



Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ

КАФЕДРА КОМПЬЮТЕРНЫЕ СИСТЕМЫ И СЕТИ (ИУ6)

НАПРАВЛЕНИЕ ПОДГОТОВКИ 09.03.01 Информатика и вычислительная техника

О Т Ч Е Т

по лабораторной работе № 2


Название: Создание БД для приложения

Дисциплина: Базы данных

Вариант: 23

Студент

ИУ6-33Б
(Группа)


(Подпись, дата)

Д. И. МIRONENKO
(И.О. Фамилия)

Преподаватель

(Подпись, дата)

М. А. Скворцова
(И.О. Фамилия)

Часть 1. Проектирование схемы базы данных

Задание

Каждый индивидуальный вариант содержит предметную область, из которой должна быть проектируемая база данных. К данной предметной области необходимо добавить не менее 2-х дополнительных таблиц (сущностей), необходимых для детального решения поставленной задачи. Задачей студента является решить, для чего будет использоваться создаваемая база данных, и, исходя из этого, построить её концептуальную схему. Результатом данной части лабораторной работы является схема базы данных (в виде ER-диаграммы, содержащей таблицы и связи между ними, с уточнением типов столбцов, с описание внешних и первичных ключей). При сдаче задания студент должен обосновать соответствие созданной схемы поставленной задаче.

