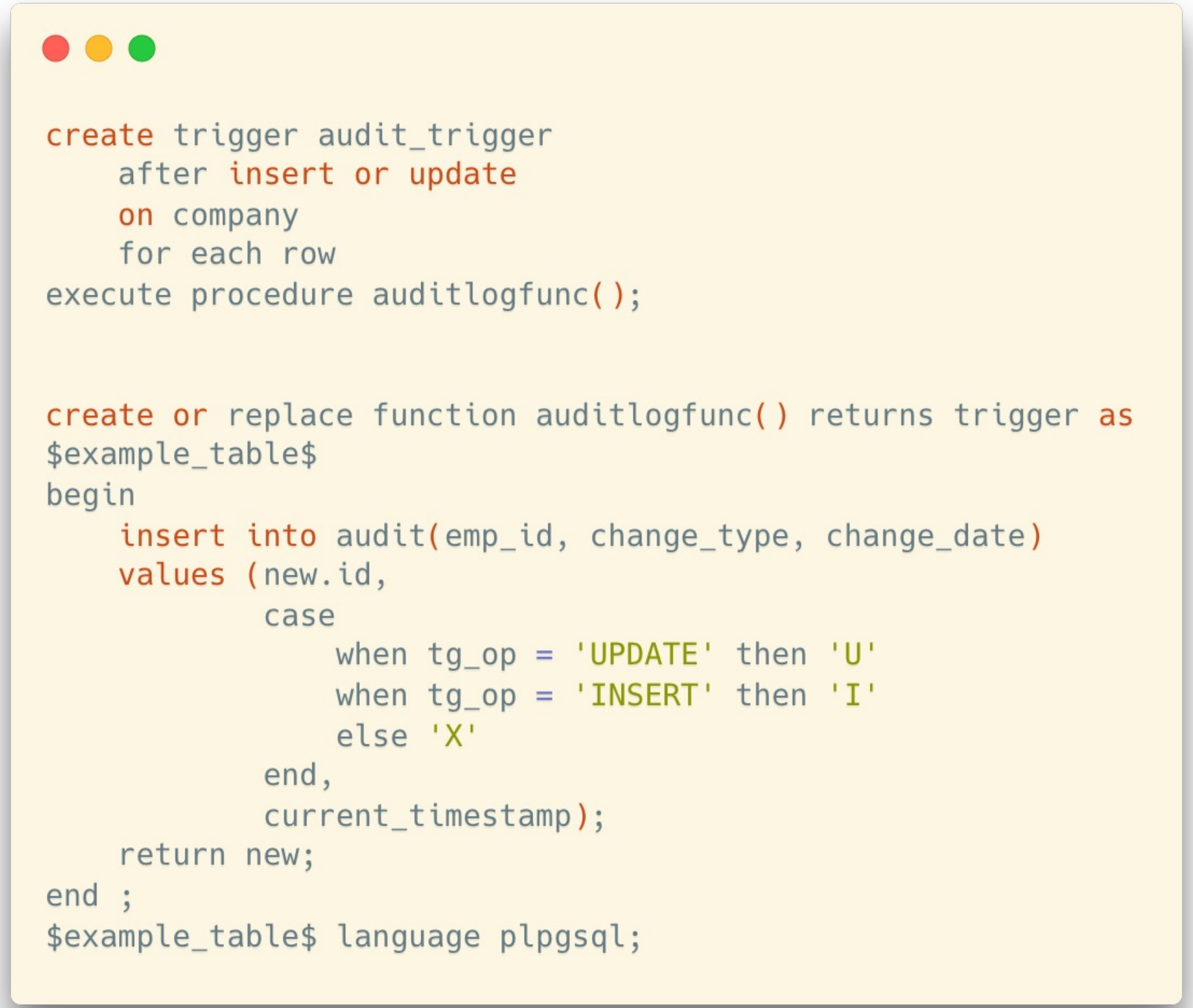


```
create table company(
   id int primary key     not null,
   name              text     not null,
   age               int      not null,
   address           char(50),
   salary            real
);


create table audit(
   emp_id int not null,
   change_type char(1) not null,
   change_date text not null
);
```

