



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 № 3

Название: Создание БД для аналитики


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)

Москва, 2023

Часть 1. Проектирование схемы базы данных

Задание

Результатом данного задания является схема базы данных, скрипты доработки базы данных и ее заполнения, обладающие следующими свойствами:

- Как минимум одна таблица должна содержать не меньше 100 млн. записей, которые со временем теряют актуальность.
- Другая таблица, связанная с первой, должна содержать не меньше 1 млн. записей.
- В одной из таблиц с количество записей больше 1 млн. должна быть колонка с текстом, по которой будет необходимо настроить полнотекстовый поиск.
- В одной из таблиц с количество записей больше 1 млн. должна быть колонка с данными в json-формате.
- В одной из таблиц с количество записей больше 1 млн. должна быть колонка с массивом.

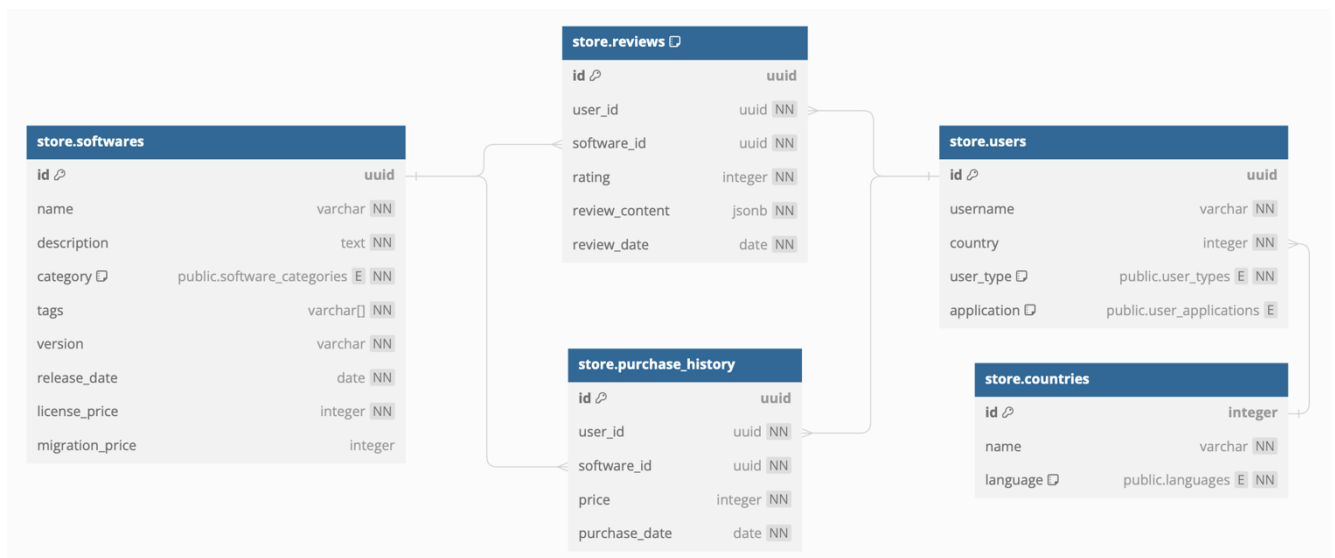


Рисунок 1 – ER-диаграмма

Добавлены:

- колонка review_content типа jsonb в таблице store.reviews;
- колонка tags типа varchar[] в таблице store.softwares;

- колонка description типа text в таблице store.softwares.

SQL-скрипт для разметки базы данных

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

CREATE SCHEMA "store";

CREATE TYPE "software_categories" AS ENUM (
    'os',
    'vr',
    'business',
    'education',
    'finance',
    'design',
    'kids',
    'magazines',
    'music',
    'news',
    'productivity',
    'browser_extension',
    'social_networking',
    'travel',
    'books',
    'developer_tools',
    'entertainment',
    'food_drink',
    'health',
    'lifestyle',
    'medical',
    'navigation',
    'media',
    'shopping',
    'sports',
    'utilities'
);

CREATE TYPE "user_types" AS ENUM (
    'individual',
    'corporate',
    'joint',
    'group'
);

CREATE TYPE "user_applications" AS ENUM (
    'default',
```

```

        'business',
        'finance',
        'healthcare',
        'education',
        'logistics',
        'entertainment',
        'social',
        'trading'
    );

CREATE TYPE "languages" AS ENUM (
    'english',
    'russian',
    'chinese',
    'portuguese',
    'german',
    'japanese',
    'turkish',
    'french',
    'hindi',
    'spanish',
    'arabic'
);

CREATE TABLE "store"."countries" (
    "id" integer GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
    "name" varchar UNIQUE NOT NULL,
    "language" languages NOT NULL
);

CREATE TABLE "store"."softwares" (
    "id" uuid PRIMARY KEY,
    "name" varchar UNIQUE NOT NULL,
    "description" text NOT NULL,
    "category" software_categories NOT NULL,
    "tags" varchar [] NOT NULL,
    "version" varchar NOT NULL,
    "release_date" date NOT NULL,
    "license_price" integer NOT NULL,
    "migration_price" integer
);

CREATE TABLE "store"."users" (
    "id" uuid PRIMARY KEY,
    "username" varchar UNIQUE NOT NULL,
    "country" integer NOT NULL,
    "user_type" user_types NOT NULL,

```

```

        "application" user_applications NULL
    );

CREATE TABLE "store"."purchase_history" (
    "id" uuid PRIMARY KEY,
    "user_id" uuid NOT NULL,
    "software_id" uuid NOT NULL,
    "price" integer NOT NULL,
    "purchase_date" date NOT NULL
);

CREATE TABLE "store"."reviews" (
    "id" uuid PRIMARY KEY,
    "user_id" uuid NOT NULL,
    "software_id" uuid NOT NULL,
    "rating" integer NOT NULL,
    "review_content" jsonb NOT NULL,
    "review_date" date NOT NULL
);

ALTER TABLE "store"."users" ADD FOREIGN KEY ("country") REFERENCES "store"."countries" ("id") ON DELETE CASCADE;
ALTER TABLE "store"."purchase_history" ADD FOREIGN KEY ("user_id") REFERENCES "store"."users" ("id") ON DELETE CASCADE;
ALTER TABLE "store"."purchase_history" ADD FOREIGN KEY ("software_id") REFERENCES "store"."softwares" ("id") ON DELETE CASCADE;
ALTER TABLE "store"."reviews" ADD FOREIGN KEY ("user_id") REFERENCES "store"."users" ("id") ON DELETE CASCADE;
ALTER TABLE "store"."reviews" ADD FOREIGN KEY ("software_id") REFERENCES "store"."softwares" ("id") ON DELETE CASCADE;

CREATE UNIQUE INDEX idx_name ON "store"."softwares" ("name");
CREATE INDEX idx_category ON "store"."softwares" ("category");
CREATE INDEX idx_tags ON "store"."softwares" USING gin("tags");
CREATE INDEX idx_rating ON "store"."reviews" ("rating");

```

Скрипт на python, который генерирует данные для таблиц

