

Лабораторна робота № 3.

Засоби оптимізації роботи СУБД PostgreSQL

Метою роботи є здобуття практичних навичок використання засобів оптимізації СУБД PostgreSQL.

Завдання роботи полягає у наступному:

1. Перетворити модуль “Модель” з шаблону MVC лабораторної роботи №2 у вигляд об’єктно-реляційної проекції (ORM).
2. Створити та проаналізувати різні типи індексів у PostgreSQL.
3. Розробити тригер бази даних PostgreSQL.

Вимоги до пункту завдання №1

Для перетворення функцій, що реалізують запити до об’єктної бази даних, необхідно встановити бібліотеку sqlalchemy, налаштувати програму на роботу з ORM, розробити класи-сутності для об’єктів-сутностей, представлених відповідними таблицями БД та пов’язаних зв’язками 1:M, M:M та 1:1 виконати опис схеми бази даних. Особливу увагу приділити контролю зовнішніх зв’язків між таблицями засобами ORM.

Замінити виклики запитів мовою SQL на відповідні запити засобами SQLAlchemy по роботі з об’єктами. Обов’язковим є реалізація вставки, вилучення та редагування екземплярів класів-сутностей. Розробка запитів на генерацію даних та пошук екземплярів класів-сутностей вітається, але не є обов’язковою.

Інтерфейси функцій (вхідні та вихідні аргументи функцій модуля “Модель”) мають залишитись без змін.

Вимоги до пункту завдання №2

Відповідно до варіанту індексування продемонструвати на прикладах запитів SQL SELECT підвищення швидкодії їх виконання з використанням індексів, а також пояснити чому для деяких випадків індексування використовувати недоцільно. При цьому для наочного представлення слід використати функцію генерування рандомізованих даних з лабораторної роботи №2, створивши необхідну кількість тестових даних. Навести 4-5 прикладів запитів SELECT (із виведенням результуючих даних), що містять фільтрацію, агрегатні функції, групування та сортування (у необхідних комбінаціях).

