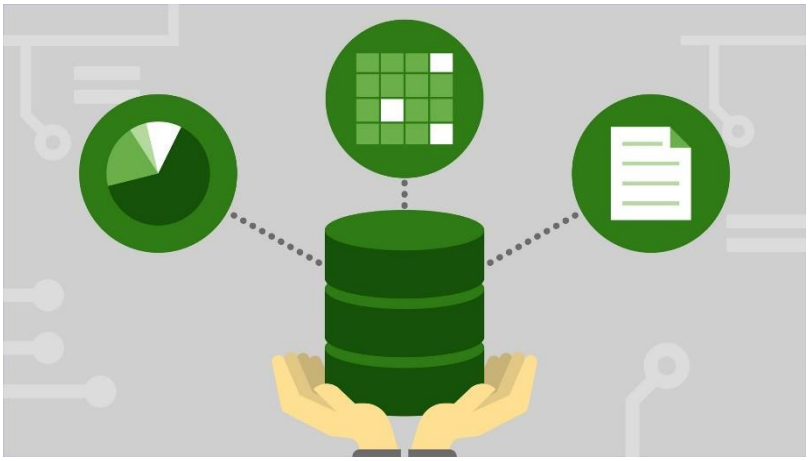
به نام خدا

دانشگاه تهران
پردیس دانشکده‌های فنی
دانشکده برق و کامپیوتر



آزمایشگاه پایگاه داده

دستور کار شماره ۱

سجاد علی‌زاده
۸۱۰۱۹۷۵۴۷

آذرماه ۱۴۰۰

گزارش دستورکار انجام شده

بخش اول)

قسمت اول)

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

Updated Rows	0
Query	CREATE TABLE person (login_name varchar(9) not null primary key, display_name text)
Finish time	Thu Dec 16 20:11:03 IRST 2021

با استفاده از این دستور یک جدول با نام person ساخته میشود.

قسمت دوم)

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

SQL Error [23502]: ERROR: null value in column "login_name" of relation "person" violates not-null constraint
Detail: Failing row contains (null, Felonious Erroneous).

Details >>

این دستور میخواهد یک ردیف به جدول person اضافه کند اما چون مقدار login_name آن نال است و هنگام تعریف جدول شرط not null روی آن گذاشتیم به این ارور برخوردیم.

قسمت سوم)

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

SQL Error [22001]: ERROR: value too long for type character varying(9)

Details >>

این دستور میخواهد یک ردیف به جدول person اضافه کند اما چون مقدار login_name آن از ۹ کاراکتر بیشتر است و هنگام تعریف جدول محدودیت ۹ کاراکتر روی آن گذاشتیم به این ارور برخوردیم.

قسمت چهارم)

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

Name	Value
Updated Rows	0
Query	ALTER TABLE person ADD CONSTRAINT PERSON_LOGIN_NAME_NON_NULL CHECK (LENGTH(login_name) > 0)
Finish time	Thu Dec 16 20:41:37 IRST 2021

در اینجا محدودیتی روی login_name تعریف کردیم. بر اساس این محدودیت طول این رشته حتما باید بیشتر از ۰ باشد.

قسمت پنجم)

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

Name	Value
Updated Rows	0
Query	ALTER TABLE person ADD CONSTRAINT person_login_name_no_space CHECK (POSITION(' ' IN login_name) = 0)
Finish time	Thu Dec 16 20:43:32 IRST 2021

در اینجا محدودیتی روی login_name تعریف کردیم. بر اساس این محدودیت نباید در رشته login_name هیچگونه کاراکتر فاصله (اسپیس) وجود داشته باشد.

قسمت ششم)

```
ALTER TABLE PERSON DROP CONSTRAINT person_login_name_no_space;
ALTER TABLE PERSON DROP CONSTRAINT person_login_name_non_null;
```

Name	Value
Updated Rows	0
Query	ALTER TABLE PERSON DROP CONSTRAINT person_login_name_no_space
Finish time	Thu Dec 16 20:53:04 IRST 2021

Name	Value
Updated Rows	0
Query	ALTER TABLE PERSON DROP CONSTRAINT person_login_name_non_null
Finish time	Thu Dec 16 20:53:44 IRST 2021

با استفاده از این دو دستور محدودیت‌هایی که در قسمت‌های قبل اعمال کردیم را drop می‌کنیم.

قسمت هفتم)

```
CREATE OR REPLACE FUNCTION person_bit()
    RETURNS TRIGGER
    SET SCHEMA 'public'
    LANGUAGE plpgsql
    SET search_path = public
    AS '
    BEGIN
    END;
    ';
```

Name	Value
Updated Rows	0

Query	CREATE OR REPLACE FUNCTION person_bit()
	RETURNS TRIGGER
	SET SCHEMA 'public'
	LANGUAGE plpgsql
	SET search_path = public
	AS '
	BEGIN
	END;
	'

Finish time	Thu Dec 16 20:58:41 IRST 2021

در این قسمت صرفا یک تابع خالی ساخته شده تا در قسمت های بعدی استفاده شود.