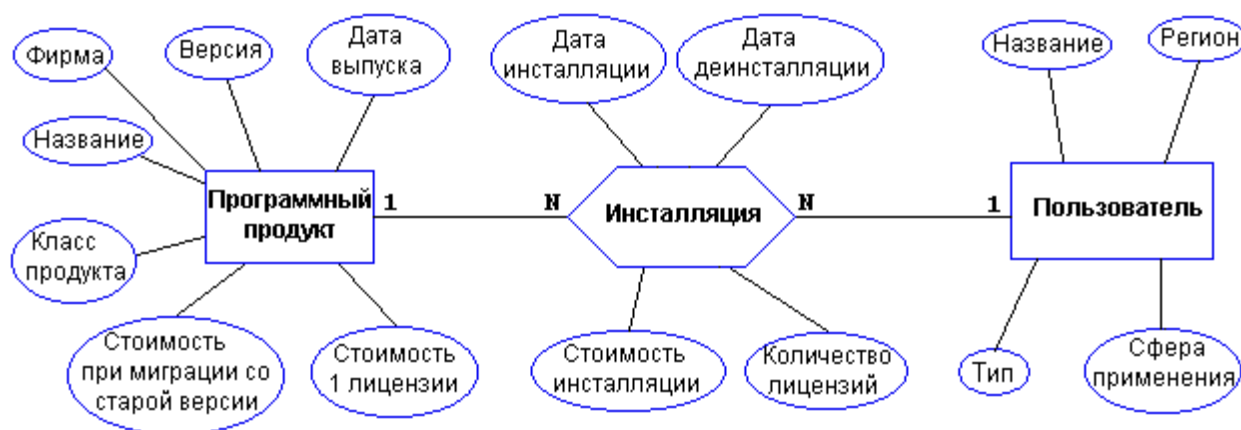


Рисунок 1 – предметная схема

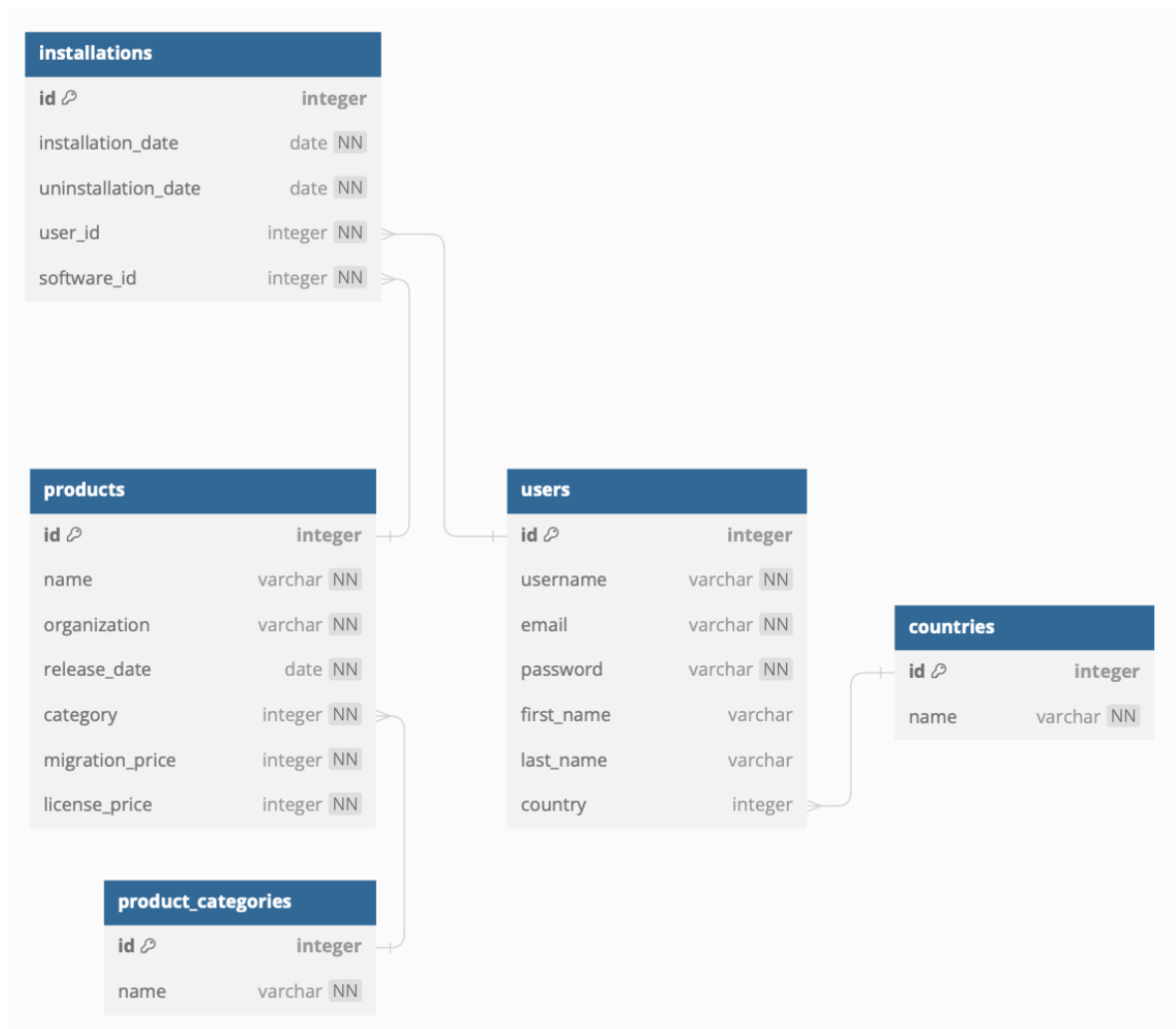


Рисунок 2 – ER-диаграмма

Часть 2. Создание и заполнения таблиц

Задание

Необходимо определить первичные и внешние ключи, а также декларативные ограничения целостности (возможность принимать неопределенное значение, уникальные ключи, проверочные ограничения и т. д.). Таблицы следует создавать в отдельной базе данных. Кроме того, нужно подготовить данные для заполнения созданных таблиц. Кроме того, нужно подготовить данные для заполнения созданных таблиц. Объем подготовленных данных должен составлять не менее 10 экземпляров для каждой из стержневых сущностей и 1000 экземпляров для целевой сущности. На основе этих данных необходимо создать SQL-скрипт для вставки соответствующих строк в таблицы БД.

SQL-скрипт для разметки таблицы

```
DROP SCHEMA IF EXISTS "public" CASCADE;
```

```
CREATE SCHEMA IF NOT EXISTS "public";
```

```
CREATE TABLE "products" (  
    "id" integer PRIMARY KEY,  
    "name" varchar UNIQUE NOT NULL,  
    "organization" varchar NOT NULL,  
    "release_date" date NOT NULL,  
    "category" integer NOT NULL,  
    "migration_price" integer NOT NULL,  
    "license_price" integer NOT NULL  
);
```

```
CREATE TABLE "users" (  
    "id" integer PRIMARY KEY,  
    "username" varchar UNIQUE NOT NULL,  
    "email" varchar UNIQUE NOT NULL,  
    "password" varchar NOT NULL,  
    "first_name" varchar,  
    "last_name" varchar,  
    "country" integer  
);
```

```
CREATE TABLE "installations" (  
    "id" integer PRIMARY KEY,  
    "installation_date" date NOT NULL,  
    "uninstallation_date" date NOT NULL,  
    "user_id" integer NOT NULL,  
    "software_id" integer NOT NULL  
);
```

```
CREATE TABLE "product_categories" (  
    "id" integer PRIMARY KEY,  
    "name" varchar NOT NULL  
);
```

```

CREATE TABLE "countries" (
    "id" integer PRIMARY KEY,
    "name" varchar NOT NULL
);

ALTER TABLE "products" ADD FOREIGN KEY ("category") REFERENCES
"product_categories" ("id") ON DELETE CASCADE ON UPDATE CASCADE;

ALTER TABLE "users" ADD FOREIGN KEY ("country") REFERENCES "countries" ("id") ON
DELETE CASCADE ON UPDATE CASCADE;

ALTER TABLE "installations" ADD FOREIGN KEY ("user_id") REFERENCES "users" ("id")
ON DELETE CASCADE ON UPDATE CASCADE;

ALTER TABLE "installations" ADD FOREIGN KEY ("software_id") REFERENCES "products"
("id") ON DELETE CASCADE ON UPDATE CASCADE;

```

SQL-скрипт для вставки значений в таблицу

```

INSERT INTO "public"."countries" ("id", "name") VALUES ('1', 'Russia');
INSERT INTO "public"."countries" ("id", "name") VALUES ('2', 'Argentina');
INSERT INTO "public"."countries" ("id", "name") VALUES ('3', 'Brazil');
INSERT INTO "public"."countries" ("id", "name") VALUES ('4', 'Canada');
INSERT INTO "public"."countries" ("id", "name") VALUES ('5', 'Denmark');
INSERT INTO "public"."countries" ("id", "name") VALUES ('6', 'Estonia');
INSERT INTO "public"."countries" ("id", "name") VALUES ('7', 'France');
INSERT INTO "public"."countries" ("id", "name") VALUES ('8', 'Germany');
INSERT INTO "public"."countries" ("id", "name") VALUES ('9', 'Honduras');
INSERT INTO "public"."countries" ("id", "name") VALUES ('10', 'India');
INSERT INTO "public"."countries" ("id", "name") VALUES ('11', 'Japan');
INSERT INTO "public"."countries" ("id", "name") VALUES ('12', 'Kenya');
INSERT INTO "public"."countries" ("id", "name") VALUES ('13', 'Lithuania');
INSERT INTO "public"."countries" ("id", "name") VALUES ('14', 'Malaysia');
INSERT INTO "public"."countries" ("id", "name") VALUES ('15', 'Netherlands');
INSERT INTO "public"."countries" ("id", "name") VALUES ('16', 'Oman');
INSERT INTO "public"."countries" ("id", "name") VALUES ('17', 'Poland');
INSERT INTO "public"."countries" ("id", "name") VALUES ('18', 'Qatar');