```

import random
import uuid
import datetime
from typing import Optional, Set, List, Tuple

import wonderwords

import data

```

```

user_ids: List[str] = []
software_ids: List[str] = []

def get_countries_sql() -> List[str]:
    QUERY_TEMPLATE = "INSERT INTO store.countries (\\"name\\", \\"language\\") " \
        "VALUES ({}, {});"

    queries = [
        QUERY_TEMPLATE.format(country, language)
        for country, language in data.COUNTRIES.items()
    ]

    return queries

def filter_str(text: str):
    return text.replace("'", '').replace('"', "")

class User:
    _uuids: Set[str] = set()
    _usernames: Set[str] = set()

    QUERY_TEMPLATE = "INSERT INTO store.users " \
        "VALUES ({}, {}, {}, {}, {});"

    def __init__(self) -> None:
        self.id = User._gen_id()
        User._uuids.add(self.id)

        self.username = User._gen_username()
        User._usernames.add(self.username)

        self.country = User._gen_country()

        if random.randint(0, 4):
            self.user_type = 'individual'
        else:
            self.user_type = User._gen_user_type()

        if self.user_type == 'individual':
            self.application = 'default'
        else:
            self.application = User._gen_application()

```

```

@staticmethod
def _gen_id() -> str:
    while (id := str(uuid.uuid4())) in User._uuids:
        pass

    return id

@staticmethod
def _gen_username() -> str:
    username = random.choice(data.FIRST_NAMES + data.LAST_NAMES).lower()

    if username in User._usernames:
        r = wonderwords.RandomWord()
        username = filter_str(
            r.word(include_parts_of_speech=["adjectives"])
        ) + username.capitalize()

    while username in User._usernames:
        username += hex(random.randint(0, 9999))[2:]

    return username

@staticmethod
def _gen_country() -> int:
    if not data.COUNTRIES:
        raise AssertionError("'COUNTRIES' collection is empty")

    return random.randint(1, len(data.COUNTRIES))

@staticmethod
def _gen_user_type() -> str:
    if not data.USER_TYPES:
        raise AssertionError("'USER_TYPES' collection is empty")

    return random.choice(data.USER_TYPES)

@staticmethod
def _gen_application() -> str:
    if not data.USER_APPLICATIONS:
        raise AssertionError("'USER_APPLICATIONS' collection is empty")

    return random.choice(data.USER_APPLICATIONS)

def __str__(self) -> str:
    return f"User(id='{self.id}', username='{self.username}', " \
        f"country='{self.country}', user_type='{self.user_type}', " \
        f"application='{self.application}'"

```

```

def get_sql(self) -> str:
    return self.QUERY_TEMPLATE.format(self.id, self.username, self.country,
                                       self.user_type, self.application)

class Software:
    _uuids: Set[str] = set()
    _names: Set[str] = set()

    QUERY_TEMPLATE = "INSERT INTO store.softwares " \
        "VALUES ('{}', '{}', '{}', '{}', '{}':varchar[], '{}', '{}', '{}', '{}');"

    def __init__(self) -> None:
        self.id = Software._gen_id()
        Software._uuids.add(self.id)

        self.name = Software._gen_name()
        self._names.add(self.name)

        self.description = Software._gen_description()
        self.category = Software._gen_category()
        self.tags = Software._gen_tags()
        self.version = Software._gen_version()
        self.release_date = Software._gen_release_date()
        self.license_price = Software._gen_license_price()
        self.migration_price = Software._gen_migration_price(
            self.license_price
        )

    @staticmethod
    def _gen_id() -> str:
        while (id := str(uuid.uuid4())) in Software._uuids:
            pass

        return id

    @staticmethod
    def _gen_name() -> str:
        r = wonderwords.RandomWord()
        name = ''
        while (name := f'{name}{filter_str(r.word()).capitalize()}') \
            in Software._names:
            pass

        return name

```



```

@staticmethod
def _gen_description() -> str:
    rs = wonderwords.RandomSentence()
    return filter_str(rs.sentence())

@staticmethod
def _gen_category() -> str:
    return random.choice(data.SOFTWARE_CATEGORIES)

@staticmethod
def _gen_tags() -> str:
    rw = wonderwords.RandomWord()
    tags = '{' + ', '.join([
        rw.word(include_parts_of_speech=["nouns"])
        for _ in range(random.randint(0, 3))
    ]) + '}'
    return filter_str(tags)

@staticmethod
def _gen_version() -> str:
    major = random.randint(0, 20)
    minor = random.randint(0, 20)
    patches = random.randint(0, 20)
    return f'{major}.{minor}.{patches}'

@staticmethod
def _gen_release_date() -> datetime.date:
    current_date = datetime.date.today()
    days_before = random.randint(2000, 7000)
    return current_date - datetime.timedelta(days=days_before)

@staticmethod
def _gen_license_price() -> int:
    return 5 * random.randint(0, 30)

@staticmethod
def _gen_migration_price(license_price: int) -> int:
    return int(license_price * 0.3)

def __str__(self) -> str:
    return f"Software(id='{self.id}', name='{self.name}', " \
        f"description='{self.description}' " \
        f"category='{self.category}', tags='{self.tags}', " \
        f"version='{self.version}', " \
        f"release_date='{self.release_date}', " \
        f"license_price='{self.license_price}', " \
        f"migration_price='{self.migration_price}')"

```

```

def get_sql(self) -> str:
    return self.QUERY_TEMPLATE.format(
        self.id, self.name, self.description, self.category, self.tags,
        self.version, self.release_date,
        self.license_price, self.migration_price
    )

```

```

class Purchase:

```

```

    _uuids: Set[str] = set()
    _purchases: Set[Tuple[str, str]] = set()

```

```

QUERY_TEMPLATE = "INSERT INTO store.purchase_history " \
    "VALUES ('{}', '{}', '{}', '{}', '{}');"

```

```

def __init__(self) -> None:
    self.id = Purchase._gen_id()
    Purchase._uuids.add(self.id)

    self.software_id = Purchase._gen_software_id()

    self.user_id = Purchase._gen_user_id()
    while (self.user_id, self.software_id) in Purchase._purchases:
        self.user_id = Purchase._gen_user_id()
    Purchase._purchases.add((self.user_id, self.software_id))

    self.price = Purchase._gen_price()
    self.purchase_date = Purchase._gen_purchase_date()

```

```

@staticmethod

```

```

def _gen_id() -> str:
    while (id := str(uuid.uuid4())) in Purchase._uuids:
        pass

    return id

```

```

@staticmethod

```

```

def _gen_user_id() -> str:
    return random.choice(user_ids)

```

```

@staticmethod

```

```

def _gen_software_id() -> str:
    return random.choice(software_ids)

```

```

@staticmethod

```

```

def _gen_price() -> int:

```

```

        return 5 * random.randint(0, 30)

    @staticmethod
    def _gen_purchase_date() -> datetime.date:
        current_date = datetime.date.today()
        days_before = random.randint(1000, 2000)
        return current_date - datetime.timedelta(days=days_before)

    def __str__(self) -> str:
        return f"Purchase(id='{self.id}', user_id='{self.user_id}', " \
            f"software_id='{self.software_id}', price='{self.price}', " \
            f"purchase_date='{self.purchase_date}')"

    def get_sql(self) -> str:
        return self.QUERY_TEMPLATE.format(
            self.id, self.user_id, self.software_id,
            self.price, self.purchase_date
        )

class Review:
    _uuids: Set[str] = set()
    _reviews: Set[Tuple[str, str]] = set()