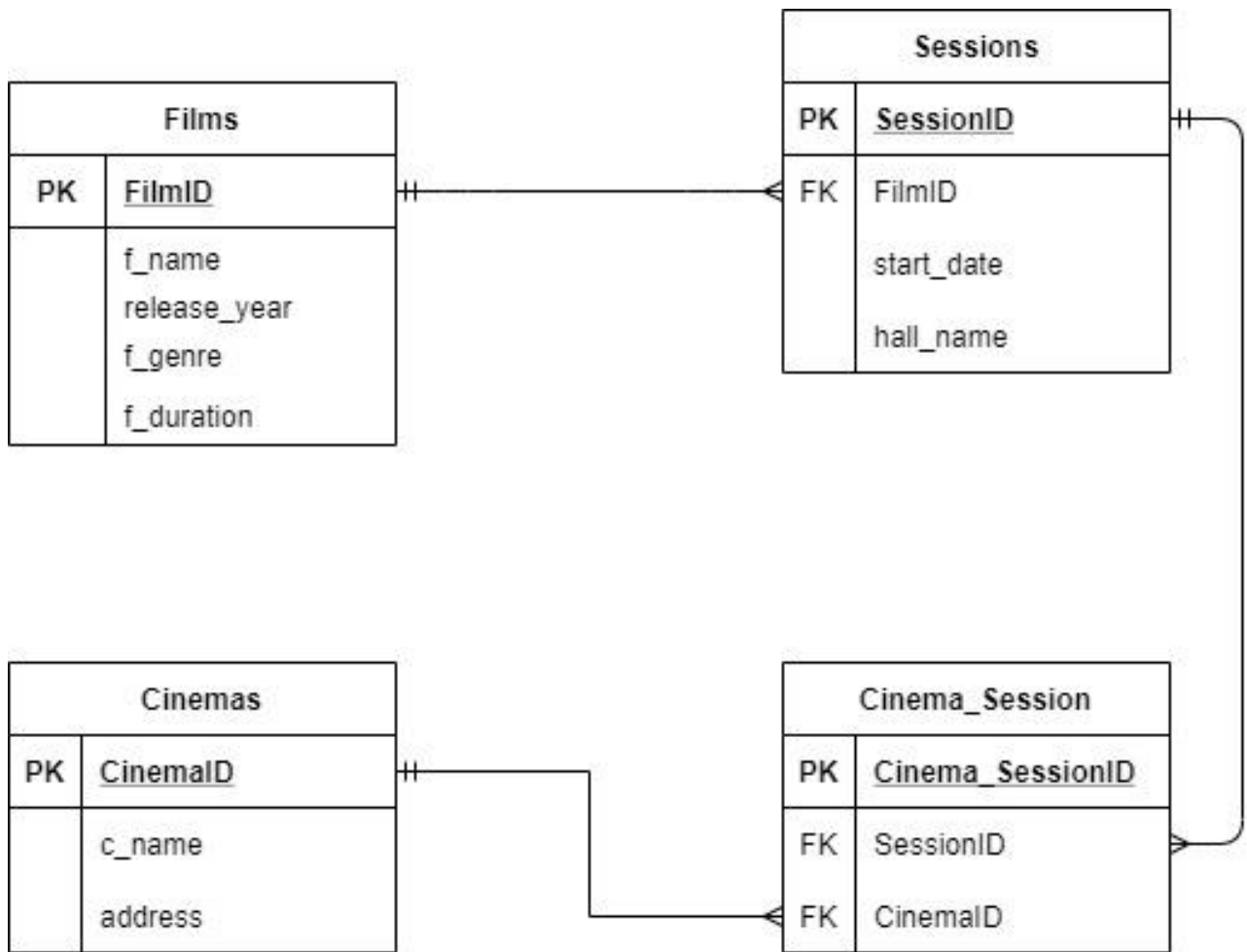
Вимоги до пункту завдання №3

Створити тригер бази даних PostgreSQL відповідно до варіанта. Тригерна функція має включати обробку запису, що модифікується (вставляється або вилучається), умовні оператори, курсорні цикли та обробку виключних ситуацій. Виконати відлагодження тригера при різних вхідних даних, навівши 2-3 приклади його використання.

Варіант 17

<i>17</i>	<i>GIN, BRIN</i>	<i>before update, delete</i>
-----------	------------------	------------------------------

Завдання 1



Зміст файлу base.py

```
from sqlalchemy import create_engine
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import sessionmaker

engine =
create_engine('postgresql://postgres:qwerty@localhost:5432/tic
kets')
Session = sessionmaker(bind=engine)

Base = declarative_base()
```

Перетворений модуль “Модель” з шаблону MVC лабораторної роботи №2 у вигляд об’єктно-реляційної проєкції (ORM). Функції, що виконують вставку, вилучення, модифікації та отримання необхідних даних.

Деякі основні функції з файлу `orm_model.py`

Створити запис

```
def create_item(self, table_name, columns, item):
```

```
    obj = self._tables[table_name]()
    for i in range(len(columns)):
        obj.__dict__[columns[i]] = item[i]
    self._session.add(obj)
    self._session.commit()
```

Створити декілька записів

```
def create_items(self, table_name, columns, items):
```

```
    for j in range(len(items)):
        obj = self._tables[table_name]()
        for i in range(len(columns)):
            obj.__dict__[columns[i]] = items[j][i]
        self._session.add(obj)
    self._session.commit()
```

Взяти дані про запис з бази

```
def read_item(self, table_name, columns, item_id):
```

```
    col_names = []
    tbl_entity = self._tables[table_name]
    for i in range(len(columns)):
        col_names.append(tbl_entity.__dict__[columns[i]])
```

```
    query = self._session.query(*col_names).filter(tbl_entity.id ==
item_id)
    return query.all()
```