```

```
INSERT INTO "public"."countries" ("id", "name") VALUES ('19', 'South Korea');
INSERT INTO "public"."countries" ("id", "name") VALUES ('20', 'Tanzania');
INSERT INTO "public"."countries" ("id", "name") VALUES ('21', 'Venezuela');
INSERT INTO "public"."countries" ("id", "name") VALUES ('22', 'Wales');
INSERT INTO "public"."countries" ("id", "name") VALUES ('23', 'Xinjiang');
INSERT INTO "public"."countries" ("id", "name") VALUES ('24', 'Yemen');
INSERT INTO "public"."countries" ("id", "name") VALUES ('25', 'Zimbabwe');
INSERT INTO "public"."product_categories" ("id", "name") VALUES ('1', 'OS');
INSERT INTO "public"."product_categories" ("id", "name") VALUES ('2', 'VR');
INSERT INTO "public"."product_categories" ("id", "name") VALUES ('3', 'Business');
INSERT INTO "public"."product_categories" ("id", "name") VALUES ('4',
'Education');
INSERT INTO "public"."product_categories" ("id", "name") VALUES ('5', 'Finance');
INSERT INTO "public"."product_categories" ("id", "name") VALUES ('6', 'Graphics &
Design');
INSERT INTO "public"."product_categories" ("id", "name") VALUES ('7', 'Kids');
INSERT INTO "public"."product_categories" ("id", "name") VALUES ('8', 'Magazines &
Newspapers');
INSERT INTO "public"."product_categories" ("id", "name") VALUES ('9', 'Music');
INSERT INTO "public"."product_categories" ("id", "name") VALUES ('10', 'News');
INSERT INTO "public"."product_categories" ("id", "name") VALUES ('11',
'Productivity');
INSERT INTO "public"."product_categories" ("id", "name") VALUES ('12', 'Browser
Extension');
INSERT INTO "public"."product_categories" ("id", "name") VALUES ('13', 'Social
Networking');
INSERT INTO "public"."product_categories" ("id", "name") VALUES ('14', 'Travel');
INSERT INTO "public"."product_categories" ("id", "name") VALUES ('15', 'Books');
INSERT INTO "public"."product_categories" ("id", "name") VALUES ('16', 'Developer
Tools');
INSERT INTO "public"."product_categories" ("id", "name") VALUES ('17',
'Entertainment');
INSERT INTO "public"."product_categories" ("id", "name") VALUES ('18', 'Food &
Drink');
...
```

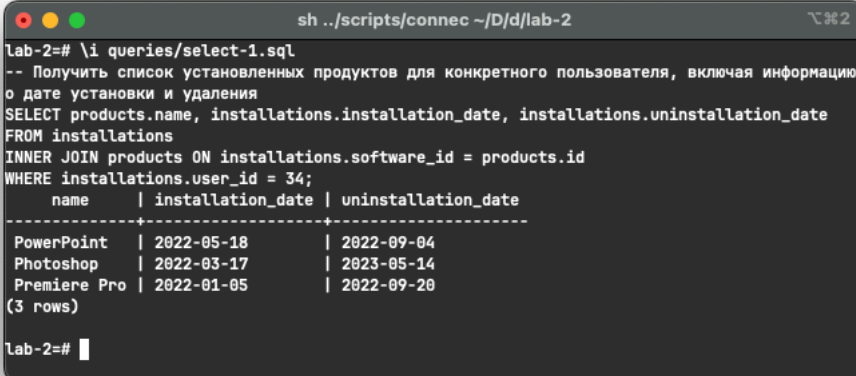
Часть 3. Создание и заполнения таблиц

Задание

- Подготовить 3-4 выборки, которые имеют осмысленное значение для предметной области, и также составить для них SQL-скрипты.
- Сформулировать 3-4 запроса на изменение и удаление из базы данных. Запросы должны быть сформулированы в терминах предметной области. Среди запросов обязательно должны быть такие, которые будут вызывать срабатывание ограничений целостности. Составить SQL-скрипты для выполнения этих запросов.

Получение списка установленных продуктов для конкретного пользователя, включая информацию о дате установки и удаления

```
SELECT products.name, installations.installation_date,  
installations.uninstallation_date  
FROM installations  
INNER JOIN products ON installations.software_id = products.id  
WHERE installations.user_id = 34;
```



The screenshot shows a terminal window with the following content:

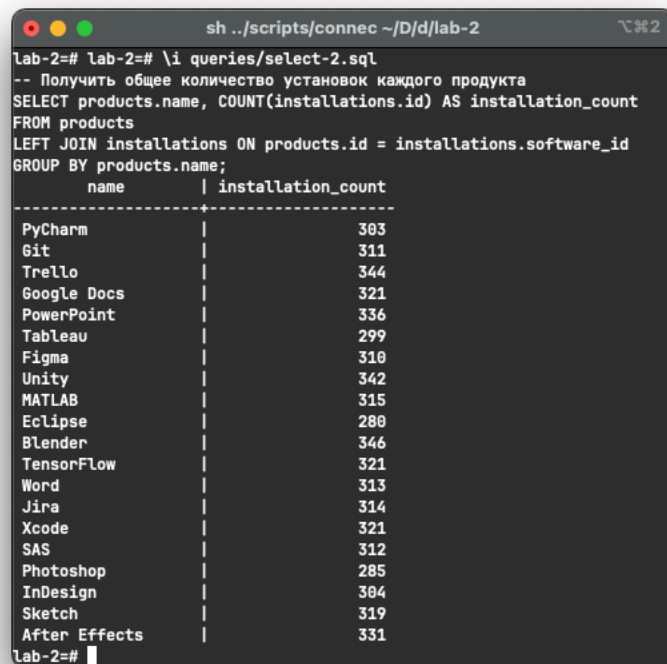
```
sh ../scripts/connec ~/D/d/lab-2  
lab-2=# \i queries/select-1.sql  
-- Получить список установленных продуктов для конкретного пользователя, включая информацию  
-- о дате установки и удаления  
SELECT products.name, installations.installation_date, installations.uninstallation_date  
FROM installations  
INNER JOIN products ON installations.software_id = products.id  
WHERE installations.user_id = 34;  
name | installation_date | uninstallation_date  
-----+-----+-----  
PowerPoint | 2022-05-18 | 2022-09-04  
Photoshop | 2022-03-17 | 2023-05-14  
Premiere Pro | 2022-01-05 | 2022-09-20  
(3 rows)  
lab-2=#
```