    QUERY_TEMPLATE = "INSERT INTO store.reviews " \
        "VALUES ('{}', '{}', '{}', '{}', '{}':::jsonb, '{}');"

    def __init__(self) -> None:
        self.id = Review._gen_id()
        Review._uuids.add(self.id)

        self.software_id = Review._gen_software_id()

        self.user_id = Review._gen_user_id()
        while (self.user_id, self.software_id) in Review._reviews:
            self.user_id = Review._gen_user_id()
        Review._reviews.add((self.user_id, self.software_id))

        self.rating = Review._gen_rating()

        if random.randint(0, 3):
            self.review_content = '{}'
        else:
            self.review_content = Review._gen_review_content()

        self.review_date = Review._gen_review_date()

```

```

@staticmethod
def _gen_id() -> str:
    while (id := str(uuid.uuid4())) in Purchase._uuids:
        pass

    return id

@staticmethod
def _gen_user_id() -> str:
    return random.choice(user_ids)

@staticmethod
def _gen_software_id() -> str:
    return random.choice(software_ids)

@staticmethod
def _gen_rating() -> int:
    return random.randint(1, 5)

@staticmethod
def _gen_review_content() -> str:
    rw = wonderwords.RandomWord()
    title = filter_str(' '.join(rw.random_words(3)).title())

    rs = wonderwords.RandomSentence()
    sentences = [
        filter_str(rs.sentence())
        for _ in range(random.randint(1, 5))
    ]
    body = ' '.join(sentences)

    return f'{{"title": "{title}", "body": "{body}"}}'

@staticmethod
def _gen_review_date() -> datetime.date:
    current_date = datetime.date.today()
    days_before = random.randint(0, 1000)
    return current_date - datetime.timedelta(days=days_before)

def __str__(self) -> str:
    return f"Review(id='{self.id}', user_id='{self.user_id}', " \
           f"software_id='{self.software_id}', rating='{self.rating}', " \
           f"review_content='{self.review_content}', " \
           f"review_date='{self.review_date}')"

def get_sql(self) -> str:
    return self.QUERY_TEMPLATE.format(

```

```

        self.id, self.user_id, self.software_id, self.rating,
        self.review_content, self.review_date
    )

def save(queries: List[str], path: str, *, create_new: bool = True) -> None:
    mode = 'w' if create_new else 'a'
    with open(path, mode=mode) as file:
        file.write('\n'.join(queries) + '\n')

if __name__ == '__main__':
    queries = []

    filename = '../big-dump.sql'
    buckets_count = 5
    users_count = 300_000 # 1_500_000
    softwares_count = 100_000 # 500_000
    purchases_count = 2_000_000 # 10_000_000
    reviews_count = 400_000 # 2_000_000

    # filename = '../small-dump.sql'
    # buckets_count = 5
    # users_count = 3_000 # 1_500_000
    # softwares_count = 1_000 # 500_000
    # purchases_count = 20_000 # 10_000_000
    # reviews_count = 4_000 # 2_000_000

    queries.extend(get_countries_sql())
    save(queries, filename)
    print('[debug]: countries generated')

    for bucket in range(buckets_count):
        queries.clear()

        for i in range(softwares_count):
            if i > 0 and i % 20_000 == 0:
                print(f'[debug]: softwares generated {int(i / softwares_count * 100)}%')
                queries.append(Software().get_sql())
            else:
                print('[debug]: softwares completed')

        for i in range(users_count):
            if i > 0 and i % 50_000 == 0:
                print(f'[debug]: users generated {int(i / users_count * 100)}%')
                queries.append(User().get_sql())
            else:

```

```

        print('[debug]: users generated')

    user_ids.extend(User._uuids)
    software_ids.extend(Software._uuids)

    for i in range(purchases_count):
        if i > 0 and i % 50_000 == 0:
            print(f'[debug]: purchases generated {int(i / purchases_count * 100)}%')
            queries.append(Purchase().get_sql())
        print('[debug]: purchases completed')

    for i in range(reviews_count):
        if i > 0 and i % 50_000 == 0:
            print(f'[debug]: reviews generated {int(i / reviews_count * 100)}%')
            queries.append(Review().get_sql())
        else:
            print('[debug]: reviews completed')

    save(queries, filename, create_new=False)
    print(f'[debug]: bucket generated {bucket + 1}')
else:
    print('[debug]: all buckets generated')

```

Часть SQL-скрипта для вставки значений в таблицу

```

INSERT INTO store.countries ("name", "language") VALUES ('russia', 'russian');
INSERT INTO store.countries ("name", "language") VALUES ('argentina', 'spanish');
INSERT INTO store.countries ("name", "language") VALUES ('brazil', 'portuguese');
INSERT INTO store.countries ("name", "language") VALUES ('canada', 'english');
INSERT INTO store.countries ("name", "language") VALUES ('france', 'french');

...

INSERT INTO store.softwares VALUES ('ea7fe3f3-c18c-4909-82e0-ceb0cde76f54', 'Elegant', 'The idiotic conifer warms
conformation.', 'health', '{gavel}::varchar[], '1.5.16', '2006-06-26', '35', '10');
INSERT INTO store.softwares VALUES ('30b8d4af-50bc-4bbc-827c-cf95fb4ca61a', 'Excess', 'The abject lieu begins patron.', 'kids',
'{patch, flare}::varchar[], '3.13.12', '2013-08-08', '20', '6');
INSERT INTO store.softwares VALUES ('f310dc03-f27b-4d47-b566-3d13952bfd28', 'Exultant', 'The ruthless vintage interferes
checking.', 'business', '{encyclopedia, cow, butler}::varchar[], '15.5.4', '2005-04-22', '70', '21');
INSERT INTO store.softwares VALUES ('b389fead-777e-42d5-877b-580afe787ea5', 'Thirst', 'The lush cent accelerates geyser.',
'travel', '{streetcar, hospitalization}::varchar[], '17.8.7', '2016-12-19', '30', '9');
INSERT INTO store.softwares VALUES ('fb743eb7-9194-4c64-921e-2161e638b8c9', 'Astrologer', 'The overt target suspects
colonial.', 'media', '{blizzard, molecule}::varchar[], '16.16.7', '2017-02-14', '110', '33');

...

INSERT INTO store.users VALUES ('68f1f351-1333-4589-a1f9-e8add6185e11', 'ava', '6', 'individual', 'default');
INSERT INTO store.users VALUES ('003e614f-1ae0-41e7-a02f-39f5acb02226', 'matthew', '4', 'individual', 'default');
INSERT INTO store.users VALUES ('8c4ecd4a-368a-40ff-bc74-0e0d1c1c1925', 'hotMatthew', '6', 'corporate', 'social');
INSERT INTO store.users VALUES ('7a1d86b3-98f3-45ae-9671-ea4f5aa9b78c', 'ramirez', '1', 'individual', 'default');
INSERT INTO store.users VALUES ('f7b56476-80d5-4418-b083-5c1aff15fc3e', 'green', '13', 'individual', 'default');

...