قسمت هشتم)

```
CREATE TRIGGER person_bit
    BEFORE INSERT ON person
    FOR EACH ROW EXECUTE PROCEDURE person_bit();
```

Name	Value
Updated Rows	0
Query	CREATE TRIGGER person_bit
	BEFORE INSERT ON person
	FOR EACH ROW EXECUTE PROCEDURE person_bit()
Finish time	Thu Dec 16 21:05:57 IRST 2021

در این قسمت یک تریگر تعریف شد تا قبل از هر بار اضافه کردن ردیف به جدول person تابعی که در قسمت قبل نوشته شد یعنی تابع person_bit() اجرا شود.

قسمت نهم)

```
CREATE OR REPLACE FUNCTION person_bit()
    RETURNS TRIGGER
    SET SCHEMA 'public'
    LANGUAGE plpgsql
    AS $$
    BEGIN
    IF LENGTH(NEW.login_name) = 0 THEN
        RAISE EXCEPTION 'Login name must not be empty.';
    END IF;

    IF POSITION(' ' IN NEW.login_name) > 0 THEN
        RAISE EXCEPTION 'Login name must not include white space.';
    END IF;
    RETURN NEW;
    END;
    $$;
```

Updated Rows	0
Query	CREATE OR REPLACE FUNCTION person_bit()
	RETURNS TRIGGER
	SET SCHEMA 'public'
	LANGUAGE plpgsql
	AS $$
	BEGIN
	IF LENGTH(NEW.login_name) = 0 THEN
	RAISE EXCEPTION 'Login name must not be empty.';
	END IF;
	IF POSITION(' ' IN NEW.login_name) > 0 THEN
	RAISE EXCEPTION 'Login name must not include white space.';
	END IF;
	RETURN NEW;
	END;
	$$
Finish time	Thu Dec 16 21:25:01 IRST 2021

در این قسمت تابعی تعریف شد که قبل از هر insert صدا زده میشود. در این تابع از new استفاده شده که همان مقداری است که میخواهد اضافه شود. اگر طول رشته login_name برابر صفر باشد یا این رشته حاوی فاصله باشد مقدار جدید به جدول اضافه نمیشود و یک استثنا پرتاب میشود. در غیر این صورت مشکلی وجود ندارد و داده جدید بازگردانده میشود.

قسمت دهم)

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

SQL Error [P0001]: ERROR: Login name must not be empty.
Where: PL/pgSQL function person_bit() line 4 at RAISE

Details >>

همانطور که مشاهده میشود چون طول رشته login_name برابر صفر است در تابعی که نوشتیم به ارور میخورد و عملیات کنسل میشود.

قسمت یازدهم)

```
INSERT INTO person VALUES ('space man', 'Major Tom');
```

SQL Error [P0001]: ERROR: Login name must not include white space.
Where: PL/pgSQL function person_bit() line 8 at RAISE

Details >>

همانطور که مشاهده میشود چون رشته login_name دارای اسپیس است در تابعی که نوشتیم به ارور میخورد و عملیات کنسل میشود.

قسمت دوازدهم)

```
CREATE TABLE person_audit (
    login_name varchar(9) not null,
    display_name text,
    operation varchar,
    effective_at timestamp not null default now(),
    userid name not null default session_user
);
```

Updated Rows	0
Query	CREATE TABLE person_audit (login_name varchar(9) not null, display_name text, operation varchar, effective_at timestamp not null default now(), userid name not null default session_user)
Finish time	Fri Dec 17 10:57:15 IRST 2021

در این مثال جدولی به نام audit_person ساختیم که اتفاقات روی جدول person را در خود ذخیره کند که علاوه بر اطلاعات سطر مورد تغییر، نوع اتفاق انجام شده و زمان انجام آن و کاربر انجام دهنده اتفاق را در خود دارد.

قسمت سیزدهم)

```
CREATE OR REPLACE FUNCTION person_bit()
    RETURNS TRIGGER
    SET SCHEMA 'public'
    LANGUAGE plpgsql
    AS $$
    BEGIN
    IF LENGTH(NEW.login_name) = 0 THEN
        RAISE EXCEPTION 'Login name must not be empty.';
    END IF;

    IF POSITION(' ' IN NEW.login_name) > 0 THEN
        RAISE EXCEPTION 'Login name must not include white space.';
    END IF;

    -- New code to record audits

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

    RETURN NEW;
    END;
    $$;
```

حال تابع person_bit را به گونه‌ای تغییر دادیم تا اگر در صورتی که در داده ورودی مشکلی نبود اطلاعات مورد نظر را به جدول person_audit اضافه کند.