Рисунок 3 – результат SELECT запроса

Получение общего количества установок каждого продукта

```
SELECT products.name, COUNT(installations.id) AS installation_count  
FROM products
```

```
LEFT JOIN installations ON products.id = installations.software_id  
GROUP BY products.name;
```



The screenshot shows a terminal window with the title bar "sh ../scripts/connec ~/D/d/lab-2". The prompt is "lab-2=#". The user enters the command "\i queries/select-2.sql". The terminal shows the SQL query being executed, followed by the results of the query. The results are displayed as a table with two columns: "name" and "installation_count".

```
lab-2=# lab-2=# \i queries/select-2.sql  
-- Получить общее количество установок каждого продукта  
SELECT products.name, COUNT(installations.id) AS installation_count  
FROM products  
LEFT JOIN installations ON products.id = installations.software_id  
GROUP BY products.name;  
name | installation_count  
-----+-----  
PyCharm | 303  
Git | 311  
Trello | 344  
Google Docs | 321  
PowerPoint | 336  
Tableau | 299  
Figma | 310  
Unity | 342  
MATLAB | 315  
Eclipse | 280  
Blender | 346  
TensorFlow | 321  
Word | 313  
Jira | 314  
Xcode | 321  
SAS | 312  
Photoshop | 285  
InDesign | 304  
Sketch | 319  
After Effects | 331  
lab-2=#
```

Рисунок 4 – результат SELECT запроса

Получение списка всех установок продуктов, которые были установлены после определенной даты, включая информацию о пользователе и продукте

```
SELECT users.username, products.name, installations.installation_date  
FROM installations  
INNER JOIN users ON installations.user_id = users.id  
INNER JOIN products ON installations.software_id = products.id  
WHERE installations.installation_date > '2023-11-1';
```



```
sh ../scripts/connec ~/D/d/lab-2
lab-2=# \i queries/select-3.sql
-- Получить список всех установок продуктов, которые были установлены п
осле определенной даты, включая информацию о пользователе и продукте
SELECT users.username, products.name, installations.installation_date
FROM installations
INNER JOIN users ON installations.user_id = users.id
INNER JOIN products ON installations.software_id = products.id
WHERE installations.installation_date > '2023-11-1';
  username |      name      | installation_date
-----+-----+-----
sophia8948 | Jira            | 2023-11-05
david2806  | PyCharm         | 2023-11-05
william2600 | Tableau         | 2023-11-02
jane7854   | Word            | 2023-11-03
emily4438  | Premiere Pro    | 2023-11-03
david840   | MATLAB          | 2023-11-02
mia5566    | Git             | 2023-11-03
chris2504  | Tableau         | 2023-11-05
ella5493   | PyCharm         | 2023-11-03
emily7849  | Premiere Pro    | 2023-11-05
matthew6052 | TensorFlow      | 2023-11-05
jacob4596  | PyCharm         | 2023-11-04
(12 rows)
lab-2=#
```

Рисунок 5 – результат SELECT запроса

Обновление цены миграции и лицензии для продукта с определенным идентификатором

```
UPDATE products SET migration_price = 30, license_price = 100 WHERE id = 6;
```

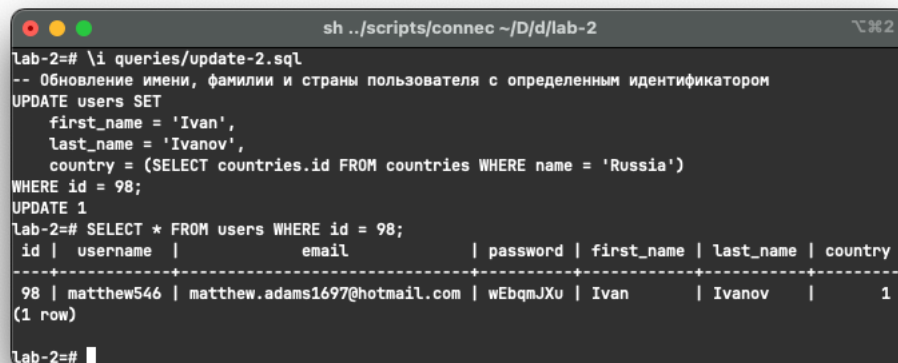
```
sh ../scripts/connec ~/D/d/lab-2
lab-2=# \i queries/update-1.sql
-- Обновление цены миграции и лицензии для продукта с определенным идентификатором:
UPDATE products SET migration_price = 30, license_price = 100 WHERE id = 6;
UPDATE 1
lab-2=# SELECT * FROM products WHERE id = 6;
 id |      name      | organization | release_date | category | migration_price | license_price
-----+-----+-----+-----+-----+-----+-----
  6 | Premiere Pro   | McDonalds   | 2020-10-27  |      17 |          30     |          100
(1 row)
lab-2=#
```

Рисунок 6– результат UPDATE запроса

Обновление имени, фамилии и страны пользователя с определенным идентификатором

```
UPDATE users SET
    first_name = 'Ivan',
    last_name = 'Ivanov',
```

```
country = (SELECT countries.id FROM countries WHERE name = 'Russia')
WHERE id = 98;
```