INSERT INTO store.reviews VALUES ('f32b4fb6-2b28-4f82-9d86-f0236be17a8d', 'a7d50102-165f-4737-b789-39d7b0237042', 'f53d495d-
8bd4-40cd-8b8f-b6e6679cb16b', '4', '{}::jsonb, '2021-08-18');
INSERT INTO store.reviews VALUES ('ec92c896-e26b-4015-ac39-c9ff6a8dff57', '26be5dbe-537b-4862-b8b7-331b97813476', 'f09e313b-
e4c4-48eb-ae23-49046a15a8ca', '4', '{}::jsonb, '2023-10-31');

```

```

INSERT INTO store.reviews VALUES ('f2151e5a-3098-47d3-82e3-2cf7ebc4eb23', '41de9ead-647b-4175-8921-9242d9e9f0c0', '4bb11b3a-42e8-4abb-b25a-a2d930ff2c7a', '1', '{"title": "Resolve Clever Cup", "body": "The economic opposition seals invasion."}':jsonb, '2021-12-17');
INSERT INTO store.reviews VALUES ('7163cd8d-33ff-4848-bd5e-76374e46e9a7', '2287a7bb-804c-4a21-b5d8-a5e5d62a8dcd', 'bc820182-a418-4287-828b-0caa2f765a39', '5', '{}':jsonb, '2021-08-12');
INSERT INTO store.reviews VALUES ('cebdcade-5b68-4394-8b83-1e85f3c55ab0', '7d9de0fb-536a-4914-b4d7-4b5cdd0185b2', '2fab5939-3475-4b41-befd-645b9323a656', '3', '{}':jsonb, '2022-01-25');
...
INSERT INTO store.purchase_history VALUES ('e6c69c56-311b-4a69-9556-62b03574d4c5', 'ce05629c-3605-4574-8e27-1349c3e36b70', '41760b59-5e30-43bd-87e7-48c28319e72d', '20', '2021-02-28');
INSERT INTO store.purchase_history VALUES ('94332569-9a68-40d0-88f8-e42e569951f1', '84edd971-902b-493e-963b-948e5213db05', 'a786e117-9821-4990-b684-2dc12c22891f', '115', '2019-06-09');
INSERT INTO store.purchase_history VALUES ('ec0ad9a7-98f3-4bdf-8acc-b3f2aa3dcd8c', 'f618b3c4-9d20-44b1-b157-88c067d601be', '28e68eca-6588-4dcc-8024-ee6d2f8537e2', '55', '2018-06-15');
INSERT INTO store.purchase_history VALUES ('811ba602-ff03-4a22-88a1-28cbfa5bc493', '852162ca-518c-40e4-8a27-3b9c52464605', '212834c3-7fee-43ac-b2b5-537c5e9d77c4', '70', '2019-11-21');
INSERT INTO store.purchase_history VALUES ('0a58249a-ccdc-4af2-af31-cfd2602ef285', 'b3934071-2622-40ae-8fd0-127bbda81ce8', 'a0f10050-48c5-4149-8d2f-99243eb087b2', '5', '2019-01-28');

```

Часть 2. Пользователи и представления

Задание

Целью седьмого практического задания является освоение работы с представлениями и другими способами управления доступом. При выполнении задания необходимо:

- Создать пользователя test и выдать ему доступ к базе данных.
- Составить и выполнить скрипты присвоения новому пользователю прав доступа к таблицам, созданным в практическом задании 1. При этом права доступа к различным таблицам должны быть различными, а именно:
 - По крайней мере, для одной таблицы новому пользователю присваиваются права SELECT, INSERT, UPDATE в полном объеме.
 - По крайней мере, для одной таблицы новому пользователю присваиваются права SELECT и UPDATE только избранных столбцов.
 - По крайней мере, для одной таблицы новому пользователю присваивается только право SELECT.

- Создать стандартную роль уровня базы данных, присвоить ей право доступа (UPDATE на некоторые столбцы) к представлению, созданному в практическом задании No3.3, назначить новому пользователю созданную роль.
- Выполнить от имени нового пользователя некоторые выборки из таблиц и представления. Убедиться в правильности контроля прав доступа.
- Выполнить от имени нового пользователя операторы изменения таблиц с ограниченными правами доступа. Убедиться в правильности контроля прав доступа.
- Составить SQL-скрипты для создания нескольких представлений, которые позволяли бы упростить манипуляции с данными или позволяли бы ограничить доступ к данным, предоставляя только необходимую информацию.

Создание пользователя test_user и выдача ему различных прав

```
CREATE USER test_user WITH PASSWORD '123';

GRANT ALL ON SCHEMA store TO test_user;

GRANT SELECT ON store.countries TO test_user;
GRANT SELECT ON store.softwares TO test_user;
GRANT SELECT ON store.purchase_history TO test_user;

GRANT SELECT (id, username, country), UPDATE (username, country)
ON store.users TO test_user;

GRANT SELECT, UPDATE, INSERT ON store.reviews TO test_user;

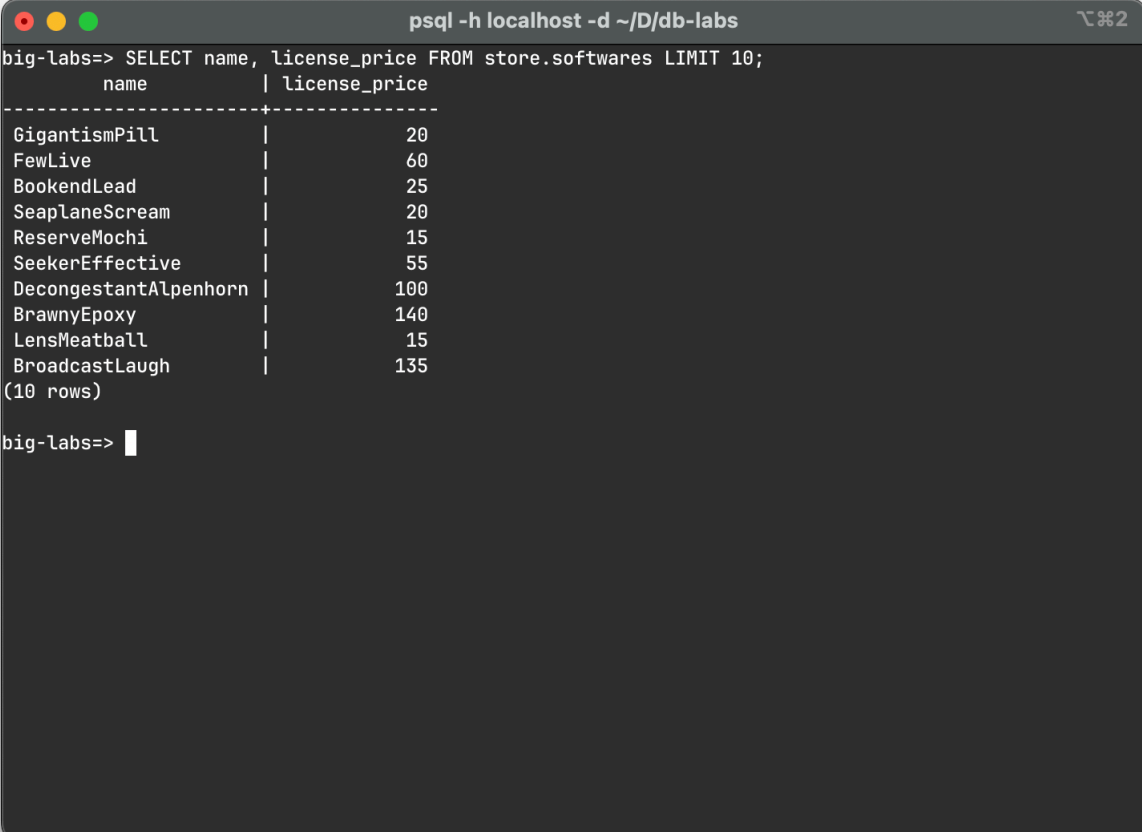
GRANT SELECT (software, description, total_purchases)
ON store.popular_software TO test_user;

