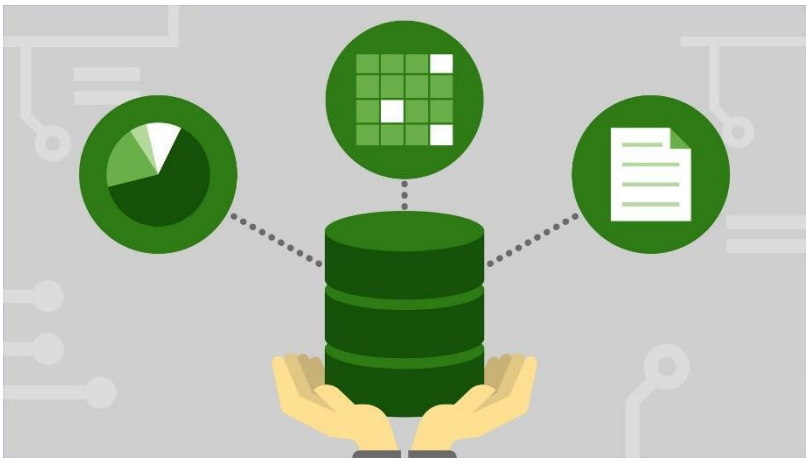
به نام خدا

دانشگاه تهران
پردیس دانشکده‌های فنی
دانشکده برق و کامپیوتر



آزمایشگاه پایگاه داده

دستور کار شماره 5

حمیدرضا خدادادی
810197499

دی ماه ۱۴۰۰

گزارش دستور کار انجام شده

توابع و تریگر ها:

- تعریف کردن یک trigger:

```
CREATE TRIGGER name { BEFORE | AFTER | INSTEAD OF } { event [ OR ... ] }
    ON table_name
    FOR EACH ROW EXECUTE PROCEDURE function_name()

where event can be one of:

    INSERT
    UPDATE [ OF column_name [, ... ] ]
    DELETE
    TRUNCATE
```

- تعریف کردن یک function:

```
CREATE [ OR REPLACE ] FUNCTION
    name () RETURNS TRIGGER
  { LANGUAGE lang_name
    | SECURITY DEFINER
    | SET configuration_parameter { TO value | = value | FROM CURRENT }
    | AS 'definition'
  }...
```

در ابتدا به یک جدول برای کار کردن روی آن نیاز داریم. به عنوان مثال، فرض کنید باید داده‌های هویتی اولیه حساب را ذخیره کنیم. بدین منظور یک جدول person ایجاد می کنیم.

```
CREATE TABLE person (
    login_name varchar(9) not null primary key,
    display_name text
);
```

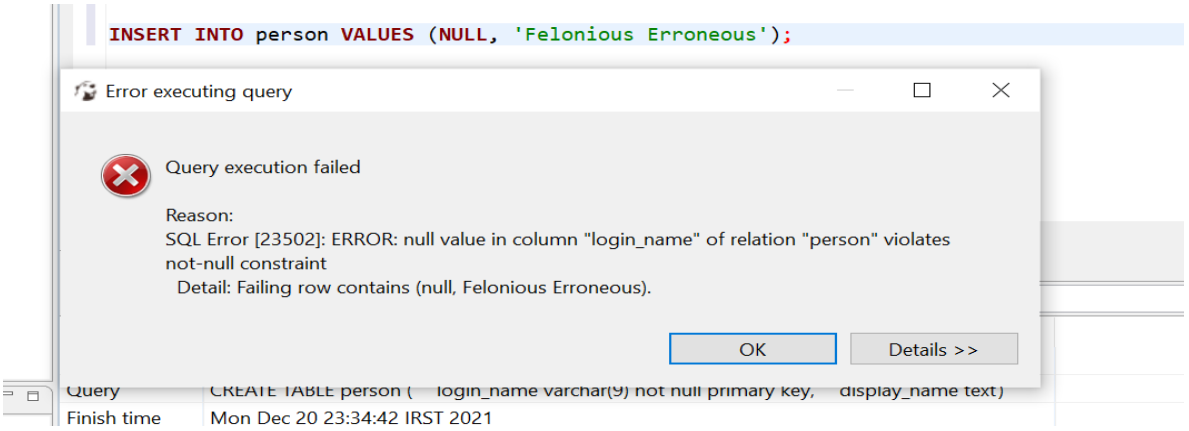
Statistics

CREATE TABLE person (login_name varcha | Enter a SQL expression to filter results (use Ctrl+Space)

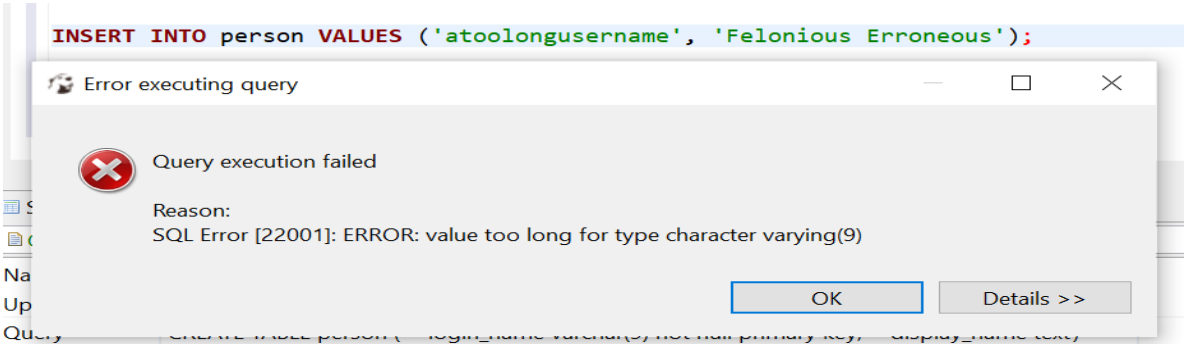
Name	Value
Updated Rows	0
Query	CREATE TABLE person (login_name varchar(9) not null primary key, display_name text)
Finish time	Mon Dec 20 23:34:42 IRST 2021

همانطور که مشاهده می شود، در دستور ساخت این جدول، برای ستون login_name سایز حداکثر 9 تعیین شده و اینکه نباید null باشد.

پس اگر مقدار null به جای login_name در جدول insert کنیم، از اضافه شدن آن جلوگیری می شود.



و اگر مقداری بزرگ تر از سایز 9 برای login_name در جدول insert کنیم، از اضافه شدن آن جلوگیری می شود.



سایر محدودیت ها، مانند نیاز به حداقل طول و رد کردن کارکتر های خاص را می‌توان با محدودیت‌های constraint اعمال کرد.

در ادامه دو constraint روی صفت login_name از ستون person اعمال می کنیم تا محدودیت اول اینکه بررسی کند اندازه آن بزرگ تر از صفر باشد (استرینگ خالی نباشد)، و محدودیت دوم اینکه بررسی کند تا کارکتر space در استرینگ آن وجود نداشته باشد.

```
-----------------------------------------------------------------------

ALTER TABLE person
    ADD CONSTRAINT PERSON_LOGIN_NAME_NON_NULL
    CHECK (LENGTH(login_name) > 0);

-----------------------------------------------------------------------
```

Statistics

ALTER TABLE person ADD CONSTRAINT P | Enter a SQL expression to filter results (use Ctrl+Space)

Name	Value
Updated Rows	0
Query	ALTER TABLE person ADD CONSTRAINT PERSON_LOGIN_NAME_NON_NULL CHECK (LENGTH(login_name) > 0)
Finish time	Mon Dec 20 23:38:26 IRST 2021

```
-----------------------------------------------------------------------

ALTER TABLE person
    ADD CONSTRAINT person_login_name_no_space
    CHECK (POSITION(' ' IN login_name) = 0);

-----------------------------------------------------------------------
```

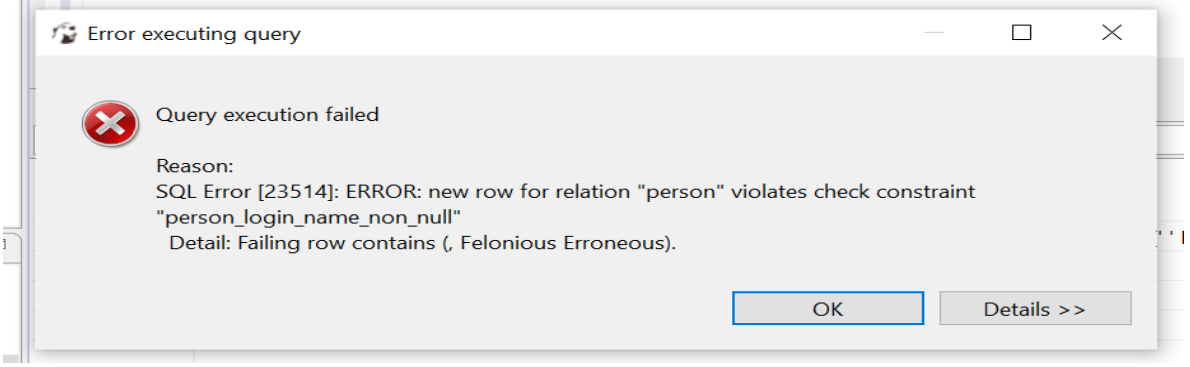
Statistics

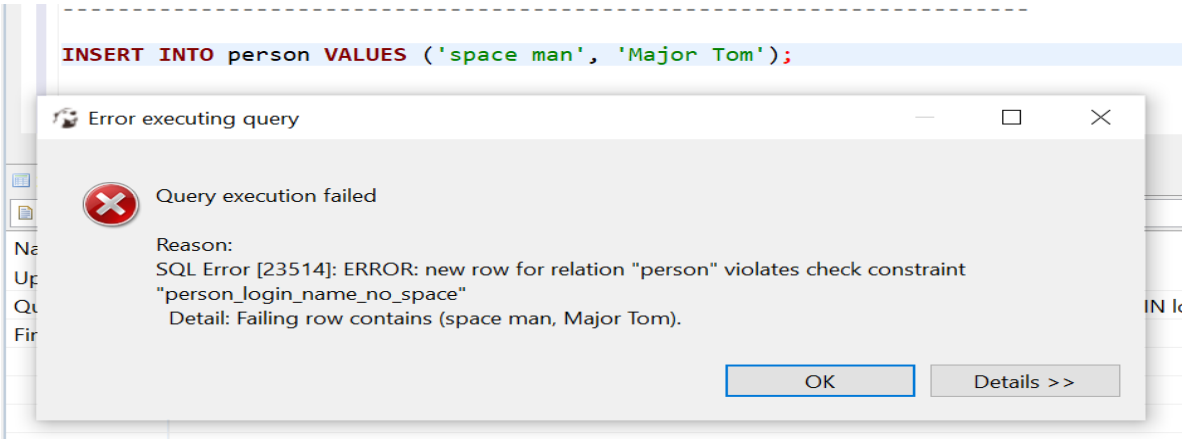
ALTER TABLE person ADD CONSTRAINT p | Enter a SQL expression to filter results (use Ctrl+Space)

Name	Value
Updated Rows	0
Query	ALTER TABLE person ADD CONSTRAINT person_login_name_no_space CHECK (POSITION(' ' IN login_name) = 0)
Finish time	Mon Dec 20 23:39:12 IRST 2021

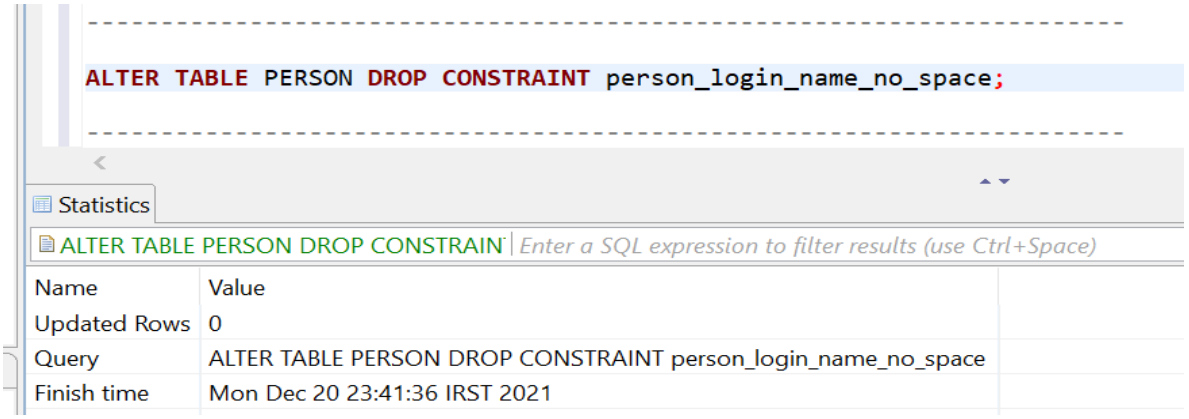
در ادامه خواهیم دید که اگر برای login_name یک مقدار برابر استرینگ خالی یا یک مقدار دارای کارکتر ' '، اعمال کنیم، با خطا مواجه خواهیم شد و از اضافه شدن آن جلوگیری به عمل می آید.