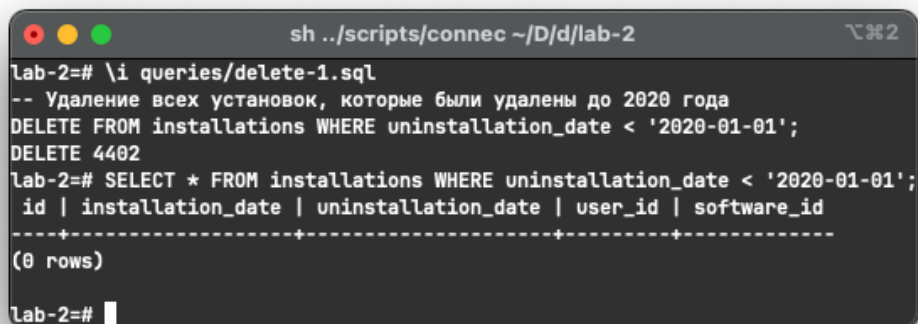


```
sh ../scripts/connec ~/D/d/lab-2
lab-2=# \i queries/update-2.sql
-- Обновление имени, фамилии и страны пользователя с определенным идентификатором
UPDATE users SET
  first_name = 'Ivan',
  last_name = 'Ivanov',
  country = (SELECT countries.id FROM countries WHERE name = 'Russia')
WHERE id = 98;
UPDATE 1
lab-2=# SELECT * FROM users WHERE id = 98;
 id | username | email | password | first_name | last_name | country
-----+-----+-----+-----+-----+-----+-----
  98 | matthew546 | matthew.adams1697@hotmail.com | wEbqmJXu | Ivan | Ivanov | 1
(1 row)
lab-2=#
```

Рисунок 7 – результат UPDATE запроса

Удаление всех установок, которые были удалены до 2020 года

```
DELETE FROM installations WHERE uninstallation_date < '2020-01-01';
```



```
sh ../scripts/connec ~/D/d/lab-2
lab-2=# \i queries/delete-1.sql
-- Удаление всех установок, которые были удалены до 2020 года
DELETE FROM installations WHERE uninstallation_date < '2020-01-01';
DELETE 4402
lab-2=# SELECT * FROM installations WHERE uninstallation_date < '2020-01-01';
 id | installation_date | uninstallation_date | user_id | software_id
-----+-----+-----+-----+-----
(0 rows)
lab-2=#
```

Рисунок 8 – результат DELETE запроса

Часть 4. Контроль целостности данных

Задание

Необходимо подготовить SQL-скрипты для проверки наличия аномалий (потерянных изменений, грязных чтений, неповторяющихся чтений, фантомов)

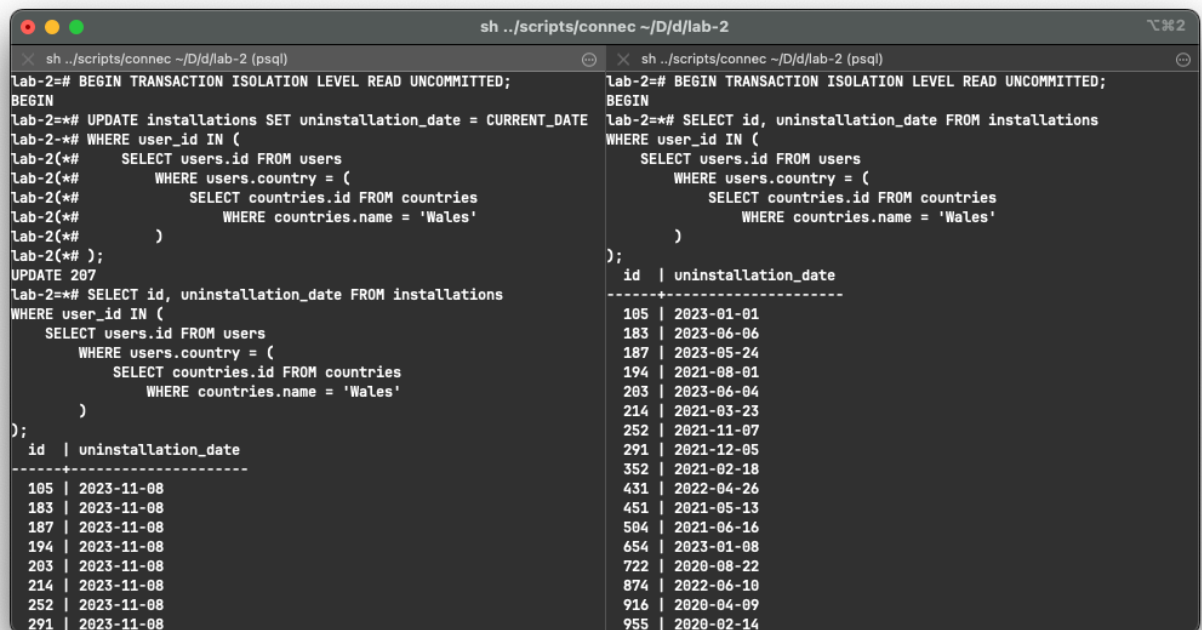
при параллельном исполнении транзакций на различных уровнях изолированности SQL/92 (READ UNCOMMITTED, READ COMMITTED, REPEATABLE READ, SERIALIZABLE). Подготовленные скрипты должны работать с одной из таблиц, созданных в практическом задании №2.1. Для проверки наличия аномалий потребуются два параллельных сеанса, операторы в которых выполняются пошагово:

- ☐ Установить в обоих сеансах уровень изоляции READ UNCOMMITTED. Выполнить сценарии проверки наличия аномалий потерянных изменений и грязных чтений.
- ☐ Установить в обоих сеансах уровень изоляции READ COMMITTED. Выполнить сценарии проверки наличия аномалий грязных чтений и неповторяющихся чтений.
- ☐ Установить в обоих сеансах уровень изоляции REPEATABLE READ. Выполнить сценарии проверки наличия аномалий неповторяющихся чтений и фантомов.
- ☐ Установить в обоих сеансах уровень изоляции SERIALIZABLE. Выполнить сценарий проверки наличия фантомов.

Необходимо составить скрипт для создания триггера, а также подготовить несколько запросов для проверки и демонстрации его полезных свойств:

- ☐ Изменение данных для сохранения целостности.
- ☐ Проверка транзакций и их откат в случае нарушения целостности.