قسمت چهاردهم)

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

Updated Rows	0
Query	CREATE TRIGGER person_biut
	BEFORE INSERT OR UPDATE ON person
	FOR EACH ROW EXECUTE PROCEDURE person_bit()
Finish time	Fri Dec 17 11:14:19 IRST 2021

در این قسمت تریگر قبلی را حذف کردیم و یک تریگر جدید ساختیم که بر اساس آن قبل از هر insert یا update تابع person_bit را صدا بزند.

قسمت پانزدهم)

```
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;
    $$;
```

Name	Value
Updated Rows	0

Query	CREATE OR REPLACE FUNCTION person_bdt()
	RETURNS TRIGGER
	SET SCHEMA 'public'
	LANGUAGE plpgsql
	AS $$
	BEGIN
	-- Record deletion in audit table
	INSERT INTO person_audit (login_name, display_name, operation)
	VALUES (OLD.login_name, OLD.display_name, TG_OP);
	RETURN OLD;
	END;
	$$

Finish time	Fri Dec 17 11:17:01 IRST 2021

در این قسمت یک تابع تعریف کردیم که قبل از هر delete صدا زده شود و اطلاعات مورد نظر را در جدول person_audit ذخیره کند. در این قسمت از OLDاستفاده کردیم که اشاره به داده‌ای دارد که در حال حذف شدن است.

قسمت شانزدهم)

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

Updated Rows	0
Query	CREATE TRIGGER person_bdt BEFORE DELETE ON person FOR EACH ROW EXECUTE PROCEDURE person_bdt()
Finish time	Fri Dec 17 11:26:49 IRST 2021

در این قسمت یک تریگر تعریف کردیم که قبل از هر delete اجرا شود و به ازای هر ردیفی که پاک میشود تابعی که در قسمت قبل تعریف کردیم را صدا کند.

قسمت هفدهم)

```
INSERT INTO person VALUES ('dfunny', 'Doug Funny');
INSERT INTO person VALUES ('pmayo', 'Patti Mayonnaise');
SELECT * FROM person;
SELECT * FROM person_audit;
```

login_name	display_name
dfunny	Doug Funny
pmayo	Patti Mayonnaise

login_name	display_name	operation	effective_at	userid
dfunny	Doug Funny	INSERT	2021-12-17 11:51:29.331	postgres
pmayo	Patti Mayonnaise	INSERT	2021-12-17 11:51:32.644	postgres

در این مرحله دو ردیف به جدول person اضافه کردیم تا ببینیم توابعی که تعریف کردیم درست کار میکنند یا خیر. همانطور که مشاهده میشود درست کار میکنند زیرا ردیف‌های متناظر در جدول person_audit ذخیره شده است.

قسمت هجدهم)

```
UPDATE person SET display_name = 'Doug Yancey Funny' WHERE login_name = 'dfunny';
SELECT * FROM person;
SELECT * FROM person_audit ORDER BY effective_at;
```

login_name	display_name
pmayo	Patti Mayonnaise
dfunny	Doug Yancey Funny

login_name	display_name	operation	effective_at	userid
dfunny	Doug Funny	INSERT	2021-12-17 11:51:29.331	postgres
pmayo	Patti Mayonnaise	INSERT	2021-12-17 11:51:32.644	postgres
dfunny	Doug Yancey Funny	UPDATE	2021-12-17 11:56:02.003	postgres

در این قسمت یک آپدیت انجام دادیم. مشاهده میکنیم نتیجه این آپدیت در جدول person حاصل شده است. همچنین در جدول person_audit نیز مشاهده میشود یک ردیف که متعلق به همین آپدیت است ذخیره شده است.

قسمت نوزدهم)

```
DELETE FROM person WHERE login_name = 'pmayo';
SELECT * FROM person;
SELECT * FROM person_audit ORDER BY effective_at;
```

login_name	display_name
dfunny	Doug Yancey Funny

login_name	display_name	operation	effective_at	userid
dfunny	Doug Funny	INSERT	2021-12-17 11:51:29.331	postgres
pmayo	Patti Mayonnaise	INSERT	2021-12-17 11:51:32.644	postgres
dfunny	Doug Yancey Funny	UPDATE	2021-12-17 11:56:02.003	postgres
pmayo	Patti Mayonnaise	DELETE	2021-12-17 11:59:25.162	postgres

در این قسمت یک دیلیت انجام دادیم. مشاهده میشود ردیف متناظر از جدول person حذف شده است و در جدول person_audit گزارش این دیلیت کردن اضافه شده است.

قسمت بیستم)

```
ALTER TABLE person ADD COLUMN abstract TEXT;
ALTER TABLE person ADD COLUMN ts_abstract TSVECTOR;

ALTER TABLE person_audit ADD COLUMN abstract TEXT;
```

در این قسمت دو ردیف به جدول person اضافه کردیم. یکی به صورت متنی است و دیگری برای جستجو در آن متن به کار می‌رود. همچنین ردیف متنی را به جدول person_audit نیز اضافه کردیم.