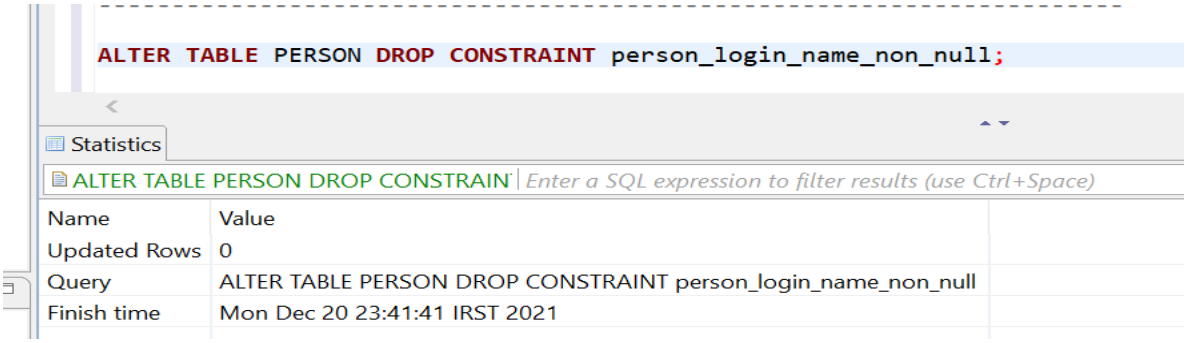
```
-----------------------------------------------------------------------

INSERT INTO person VALUES ('', 'Felonious Erroneous');
```





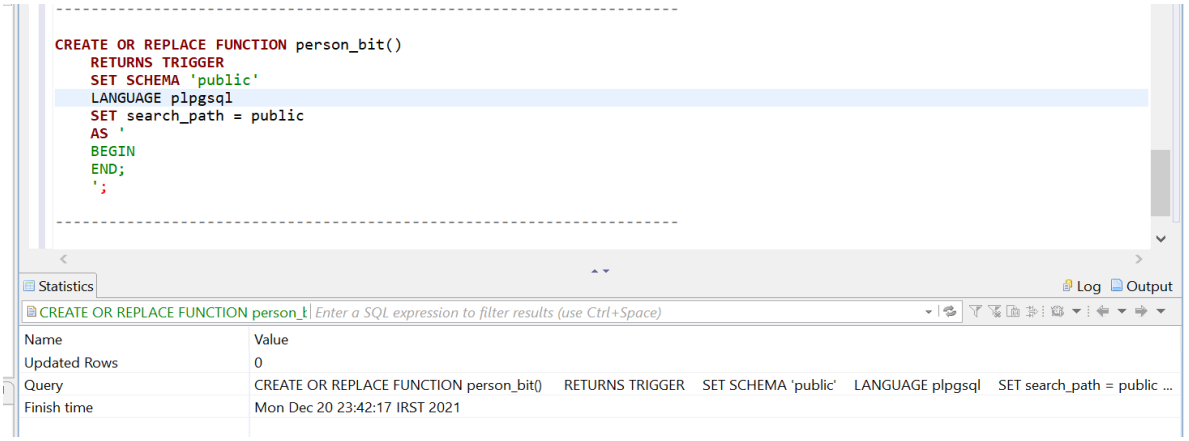
حال این دو constraint اضافه شده روی login_name را پاک می کنیم.



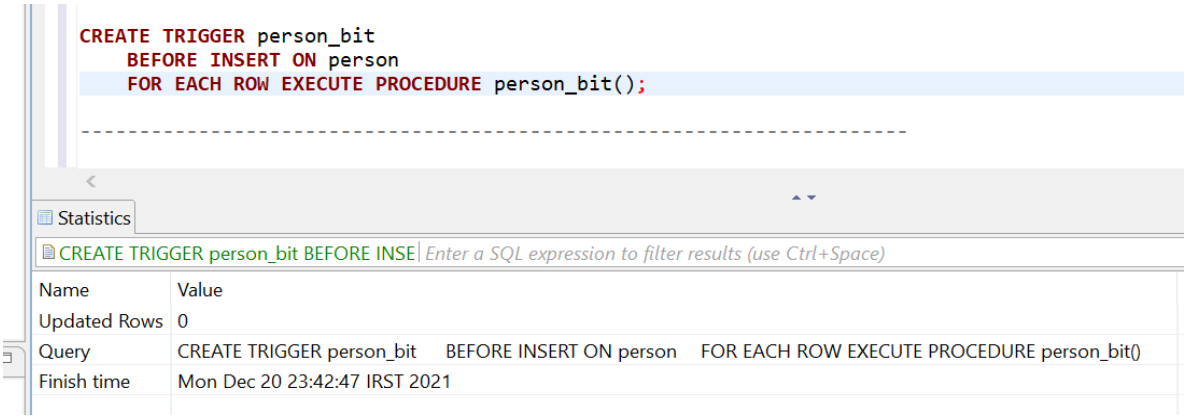


اما پیام خطا به اندازه کافی گویا نیست و ما می خواهیم با استفاده از توابع و تریگر ها به اطلاعات بیشتری برسیم.

برای این کار در ابتدا تابع person_bit() با بدنه خالی ایجاد می کنیم. و در تریگر ها از آن استفاده می کنیم.



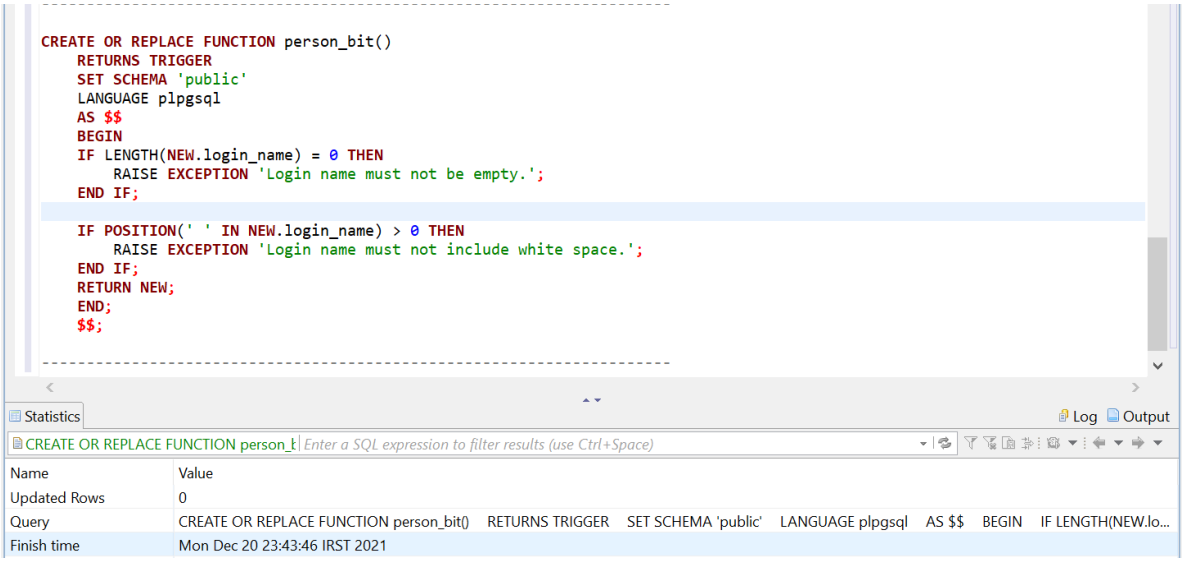
در ادامه یک trigger هم نام با تابع ایجاد شده در عکس قبل یعنی person_bit ایجاد می کنیم. و این trigger بدین صورت کار خواهد کرد که قبل از هر insert روی جدول person فعال می شود و به ازای هر سطر آن تابع person_bit() را اجرا می کند.



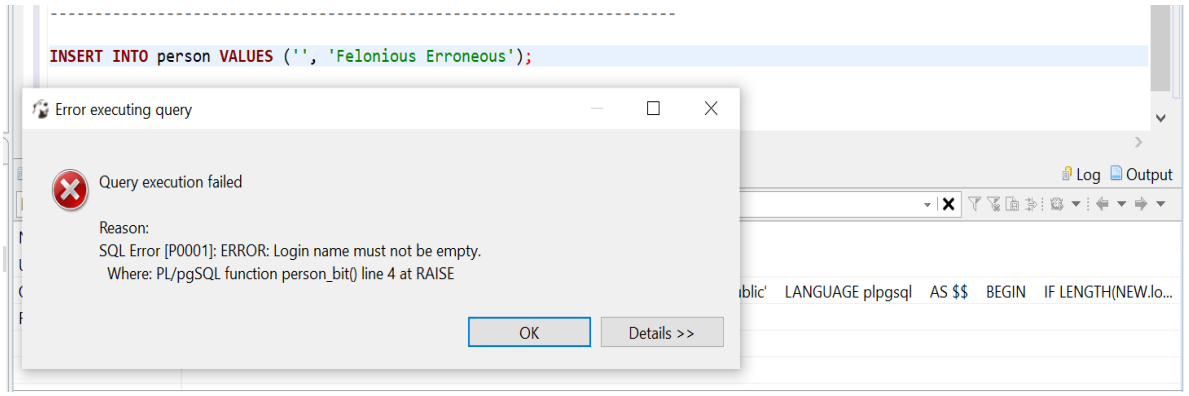
- **مثال 0: اعتبار سنجی داده ها**

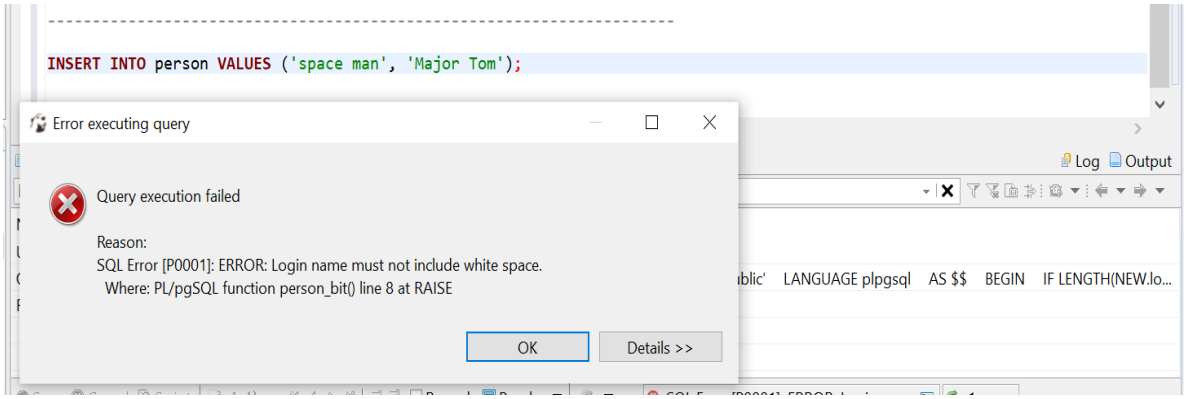
در این مثال، بدنه تابع ایجاد شده را به گونه ای تغییر می دهیم تا دو constraint ایجاد شده در مراحل قبل را بررسی کند و در صورت نقض شدن exception مناسب بدهد.

این تابع را به گونه ای عوض می کنیم تا اگر سطر جدیدی که قرار است در جدول insert شود (متغیر NEWاشاره گر به آن سطر است.)، دارای login_name با طول صفر یا همان برابر استرینگ خالی بود یا اینکه دارای کارکتر ' ' یا همان کارکتر space در استرینگ آن بود، با خطا و exception مناسب مواجه خواهیم شد و از اضافه شدن آن جلوگیری به عمل می آید.



خواهیم دید که اگر برای login_name یک مقدار برابر استرینگ خالی یا یک مقدار دارای کارکتر ' '، اعمال کنیم، با خطا مواجه خواهیم شد و از اضافه شدن آن جلوگیری به عمل می آید. و این بار خطای مناسب تری خواهیم دید.

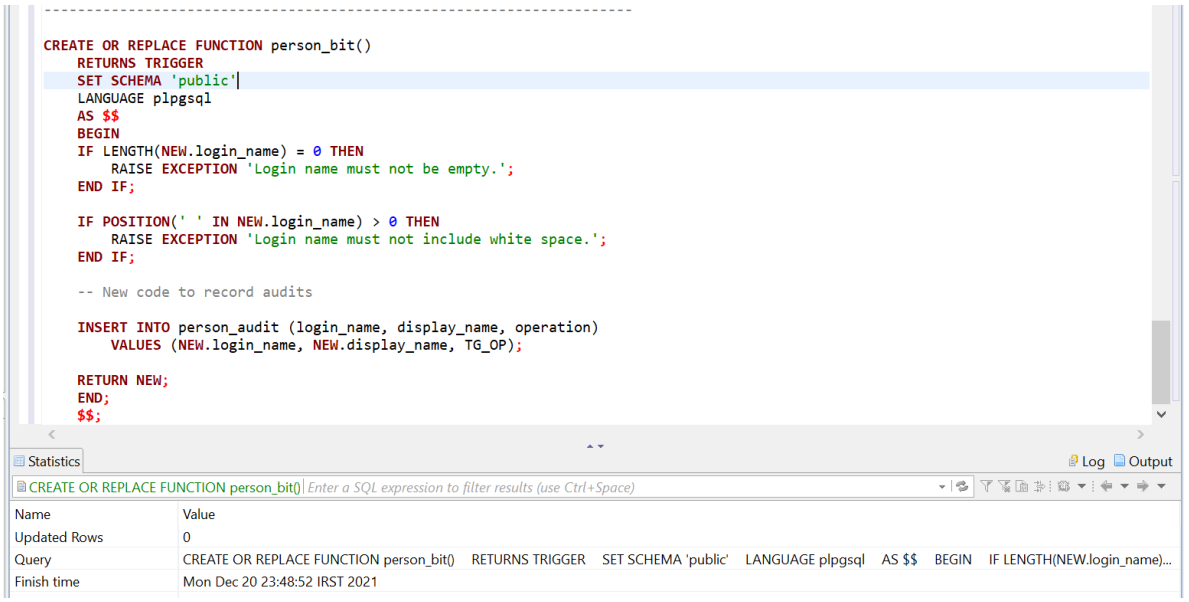




- **مثال 1: ثبت لاگ ها**

در این مثال ما می خواهیم یک جدول جدید با نام person_audit ایجاد کنیم و لاگ های اجرای عملیات ها را در آن ذخیره کنیم.

