Санкт-Петербургский политехнический университет Петра Великого
Институт компьютерных наук и технологий
Кафедра компьютерных систем и программных технологий

ОТЧЕТ
по лабораторной работе

Базы данных

(наименование дисциплины)

Работу выполнил:

43501.3	Волкова М.Д.
группа	Ф.И.О.

Преподаватель:

	Мяснов А.В.
подпись	Ф.И.О.

Санкт-Петербург
2017

1. Цели работ

1. Познакомиться основами проектирования схемы БД, способами организации данных в SQL-БД.
2. Познакомиться с основами проектирования схемы БД, языком описания сущностей и ограничений БД SQL-DDL.
3. Сформировать набор данных, позволяющий производить операции на реальных объемах данных.
4. Познакомить студентов с языком создания запросов управления данными SQL-DML.
5. Познакомить студентов с возможностями реализации более сложной обработки данных на стороне сервера с помощью хранимых процедур.
6. Познакомить студентов с возможностями реализации более сложной обработки данных на стороне сервера с помощью хранимых процедур и триггеров.

2. Программа работы

1.

1.1. Выбрать задание, описать набор данных и требования к хранимым данных в свободном формате в wiki своего проекта в gitlab.
1.2. Сформировать в свободном формате (предпочтительно в виде графической схемы) схему БД, соответствующую заданию (должно получиться не менее 7 таблиц)
1.3. Согласовать с преподавателем схему БД. Обосновать принятые решения и соответствие требованиям выбранного задания.
1.4. Создать проект для работы в gitlab и выложить схему БД в данный проект.
1.5. Продемонстрировать результаты преподавателю

2.

2.1. Самостоятельное изучение SQL-DDL
2.2. Создание скрипта БД в соответствии с согласованной схемой (должны присутствовать первичные и внешние ключи, ограничения на диапазоны значений). Продемонстрировать скрипт преподавателю.
2.3. Создайте скрипт, заполняющий все таблицы БД данными
2.4. Выполнение SQL-запросов, изменяющих схему созданной БД '"по заданию преподавателя"'. Продемонстрировать их работу преподавателю.

3.

3.1. Реализовать в виде программы параметризуемый генератор, который позволит сформировать набор связанных данных в каждой таблице.
3.2. Частные требования к генератору, набору данных и результирующему набору данных:
- количество записей в справочных таблицах должно соответствовать ограничениям предметной области
- количество записей в таблицах, хранящих информацию об объектах или субъектах должно быть параметром генерации
- значения для внешних ключей необходимо брать из связанных таблиц

4.

4.1. Изучите SQL-DML
4.2. Выполните все запросы из списка стандартных запросов. Продемонстрируйте результаты преподавателю.
4.3. Получите у преподавателя и реализуйте SQL-запросы в соответствии с '"индивидуальным"' заданием. Продемонстрируйте результаты преподавателю.
4.4. Выполненные запросы SELECT сохраните в БД в виде представлений, запросы INSERT, UPDATE или DELETE -- в виде ХП. Выложите скрипт в GitLab.

5.

5.1. Изучить возможности языка PL/pgSQL
5.2. Создать две хранимые процедуры в соответствии с индивидуальным заданием, полученным у преподавателя
5.3. Выложить скрипт с созданными сущностями в репозиторий
5.4. Продемонстрировать результаты преподавателю

6.

6.1. Создать два триггера: один триггер для автоматического заполнения ключевого поля, второй триггер для контроля целостности данных в подчиненной таблице при удалении/изменении записей в главной таблице
6.2. Создать триггер в соответствии с индивидуальным заданием, полученным у преподавателя
6.3. Создать триггер в соответствии с индивидуальным заданием, вызывающий хранимую процедуру
6.4. Выложить скрипт с созданными сущностями в GitLab
6.5. Продемонстрировать результаты преподавателю

3. Техническое задание

Разработать базу данных, которая хранит сведения о фильмах, актерах, наградах и жанрах.

4. Индивидуальные задания

1. Изменение схемы БД
    1. Ввести единообразный учет людей, участвующих в производстве кино.
    2. Добавить возможность участия человека в фильме в заданном качестве: актер, оператор, режиссер и пр.
    3. Добавить учет видов типов кинопремий (разные фестивали и номинации) и получения фильмами премий.

2. Запросы к БД
    1. Вывести 10 человек, которые получали какие-либо премии за фильмы, в которых участвовали в двух или более качествах (например, актер и режиссер).
    2. Вывести 10 самых успешных деятелей кино (максимальное количество наград) за заданный период времени. При одинаковом количестве наград вторым показателем успешности необходимо считать общее количество фильмов, в которых принималось участие.

3. Хранимые процедуры и триггеры
    1. В виде триггера реализовать проверку: должно быть нельзя дважды добавить одного человека в один фильм в одном качестве.
    2. При суммарном количестве наград более заданного присуждать фильму премию за лучший фильм.

4. Ход работы

Разработаем базу данных из восьми таблиц, связанных между собой и реализующие техническое задание.