SQL транзакция с уровнем READ UNCOMMITTED

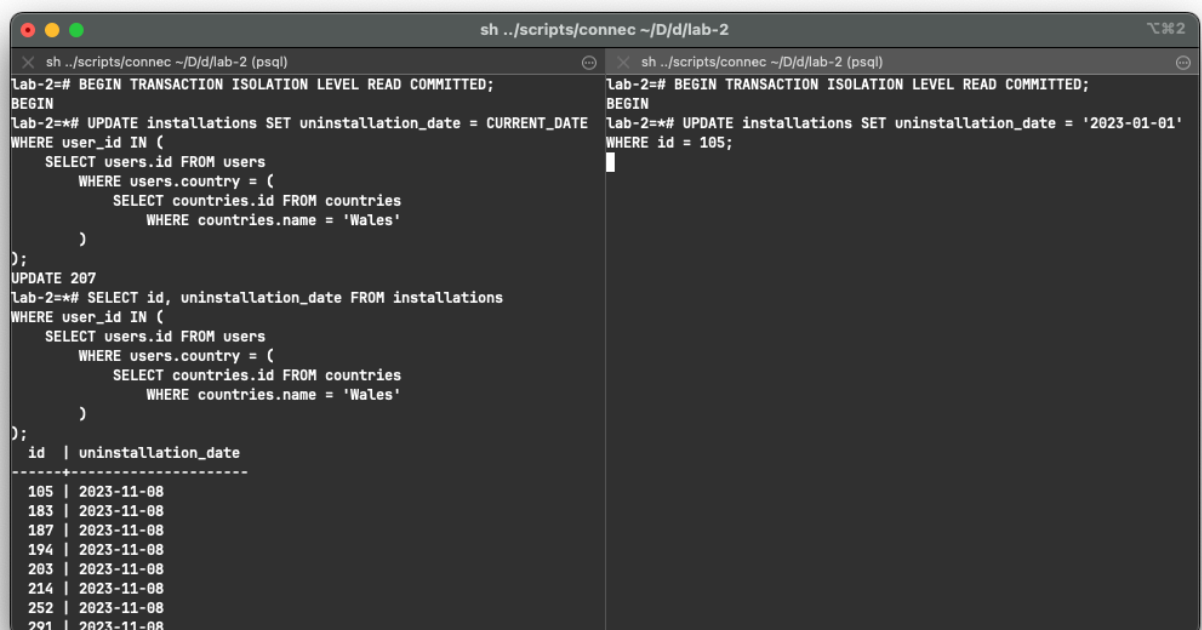


```
sh ../scripts/connec ~/D/d/lab-2 (psql)
lab-2=# BEGIN TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;
BEGIN
lab-2==# UPDATE installations SET uninstallation_date = CURRENT_DATE
lab-2-*# WHERE user_id IN (
lab-2(*## SELECT users.id FROM users
lab-2(*## WHERE users.country = (
lab-2(*## SELECT countries.id FROM countries
lab-2(*## WHERE countries.name = 'Wales'
lab-2(*## )
lab-2(*## );
UPDATE 207
lab-2==# SELECT id, uninstallation_date FROM installations
WHERE user_id IN (
SELECT users.id FROM users
WHERE users.country = (
SELECT countries.id FROM countries
WHERE countries.name = 'Wales'
)
);
id | uninstallation_date
-----+-----
105 | 2023-11-08
183 | 2023-11-08
187 | 2023-11-08
194 | 2023-11-08
203 | 2023-11-08
214 | 2023-11-08
252 | 2023-11-08
291 | 2023-11-08

sh ../scripts/connec ~/D/d/lab-2 (psql)
lab-2=# BEGIN TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;
BEGIN
lab-2==# SELECT id, uninstallation_date FROM installations
WHERE user_id IN (
SELECT users.id FROM users
WHERE users.country = (
SELECT countries.id FROM countries
WHERE countries.name = 'Wales'
)
);
id | uninstallation_date
-----+-----
105 | 2023-01-01
183 | 2023-06-06
187 | 2023-05-24
194 | 2021-08-01
203 | 2023-06-04
214 | 2021-03-23
252 | 2021-11-07
291 | 2021-12-05
352 | 2021-02-18
431 | 2022-04-26
451 | 2021-05-13
504 | 2021-06-16
654 | 2023-01-08
722 | 2020-08-22
874 | 2022-06-10
916 | 2020-04-09
955 | 2020-02-14
```