قسمت بیست و یکم)

```
CREATE OR REPLACE FUNCTION person_bit()
    RETURNS TRIGGER
    LANGUAGE plpgsql
    SET SCHEMA 'public'
    AS $$
    BEGIN
    IF LENGTH(NEW.login_name) = 0 THEN
        RAISE EXCEPTION 'Login name must not be empty.';
    END IF;

    IF POSITION(' ' IN NEW.login_name) > 0 THEN
        RAISE EXCEPTION 'Login name must not include white space.';
    END IF;

    -- Modified audit code to include text abstract

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

    -- New code to reduce text to text-search vector

    SELECT to_tsvector(NEW.abstract) INTO NEW.ts_abstract;

    RETURN NEW;
    END;
    $$;
```

در تابع person_bit تغییراتی اعمال میکنیم. به دلیل اضافه کردن ستون abstract باید insert مربوط به جدول person_audit را تغییر دهیم. سپس با تابع to_tsvector بردار جستجوی مربوط به متن را ساخته و به NEW اضافه میکنیم.

قسمت بیست و دوم)

```
UPDATE person SET abstract = 'Doug is depicted as an introverted,
   quiet, insecure and gullible 11 (later 12)
   year old boy who wants to fit in with the crowd.'
WHERE login_name = 'dfunny';
SELECT login_name, ts_abstract  FROM person;
```

login_name	ts_abstract
dfunny	'11':11 '12':13 'boy':16 'crowd':24 'depict':3 'doug':1 'fit':20 'gullibl':10 'insecur':8 'introvert':6 'later':12 'old':15 'quiet':7 'want':18 'year':14

در این قسمت برای یک ردیف مقدار abstract ست می‌شود و همانطور که از نتیجه مشخص است بردار جستجوی مربوط به متن آن نیز ساخته شده است.

قسمت بیست و سوم)

```
CREATE VIEW abridged_person AS SELECT login_name, display_name, abstract FROM person;
```

Name	Value
Updated Rows	0
Query	CREATE VIEW abridged_person AS SELECT login_name, display_name, abstract FROM person
Finish time	Sat Dec 18 12:52:13 IRST 2021

در این قسمت یک view برای جدول person ساختیم تا از دست ts_abstract راحت شویم و فقط فیلدهایی را که میخواهیم به کاربر نشان دهیم. رفتار با این view دقیقا مانند جدول اصلی است زیرا ts_abstract در تریگر مربوطه ساخته میشود.

قسمت بیست و چهارم)

```
INSERT INTO abridged_person VALUES ('skeeter', 'Mosquito Valentine', 'Skeeter is Doug''s best friend.
SELECT login_name, ts_abstract FROM person WHERE login_name = 'skeeter';
SELECT login_name, display_name, operation, userid FROM person_audit ORDER BY effective_at;
```

login_name	ts_abstract
skeeter	'best':5 'doug':3 'famous':9 'frequent':18 'friend':6 'honk':15 'make':19 'seri':12 'skeeter':1 'sound':16

login_name	display_name	operation	userid
dfunny	Doug Funny	INSERT	postgres
pmayo	Patti Mayonnaise	INSERT	postgres
dfunny	Doug Yancey Funny	UPDATE	postgres
pmayo	Patti Mayonnaise	DELETE	postgres
dfunny	Doug Yancey Funny	UPDATE	postgres
skeeter	Mosquito Valentine	INSERT	postgres

حال به ویوی ساخته شده یک ردیف اضافه میکنیم. با مشاهده جدول person متوجه میشویم این ردیف به درستی به این جدول اضافه شده است و با مشاهده جدول person_audit میبینیم این عملیات گزارش هم شده است یعنی تریگر مربوط به آن فراخوانی شده است.

قسمت بیست و پنجم)

```
CREATE TABLE transaction (
    login_name character varying(9) NOT NULL,
    post_date date,
    description character varying,
    debit money,
    credit money,
    FOREIGN KEY (login_name) REFERENCES person (login_name)
);
```

یک جدول جدا برای نگهداری حسابرسی‌های مربوط به افراد ساختیم. هر رکورد از این جدول از طریق کلید خارجی به یک رکورد از person متصل است. در این جدول اطلاعات transaction نگهداری شده است.