تابع person_bit() را به گونه ای تغییر می دهیم تا یک کپی از insert داده ها در جدول قبلی را در جدول person_audit که برای ذخیره لاگ ها ایجادش کردیم، insert کند.



حال برای این که بتوانیم از تابع جدید تعریف شده استفاده کنیم، یک trigger جدید ایجاد می کنیم. برای این کار trigger قبلی را حذف کرده و یک trigger جدید ایجاد می کنیم تا این trigger بدین صورت کار کند که قبل از هر insert و هر update روی جدول person فعال شود و به ازای هر سطر آن تابع person_bit() را اجرا می کند و لاگ شود.

```
-------------------------------------------------------------------

DROP TRIGGER person_bit ON person;

-------------------------------------------------------------------
```

Statistics

DROP TRIGGER person_bit ON person | Enter a SQL expression to filter results (use Ctrl+Space)

Name	Value
Updated Rows	0
Query	DROP TRIGGER person_bit ON person
Finish time	Mon Dec 20 23:49:27 IRST 2021

```
-------------------------------------------------------------------

CREATE TRIGGER person_biut
    BEFORE INSERT OR UPDATE ON person
    FOR EACH ROW EXECUTE PROCEDURE person_bit();

-------------------------------------------------------------------
```

Statistics

CREATE TRIGGER person_biut BEFORE INSER | Enter a SQL expression to filter results (use Ctrl+Space)

Name	Value
Updated Rows	0
Query	CREATE TRIGGER person_biut BEFORE INSERT OR UPDATE ON person FOR EACH ROW EXECUTE PROCEDURE person_bit()
Finish time	Mon Dec 20 23:49:48 IRST 2021

حال ممکن است علاوه بر insert و update یک سطر، نیاز باشد تا لاگ حذف یک سطر را هم ذخیره کنیم. این بار باید یک تابع و یک trigger جدید تعریف کنیم و این بار از متغیر اشاره گر OLD که اشاره گر به سطر در شرف حذف شدن است، استفاده کنیم.

```
-------------------------------------------------------------------

CREATE OR REPLACE FUNCTION person_bdt()
    RETURNS TRIGGER
    SET SCHEMA 'public'
    LANGUAGE plpgsql
    AS $$
    BEGIN

    -- Record deletion in audit table

    INSERT INTO person_audit (login_name, display_name, operation)
      VALUES (OLD.login_name, OLD.display_name, TG_OP);

    RETURN OLD;
    END;
    $$;

-------------------------------------------------------------------
```

Statistics Log Output

CREATE OR REPLACE FUNCTION person_bdt | Enter a SQL expression to filter results (use Ctrl+Space)

Name	Value
Updated Rows	0
Query	CREATE OR REPLACE FUNCTION person_bdt() RETURNS TRIGGER SET SCHEMA 'public' LANGUAGE plpgsql AS $$ BEGIN -- Record deletion in audit ...
Finish time	Mon Dec 20 23:50:49 IRST 2021

در ادامه یک trigger هم نام با تابع ایجاد شده در عکس قبل یعنی person_bdt ایجاد می کنیم. و این trigger بدین صورت کار خواهد کرد که قبل از هر delete روی جدول person فعال می شود و به ازای هر سطر آن تابع person_bdt() را اجرا می کند.

```
-----------------------------------------------------------------

CREATE TRIGGER person_bdt
    BEFORE DELETE ON person
    FOR EACH ROW EXECUTE PROCEDURE person_bdt();

-----------------------------------------------------------------
```

Statistics

CREATE TRIGGER person_bdt BEFORE DELETE | Enter a SQL expression to filter results (use Ctrl+Space)

Name	Value
Updated Rows	0
Query	CREATE TRIGGER person_bdt BEFORE DELETE ON person FOR EACH ROW EXECUTE PROCEDURE person_bdt()
Finish time	Mon Dec 20 23:51:26 IRST 2021

حال سعی می کنیم که اعمال انجام شده در این مثال را بررسی کنیم.

در ابتدا چند مقدار در جدول person اضافه می کنیم و لاگ آن ها را بررسی می کنیم.

```
-----------------------------------------------------------------

INSERT INTO person VALUES ('dfunny', 'Doug Funny');

-----------------------------------------------------------------
```

Statistics

INSERT INTO person VALUES ('dfunny', 'Doug | Enter a SQL expression to filter results (use Ctrl+Space)

Name	Value
Updated Rows	1
Query	INSERT INTO person VALUES ('dfunny', 'Doug Funny')
Finish time	Mon Dec 20 23:51:55 IRST 2021

```
-----------------------------------------------------------------

INSERT INTO person VALUES ('pmayo', 'Patti Mayonnaise');
```

Statistics

INSERT INTO person VALUES ('pmayo', 'Patti | Enter a SQL expression to filter results (use Ctrl+Space)

Name	Value
Updated Rows	1
Query	INSERT INTO person VALUES ('pmayo', 'Patti Mayonnaise')
Finish time	Mon Dec 20 23:52:08 IRST 2021

```
SELECT * FROM person;
```

person

SELECT * FROM person | Enter a SQL expression to filter results (use Ctrl+Space)

	login_name	display_name
1	dfunny	Doug Funny
2	pmayo	Patti Mayonnaise

```
SELECT * FROM person_audit;
```

person_audit

SELECT * FROM person_audit | Enter a SQL expression to filter results (use Ctrl+Space)

	login_name	display_name	operation	effective_at	userid
1	dfunny	Doug Funny	INSERT	2021-12-20 23:51:55	postgres
2	pmayo	Patti Mayonnaise	INSERT	2021-12-20 23:52:08	postgres

مشاهده می شود که trigger ها توابع را به کار انداخته اند و لاگ آن ها در جدول جدید ثبت شده است.

در ادامه یک مقدار را در جدول person آپدیت می کنیم و لاگ آن را بررسی می کنیم.

```
UPDATE person SET display_name = 'Doug Yancey Funny' WHERE login_name = 'dfunny';
```

Statistics

UPDATE person SET display_name = 'Doug Y | Enter a SQL expression to filter results (use Ctrl+Space)

Name	Value
Updated Rows	1
Query	UPDATE person SET display_name = 'Doug Yancey Funny' WHERE login_name = 'dfunny'
Finish time	Mon Dec 20 23:53:01 IRST 2021

```
SELECT * FROM person;
```

person

SELECT * FROM person | Enter a SQL expression to filter results (use Ctrl+Space)

	login_name	display_name
1	pmayo	Patti Mayonnaise
2	dfunny	Doug Yancey Funny

```
SELECT * FROM person_audit ORDER BY effective_at;
```

person_audit

SELECT * FROM person_audit ORDER BY effe | Enter a SQL expression to filter results (use Ctrl+Space)

	login_name	display_name	operation	effective_at	userid
1	dfunny	Doug Funny	INSERT	2021-12-20 23:51:55	postgres
2	pmayo	Patti Mayonnaise	INSERT	2021-12-20 23:52:08	postgres
3	dfunny	Doug Yancey Funny	UPDATE	2021-12-20 23:53:01	postgres

مشاهده می شود که trigger ها توابع را به کار انداخته اند و لاگ آن ها در جدول جدید ثبت شده است.

در ادامه یک مقدار را از جدول person حذف می کنیم و لاگ آن را بررسی می کنیم.

```
DELETE FROM person WHERE login_name = 'pmayo';
```

Statistics

DELETE FROM person WHERE login_name = | Enter a SQL expression to filter results (use Ctrl+Space)

Name	Value
Updated Rows	1
Query	DELETE FROM person WHERE login_name = 'pmayo'
Finish time	Mon Dec 20 23:53:48 IRST 2021

```
SELECT * FROM person;
```

person

SELECT * FROM person | Enter a SQL expression to filter results (use Ctrl+Space)

	login_name	display_name
1	dfunny	Doug Yancey Funny

```
SELECT * FROM person_audit ORDER BY effective_at;
```

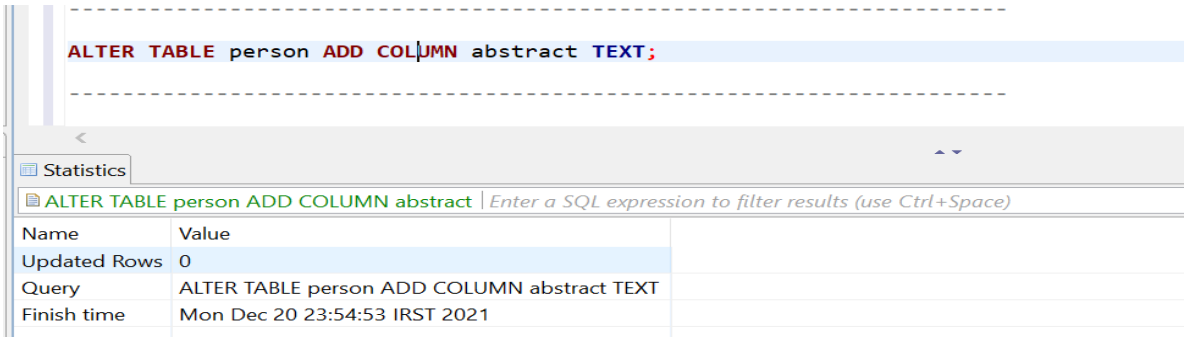
person_audit

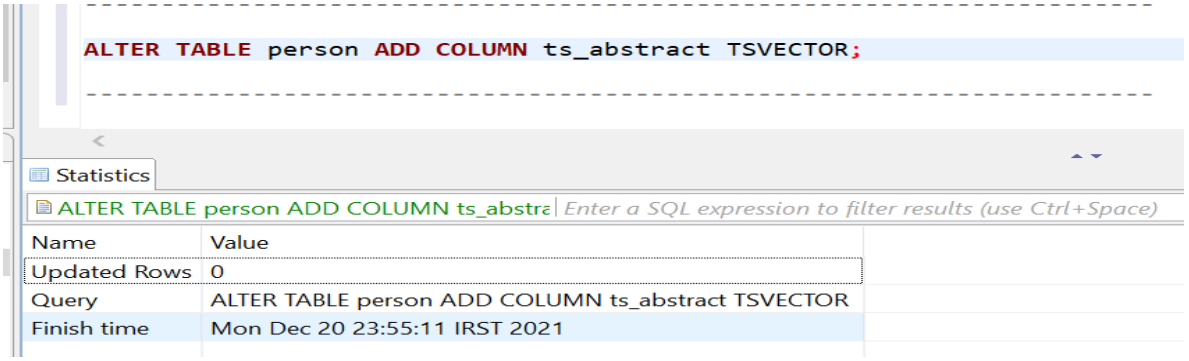
SELECT * FROM person_audit ORDER BY effe | Enter a SQL expression to filter results (use Ctrl+Space)

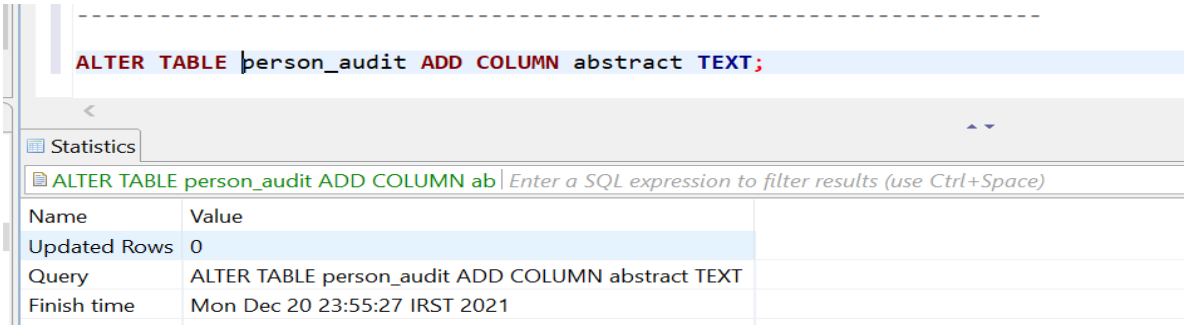
	login_name	display_name	operation	effective_at	userid
1	dfunny	Doug Funny	INSERT	2021-12-20 23:51:55	postgres
2	pmayo	Patti Mayonnaise	INSERT	2021-12-20 23:52:08	postgres
3	dfunny	Doug Yancey Funny	UPDATE	2021-12-20 23:53:01	postgres
4	pmayo	Patti Mayonnaise	DELETE	2021-12-20 23:53:48	postgres