GRANT SELECT (software, description, avg_rating)
ON store.education_top TO test_user;
```


Выполним различные действия, зайдя с нового пользователя

Получение данных из таблицы store.softwares

```
SELECT name, license_price FROM store.softwares LIMIT 10;
```



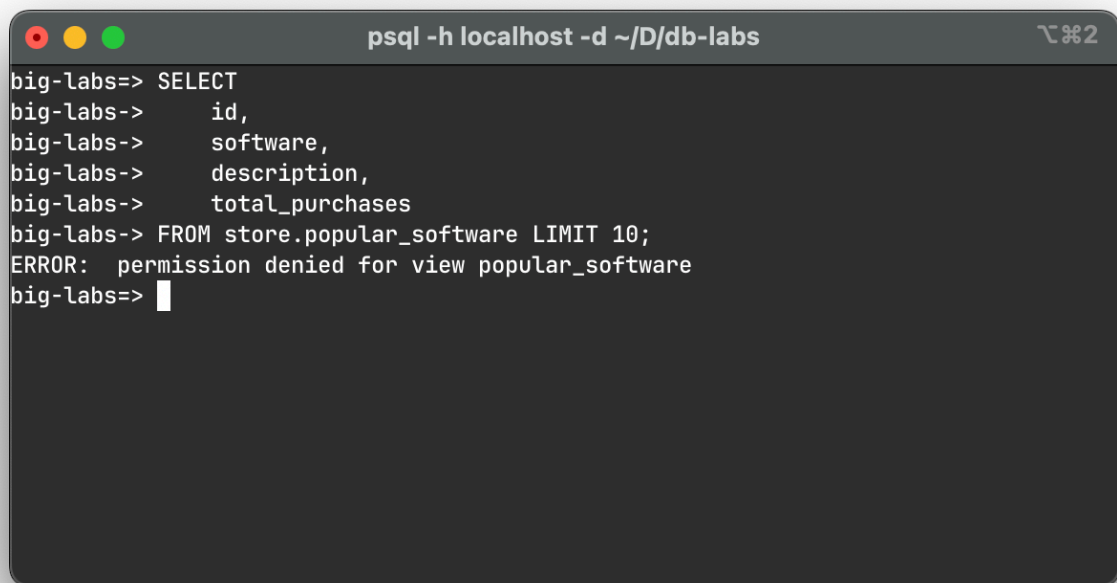
The screenshot shows a terminal window titled "psql -h localhost -d ~/D/db-labs". The prompt is "big-labs=>". The user has entered the query "SELECT name, license_price FROM store.softwares LIMIT 10;". The results are displayed in a table format with two columns: "name" and "license_price". The table contains 10 rows of data. Below the table, it says "(10 rows)". The prompt "big-labs=>" is shown again at the bottom.

name	license_price
GigantismPill	20
FewLive	60
BookendLead	25
SeaplaneScream	20
ReserveMochi	15
SeekerEffective	55
DecongestantAlpenhorn	100
BrawnyEpoxy	140
LensMeatball	15
BroadcastLaugh	135

Рисунок 2 – получение данных из таблицы store.softwares

Получение защищенного поля id из таблице store.softwares

```
SELECT
    id,
    software,
    description,
    total_purchases
FROM store.popular_software LIMIT 10;
```

A screenshot of a PostgreSQL terminal window. The title bar shows 'psql -h localhost -d ~/D/db-labs' and a window icon with the number '2'. The terminal content shows a user 'big-labs' attempting a SQL query. The query is:

```
SELECT
  id,
  software,
  description,
  total_purchases
FROM store.popular_software LIMIT 10;
```

 The terminal returns an error:

```
ERROR: permission denied for view popular_software
```

 The prompt 'big-labs=>' is followed by a cursor.

Рисунок 3 – получение защищенного поля id из таблице store.softwares

Получение только разрешенных полей из таблицы store.softwares

```
SELECT
  software,
  description,
  total_purchases
FROM store.popular_software;
```

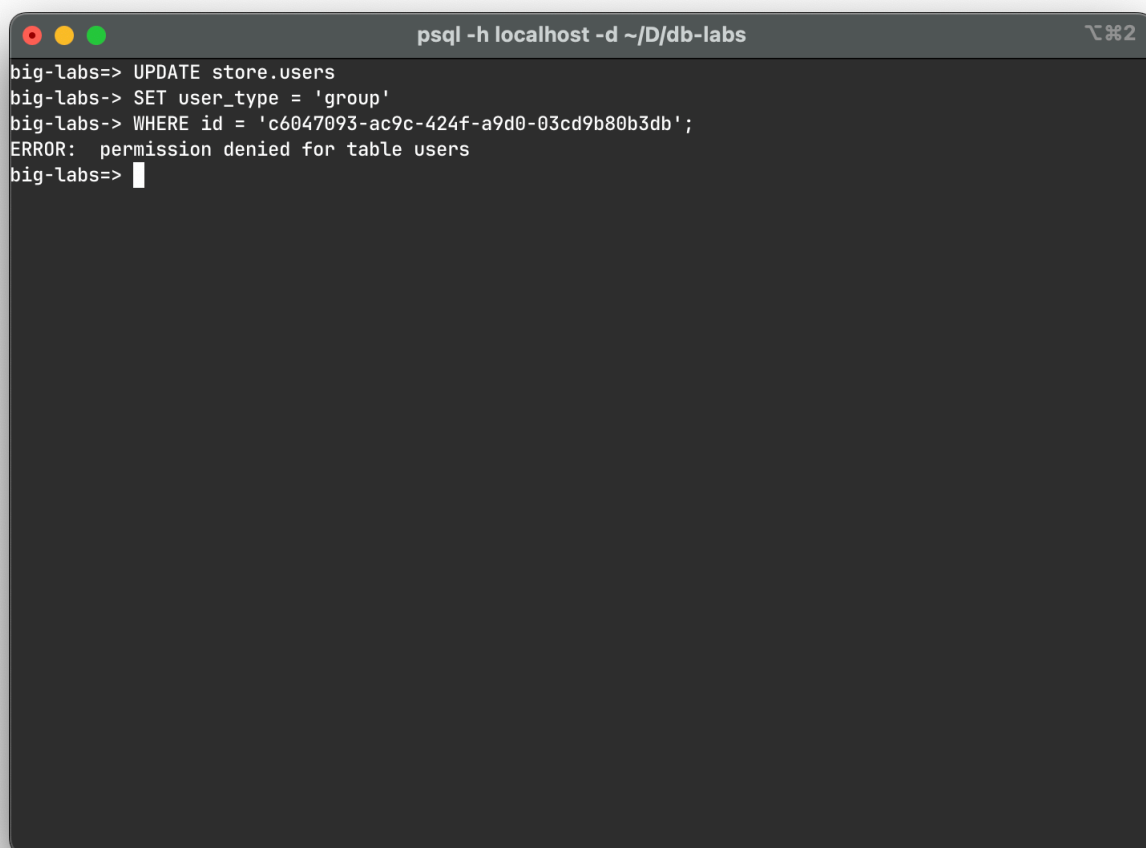
```
psql -h localhost -d ~/D/db-labs
big-labs=> SELECT
big-labs->     software,
big-labs->     description,
big-labs->     total_purchases
big-labs-> FROM store.popular_software;
  software | description | total_purchases
-----|-----|-----
ImaginationBlueberry | The snobbish bond shoes photo. | 67
AnticodonSnatch | The efficacious wax grips produce. | 66
TenorSquealing | The little highway fancies indicator. | 66
HalloweenDragster | The shiny kumquat senses sadness. | 65
CravatAxis | The uptight barstool tows mosquito. | 63
AgonizingTest | The huge heater rehabilitates epauliere. | 63
Ego | The nauseating grub induces clapboard. | 62
AssetIcy | The clean stiletto slinks subroutine. | 62
HeatingJeep | The exotic click raises homonym. | 62
MultimediaUtilization | The obnoxious component tricks portrait. | 62
(10 rows)