قسمت بیست و ششم)

```
ALTER TABLE person ADD COLUMN balance MONEY DEFAULT 0;
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-negativ
    END IF;

    IF COALESCE(NEW.credit, 0::money) < 0::money THEN
        RAISE EXCEPTION 'Credit value must be non-negati
    END IF;

    IF newbalance < 0::money THEN
        RAISE EXCEPTION 'Insufficient funds: %', NEW;
    END IF;

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

به جدول person یک ستون جدید با نام balance از نوع money اضافه می‌کنیم تا به ازای هر کاربر مقدار پولش را نگهداری کنیم. سپس یک تابع جدید تعریف میکنیم که در ابتدای آن موجودی فرد به روز می‌شود و سپس چک میشود سطری که در حال اضافه شدن است مشکلی نداشته باشد. (مثلا منفی بودن) دقت کنید هنگام آپدیت نیاز است سطر متناظر از جدول person قفل شود برای همین ما آپدیت را بالاتر از validation گذاشتیم تا سریعتر قفل را در اختیار بگیرد. این کار مشکلی پیش نمی‌آورد زیرا اگر در ادامه exception رخ داد این عملیات roll back میشود و اصطلاحا کل بدنه این تابع all or none است. یعنی یا همه آن اجرا میشود یا هیچکدام اجرا نمیشود. در نهایت نیز یک تریگر تعریف شد تا هنگام اضافه شدن مقادیر به جدول به ازای هر مقدار این تابع را صدا کند تا موجودی فرد آپدیت شود. دقت کنید ممکن است به ازای یک فرد مقدار زیادی transaction وجود داشته باشد و هنگام محاسبه پول یک فرد تمام این transactionها باید محاسبه شود اما با کاری که الان انجام دادیم همواره پول یک فرد را آپدیت نگه میداریم زیرا هر تراکنش این مقدار را آپدیت میکند.

قسمت بیست و هفتم)

```
SELECT login_name, balance FROM person WHERE login_name = 'dfunny';
INSERT INTO transaction (login_name, post_date, description, credit, debit) VALUES
    ('dfunny', '2018-01-11', 'ACH CREDIT FROM: FINANCE AND ACCO ALLOTMENT : Direct Deposit', NULL,
    '$2,000.00');
SELECT login_name, balance FROM person WHERE login_name = 'dfunny';
```

login_name	balance
dfunny	$0.00

login_name	balance
dfunny	$2,000.00

مشاهده میکنیم در ابتدا موجودی صفر است. سپس یک تراکنش به نام این کاربر به مبلغ ۲۰۰۰ دلار ثبت شد. حال با مشاهده جدول person مشاهده میکنیم مقدار پول کاربر به همین اندازه اضافه شده است. پس تریگر مورد نظر انجام شده و تابع مربوطه را صدا زده است.

قسمت بیست و هشتم)

```
INSERT INTO transaction (login_name, post_date, description, credit, debit) VALUES
    ('dfunny', '2018-01-17', 'FOR:BGE PAYMENT ACH Withdrawal', '$2780.52', NULL)
```

SQL Error [P0001]: ERROR: Insufficient funds: (dfunny,2018-01-17,"FOR:BGE PAYMENT ACH Withdrawal",,"$2,780.52")
Where: PL/pgSQL function transaction_bit() line 27 at RAISE

Details >>

همانطور که انتظار داشتیم در یک تراکنش وقتی هزینه از موجودی کاربر بیشتر باشد exception پرتاب میشود.

قسمت بیست و نهم)

```
SELECT login_name, balance FROM person WHERE login_name = 'dfunny';
INSERT INTO transaction (login_name, post_date, description, credit, debit) values
    ('dfunny', '2018-01-17', 'FOR:BGE PAYMENT ACH Withdrawal', '$278.52', NULL);
SELECT login_name, balance FROM person WHERE login_name = 'dfunny';
INSERT INTO transaction (login_name, post_date, description, credit, debit) values
    ('dfunny', '2018-01-23', 'FOR: ANNE ARUNDEL ONLINE PMT ACH Withdrawal', '$35.29', NULL);
SELECT login_name, balance FROM person WHERE login_name = 'dfunny';
```

login_name	balance
dfunny	$2,000.00

login_name	balance
dfunny	$1,721.48

login_name	balance
dfunny	$1,686.19

در ابتدا مشاهده میکنیم مقدار اعتبار این کاربر ۲۰۰۰ دلار است. یعنی همانطور که انتظار داشتیم تابع transaction_bit به صورت all or none اجرا شده است. سپس دو تراکنش دیگر را ثبت کردیم و مشاهده میکنیم نتیجه آن به درستی تاثیر داشته است.

قسمت سی‌ام)

```
BEGIN;
UPDATE person SET balance = '1000000000.00';

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

ROLLBACK;
```

login_name	balance
dfunny	$1,000,000,000.00

در این قسمت میخواهیم یک مشکل را نشان دهیم. ما دوست داریم موجودی کاربر فقط از طریق transaction قابل تغییر باشد اما در اینجا توانستیم به صورت دستی موجودی یک کاربر را خیلی زیاد کنیم. (توجه کنید از begin و rollback استفاده کردیم تا داده به حالت اولیه بازگردد) در ادامه به حل این موضوع خواهیم پرداخت.

قسمت سی و یکم)

```
CREATE OR REPLACE VIEW abridged_person AS
  SELECT login_name, display_name, abstract, balance FROM person;

BEGIN;
UPDATE person SET balance = '1000000000.00';