- **مثال 2: مقادیر مشتق شده**

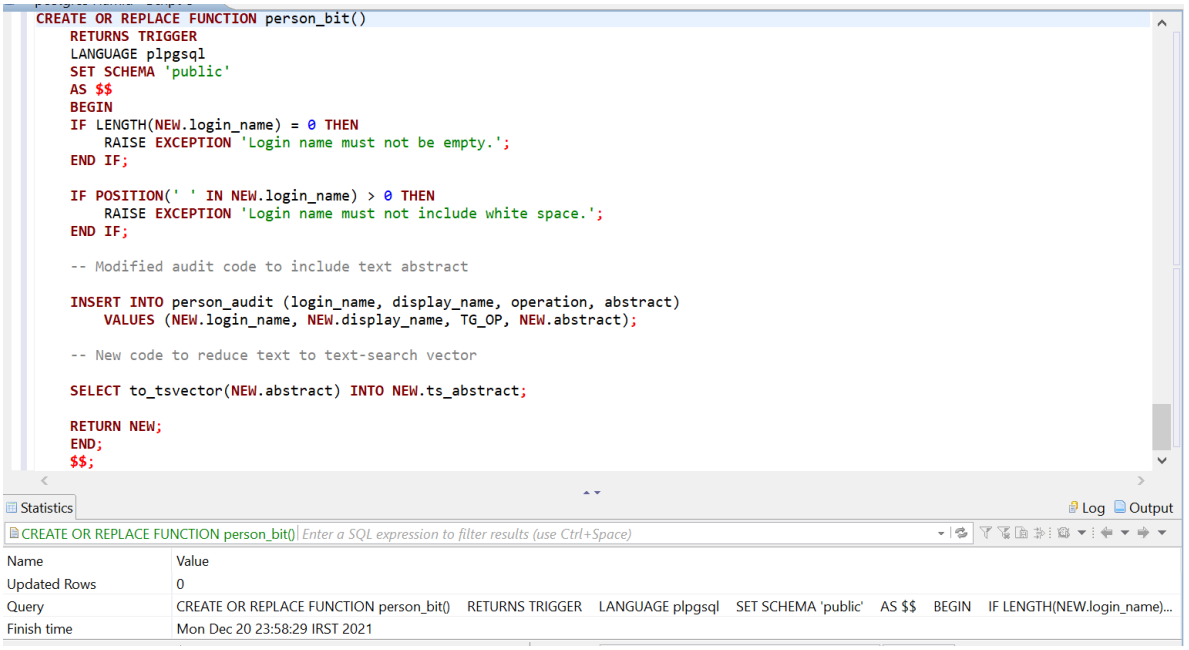
حال می‌خواهیم یک سند متنی آزاد را در هر ردیف ذخیره کنیم، مثلاً یک رزومه با قالب متن ساده یا مقاله کنفرانس یا خلاصه شخصیت داستانی. در ابتدا دو ویژگی را برای پشتیبانی از ذخیره سازی سند و یک بردار جستجوی متن مرتبط به جدول اصلی person اضافه می کنیم. از آنجایی که بردار جستجوی متن بر اساس هر ردیف مشتق شده است، ذخیره آن در جدول لاگ فایده ای ندارد، زیرا ستون ذخیره سند را به جدول لاگ مرتبط کردیم.





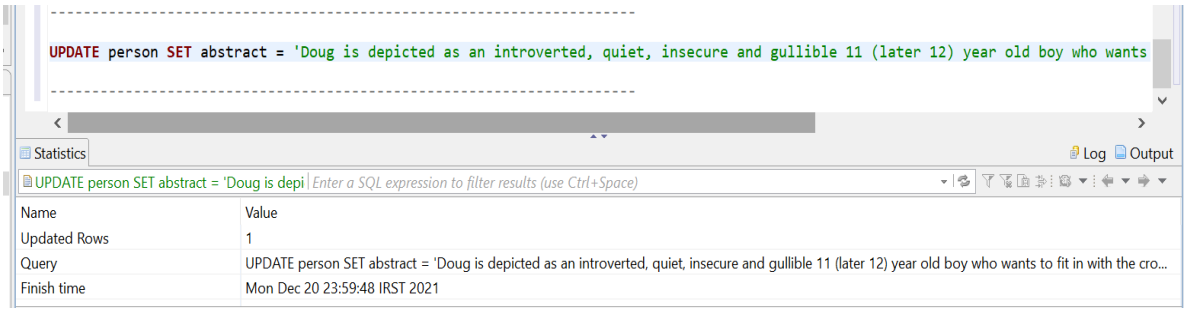


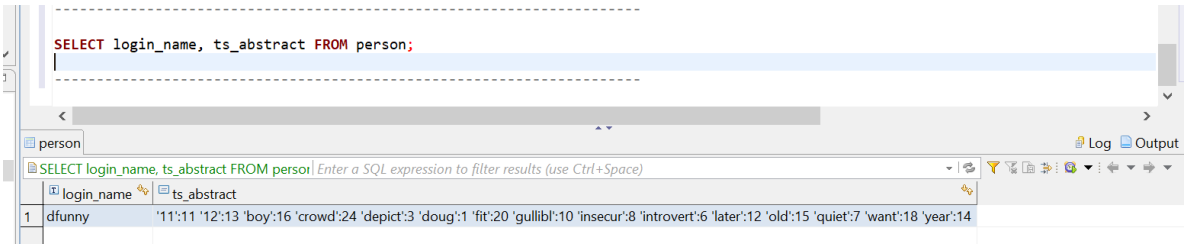
حال تابع را تغییر می دهیم. به تابع person_bit() اضافه می کنیم که فیلد abstract شده که از نوع TEXT هم هست، به وسیله متد to_tsvector به تایپ مورد نظر تبدیل شده و در فیلد ts_abstract آن را ذخیره می کنیم. این کار باعث سهولت سرچ خواهد شد.



حال برای بررسی صحت این تابع و trigger یک آپدیت روی جدول person انجام می دهیم.

که مشاهده می کنیم به درستی کار کرده است.

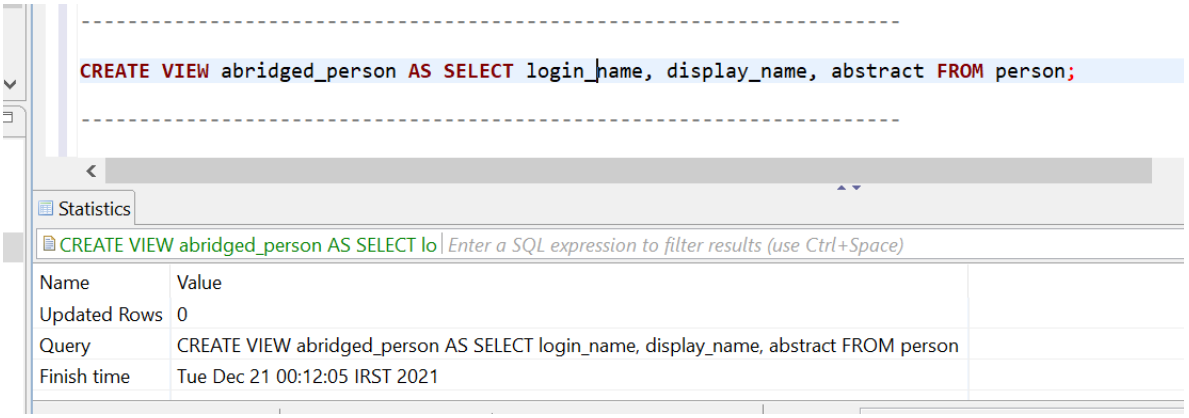




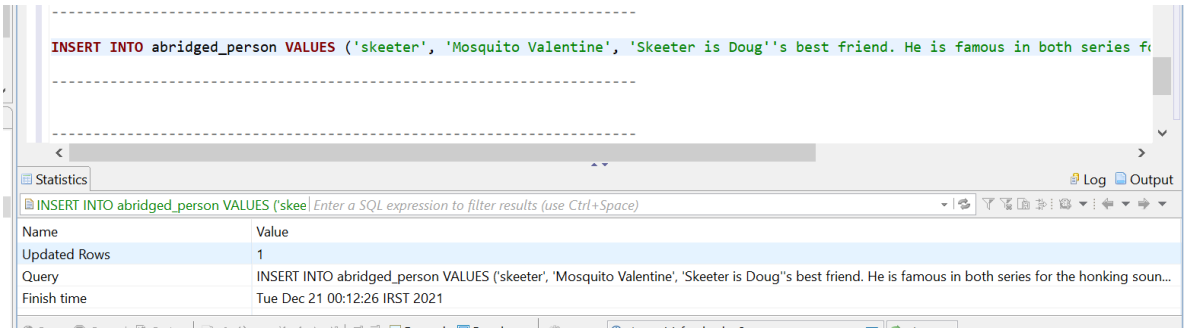
- **مثال 3: trigger ها و view ها**

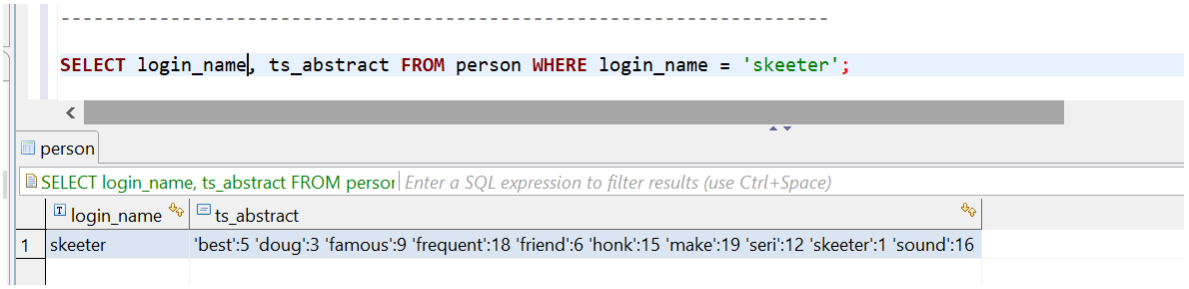
اگر کاربر سعی کند مقداری را برای ستون ts_abstract درج کند، هر چیزی که وارد می‌شود دور ریخته می‌شود و با مقداری که از داخل تابع trigger مشتق شده است جایگزین می‌شود. پس بردار جستجوی متن مشتق شده در مثال بالا برای استفاده کاربران در نظر گرفته نشده است، یعنی توسط

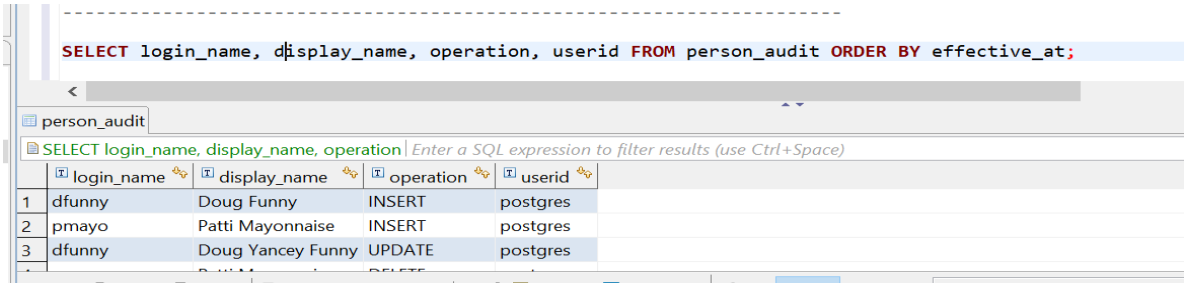
کاربر وارد نشده است، و ما هرگز انتظار نداریم که مقدار را به کاربر هدف ارائه دهیم. حال یک view ایجاد می کنیم و این ویژگی را در آن قرار نمی دهیم.



مشاهده می کنیم که وقتی به این view یک مقدار insert می کنیم، باز این trigger به درستی کار می کند.

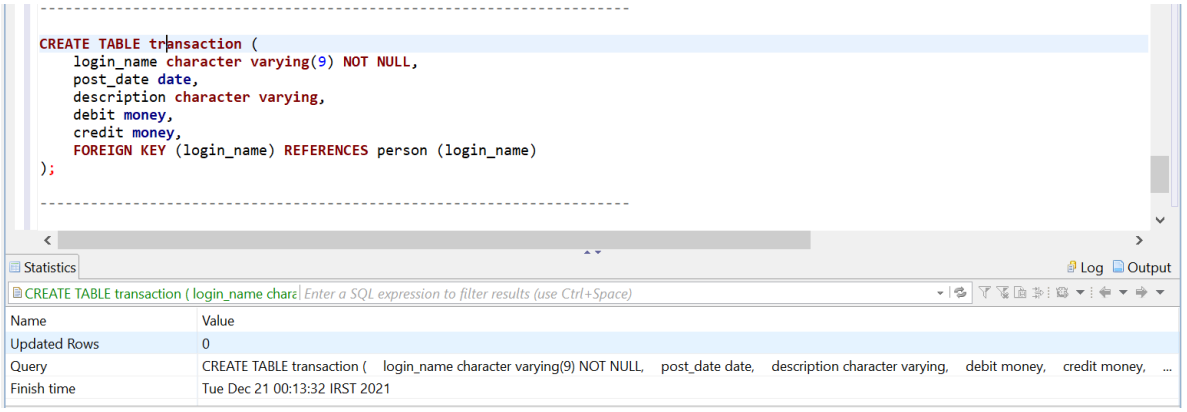




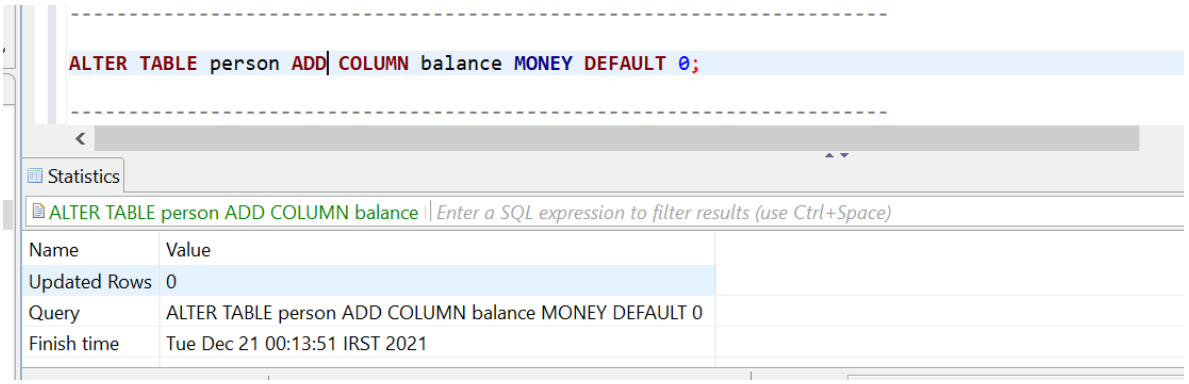


- **مثال 4: مقادیر خلاصه**

حال در این مثال جدولی برای ذخیره کردن تراکنش ها ایجاد می کنیم.