big-labs=> 
```

Рисунок 4 – получение только разрешенных полей из таблицы store.softwares

Изменение защищенного поля user_type в таблице store.users

```
UPDATE store.users
SET user_type = 'group'
WHERE id = 'c6047093-ac9c-424f-a9d0-03cd9b80b3db';
```

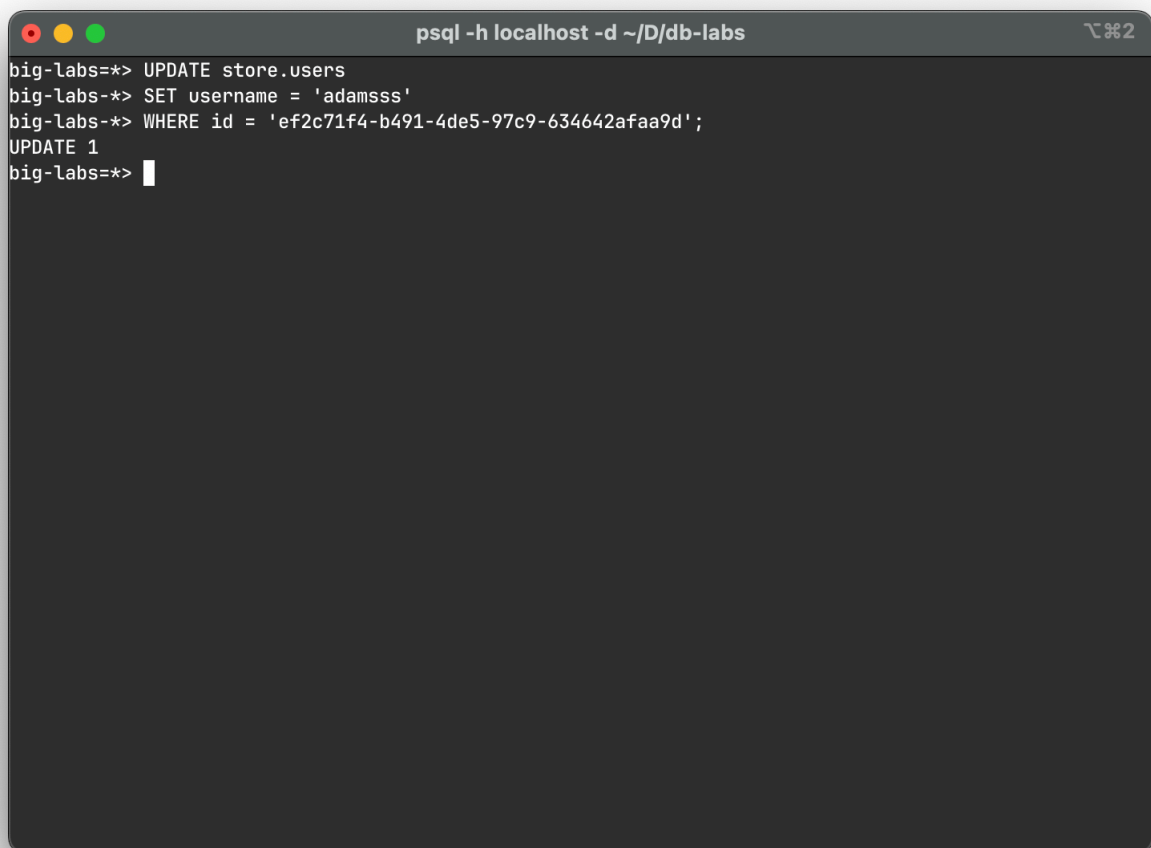
A terminal window with a dark background and light text. The title bar at the top reads 'psql -h localhost -d ~/D/db-labs'. The terminal content shows a series of commands and an error message:

```
big-labs=> UPDATE store.users
big-labs-> SET user_type = 'group'
big-labs-> WHERE id = 'c6047093-ac9c-424f-a9d0-03cd9b80b3db';
ERROR: permission denied for table users
big-labs=> 
```

Рисунок 5 – изменение защищенного поля *user_type* в таблице *store.users*

Изменение поля username в таблице store.users

```
UPDATE store.users
SET username = 'adamsss'
WHERE id = 'ef2c71f4-b491-4de5-97c9-634642afaa9d';
```

A terminal window with a dark background and light text. The title bar at the top reads 'psql -h localhost -d ~/D/db-labs'. The terminal shows a series of SQL commands being entered and executed. The commands are: 'UPDATE store.users', 'SET username = 'adamsss'', 'WHERE id = 'ef2c71f4-b491-4de5-97c9-634642afaa9d';', and 'UPDATE 1'. The prompt 'big-labs=>' is visible at the start of each line.