Прочитати дані таблиці з бази

```
def read_items(self, table_name, columns):
```

```
    tbl_entity = self._tables[table_name]
    if columns is not None:
        col_names = []
        for i in range(len(columns)):
            col_names.append(tbl_entity.__dict__[columns[i]])
        query = self._session.query(*col_names)
    else:
```

```

        query = self._session.query(tbl_entity)
        return query.all()

# Оновити запис
def update_item(self, table_name, columns, item, item_id):

    tbl_entity = self._tables[table_name]

    update_values = dict(zip(columns, item))
    self._session.query(tbl_entity) \
        .filter(tbl_entity.id == item_id) \
        .update(update_values)

    self._session.commit()

# Видалити запис за ключем
def delete_item(self, table_name, item_id):
    tbl_entity = self._tables[table_name]
    self._session.query(tbl_entity).filter(tbl_entity.id ==
item_id).delete()
    self._session.commit()

```

Класи-сутності для об'єктів-сутностей, представлених відповідними таблицями БД.

Файл cinema.py

```

from sqlalchemy import Column, String, Integer
from base import Base
class Cinema(Base):
    __tablename__ = 'Cinemas'

    id = Column(Integer, primary_key=True)
    c_name = Column(String(20))
    address = Column(String(40))

    def __repr__(self):
        return "<Cinemas('%s', '%s')>" % (self.c_name, self.address)

```

Файл film.py

```
from sqlalchemy import Column, String, Integer, Date

from base import Base

class Film(Base):
    __tablename__ = 'Films'
    id = Column(Integer, primary_key=True)
    f_name = Column(String(20))
    release_year = Column(Integer)
    f_genre = Column(String(10))
    f_duration = Column(Integer)
    def __repr__(self):
        return "<Films('%s', '%s', '%s', '%s')>" % (self.f_name,
self.release_year, self.f_genre, self.f_duration)
```

Файл session.py

```
from sqlalchemy import Column, String, Integer, Date, ForeignKey, Table
from sqlalchemy.orm import relationship

from base import Base

cinema_session_association = Table(
    'Cinema_Session', Base.metadata,
    Column('id', Integer, primary_key=True),
    Column('session_id', Integer, ForeignKey('Sessions.id')),
    Column('cinema_id', Integer, ForeignKey('Cinemas.id'))
)

class Session(Base):
    __tablename__ = 'Sessions'

    id = Column(Integer, primary_key=True)
    start_date = Column(Date)
    hall_name = Column(String(20))
    film_id = Column(Integer, ForeignKey('Films.id'))
    film = relationship("Film", uselist=False)
    cinemas = relationship("Cinema",
secondary=cinema_session_association)
```

```
def __repr__(self):
    return "<Sessions('%s', '%s', '%s')>" % (self.start_date,
self.hall_name, self.film.s_name)
```

Програма читає дані типів з бази даних і пропонує користувачу ввести відповідні дані. Приклад введення нових даних запису до таблиці Films.

```

Введіть значення поля FilmID
Тип поля число
111
Введіть значення поля f_name
Тип поля текст
Ford v Ferrari
Введіть значення поля release_year
Тип поля число
2019
Введіть значення поля f_genre
Тип поля текст
Sport
Введіть значення поля f_duration
Тип поля число
152
+++++
Успіх! Запис з ідентифікатором 111 було додано до таблиці Films !
+++++

```

Оновлення даних таблиці Films.

```
2
Введіть значення поля FilmID
Тип поля число
111
Введіть значення поля f_name
Тип поля текст
Ford v Ferrari
Введіть значення поля release_year
Тип поля число
2019
Введіть значення поля f_genre
Тип поля текст
Sport
Введіть значення поля f_duration
Тип поля число
153
--- --
Запис 111 було змінено на [111, 'Ford v Ferrari', 2019, 'Sport', 153]
--- --
Для продовження натисніть Enter...|
```