برای اینکه هر بار که به مانده خالص یک فرد نیاز داریم، لازم نباشد آن را با جمع و تفریق همه تراکنش هایش حساب کنیم، بهتر است که یک فیلد بدین منظور ایجاد کنیم.

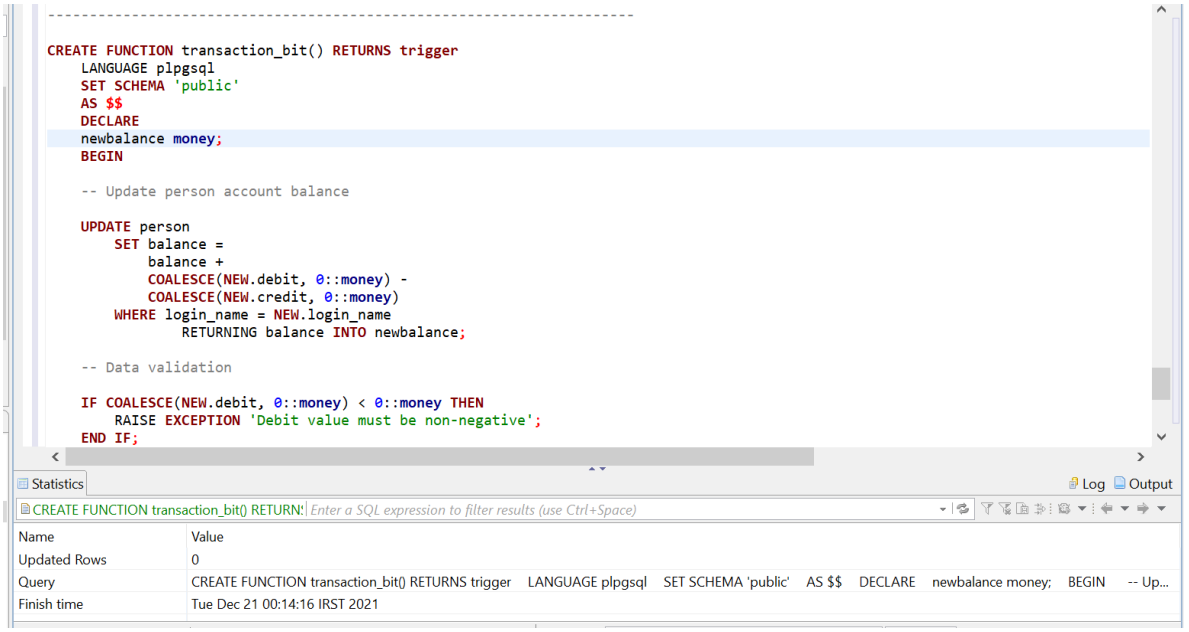


و سپس یک تابع و تریگر تعریف می کنیم تا به محض اینکه تراکنشی رخ داد، این فیلد آپدیت شود.

ما مقدار اولیه اعتبار هر شخص را برابر صفر قرار داده ایم. بعد از هر تراکنش، این مقدار با debit تراکنش آن شخص جمع شده و credit تراکنش آن شخص از اعتبارش کاسته می شود.

بررسی می کنیم که پس از آپدیت شدن مبلغ اعتبار شخص، مقدار آن منفی نشود.

فایده استفاده از UPDATE این است که به‌روزرسانی، سطر هدف را در مدت تراکنش قفل می‌کند و بنابراین هر تراکنش دیگری که سعی در به‌روزرسانی همان ردیف داشته باشد تا زمانی که تراکنش فعلی کامل شود مسدود می‌شود.

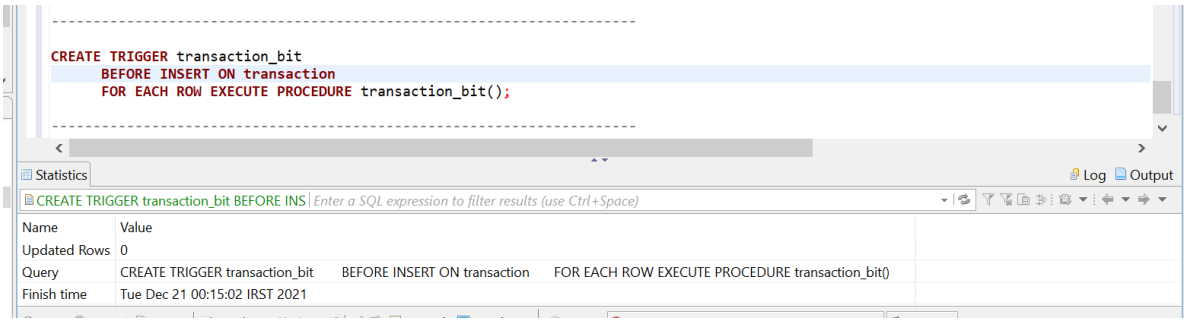


```
CREATE FUNCTION transaction_bit() RETURNS trigger
    LANGUAGE plpgsql
    SET SCHEMA 'public'
    AS $$
    DECLARE
    newbalance money;
    BEGIN

    -- Update person account balance

    UPDATE person
        SET balance =
            balance +
            COALESCE(NEW.debit, 0::money) -
            COALESCE(NEW.credit, 0::money)
        WHERE login_name = NEW.login_name
                RETURNING balance INTO newbalance;

    -- Data validation

    IF COALESCE(NEW.debit, 0::money) < 0::money THEN
        RAISE EXCEPTION 'Debit value must be non-negative';
    END IF;
```

Statistics | Log | Output

CREATE FUNCTION transaction_bit() RETURN! | Enter a SQL expression to filter results (use Ctrl+Space)

Name	Value
Updated Rows	0
Query	CREATE FUNCTION transaction_bit() RETURNS trigger LANGUAGE plpgsql SET SCHEMA 'public' AS $$ DECLARE newbalance money; BEGIN -- Up...
Finish time	Tue Dec 21 00:14:16 IRST 2021



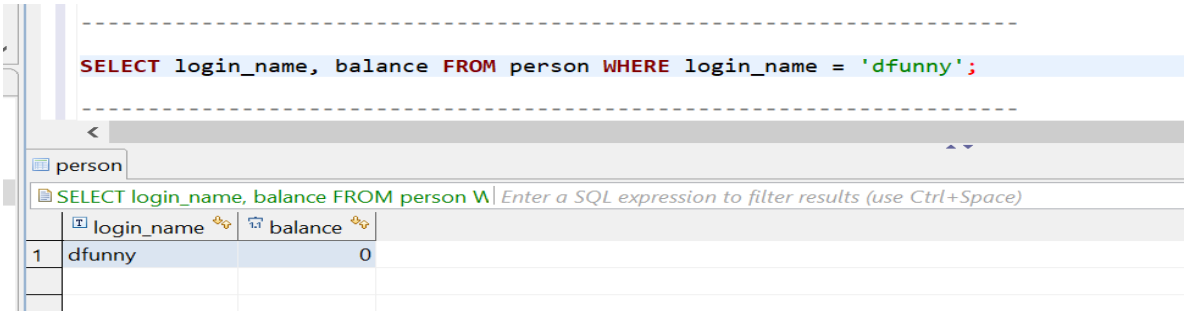
```
CREATE TRIGGER transaction_bit
      BEFORE INSERT ON transaction
      FOR EACH ROW EXECUTE PROCEDURE transaction_bit();
```

Statistics | Log | Output

CREATE TRIGGER transaction_bit BEFORE INS | Enter a SQL expression to filter results (use Ctrl+Space)

Name	Value
Updated Rows	0
Query	CREATE TRIGGER transaction_bit BEFORE INSERT ON transaction FOR EACH ROW EXECUTE PROCEDURE transaction_bit()
Finish time	Tue Dec 21 00:15:02 IRST 2021

حال سعی می کنیم چند تراکنش انجام دهیم و صحت عملکرد تابع و تریگر خود را بررسی کنیم.

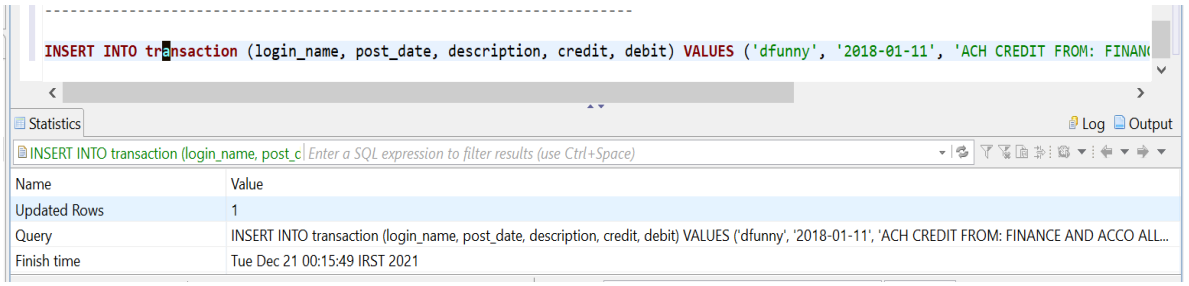


```
SELECT login_name, balance FROM person WHERE login_name = 'dfunny';
```

person

SELECT login_name, balance FROM person W | Enter a SQL expression to filter results (use Ctrl+Space)

	login_name	balance
1	dfunny	0

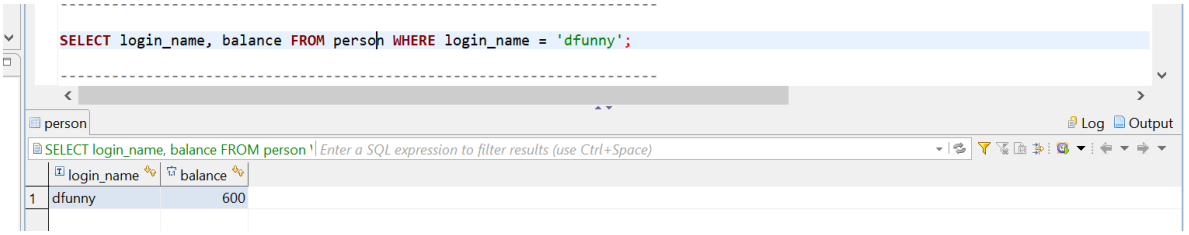


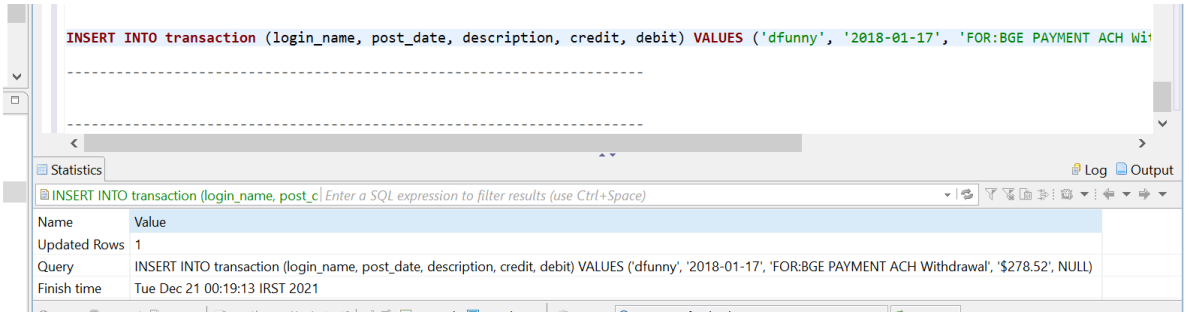
```
INSERT INTO transaction (login_name, post_date, description, credit, debit) VALUES ('dfunny', '2018-01-11', 'ACH CREDIT FROM: FINANC
```