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

ROLLBACK;
```

login_name	balance
dfunny	$1,000,000,000.00

در این قسمت به view مربوطه مقدار balance هم اضافه کردیم با این تصور که فقط میتوان عملیات خواندن را انجام داد اما اجازه نوشتن روی view را هم داریم و این کار مشکل مارا حل نمیکند.

قسمت سی و دوم)

```
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;
    $$;

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

UPDATE abridged_person SET balance = '1000000000.00';
SELECT login_name, balance FROM abridged_person WHERE login_name = 'dfunny';
```

login_name	balance
dfunny	$1,686.19

یک تریگر تعریف کردیم تا به جای آپدیت کردن به ازای هر سطر تابع abridged_person_iut را اجرا کند. در این تابع نیز از آپدیت شدن جلوگیری کردیم و مقدار جدید را همان مقدار قدیم قرار دادیم. با این تغییر مشاهده میشود در صورت آپدیت شدن مقدار موجودی کاربر تغییری نکرده است و فقط از طریق transactionها قابل محاسبه بوده است.

قسمت سی و سوم)

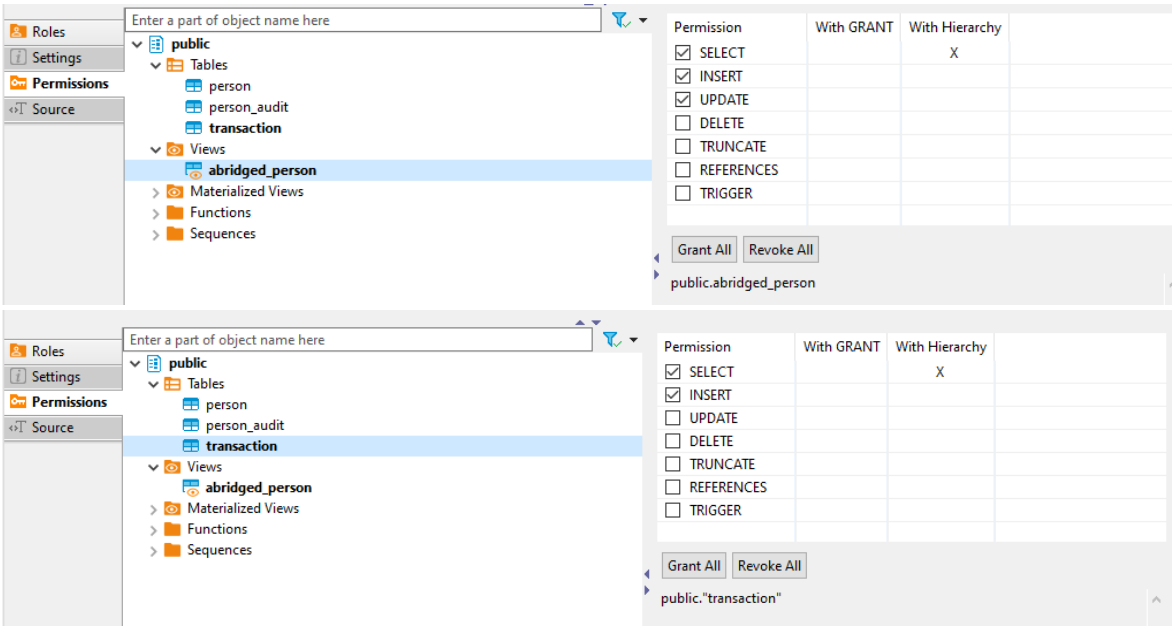
```
CREATE USER eve;
```

Permission	With GRANT	With Hierarchy
☐ SELECT		
☐ INSERT		
☐ UPDATE		
☐ DELETE		
☐ TRUNCATE		
☐ REFERENCES		
☐ TRIGGER		

در این قسمت یک role جدید با نام eve ایجاد کردیم. مشاهده میکنیم این یوزر هیچ دسترسی‌ای ندارد.

قسمت سی و چهارم)

```
GRANT SELECT,INSERT, UPDATE ON abridged_person TO eve;
GRANT SELECT,INSERT ON transaction TO eve;
```



در این قسمت دسترسی‌هایی روی abridged_person و transaction به eve دادیم. مشاهده میکنیم این دسترسی‌ها به این یوزر داده شده است.

قسمت سی و پنجم)

```
SET SESSION AUTHORIZATION eve;
SELECT * FROM person;
SELECT * from person_audit;
SELECT * FROM abridged_person;
```

SQL Error [42501]: ERROR: permission denied for table person

SQL Error [42501]: ERROR: permission denied for table person

login_name	display_name	abstract	balance
skeeter	Mosquito Valentine	Skeeter is Doug's best friend. He is famous in both series for the honking sounds he frequently makes.	$0.00
dfunny	Doug Yancey Funny	Doug is depicted as an introverted, ¶ quiet, insecure and gullible 11 (later 12)¶ year old boy who wants t	$1,686.19

در ابتدا یوزر را eve کردیم و سپس تلاش کردیم به جداول person و person_audit دسترسی داشته باشیم اما همانطور که مشاهده می‌شود به ارور برخوردیم. اما توانستیم جدول abridged_person را بخوانیم زیرا اجازه خواندن آن را داشتیم.

قسمت سی و ششم)