Вилучення запису з таблиці Films.

```
1. Додати запис
2. Оновити дані запису
3. Відобразити дані
4. Видалити дані
5. Вихід
Виберіть один з пунктів...
4
Введіть ключ для запису який необхідно видалити
111
-----
Елемент 111 було успішно видалено!
-----
Для продовження натисніть Enter...|
```

Перегляд записів.

```
Для продовження натисніть Enter...
1. Додати запис
2. Оновити дані запису
3. Відобразити дані
4. Видалити дані
5. Вихід
Виберіть один з пунктів...
3
['id', 'f_name', 'release_year', 'f_genre', 'f_duration']
--- ТАБЛИЦЯ FILMS ---
1. <Films('InterStellar', '2014', 'Fantastic', '168')>
2. <Films('Joker', '2019', 'Drama', '116')>
3. <Films('Gentlemen', '2019', 'Criminal', '113')>
4. <Films('film4', '2020', 'None', 'None')>
5. <Films('film5', '2020', 'None', 'None')>
6. <Films('film1', '2020', 'Comedy', 'None')>
```


Запити з минулої роботи на мові SQL було замінено на запити засобами SQLAlchemy по роботі з об'єктами.

Запит 1.

```
SELECT f_name, f_genre, start_date, hall_name '\
FROM "Films", "Sessions", "Cinema_Session", "Cinemas" '\
WHERE "Films"."FilmID" = "Sessions"."FilmID" '\
AND "Sessions"."SessionID" = "Cinema_Session"."SessionID" '\
AND "Cinema_Session"."CinemaID" = "Cinemas"."CinemaID" '\
AND "Cinemas".c_name = \{\}' AND "Sessions".start_date > \{\}'
```

```
def search_query1(self, after_date, cinema):
    date_obj = datetime.datetime.strptime(after_date, '%Y-%m-%d').date()
    query = self._session.query(Film.f_name, Film.f_genre,
Session.start_date, Session.hall_name) \
        .join(Session.film).join(Session.cinemas) \
        .filter(Cinema.c_name == cinema) \
        .filter(Session.start_date > date_obj)
    return query.all()
```

Запит 2.

```
'SELECT f_name, f_duration, start_date, c_name '\
FROM "Films", "Sessions", "Cinema_Session", "Cinemas" '\
WHERE "Films"."FilmID" = "Sessions"."FilmID" '\
AND "Sessions"."SessionID" = "Cinema_Session"."SessionID" '\
AND "Cinema_Session"."CinemaID" = "Cinemas"."CinemaID" '\
AND "Sessions"."start_date" > \{\}' '\
AND "Films".f_duration BETWEEN {} AND {}'
```

```
def search_query2(self, after_date, min_duration, max_duration):
    date_obj = datetime.datetime.strptime(after_date, '%Y-%m-%d').date()
    query = self._session.query(Film.f_name, Film.f_duration,
Cinema.c_name, Session.start_date) \
        .join(Session.film).join(Session.cinemas) \
        .filter(Film.f_duration.between(min_duration, max_duration)) \
        .filter(Session.start_date > date_obj)
    return query.all()
```

Запит 3.

```
'SELECT f_name, f_genre, f_duration, release_year '\n  FROM "Films"\n WHERE '\n  "Films".f_name LIKE \'%{}%\''\n  AND "Films".f_genre LIKE \'%{}%\''
```

```
def search_query3(self, str_seq, str_genre):\n\n    query = self._session.query(Film.f_name, Film.f_genre,\n    Film.f_duration, Film.release_year)\n        .filter(and_(Film.f_name.ilike('%'+str_seq+'%'),\n                        Film.f_genre.ilike('%'+str_genre+'%')))\n    return query.all()
```

Завдання 2

Для аналізу різних типів індексів у PostgreSQL було створено таблицю з двома полями.

```
CREATE TABLE test (\n    num integer,\n    txt character varying\n);\n
```