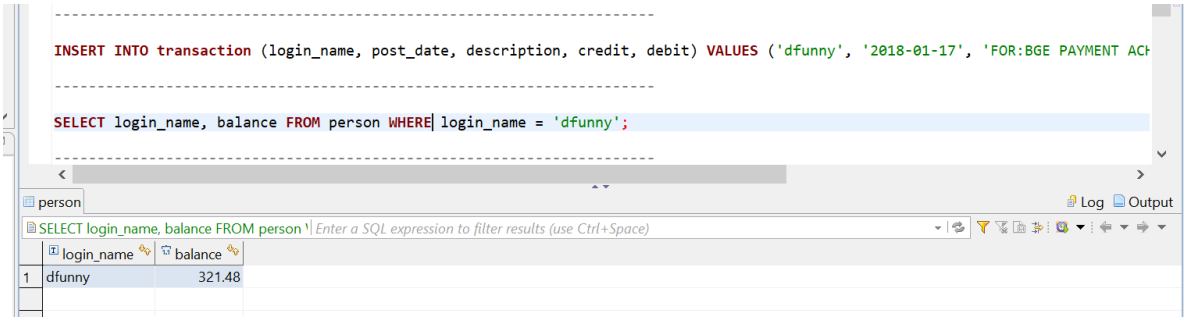
Statistics | Log | Output

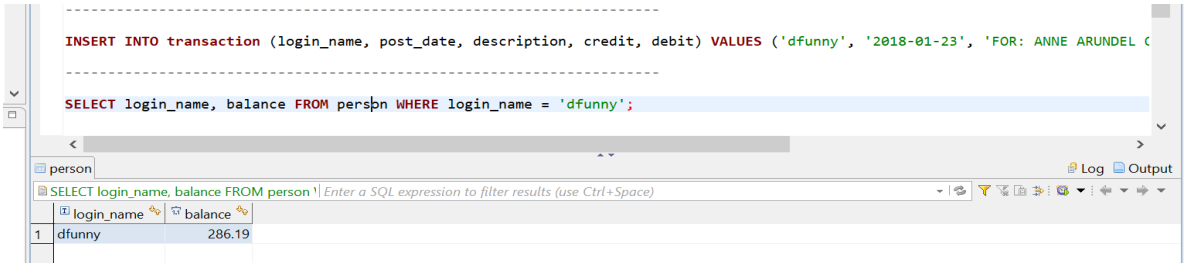
INSERT INTO transaction (login_name, post_c | Enter a SQL expression to filter results (use Ctrl+Space)

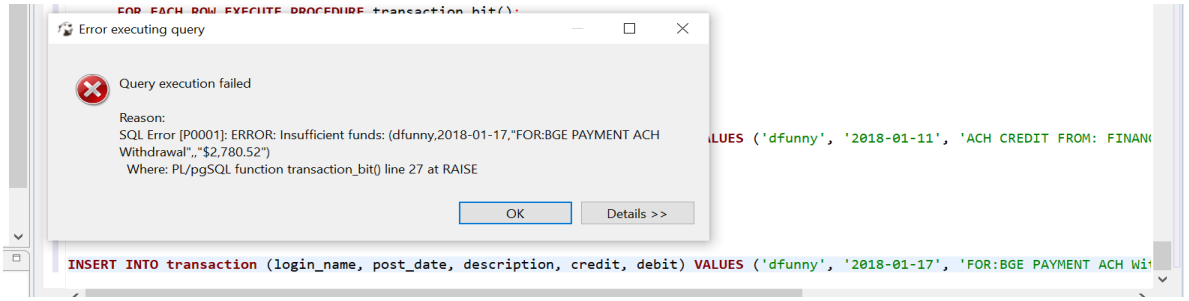
Name	Value
Updated Rows	1
Query	INSERT INTO transaction (login_name, post_date, description, credit, debit) VALUES ('dfunny', '2018-01-11', 'ACH CREDIT FROM: FINANCE AND ACCO ALL...
Finish time	Tue Dec 21 00:15:49 IRST 2021









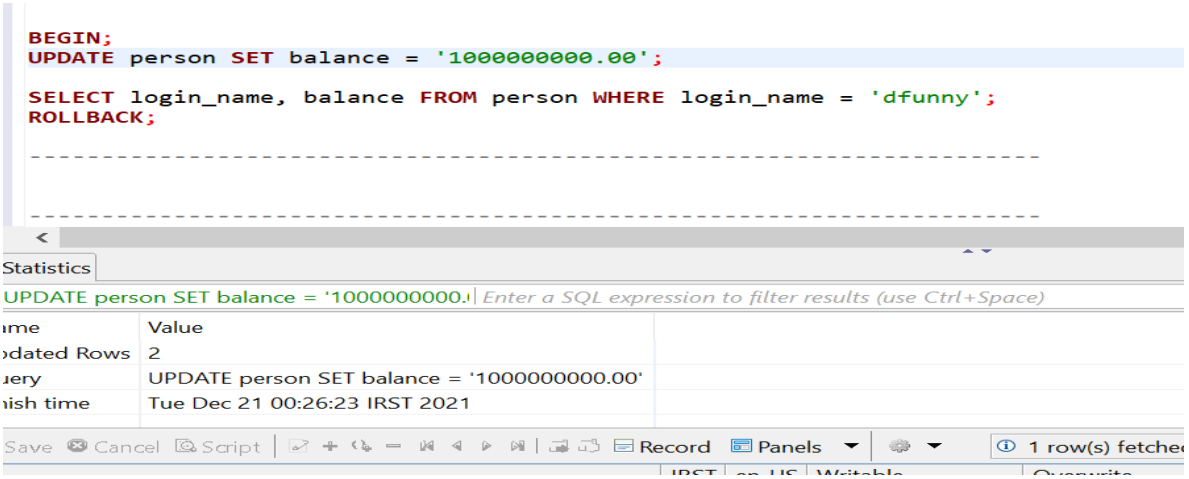


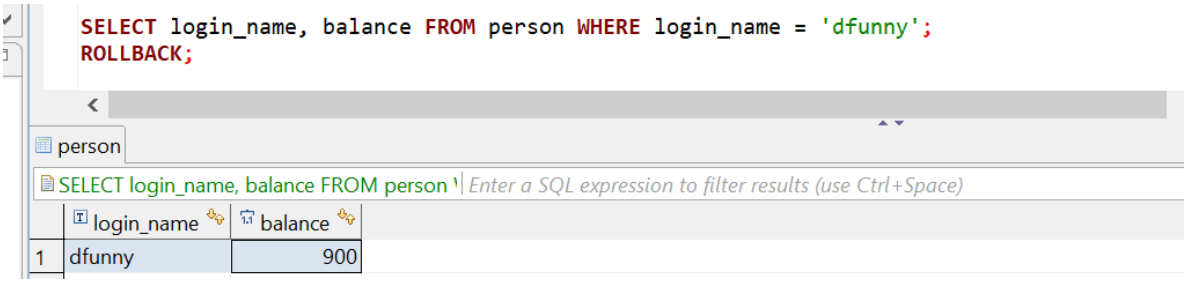
مشاهده می کنیم که در ابتدا مقدار اعتبار را با تراکنشی افزایش داده و سپس با چند تراکنش مقدار آن را کاهش داده ایم تا جایی که مقدار آن منفی شده و اکسپش دریافت کرده ایم.

- **مثال 5: تریگر ها و view redux ها**

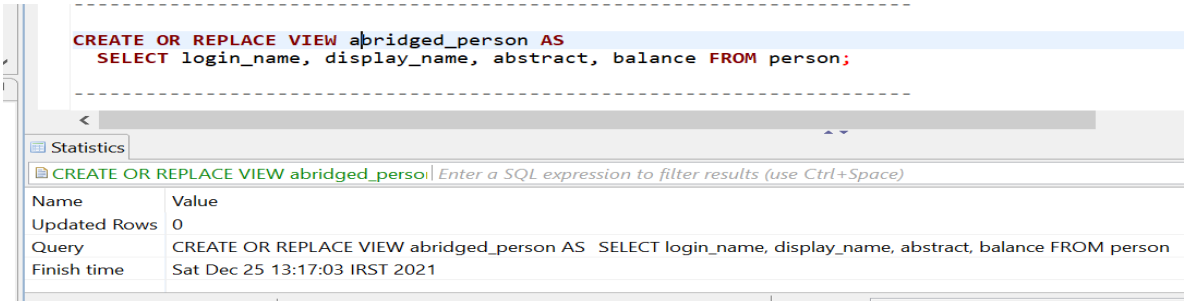
حال اگر در مثال قبل کاربر مخرب داشته باشیم، می تواند مقدار اعتبار را به صورت دستی تغییر دهد.

تصویر زیر گویای این عملکرد مخرب است. من برای مثال سعی کردم دو مقدار مختلف در این فیلد قرار دهم که در عکس دوم یکی از این دو مقدار را مشاهده می کنید.





ابتدا یک view ایجاد می کنیم تا فیلد balance را برای جدول person نشان دهد.



مشاهده می کنیم که مشکل هنوز برقرار است و می توان مقدار این فیلد را دستی تغییر داد.

```
BEGIN;
UPDATE abridged_person SET balance = '800.00';

SELECT login_name, balance FROM abridged_person WHERE login_name = 'dfunny';
ROLLBACK;
```

abridged_person

SELECT login_name, balance FROM abridge | Enter a SQL expression to filter results (use Ctrl+Space)

	login_name	balance
1	dfunny	800

حال برای حل این مشکل یک تابع و trigger تعریف می کنیم. و اگر تغییری روی این فیلد اعمال شد، مقدار قبل از تغییر را برای آن فیلد قرار می دهد.

```
CREATE FUNCTION abridged_person_iut() RETURNS TRIGGER
    LANGUAGE plpgsql
    SET search_path TO public
    AS $$
    BEGIN

    -- Disallow non-transactional changes to balance

      NEW.balance = OLD.balance;
    RETURN NEW;
    END;
    $$;
```

Statistics | Log | Output

CREATE FUNCTION abridged_person_iut() R | Enter a SQL expression to filter results (use Ctrl+Space)

Name	Value
Updated Rows	0
Query	CREATE FUNCTION abridged_person_iut() RETURNS TRIGGER LANGUAGE plpgsql SET search_path TO public AS $$ BEGIN -- Disallow non-transa...
Finish time	Sat Dec 25 15:19:41 IRST 2021

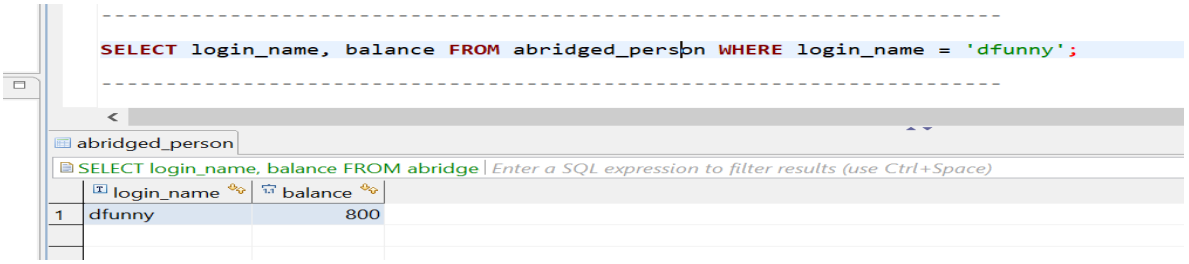
```
CREATE TRIGGER abridged_person_iut
    INSTEAD OF UPDATE ON abridged_person
    FOR EACH ROW EXECUTE PROCEDURE abridged_person_iut();
```

Statistics

CREATE TRIGGER abridged_person_iut INST | Enter a SQL expression to filter results (use Ctrl+Space)

Name	Value
Updated Rows	0
Query	CREATE TRIGGER abridged_person_iut INSTEAD OF UPDATE ON abridged_person FOR EACH ROW EXECUTE PROCEDURE abridged_person_iut()
Finish time	Sat Dec 25 15:22:46 IRST 2021

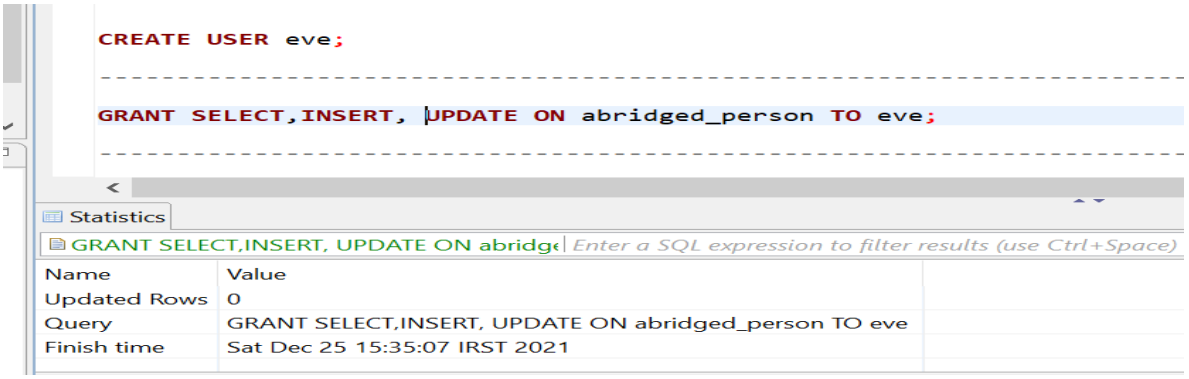
حال این تابع و trigger را تست می کنیم. مشاهده می کنیم که با اینکه مقدار balance را آپدیت کرده ایم ولی این تغییر در جدول اعمال نشده است و مقدار قبلی اش برایش ثبت شده است.