Рисунок 9 – результат транзакции с READ UNCOMMITTED

SQL транзакция с уровнем READ COMMITTED

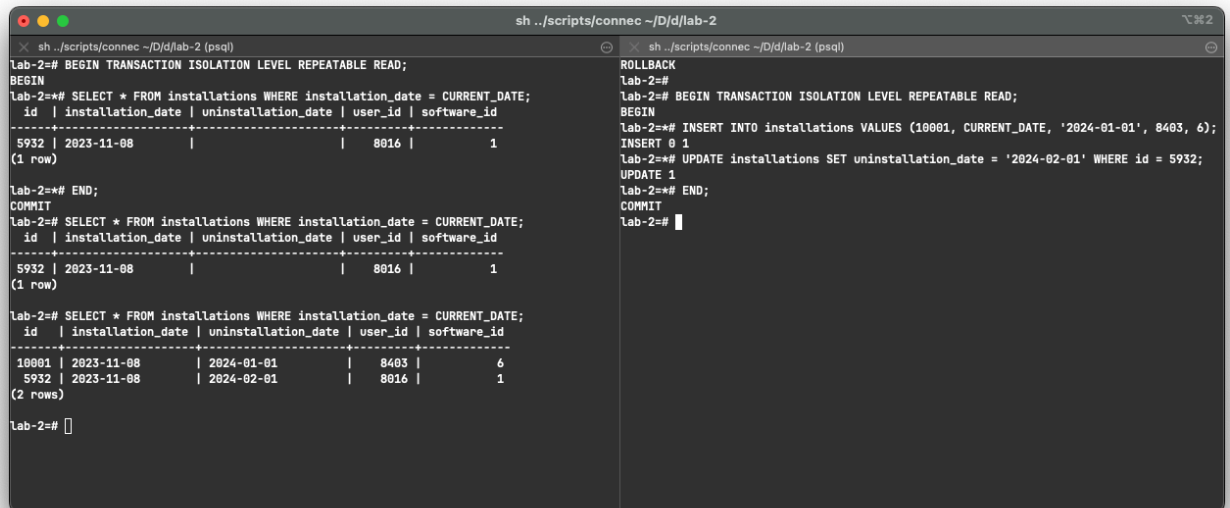


```
sh ../scripts/connec ~/D/d/lab-2 (psql)
lab-2=# BEGIN TRANSACTION ISOLATION LEVEL READ COMMITTED;
BEGIN
lab-2==# UPDATE installations SET uninstallation_date = CURRENT_DATE
WHERE user_id IN (
SELECT users.id FROM users
WHERE users.country = (
SELECT countries.id FROM countries
WHERE countries.name = 'Wales'
)
);
UPDATE 207
lab-2==# SELECT id, uninstallation_date FROM installations
WHERE user_id IN (
SELECT users.id FROM users
WHERE users.country = (
SELECT countries.id FROM countries
WHERE countries.name = 'Wales'
)
);
id | uninstallation_date
-----+-----
105 | 2023-11-08
183 | 2023-11-08
187 | 2023-11-08
194 | 2023-11-08
203 | 2023-11-08
214 | 2023-11-08
252 | 2023-11-08
291 | 2023-11-08

sh ../scripts/connec ~/D/d/lab-2 (psql)
lab-2=# BEGIN TRANSACTION ISOLATION LEVEL READ COMMITTED;
BEGIN
lab-2==# UPDATE installations SET uninstallation_date = '2023-01-01'
WHERE id = 105;
1
```

Рисунок 10 – результат транзакции с READ COMMITTED

SQL транзакция с уровнем REPEATABLE READ



```
sh ../scripts/connec ~/D/d/lab-2 (psql)
Lab-2=# BEGIN TRANSACTION ISOLATION LEVEL REPEATABLE READ;
BEGIN
Lab-2==# SELECT * FROM installations WHERE installation_date = CURRENT_DATE;
 id | installation_date | uninstallation_date | user_id | software_id
-----+-----+-----+-----+-----
5932 | 2023-11-08      |                    |    8016 |          1
(1 row)

Lab-2==# END;
COMMIT
Lab-2=# SELECT * FROM installations WHERE installation_date = CURRENT_DATE;
 id | installation_date | uninstallation_date | user_id | software_id
-----+-----+-----+-----+-----
5932 | 2023-11-08      |                    |    8016 |          1
(1 row)

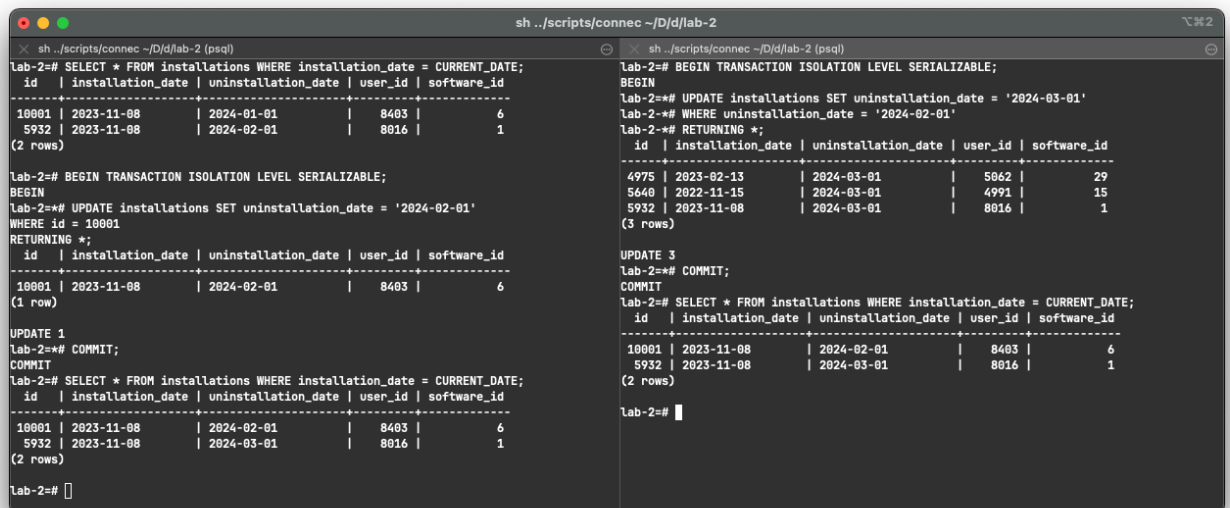
Lab-2=# SELECT * FROM installations WHERE installation_date = CURRENT_DATE;
 id | installation_date | uninstallation_date | user_id | software_id
-----+-----+-----+-----+-----
10001 | 2023-11-08      | 2024-01-01         |    8403 |          6
5932 | 2023-11-08      | 2024-02-01         |    8016 |          1
(2 rows)

Lab-2=#

sh ../scripts/connec ~/D/d/lab-2 (psql)
ROLLBACK
Lab-2=#
Lab-2=# BEGIN TRANSACTION ISOLATION LEVEL REPEATABLE READ;
BEGIN
Lab-2==# INSERT INTO installations VALUES (10001, CURRENT_DATE, '2024-01-01', 8403, 6);
INSERT 0 1
Lab-2==# UPDATE installations SET uninstallation_date = '2024-02-01' WHERE id = 5932;
UPDATE 1
Lab-2==# END;
COMMIT
Lab-2=#
```