```
SELECT * FROM transaction;
INSERT INTO transaction (login_name, post_date, description, credit, debit) VALUES ('dfunny', '2018-01-2
```

login_name	post_date	description	debit	credit
dfunny	2018-01-11	ACH CREDIT FROM: FINANCE AND ACCO ALLOTMENT : Direct Deposit	$2,000.00	[NULL]
dfunny	2018-01-17	FOR:BGE PAYMENT ACH Withdrawal	[NULL]	$278.52
dfunny	2018-01-23	FOR: ANNE ARUNDEL ONLINE PMT ACH Withdrawal	[NULL]	$35.29

```
SQL Error [42501]: ERROR: permission denied for table person
  Where: SQL statement "UPDATE person
      SET balance =
        balance +
        COALESCE(NEW.debit, 0::money) -
        COALESCE(NEW.credit, 0::money)
      WHERE login_name = NEW.login_name
```

در این قسمت ابتدا جدول transaction را خواندیم و چون اجازه خواندن داشتیم توانستیم این کار را انجام دهیم. اما هنگام اضافه کردن داده به آن به ارور خوردیم زیرا اجازه نوشتن در آن را نداریم. توجه کنید اجازه نوشتن در transaction را داریم و این ارور را هنگامی خوردیم که میخواستیم در person مقدار پول را آپدیت کنیم و اجازه نوشتن در آن جدول را نداریم.

قسمت سی و هفتم)

```
RESET SESSION AUTHORIZATION;
ALTER FUNCTION transaction_bit() SECURITY DEFINER;
SET SESSION AUTHORIZATION eve;
INSERT INTO transaction (login_name, post_date, description, credit, debit) values
    ('dfunny', '2018-01-23', 'ACH CREDIT FROM: FINANCE AND ACCO ALLOTMENT : Direct Deposit',
    NULL, '$2,000.00');
SELECT * FROM transaction;
SELECT login_name, balance FROM abridged_person WHERE login_name = 'dfunny';
```

login_name	post_date	description	debit	credit
dfunny	2018-01-11	ACH CREDIT FROM: FINANCE AND ACCO ALLOTMENT : Direct Deposit	$2,000.00	[NULL]
dfunny	2018-01-17	FOR:BGE PAYMENT ACH Withdrawal	[NULL]	$278.52
dfunny	2018-01-23	FOR: ANNE ARUNDEL ONLINE PMT ACH Withdrawal	[NULL]	$35.29
dfunny	2018-01-23	ACH CREDIT FROM: FINANCE AND ACCO ALLOTMENT : Direct Deposit	$2,000.00	[NULL]

login_name	balance
dfunny	$3,686.19

برای حل مشکل قسمت قبل از SECURITY DEFINER روی تابع استفاده کردیم. حال یوزر eve مستقیما نمیتواند جدول person را تغییر دهد اما میتواند بعد از انجام تراکنش مقداری از آن را آپدیت کند. همانطور که مشاهده میشود یک transaction بدون مشکل اضافه کردیم و بعد از آن مقدار اعتبار در جدول person نیز آپدیت شده است.

بخش دوم)

سوال اول) میخواهیم به ازای هر order_id ببینیم کدام مشتری آن را سفارش داده است و اولین سفارش آن مشتری کدام order_id بوده است.

```
SELECT order_id , customer_id ,
       FIRST_VALUE(order_id) OVER(PARTITION BY customer_id
                                  ORDER BY order_id
                                  ROWS BETWEEN UNBOUNDED PRECEDING
                                  AND CURRENT ROW) AS FirstOrderID
FROM orders o ;
```

	order_id	customer_id	firstorderid
1	10,643	ALFKI	10,643
2	10,692	ALFKI	10,643
3	10,702	ALFKI	10,643
4	10,835	ALFKI	10,643
5	10,952	ALFKI	10,643
6	11,011	ALFKI	10,643
7	10,308	ANATR	10,308
8	10,625	ANATR	10,308
9	10,759	ANATR	10,308
10	10,926	ANATR	10,308
11	10,365	ANTON	10,365

سوال دوم) میخواهیم به ازای هر مشتری سفارش او را کنارش داشته باشیم و همچنین مقدار مسافتی که برای آن سفارش طی شده (freight) و مجموع مسافت‌های طی شده برای سفارش‌های آن مشتری تا آن سفارش (مجموع تجمعی) را داشته باشیم.

```
select customer_id , order_id ,
    sum(freight) over(partition by customer_id order by order_id) as TotalDist
from orders;
```

	customer_id	order_id	totaldist
1	ALFKI	10,643	29.45999908
2	ALFKI	10,692	90.47999573
3	ALFKI	10,702	114.41999817
4	ALFKI	10,835	183.94999695
5	ALFKI	10,952	224.36999512
6	ALFKI	11,011	225.58000183
7	ANATR	10,308	1.61000001
8	ANATR	10,625	45.51000214

سوال سوم) می‌خواهیم ببینیم به ازای هر محصول از category مربوط به آن چند عدد در انبار داریم.