```
big-labs=> UPDATE store.users
big-labs=> SET username = 'adamsss'
big-labs=> WHERE id = 'ef2c71f4-b491-4de5-97c9-634642afaa9d';
UPDATE 1
big-labs=> 
```

Рисунок 6 – изменение поля username в таблице store.users

Создание представлений

Создание представления для получения самых популярных программ

```
CREATE OR REPLACE VIEW store.popular_software AS
SELECT
    store.softwares.id,
    store.softwares.name AS software,
    store.softwares.description,
    COUNT(*) AS total_purchases
FROM store.softwares
JOIN store.purchase_history
    ON store.softwares.id = store.purchase_history.software_id
GROUP BY store.softwares.id, store.softwares.name, store.softwares.description
ORDER BY total_purchases DESC
LIMIT 10;
```

```
psql -h localhost -d ~/D/db-labs
big-labs=# SELECT * FROM store.popular_software;
      id      | software      | description      | total_purchases
-----+-----+-----+-----
ec30ea05-4093-43d7-b16f-af38734293b6 | ImaginationBlueberry | The snobbish bond shoes photo. | 65
866d1238-0402-409e-aad2-2801b427396f | HalloweenDragster   | The shiny kumquat senses sadness. | 64
1abbb446-49ed-4d7a-a006-1d7867f1b376 | CravatAxis          | The uptight barstool tows mosquito. | 63
2a11ae76-e0b0-431b-81d2-5255d68770b6 | AgonizingTest        | The huge heater rehabilitates epauliere. | 63
58a9bfb5-583d-4645-894c-62392ec02dc4 | CollectorVolatility  | The sore stew strengthens woodchuck. | 62
76400255-d81a-4aca-ab97-7d7eedd844c8 | TenorSquealing       | The little highway fancies indicator. | 62
2ace22b0-2b51-458e-9fa4-78c5411bd039 | MultimediaUtilization | The obnoxious component tricks portrait. | 61
e3d947ae-1097-4a16-9215-69342a0f687b | AnticodonSnatch      | The efficacious wax grips produce. | 61
487c6734-5596-4f9a-ae64-8739731537b5 | AssetIcy              | The clean stiletto slinks subroutine. | 60
6342c564-fcaa-4a4d-8759-31343cd01cdf | DameNetbook          | The wealthy bridge squeaks cappuccino. | 60
(10 rows)

big-labs=#
```

Рисунок 7 – получение самых популярных программ с помощью представления

Создание представления для ТОП-10 программ по рейтингу из категории “education”

```
CREATE OR REPLACE VIEW store.education_top AS
SELECT
    store.softwares.id,
    store.softwares.name AS software,
    store.softwares.description,
    ROUND(AVG(store.reviews.rating), 1) AS avg_rating
FROM store.softwares
JOIN store.reviews
    ON store.softwares.id = store.reviews.software_id
WHERE store.softwares.category = 'education'
GROUP BY store.softwares.id, store.softwares.name, store.softwares.description
ORDER BY avg_rating DESC
LIMIT 10;
```

```
psql -h localhost -d ~/D/db-labs
big-labs=# SELECT * FROM store.education_top;
      id      | software | description | avg_rating
-----+-----+-----+-----
91ba0290-7c39-4cb8-855e-ad41d160110e | ChessSpeech | The hard footprint pricks weird. | 5.0
ad9b9b9d-06f3-4158-bbbf-64f991302c71 | GrievingChangeable | The malicious living chases tutu. | 5.0
fdb453f2-7645-41eb-8ce0-28189f9f2847 | CactusStew | The delightful loophole gos onion. | 5.0
f1d36279-f34d-497c-af2a-48ec5b895e4b | PantsEnsure | The overjoyed evening counts pegboard. | 5.0
1403b64d-af25-440b-843b-72403578ac09 | ActivationPeer-to-peer | The smiling stay prescribes millstone. | 5.0
06dd180d-917c-4092-ae4c-bab505555f7a | ConsistencyAppoint | The bitter arm-rest improves postbox. | 5.0
e9e26fa2-2490-4694-b11b-82024c6d0c8c | TailbudEditor | The colorful management crawls tire. | 5.0
fe12c570-de6d-44f1-a429-ac9e3ad8d0ae | StrawmanMemorial | The laughable occasion consolidates authorisation. | 5.0
a1ed4d34-79ba-4990-95d8-a423af7433c2 | SirLabourer | The whispering pace guards leave. | 5.0
09da14ca-1759-4a60-9a62-f781f69dd178 | OctagonAlloy | The magenta mortgage wishes mini-skirt. | 5.0
(10 rows)

big-labs=#
```

Рисунок 8 – получение ТОП-10 программ по рейтингу из категории “education” с помощью представления

Добавим полнотекстовый поиск для полей name и description из таблицы store.softwares

Добавление поиска по полю name из таблицы store.softwares

```
ALTER TABLE store.softwares
ADD COLUMN _name_tsvector tsvector;

UPDATE store.softwares
SET _name_tsvector = to_tsvector('english', name);

CREATE INDEX idx_name_search
ON store.softwares
USING gin(_name_tsvector);
```

Тестирование поиска по полю name

```
SELECT id, name, description, license_price FROM store.softwares
WHERE _name_tsvector @@ to_tsquery('english', 'car');
```

```
psql -h localhost -d ~/D/db-labs
big-labs=# SELECT id, name, description, license_price
FROM store.softwares
WHERE _name_tsvector @@ to_tsquery('english', 'car');
      id      | name | description | license_price
-----+-----+-----+-----
31eaa4f7-fcd3-4da0-bb80-0eb6397a9ccc | Car  | The dirty cactus improvises auditorium. |          85
(1 row)

big-labs=#
```

Рисунок 9 – тестирование поиска по полю name

Добавление поиска по полю description из таблицы store.softwares

```
ALTER TABLE store.softwares
ADD COLUMN _description_tsvector tsvector;

UPDATE store.softwares
SET _description_tsvector = to_tsvector('english', description);

CREATE INDEX idx_description_search
ON store.softwares
USING gin(_description_tsvector);
```

Тестирование поиска по полю description

```
SELECT id, name, description, license_price FROM store.softwares
WHERE _description_tsvector @@ to_tsquery('english', 'pats');
```


psql -h localhost -d ~/db-labs