Рисунок 11 – результат транзакции с REPEATABLE READ

SQL транзакция с уровнем SERIALIZABLE



```
sh ../scripts/connec ~/D/d/lab-2 (psql)
Lab-2=# SELECT * FROM installations WHERE installation_date = CURRENT_DATE;
 id | installation_date | uninstallation_date | user_id | software_id
-----+-----+-----+-----+-----
10001 | 2023-11-08      | 2024-01-01         |    8403 |          6
5932 | 2023-11-08      | 2024-02-01         |    8016 |          1
(2 rows)

Lab-2=# BEGIN TRANSACTION ISOLATION LEVEL SERIALIZABLE;
BEGIN
Lab-2==# UPDATE installations SET uninstallation_date = '2024-02-01'
WHERE id = 10001
RETURNING *;
 id | installation_date | uninstallation_date | user_id | software_id
-----+-----+-----+-----+-----
10001 | 2023-11-08      | 2024-02-01         |    8403 |          6
(1 row)

UPDATE 1
Lab-2==# COMMIT;
COMMIT
Lab-2=# SELECT * FROM installations WHERE installation_date = CURRENT_DATE;
 id | installation_date | uninstallation_date | user_id | software_id
-----+-----+-----+-----+-----
10001 | 2023-11-08      | 2024-02-01         |    8403 |          6
5932 | 2023-11-08      | 2024-03-01         |    8016 |          1
(2 rows)

Lab-2=#

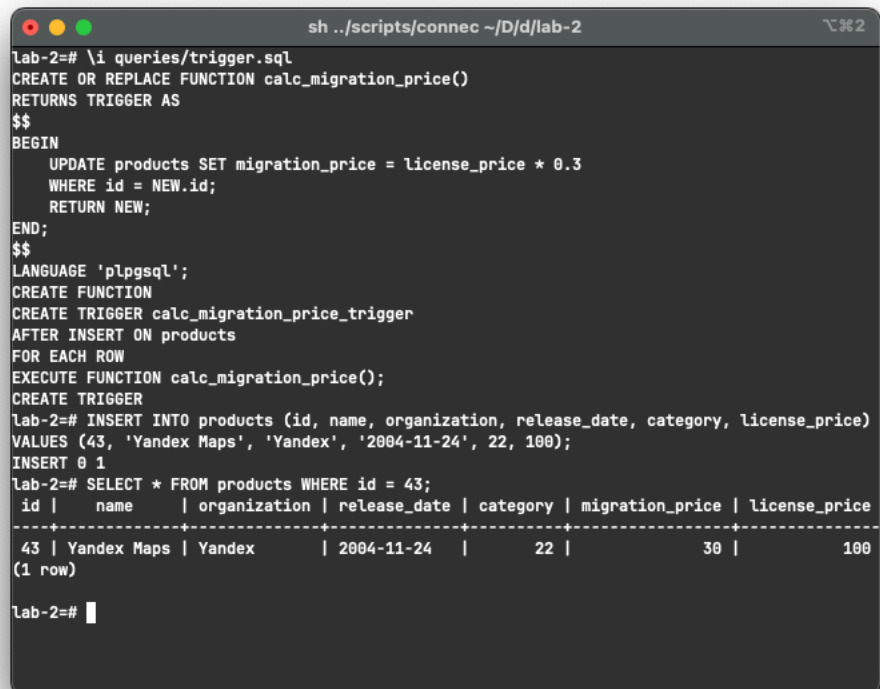
sh ../scripts/connec ~/D/d/lab-2 (psql)
Lab-2=# BEGIN TRANSACTION ISOLATION LEVEL SERIALIZABLE;
BEGIN
Lab-2==# UPDATE installations SET uninstallation_date = '2024-03-01'
WHERE uninstallation_date = '2024-02-01'
RETURNING *;
 id | installation_date | uninstallation_date | user_id | software_id
-----+-----+-----+-----+-----
4975 | 2023-02-13       | 2024-03-01         |    5062 |          29
5640 | 2022-11-15       | 2024-03-01         |    4991 |          15
5932 | 2023-11-08       | 2024-03-01         |    8016 |          1
(3 rows)

UPDATE 3
Lab-2==# COMMIT;
COMMIT
Lab-2=# SELECT * FROM installations WHERE installation_date = CURRENT_DATE;
 id | installation_date | uninstallation_date | user_id | software_id
-----+-----+-----+-----+-----
10001 | 2023-11-08      | 2024-02-01         |    8403 |          6
5932 | 2023-11-08      | 2024-03-01         |    8016 |          1
(2 rows)

Lab-2=#
```

Рисунок 12 – результат транзакции с SERIALIZABLE

SQL триггер



```
sh ../scripts/connec ~/D/d/lab-2
lab-2=# \i queries/trigger.sql
CREATE OR REPLACE FUNCTION calc_migration_price()
RETURNS TRIGGER AS
$$
BEGIN
    UPDATE products SET migration_price = license_price * 0.3
    WHERE id = NEW.id;
    RETURN NEW;
END;
$$
LANGUAGE 'plpgsql';
CREATE FUNCTION
CREATE TRIGGER calc_migration_price_trigger
AFTER INSERT ON products
FOR EACH ROW
EXECUTE FUNCTION calc_migration_price();
CREATE TRIGGER
lab-2=# INSERT INTO products (id, name, organization, release_date, category, license_price)
VALUES (43, 'Yandex Maps', 'Yandex', '2004-11-24', 22, 100);
INSERT 0 1
lab-2=# SELECT * FROM products WHERE id = 43;
 id |  name  | organization | release_date | category | migration_price | license_price
-----+-----+-----+-----+-----+-----+-----
 43 | Yandex Maps | Yandex      | 2004-11-24  |      22 |              30 |          100
(1 row)

lab-2=#
```

Рисунок 13 – результат работы триггера

Вывод

Создана и настроена реляционная база данных; реализован скрипт на python для генерации случайных данных; написаны различные SQL запросы; получены и применены на практике знания о транзакциях и триггерах.