```
select product_name ,
    sum(units_in_stock) over (partition by category_id)
from products;
```

	product_name	sum
1	Guaraná Fantástica	559
2	Ipoh Coffee	559
3	Chartreuse verte	559
4	Côte de Blaye	559
5	Steeleye Stout	559
6	Sasquatch Ale	559
7	Lakkalikööri	559
8	Rhönbräu Klosterbier	559
9	Outback Lager	559
10	Chai	559
11	Laughing Lumberjack Lager	559
12	Chang	559
13	Gula Malacca	507

بخش امتیازی سوال اول)

در این دستور ابتدا جداولی ایجاد میشود. جدول اول purchasingUsers نام دارد. در این جدول هر یوزر در کنار مجموع خریدهایی که داشته است نمایش داده می‌شود. به ازای هر یوزر با استفاده از group by مجموع خریدهایش محاسبه میشود و در صورتی نمایش داده می‌شود که این تعداد از یک بیشتر باشد (با استفاده از having).

سپس جدول دیگری با نام movingUsers ساخته شده که کاربرانی را ذخیره میکند که تعداد zipCode آنها از ۱ بیشتر باشد.

سپس جدول دیگری با نام userSessionMetrics ساختیم که به ازای هر یوزر_سشن تعداد کلیک‌ها، لاگین‌ها و خریدها را نگهداری میکند. در نهایت این سه جدول جون شدند و همه ستون‌های آن به نمایش گذاشته شدند. یعنی خروجی نهایی حاوی یوزرهایی که تعداد zipCode آنها بیشتر از یک است (moving هستند) و حداقل یک خرید (اکشن buy) داشته‌اند، سشن مربوط به آن یوزر، مجموع کلیک‌های آن یوزر، مجموع لاگین‌های آن یوزر و مجموع خریدهای آن یوزر است.

استفاده از case در به دست آوردن مجموع‌ها به ما کمک کرد. ستون actionType میتواند مقادیر مختلفی اعم از buy و login و click داشته باشد و مثلا اگر بخواهیم تعداد ردیف‌هایی که این مقدار برای آنها click است را به دست آوریم از ترکیب case و sum استفاده میکنیم. یعنی میگوییم اگر مقدار آن مورد نظر ما بود case مقدار یک را داشته باشد و با استفاده از sum مجموع این یک‌ها را محاسبه می‌کنیم.

بخش امتیازی قسمت دوم)

```
SELECT product_id,category_id, price
FROM
    (
        SELECT product_id,category_id, price,
        rank() OVER (PARTITION BY category_id ORDER BY price) AS ranking_asc,
        rank() OVER (PARTITION BY category_id ORDER BY price DESC) AS ranking_desc
        FROM products
    ) AS temp_table
WHERE temp_table.ranking_asc < 11 OR temp_table.ranking_desc < 11;
```

product_id	category_id	price
112951	2	3115000
174928	2	3257000
161592	2	3361000
178099	2	3433000
174720	2	3757000
128897	2	3814000
102570	2	3937000
171412	2	3992000

ابتدا یک جدول جدید میسازیم. در این جدول بر اساس کتگوری داده‌ها را پارتیشن میکنیم و در کنار آن رنک آنرا نیز قرار میدهیم. یک بار رنک به صورت نزولی و یکبار به صورت صعودی. در نهایت ردیف‌هایی را نگه میداریم که یکی از رنکینگ‌های آن کمتر مساوی ۱۰ باشد. در این صورت ده قیمت صعودی و ده قیمت نزولی در نتیجه نهایی باقی میمانند و بقیه حذف میشوند.

مشکلات و توضیحات تکمیلی

اگر در کار با نرم افزار دچار مشکل شده بودید و وقت زیادی از شما گرفته شد و یا توضیحات دستورالعمل، ناقص و یا مبهم بود در این قسمت، آنها را ذکر کنید که هم در تصحیح گزارش کار مد نظر قرار گیرد و هم برای ادامه کار، تا حد امکان، این مسایل جانبی به حداقل برسد.
این بخش، اختیاری است و در صورت لزوم، آنرا پر کنید.

آنچه آموختم / پیشنهادات

در این بخش که البته مانند بخش قبل، اختیاری است مهم‌ترین مطلبی که از دستورکار جاری یاد گرفته اید را می توانید ذکر کنید.

این مساله باعث میشود که فیدبک مناسبی از کیفیت و کارآیی دستورالعمل ها داشته باشیم. اگر پیشنهادی هم برای ارائه بهتر این بخش از آزمایشگاه داده در سالیان آتی دارید، در این قسمت آنرا ذکر نمایید.