Таблицю було заповнено 1000000 записів. Для заповнення таблиці було використано операції:

```
TRUNCATE test;\nINSERT INTO test (num, txt)\nSELECT\n    floor(random() * 100 + 1)::int AS num,\n    md5(random())::character varying\nFROM generate_series(1,1000000) ORDER BY num;
```

Для створення індексів GIN, BRIN по текстовому полю використовувалися такі операції:




```
CREATE INDEX txt_idx ON test USING gin (txt gin_trgm_ops);
```

```
CREATE INDEX txt_idx ON test USING brin (txt);
```

Для створення індексів по числовому полю для BRIN (для індексу GIN не розглядалося):

```
CREATE INDEX num_idx ON test USING brin (num);
```

Результат запиту `SELECT * FROM test`.

	 num integer		txt character varying	
1		1	54561684b90d9828fca25191bddfcdb7	
2		1	1a9136e816ce92efc35676b2939b60c2	
3		1	8b45410e8d5519b0aa6255610a5bbf9d	
4		1	68266824bfcdd35c79a9688d1f2ec21e	
5		1	720afd13d34ada88d294e62a486c14a7	
6		1	3eedf38cddb68e438e54537b56510718	
7		1	f11698db12b102e4bea2f5380432fc70	
8		1	3222e6181dabb5318c3cab8c772e4855	
9		1	fbae1b693c177b12a9de116e8aa8ba60	
10		1	a970a6f80cb7bf98e52dc745b3c1f2ff	

Було розглянуто такі запити.

1. `SELECT count(*) FROM test WHERE txt ILIKE '%abc%';`
2. `SELECT count(*) FROM test WHERE txt = '34c6b4e97f3f5a72d8e1e7df9dbf0367';`
3. `SELECT txt FROM test ORDER BY txt`
4. `SELECT num FROM test WHERE num > 33`
5. `SELECT num, count(num) FROM test WHERE num > 70 GROUP BY num`

Кожен запит виконувався декілька разів (5 разів) для визначення більш точного результату. У таблиці наведено середній час виконання запиту.

№ запиту	Без індексування	GIN	BRIN
1	1171,8	193	1131,4
2	306,4	273,6	286,4
3	4562,2	4239,6	4231,4
4	353,4		325,4
5	322,8		282

Виведення результуючих даних для варіанту без індексів.

1. `SELECT count(*) FROM test WHERE txt ILIKE '%abc%';`

	count bigint
1	7324

✓ Successfully run. Total query runtime: 1 secs 206 msec. 1 rows affected.

2. `SELECT count(*) FROM test WHERE txt = '34c6b4e97f3f5a72d8e1e7df9dbf0367';`

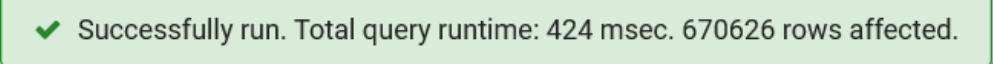
	count bigint
1	1

✓ Successfully run. Total query runtime: 366 msec. 1 rows affected.


3. SELECT txt FROM test ORDER BY txt

	txt character varying	
1	0000062f3d755e19afc941d9c9df649a	
2	000029106c817ad5bff9085cb15d46ab	
3	000039473d2082b3da186815c47e973b	
4	00003f9fd53018fb2b24f3e3c92f7098	
5	0000438c3bf509a72dee70b139b5eaa7	
6	00005d841cea3226db01ede0d0f37b96	
7	0000665b5e9b46d9fbdc47b1d72485d7	
8	0000774b1b1b1b1b1b1b1b1b1b1b1b1b	

4. SELECT num FROM test WHERE num > 33

	num integer	
1	34	
2	34	
3	34	
4	34	
5	34	
6	34	
7	34	
8	34	

5. SELECT num, count(num) FROM test WHERE num > 70 GROUP BY

	num integer	count bigint	
1	71	10145	
2	72	9946	
3	73	9914	
4	74	10034	
5	75	9997	
6	76	9813	
7	77	9897	
8	78	1003	

Варіант з індексом GIN.