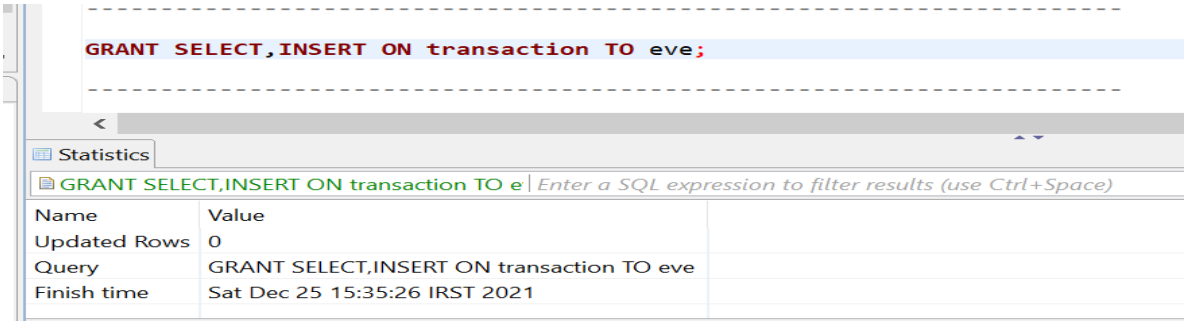


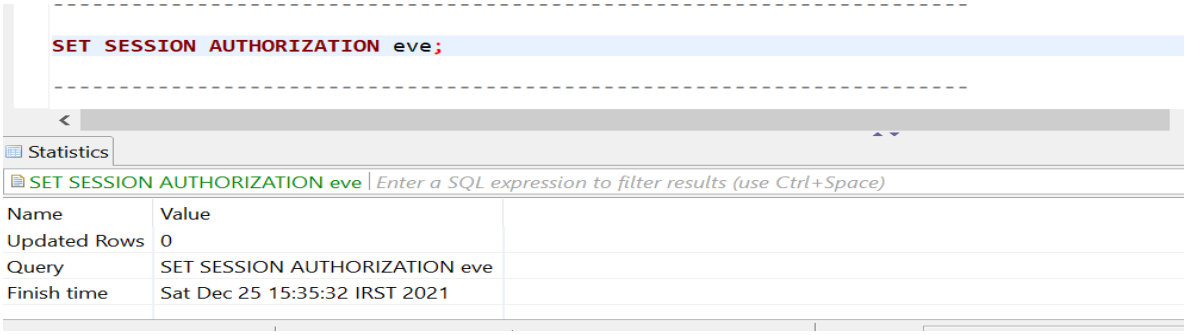
- **مثال 6: فواید دیگر**

تاکنون تمام مثال ها در سطح مالک پایگاه داده و با دسترسی کامل بودند. حال می خواهیم یک کاربر سطح پایین تر تعریف کنیم که برای یک سری از اعمال نیاز به سطح دسترسی بالاتری داشته باشد.

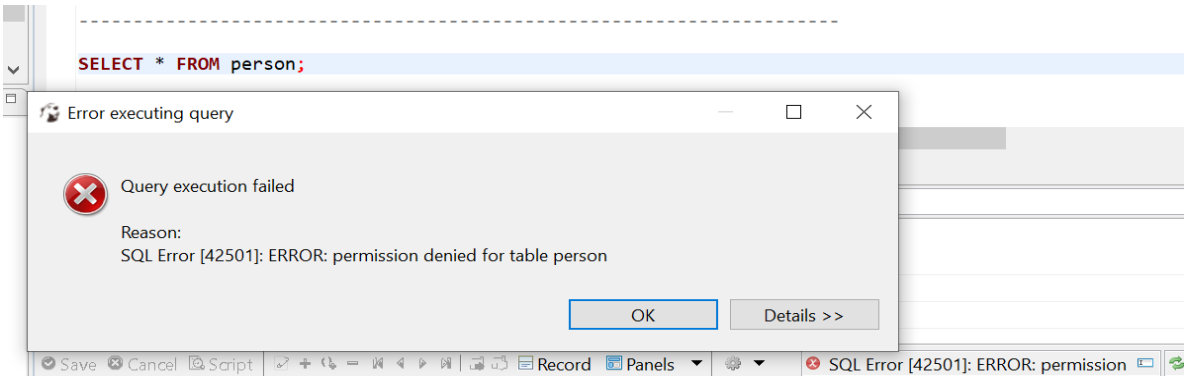
یک user به نام eve می سازیم و به او سطح دسترسی های مختلفی روی جدول های مختلف می دهیم.

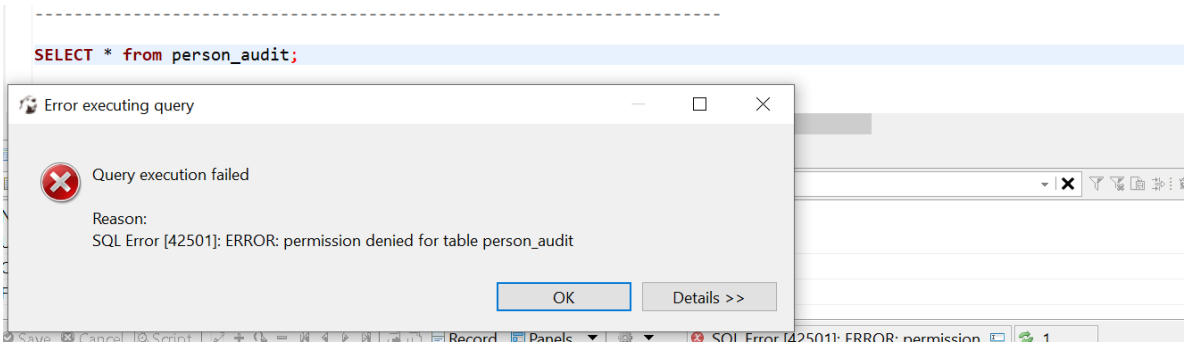




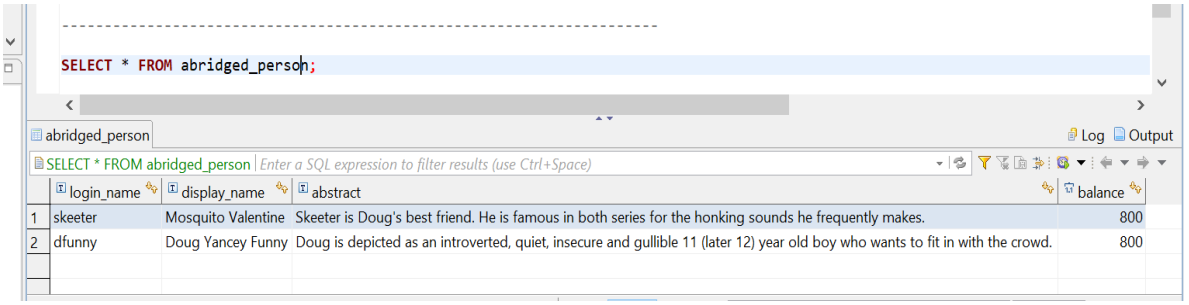


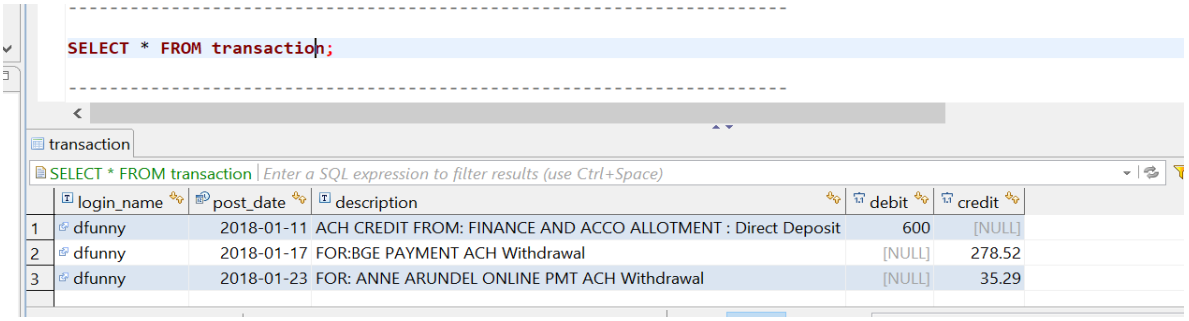
مشاهده می کنیم که وقتی دسترسی لازم برای انجام عمل دلخواه روی جدول برای کاربر موجود نباشد، با خطا رو به رو می شویم.



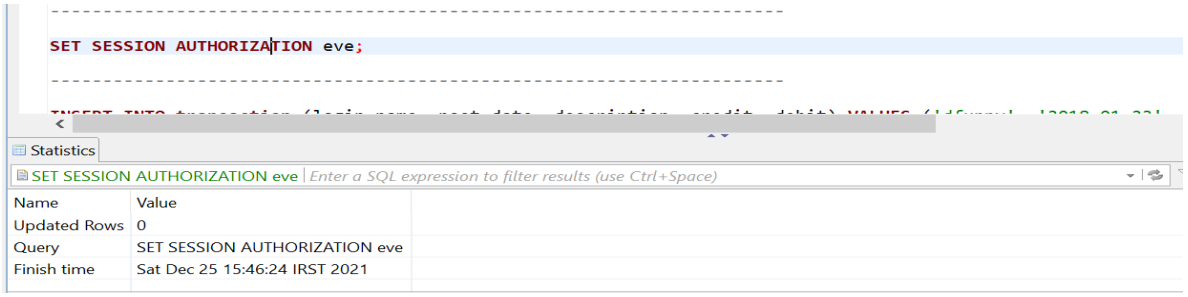


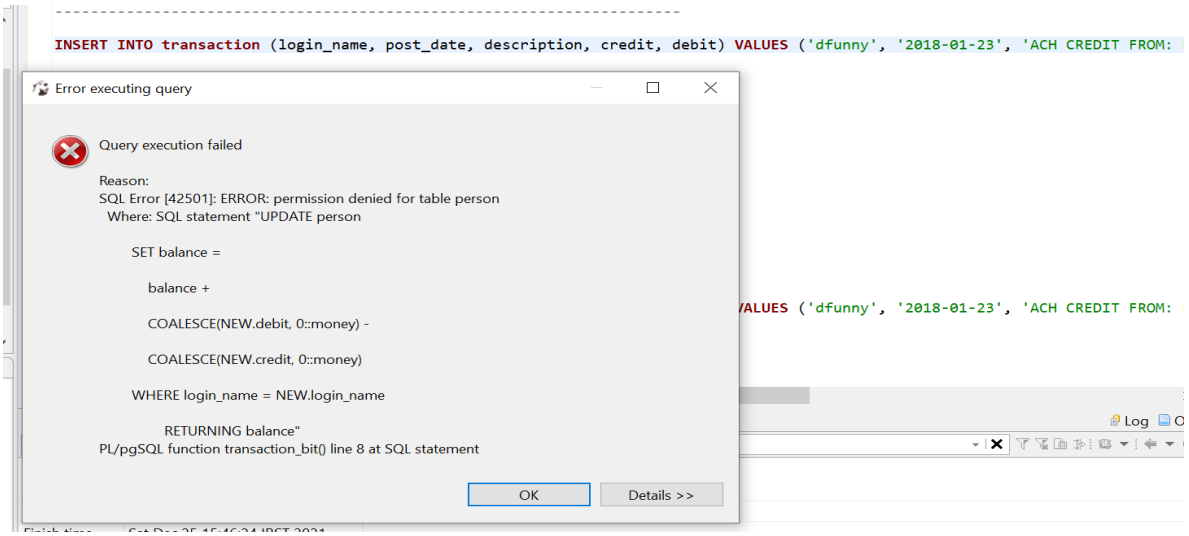
و اگر دسترسی لازم را روی جدول مورد نظر داشته باشیم، با موفقیت اعمال خود را انجام می دهیم.





مشاهده می کنیم که هر چند کاربر دسترسی به جدول تراکنش ها دارد ولی نمی تواند چیزی به آن اضافه کند؛ چون نیاز به آپدیت کردن فیلد balance در جدول person خواهد بود و به آن دسترسی لازم را ندارد.





حال سعی می کنیم این مسئله را حل کنیم. از security definer استفاده می کنیم. و با اضافه کردن این امکان به تابع باعث می شود که دستور با سطح دسترسی کسی که آن تابع را تعریف کرده اجرا شود. در ادامه مشاهده می کنیم که مشکل برطرف شده است.

```
RESET SESSION AUTHORIZATION;
```

Statistics

RESET SESSION AUTHORIZATION | Enter a SQL expression to filter results (use Ctrl+Space)

Name	Value
Updated Rows	0
Query	RESET SESSION AUTHORIZATION
Finish time	Sat Dec 25 15:57:38 IRST 2021

```
ALTER FUNCTION transaction_bit() SECURITY DEFINER;
```

Statistics

ALTER FUNCTION transaction_bit() SECURIT | Enter a SQL expression to filter results (use Ctrl+Space)

Name	Value
Updated Rows	0
Query	ALTER FUNCTION transaction_bit() SECURITY DEFINER
Finish time	Sat Dec 25 15:57:46 IRST 2021

```
SET SESSION AUTHORIZATION eve;
```

Statistics

SET SESSION AUTHORIZATION eve | Enter a SQL expression to filter results (use Ctrl+Space)

Name	Value
Updated Rows	0
Query	SET SESSION AUTHORIZATION eve
Finish time	Sat Dec 25 15:57:52 IRST 2021

```
INSERT INTO transaction (login_name, post_date, description, credit, debit) VALUES ('dfunny', '2018-01-23', 'ACH CREDIT FROM: FI
```

Statistics | Log | Output

INSERT INTO transaction (login_name, post_ | Enter a SQL expression to filter results (use Ctrl+Space)

Name	Value
Updated Rows	1
Query	INSERT INTO transaction (login_name, post_date, description, credit, debit) VALUES ('dfunny', '2018-01-23', 'ACH CREDIT FROM: FINANCE AND ACCO ...
Finish time	Sat Dec 25 15:57:58 IRST 2021

```
SELECT * FROM transaction;
```

transaction

SELECT * FROM transaction | Enter a SQL expression to filter results (use Ctrl+Space)

	login_name	post_date	description	debit	credit
1	dfunny	2018-01-11	ACH CREDIT FROM: FINANCE AND ACCO ALLOTMENT : Direct Deposit	600	[NULL]
2	dfunny	2018-01-17	FOR:BGE PAYMENT ACH Withdrawal	[NULL]	278.52
3	dfunny	2018-01-23	FOR: ANNE ARUNDEL ONLINE PMT ACH Withdrawal	[NULL]	35.29
4	dfunny	2018-01-23	ACH CREDIT FROM: FINANCE AND ACCO ALLOTMENT : Direct Deposit	[NULL]	[NULL]

```
SELECT login_name, balance FROM abridged_person WHERE login_name = 'dfunny';
```

abridged_person

SELECT login_name, balance FROM abridge | Enter a SQL expression to filter results (use Ctrl+Space)

	login_name	balance
1	dfunny	650

توابع پنجره ای:

1. در مورد اول، خواستم تاریخ استخدام پیرترین کارمند هر منطقه و سایر اطلاعات مورد نیاز از آن کارمند را پیدا کنم تا در قرعه کشی ای شرکت داده بشوند.