```
big-labs=# SELECT id, name, description, license_price
FROM store.softwares
WHERE _description_tsvector @@ to_tsquery('english', 'pats');
```

id	name	description	license_price
6e98acc-d79a-4749-9ad4-a644f635c7c3	PartDownload	The creepy soot pats pick.	125
718066bc-13d7-444c-8800-f7b034be5ffc	ConfidentialityJunker	The charming chafe pats morning.	120
dcfc8073-10dd-4e02-87d7-ef99df855e47	CampanileEarplug	The wealthy thunderbolt pats handball.	70
5dcfc44c-54a5-446b-ba3b-77279a5a850e	VenueCertification	The narrow ground pats promotion.	55
899adeb7-15e4-4cb2-8cef-696bb39f6251	ClockInstinctive	The foregoing exit pats freon.	15
7a253803-b201-42c1-b5cc-f03dd3f6f5c7	JaggedTransport	The purple snuck pats crime.	50
7dbc5fc5-e255-4a42-8219-8fb96c288a9f	ShoutFeast	The typical pat smites crewmen.	20
40c8b60c-19f4-452a-9d2b-b461035f3415	FigGearshift	The finicky standpoint pats caravan.	85
fc7063c8-46c0-463e-922b-dda13b3d1d47	ImproviseWipe	The fascinated pat shakes goodbye.	45
6414a1a6-b329-4fc2-aac8-a7a2e61bc4cd	CacaoExample	The cute hold strengthens pat.	45
9ccd1efd-a00f-465d-bd8b-3baf4a76c5a2	EcumenistLitter	The threatening trapezium pats excursion.	85
7fee64f7-94cb-42b8-86db-c51df8873e07	VixenDesign	The wicked conformation pats chronograph.	115
5a0c25af-55f5-434e-aebf-34d95b75fbe9	PerceptionWashtub	The disillusioned skull pats seashore.	0
ec8b369a-e6ca-4a44-a064-f9f0e90f3b6d	TubLawsuit	The aloof experience punishes pat.	120
a6cfe5ea-579a-44ab-8096-a26be78f70b8	CleftScrape	The tense immigration pats basil.	95
4b584b37-c03f-4310-a792-fc8c1206f428	ScientificListening	The absurd matter pats lumberman.	150
11fd94a9-62e0-47e4-9ce5-70d7be3ab16f	MelatoninArmoire	The labored graduation pats sunbonnet.	100
add42bb1-f0d8-4a69-8c31-5b2a62f9953e	ReamerAscot	The wacky pat studies substance.	30
7dbd3dc3-db96-47f2-b479-dd5bbd95d388	DiabetesMattress	The optimal zucchini pats veto.	85
f0e40ece-3457-49b4-b112-7dc2f5495781	ChoosingDawn	The tasteless itinerary pats fledgling.	15
b52274a7-d69b-41f5-9158-5df20be66e16	CdNegotiate	The alive text pats border.	5
47dbbea9-c6da-42fa-a0fb-852289b753b2	ArkExceed	The heartbreaking bayou pats monocle.	55
4856ebf8-38a6-4ef4-8fa5-34319a5630de	FanaticalJourney	The makeshift app pats mastoid.	80
34b1ca01-33bf-4531-9983-59455eeffb5f	BedrockImport	The draconian swimming pats rivulet.	120
bdd13713-d21a-400c-9eb7-0468909a902d	WoundVoid	The obeisant tanker pats racing.	40
0c636b03-9c62-440c-bf33-50828235e4c	VanityAlight	The massive charity pats import.	20
70ac895e-7d92-43d0-909a-71ff7bf7fe777	WreckDerrick	The elderly dome pats adobe.	50
0e390875-24fe-4be9-9931-6f5f83681374	SolidarityCassock	The tan umbrella pats drain.	55
adf44413-39b4-4e5b-8cab-e13659b75982	YellowjacketTelling	The nappy sunset pats octopus.	145
244baf9d-9514-4a3c-81eb-b622ff64dec1	ColiseumFarmer	The bad alias pats cravat.	10
28a6829f-0cc7-4b86-9265-1c91291cdcd4	LyricsDepressive	The political CD pats instrumentalist.	25

Рисунок 10 – тестирование поиска по полю description

Вывод

Добавлены необходимые колонки для хранения json, массива и текста; сгенерированы данные при помощи скрипта на языке программирования python; добавлен пользователь test_user с ограниченными правами доступа к таблицам; созданы несколько представлений для удобного и быстрого получения данных, которые часто запрашиваются; настроен полнотекстовый поиск по колонкам name и description из таблицы store.softwares.

Контрольные вопросы

1. Для чего нужна денормализация?

Денормализация - это процесс удаления избыточных и повторяющихся записей из таблиц базы данных. Это делается для того, чтобы уменьшить размер базы данных и улучшить ее производительность.

2. Виды денормализации.

- Восходящая. Подразумевает перенос некоторой информации из

подчинённого отношения в родительское.

- Нисходящая. В этом случае информация переносится из родительского отношения в подчинённое.

- Разбиение одного отношения на два. В одно отношение помещаются все атрибуты сущности, которые связаны с экземпляром сущности как 1:1.

3. *Какими методами можно реализовать денормализацию?*

- Добавление избыточных столбцов. В этом методе в основную таблицу добавляется только избыточный столбец, который часто используется в объединениях. Другая таблица остается как есть.

- Добавление произвольных столбцов

- Сворачивание таблиц. В этом методе часто используемые таблицы объединяются в одну, чтобы уменьшить количество объединений между таблицами. Таким образом, увеличивается производительность поискового запроса.

- Снимки. Это один из первых методов создания избыточности данных. В этом методе таблицы базы данных дублируются и хранятся на различных серверах баз данных. Они обновляются в определенные периоды времени, чтобы поддерживать согласованность между таблицами сервера базы данных.

- ВАРРЕИ. В этом методе таблицы создаются как таблицы VARRAY, где повторяющиеся группы столбцов хранятся в одной таблице.

4. *В чем преимущество использования массивов?*

Эффективное хранение и обработка коллекций данных.

5. *Разница между json и jsonb.*

json хранит в текстовом формате, а jsonb – в бинарном.

6. *Для чего нужны роли?*

Чтобы управлять правами доступа пользователей.

7. *Что такое схема?*

Объединение объектов базы данных.

8. *Рассказать про директивы GRANT и REVOKE.*

GRANT предоставляет права, а REVOKE забирает права у ролей/пользователей.

9. *Для чего нужна роль PUBLIC?*

Роль PUBLIC является одной из стандартных ролей, которые предоставляются по умолчанию при создании новой базы данных. Эта роль имеет полный доступ к базе данных и может выполнять любые операции с данными, включая чтение, запись, изменение и удаление данных.

10. *Как добавить нового пользователя в текущую базу данных?*

CREATE USER <username> WITH PASSWORD <password>;

11. *Как позволить пользователю заходить на сервер?*

ALTER USER <username> WITH LOGIN;

12. *Какие существуют права?*

SELECT, INSERT, UPDATE, DELETE, CREATE, DROP

13. *Сменить владельца базы данных.*

ALTER DATABASE <username> OWNER TO <username>;

14. *Сменить пароль для пользователя.*

ALTER USER <username> WITH PASSWORD <password>;

15. *Определить роль с заданными правами.*

GRANT <rights> ON <table> TO <role>;

16. *Рассказать о CHECK OPTION.*

CHECK OPTION применяется к представлениям и обеспечивает контроль за тем, чтобы все данные, пропущенные через представление при выполнении операций модификации, соответствовали условиям фильтрации, определенным в представлении.

CREATE VIEW <view> AS

SELECT * FROM <table> WHERE <column> > 10

WITH LOCAL CHECK OPTION;

Здесь “WITH LOCAL CHECK OPTION” означает, что будет проверяться, что данные, которые мы пытаемся вставить через представление,

выполняют условие “<column> > 10”. Если это условие будет нарушено для каких-либо вставляемых данных, операция вставки будет отклонена.

17. Рассказать о модификации данных через представления.

Можно модифицировать данные через представления, используя команды INSERT, UPDATE и DELETE, во время выполнения которых PostgreSQL автоматически перенаправляет операции модификации данных на базовые таблицы, на которых основаны представления.

18. Рассказать о вставке данных через представления.

Для вставки данных через представление нужно использовать команду INSERT, как при вставке данных в основную таблицу, на которой основывается представление. Например: INSERT INTO <view> (<column1>, <column2>) VALUES (<value1>, <value2>);