1. `SELECT count(*) FROM test WHERE txt ILIKE '%abc%';`

	count bigint	
1	7324	

✓ Successfully run. Total query runtime: 212 msec. 1 rows affected.

2. `SELECT count(*) FROM test WHERE txt = '34c6b4e97f3f5a72d8e1e7df9dbf0367';`

	count bigint	
1	1	

✓ Successfully run. Total query runtime: 406 msec. 1 rows affected.

3. `SELECT txt FROM test ORDER BY txt`

	txt character varying	
1	0000062f3d755e19afc941d9c9df649a	
2	000029106c817ad5bff9085cb15d46ab	
3	000039473d2082b3da186815c47e973b	
4	00003f9fd53018fb2b24f3e3c92f7098	
5	0000438c3bf509a72dee70b139b5eaa7	
6	00005d841cea3226db01ede0d0f37b96	
7	0000665b5e8b46d8bdc47b1d72485d7	
8	000	

✓ Successfully run. Total query runtime: 4 secs 835 msec. 1000000 rows affected.

Варіант з індексом BRIN.

1. SELECT count(*) FROM test WHERE txt ILIKE '%abc%';

	count bigint	
1	7324	

✓ Successfully run. Total query runtime: 1 secs 220 msec. 1 rows affected.

2. SELECT count(*) FROM test WHERE txt = '34c6b4e97f3f5a72d8e1e7df9dbf0367';

	count bigint	
1	1	

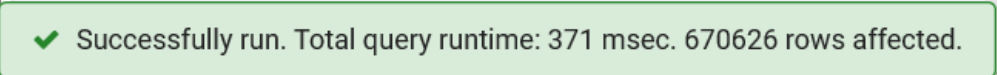
✓ Successfully run. Total query runtime: 410 msec. 1 rows affected.

3. SELECT txt FROM test ORDER BY txt

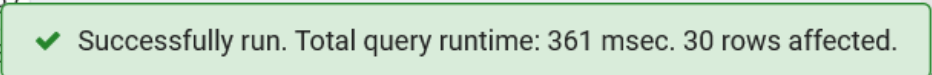
	txt character varying	
1	0000062f3d755e19afc941d9c9df649a	
2	000029106c817ad5bff9085cb15d46ab	
3	000039473d2082b3da186815c47e973b	
4	00003f9fd53018fb2b24f3e3c92f7098	
5	0000438c3bf509a72dee70b139b5eaa7	
6	00005d841cea3226db01ede0d0f37b96	
7	0000665b5e9b46d08bdc47b1d72485d7	
8	0000776c6f0b46d08bdc47b1d72485d7	

✓ Successfully run. Total query runtime: 4 secs 478 msec. 1000000 rows affected.

4. SELECT num FROM test WHERE num > 33

	num integer	
1	34	
2	34	
3	34	
4	34	
5	34	
6	34	
7	34	
8	34	

5. SELECT num, count(num) FROM test WHERE num > 70 GROUP BY num

	num integer	count bigint	
1	71	10145	
2	72	9946	
3	73	9914	
4	74	10034	
5	75	9997	
6	76	9813	
7	77	9897	
8	78	1003	

Як бачимо індекс GIN проявив себе досить добре при пошуку рядку символів у тексті. Але на GIN можливо розраховувати лише при повнотекстовому пошуку, або при використанні операцій LIKE, ILIKE. Та пошуку змісту полів типу JSON. В інших випадках його не слід застосовувати, а також він не може бути створений для чисельних типів стовпців. Також недоліком GIN досить довге створення індексу.

Індекс BRIN показав не досить значне підвищення швидкодії, але його було розраховано на застосування до дуже великих таблиць і 100000 записів не так вже і багато. Також загальна ідея BRIN це не збільшення швидкодії, а уникнення непотрібних рядків.

BRIN-індекс має сенс застосовувати для таблиць, в яких частина даних вже за своєю природою якось відсортована. Наприклад, це характерно для логів або для історії замовлень.