*<postgres Hamid> Script-5 *<postgres Hamid> Script-6 employees

```
select first_name, last_name, region, home_phone,
first_value(hire_date) over(partition by region order by birth_date asc) as oldest_employees_in_region
from employees
```

employees

select first_name, last_name, region, home_ | Enter a SQL expression to filter results (use Ctrl+Space)

	first_name	last_name	region	home_phone	oldest_employees_in_region
1	Margaret	Peacock	WA	(206) 555-8122	1993-05-03
2	Nancy	Davolio	WA	(206) 555-9857	1993-05-03
3	Andrew	Fuller	WA	(206) 555-9482	1993-05-03
4	Laura	Callahan	WA	(206) 555-1189	1993-05-03
5	Janet	Leverling	WA	(206) 555-3412	1993-05-03
6	Steven	Buchanan	[NULL]	(71) 555-4848	1993-10-17
7	Robert	King	[NULL]	(71) 555-5598	1993-10-17
8	Michael	Suyama	[NULL]	(71) 555-7773	1993-10-17
9	Anne	Dodsworth	[NULL]	(71) 555-4444	1993-10-17

2. در مورد دوم، خواستم به ازای هر تهیه کننده کالا های او را به ترتیب مقدار حضورشان در سفارشات مختلف (پر فروش بودن) رنک بندی کنم و سایر اطلاعات به درد بخور آن را در کنارش نشان دهم.

```
-----------------------------------------------------------------


select supplier_id, category_id, product_id, product_name, unit_price,
rank() over (partition by supplier_id order by units_on_order desc) as best_selling_product_of_supplier
from products
```

products

select supplier_id, category_id, product_id, | Enter a SQL expression to filter results (use Ctrl+Space)

	supplier_id	category_id	product_id	product_name	unit_price	best_selling_product_of_supplier
1	1	2	3	Aniseed Syrup	10	1
2	1	1	2	Chang	19	2
3	2	2	66	Louisiana Hot Spiced Okra	17	1
4	2	2	65	Louisiana Fiery Hot Pepper Sauce	21.05	2
5	2	2	4	Chef Anton's Cajun Seasoning	22	2
6	2	2	5	Chef Anton's Gumbo Mix	21.35	2
7	3	2	6	Grandma's Boysenberry Spread	25	1
8	3	2	8	Northwoods Cranberry Sauce	40	1
9	3	7	7	Uncle Bob's Organic Dried Pears	30	1
10	4	7	74	Longlife Tofu	10	1
11	4	6	9	Mishi Kobe Niku	97	2
12	4	8	10	Ikura	31	2
13	5	4	11	Queso Cabrales	21	1

Save Cancel Script Record Panels Grid Text 77 row(s) fetched - 0ms

3. در مورد سوم، خواستم که به ازای هر مشتری، سفارشاتش را به ترتیب تاریخ ارسال آن ها رتبه بندی کنم تا در آنالیز مشتری ها بتوانم از این اطلاعات استفاده کنم.

```
select customer_id, order_id, shipped_date,
rank() over (partition by customer_id order by shipped_date asc) as soon
from orders
```

orders

select customer_id, order_id, shipped_date, Enter a SQL expression to filter results (use Ctrl+Space)

	customer_id	order_id	shipped_date	soon
1	ALFKI	10,643	1997-09-02	1
2	ALFKI	10,692	1997-10-13	2
3	ALFKI	10,702	1997-10-21	3
4	ALFKI	10,835	1998-01-21	4
5	ALFKI	10,952	1998-03-24	5
6	ALFKI	11,011	1998-04-13	6
7	ANATR	10,308	1996-09-24	1
8	ANATR	10,625	1997-08-14	2
9	ANATR	10,759	1997-12-12	3
10	ANATR	10,926	1998-03-11	4
11	ANTON	10,365	1996-12-02	1
12	ANTON	10,507	1997-04-22	2
13	ANTON	10,535	1997-05-21	3

- **پرسش امتیازی اول:**

به صورت موقت جدول purchasingUsers را بدین صورت ایجاد می کنیم که از جدول clickstream که رکورد های آن را بر اساس شناسه کاربران گروه بندی کرده ایم، کاربرانی را انتخاب می کنیم که حداقل یک خرید داشته باشند. برای محاسبه تعداد خرید های هر کاربر از sum تعداد خرید های هر کاربر استفاده کرده ایم. در اینجا فقط کلیک هایی که دارای تایپ "buy" بوده اند را در شمارش و جمع زدن استفاده کرده ایم و بدین منظور از case استفاده کرده ایم.

و به صورت موقت جدول movingUsers را بدین صورت ایجاد می کنیم که از جدول geolocation که رکورد های آن را بر اساس شناسه کاربران گروه بندی کرده ایم، کاربرانی را انتخاب می کنیم که بیش از یک zipCode یا کدپستی یکتا داشته باشند.

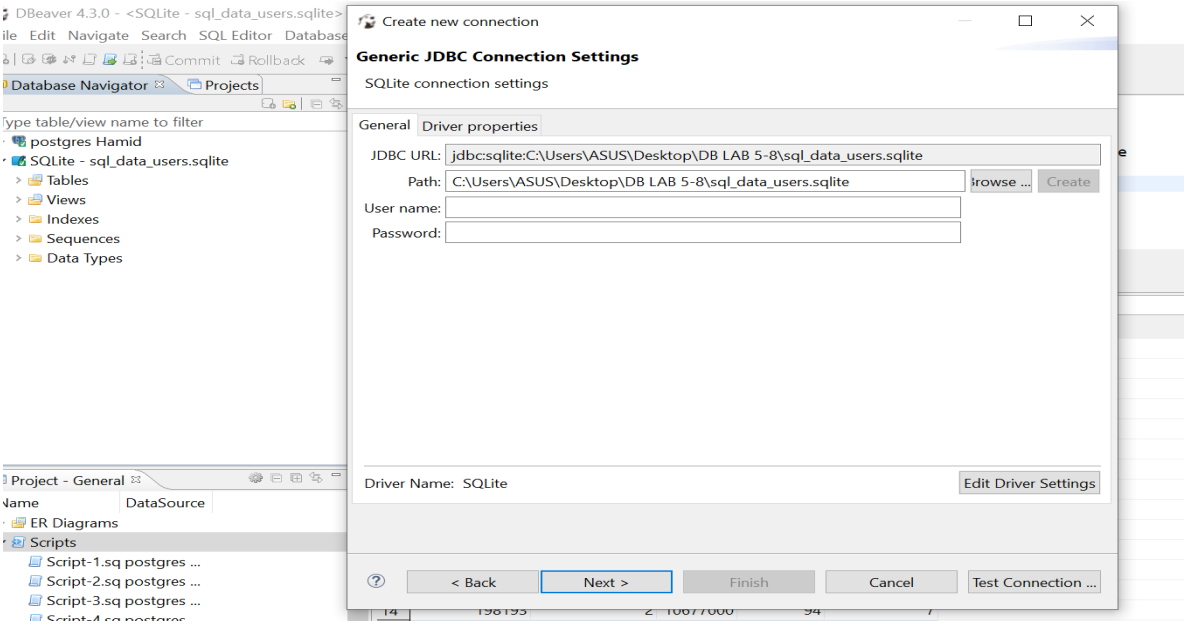
و به صورت موقت جدول userSessionMetrics را بدین صورت ایجاد می کنیم که از جدول clickstream که رکورد های آن را بر اساس شناسه کاربران و سشن ها گروه بندی کرده ایم، به ازای کاربران و سشن ها، تعداد کل کلیک های کاربر، تعداد کل لاگین های کاربر و تعداد کل خرید های کاربر را محاسبه می کند. برای محاسبه تعداد کل اکشن ها با تایپ های گوناگون هر کاربر از sum استفاده کرده ایم. در اینجا بدین منظور از case استفاده کرده ایم که اگر تایپ اکشن مورد نظر ما با تایپ مورد نظر ما همخوانی داشت، یکی به تعداد کل آن اکشن ها افزوده شود و در غیر این صورت اتفاقی نیفتد.

در نهایت این سه جدول را با هم دیگر جوین می کنیم که خروجی آن کاربران به همراه سشن های متناظرشان، که کاربر حداقل یک خرید و بیش از یک zipCode یا کدپستی یکتا داشته است و برای آن کاربر، تعداد کل کلیک های کاربر، تعداد کل لاگین های کاربر و تعداد کل خرید را هم نشان می دهیم.

- **پرسش امتیازی دوم:**

در ابتدا یک کانشکن جدید SQLite ایجاد کردم و فایل داده شده را خواندم و جدول مربوطه را ایجاد کردم. که در صفحه بعد نحوه ایجاد این کانکشن و در سمت چپ تصویر، پس از ایجاد آن را نشان داده ام. سپس یک کوئری بدین صورت نوشتم که با تابع پنجره ای یک جدول موقتی مانند پرسش قبل ایجاد کردم که رکورد ها را بر اساس شناسه کتگوری آن ها گروه بندی کرده و سپس دو ستون رنکینگ قیمت صعودی و قیمت نزولی برای این رکورد ها ایجاد کردم. سپس یک کوئری دیگر زدم و از این جدول موقتی ایجاد شده استفاده کردم وبه ازای هر کتگوری، ده رکورد ارزان به صورت نزولی و ده

رکورد گران به صورت صعودی را نشان دادم. یک بار دیگر با متد ROW_NUMBER() به جای RANK() این کوئری را پیاده سازی کردم، که نتیجه یکسانی کسب کردم.




```
WITH ranking_price AS (
    SELECT category_id, product_id, price,
    RANK() over(PARTITION BY category_id ORDER BY price ASC) AS cheap,
    RANK() over(PARTITION BY category_id ORDER BY price DESC) AS expensive
    FROM products
)
SELECT product_id, category_id, price, cheap, expensive
FROM ranking_price
WHERE ranking_price.cheap <= 10 OR ranking_price.expensive <= 10;
```

products

WITH ranking_price AS (SELECT category_i | Enter a SQL expression to filter results (use Ctrl+Space)

	product_id	category_id	price	cheap	expensive
1	112951	2	3115000	1	100
2	174928	2	3257000	2	99
3	161592	2	3361000	3	98
4	178099	2	3433000	4	97
5	174720	2	3757000	5	96
6	128897	2	3814000	6	95
7	102570	2	3937000	7	94
8	171412	2	3992000	8	93
9	149053	2	4007000	9	92
10	101280	2	4018000	10	91
11	106295	2	10589000	91	10
12	108275	2	10613000	92	9
13	158614	2	10668000	93	8
14	198193	2	10677000	94	7
15	180677	2	11009000	95	6
16	175427	2	11414000	96	5

Save Cancel Script Record Panels Grid Text

*<SQLite - sql_data_users.sqlite> Script-7 products

```
WITH ranking_price AS (
    SELECT category_id, product_id, price,
    ROW_NUMBER() over(PARTITION BY category_id ORDER BY price ASC) AS cheap,
    ROW_NUMBER() over(PARTITION BY category_id ORDER BY price DESC) AS expensive
    FROM products
)
SELECT product_id, category_id, price, cheap, expensive
FROM ranking_price
WHERE ranking_price.cheap <= 10 OR ranking_price.expensive <= 10;
```

products

WITH ranking_price AS (SELECT category_i | Enter a SQL expression to filter results (use Ctrl+Space)

	product_id	category_id	price	cheap	expensive
1	112951	2	3115000	1	100
2	174928	2	3257000	2	99
3	161592	2	3361000	3	98
4	178099	2	3433000	4	97
5	174720	2	3757000	5	96
6	128897	2	3814000	6	95
7	102570	2	3937000	7	94
8	171412	2	3992000	8	93
9	149053	2	4007000	9	92
10	101280	2	4018000	10	91
11	106295	2	10589000	91	10
12	108275	2	10613000	92	9
13	158614	2	10668000	93	8
14	198193	2	10677000	94	7
15	180677	2	11009000	95	6
16	175427	2	11414000	96	5

Save Cancel Script Record Panels Grid Text 200 row(s) fet