Example: Auditing

- Trace the insertions and updates to employees in a company

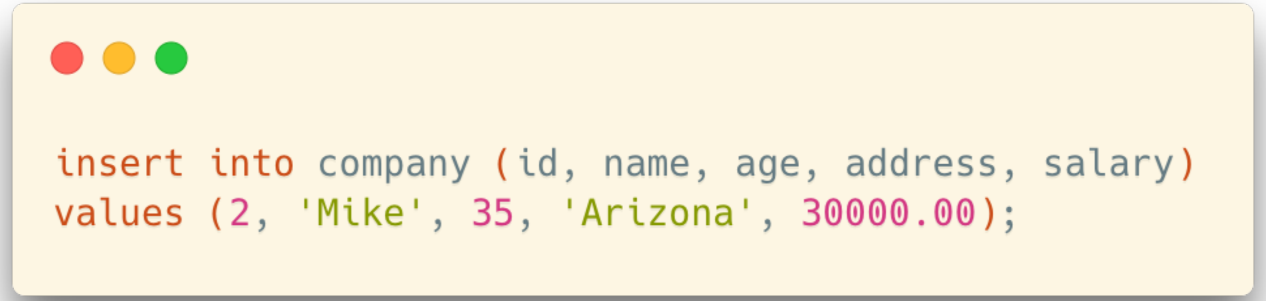


```
create trigger audit_trigger
    after insert or update
    on company
    for each row
execute procedure auditlogfunc();


create or replace function auditlogfunc() returns trigger as
$example_table$
begin
    insert into audit(emp_id, change_type, change_date)
    values (new.id,
            case
                when tg_op = 'UPDATE' then 'U'
                when tg_op = 'INSERT' then 'I'
                else 'X'
            end,
            current_timestamp);
    return new;
end ;
$example_table$ language plpgsql;
```

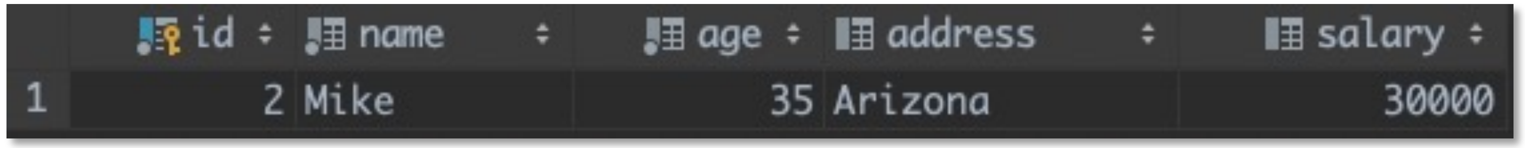
Example: Auditing

- Trace the insertions and updates to employees in a company



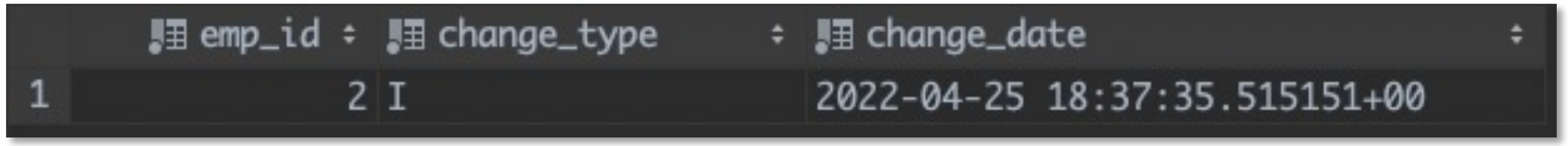
```sql
insert into company (id, name, age, address, salary)
values (2, 'Mike', 35, 'Arizona', 30000.00);
```

company



	id	name	age	address	salary
1	2	Mike	35	Arizona	30000

audit



	emp_id	change_type	change_date
1	2	I	2022-04-25 18:37:35.515151+00