BRIN прискорює оператори порівняння, але не впливає на like-запити. BRIN - не є унікальним індексом, тому не може використовуватися в якості індексу первинного ключа.

Завдання 3

Створення тригерної функції.

```
CREATE FUNCTION check_errors() RETURNS trigger
    LANGUAGE plpgsql
    AS $$
DECLARE
cur_films cursor for select * from "Films";
is_found boolean;
sumstr varchar;
BEGIN
    sumstr = '';
    is_found = false;
    IF (TG_OP = 'UPDATE') THEN
        IF NEW.start_date < CURRENT_DATE OR NEW.start_date is NULL THEN
            sumstr := sumstr || ' start_date';
            RAISE NOTICE 'Start date % is not correct!', NEW.start_date;
        END IF;
        IF NEW.hall_name = '' OR NEW.hall_name is NULL THEN
            sumstr := sumstr || ' hall_name';
            RAISE NOTICE 'Hall name must be not empty!';
        END IF;
        FOR record IN cur_films LOOP
            IF NEW.film_id = record.id THEN
                is_found = true;
                EXIT;
            END IF;
        END LOOP;
        IF is_found = false THEN
            sumstr := sumstr || ' film_id';
            RAISE NOTICE 'Film must be present in database!';
        END IF;
        IF sumstr <> '' THEN
            sumstr := TG_OP || ' Sessions' || sumstr;
            INSERT INTO logs(text,date_added) values (sumstr,NOW());
            --RAISE EXCEPTION 'Row % cannot updated!', NEW;
```

```

        END IF;
        RETURN NEW;
    ELSIF TG_OP = 'DELETE' THEN
        sumstr := TG_OP||' Sessions '|| OLD;
        INSERT INTO logs(text,date_added) values (sumstr,NOW());
        RETURN OLD;
    END IF;
    RETURN NEW;
END;
$$;

```

```

CREATE TRIGGER check_errors BEFORE DELETE OR UPDATE ON "Sessions"
FOR EACH ROW EXECUTE FUNCTION check_errors();

```

Даний тригер записує до таблиці logs всі видалення з таблиці «Sessions», спроби оновлення цієї таблиці з помилковими даними та сповіщає про помилки введення даних.

Перший приклад

```

1  UPDATE "Sessions" SET
2  hall_name = 'Emerald',
3  film_id = 11
4  WHERE id = 10;
5

```

Data Output
Explain
Messages
Notifications

ЗАМЕЧАНИЕ: Film must be present in database!

ERROR: ОШИБКА: INSERT или UPDATE в таблице "Sessions" нарушает ограничение внешнего ключа "Sessions_film_id_fkey"

DETAIL: Ключ (film_id)=(11) отсутствует в таблице "Films".

SQL state: 23503






Другий приклад

```
5 UPDATE "Sessions" SET
6 hall_name = 'Emerald',
7 start_date = '2020-11-25'
8 WHERE id = 10;
9
```

Data Output Explain Messages Notifications

ЗАМЕЧАНИЕ: Start date 2020-11-25 is not correct!
UPDATE 1

Query returned successfully in 52 msec.

				
	id [PK] integer	start_date date	hall_name character varying (20)	film_id integer
1	1	2020-09-17	Almandine	1
2	2	2020-09-17	Ultramarine	2
3	3	2020-09-18	Terracotta	3
4	10	2020-11-25	Emerald	8

Третій приклад

```
9 DELETE FROM "Sessions" WHERE id = 100;
10
```

Data Output Explain Messages Notifications

DELETE 1

Query returned successfully in 140 msec.

Зміст таблиці «logs»

```
SELECT * FROM public.logs
```

Output Explain Messages Notifications

text	date_added
text	timestamp without time zone
UPDATE Sessions start_date	2020-11-26 03:27:29.555228
DELETE Sessions (100,2020-12-12,Test,8)	2020-11-26 03:27:45.342952