Таблица 1. Таблицы базы данных

Название таблицы	Поля таблицы	Типы полей	Описание
movie	movieID	SERIAL	Содержит данные о фильмах
	name	VARCHAR(100)	
	releasyName	INTEGER	
	rating	NUMERIC	
	movieLength	INTEGER	
	description	VARCHAR(300)	
people	peopleID	SERIAL	Содержит данные о людях
	firstName	VARCHAR(50)	
	lastName	VARCHAR(50)	
	birth	DATE	
genres	genresID	SERIAL	Содержит список жанров
	genresName	VARCHAR(40)	
awards	awardsID	SERIAL	Содержит список наград
	name	VARCHAR(100)	
profession	professionID	SERIAL	Содержит список профессий
	name	VARCHAR(70)	
movieGenres	movieGenresID	SERIAL	Содержит связи всех жанров фильмов
	movieID	SERIAL	
	genresID	SERIAL	
movieAwards	movieAwardsID	SERIAL	Содержит связи всех нагарах за фильмы
	movieID	SERIAL	
	awardsID	SERIAL	
moviePeople	moviePeopleID	SERIAL	Содержит связи о наградах и качестве в которой задействован человек в фильме
	movieID	SERIAL	
	peopleID	SERIAL	
	professionID	SERIAL	
	awardsID	SERIAL	

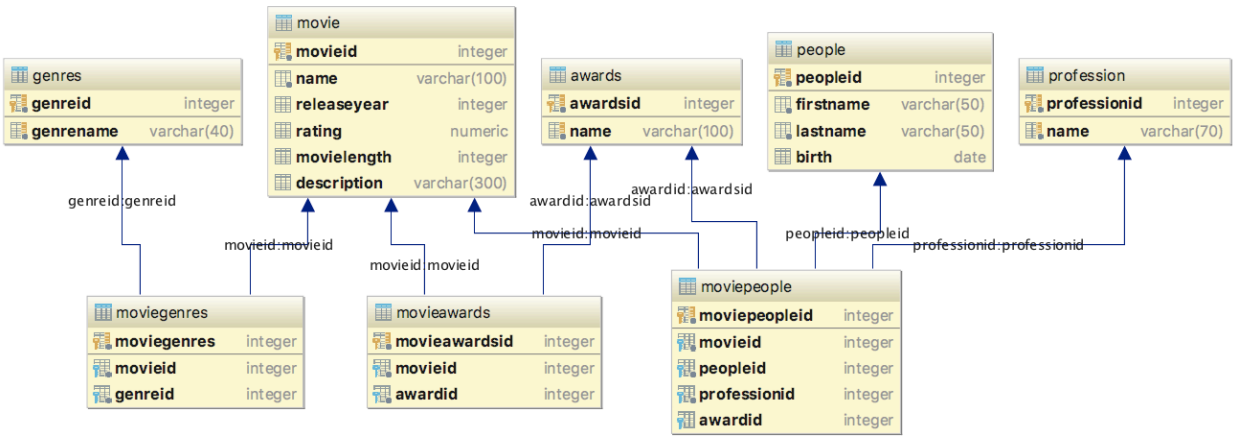

Рис. 1. SQL-диаграмма базы данных

В соответствии с диаграммой базы данных из предыдущей работы (Рис. 1), были написаны SQL-скрипты, создающие таблицы в схеме базы данных (tables.sql на gitlab).

Скрипты создают таблицы в схеме базы данных, если их не было и создает таблицы со связями. Для заполнения таблиц данным, были написаны функции (класс Generator на gitlab).

Позже был внедрен модифицирующий скрипт в таблицы и добавлены скрипты для новых таблицы, где это требовалось индивидуальным заданием №1 “Изменение схемы БД”.

Для генерации тестовых данных было написано приложение, с возможность как генерации тестовых данных, так и самостоятельного добавления необходимых данных. Для генерации были использованы список имен и фамилий, взятых с интернет-ресурсов.

Код с текстом программы можно найти на gitLab-е в папке с Java проектом.

-- выборка всех данных из каждой таблицы

```
TABLE genres;
TABLE movie;
TABLE people;
TABLE movieawards;
TABLE moviepeople;
TABLE moviegenres;
```

--выборка данных из одной таблицы при нескольких условиях, с использованием логических операций, LIKE, BETWEEN, IN

```
SELECT firstname FROM people WHERE lastname LIKE 'Deckow';
SELECT name FROM movie WHERE description LIKE '%war%';
SELECT firstname  FROM people WHERE firstname LIKE 'Al%';
SELECT name FROM movie WHERE rating BETWEEN 3 and 5;
SELECT name FROM movie WHERE movielength BETWEEN 77 and 100;
SELECT people.firstname , people.lastname FROM    people WHERE    birth BETWEEN '1949-06-10' AND '1997-11-28';
```

-- Сделайте выборку всех данных с сортировкой по нескольким полям

```
SELECT * FROM movie ORDER BY name ASC;
SELECT * FROM people ORDER BY birth DESC;
SELECT * FROM movie ORDER BY rating;
```

-- Создайте запрос, вычисляющий несколько совокупных характеристик таблиц

```
SELECT AVG(movie.rating) FROM movie;
SELECT SUM(movie.movielength) FROM movie WHERE rating BETWEEN 5 AND 10;
SELECT MAX(movie.rating) FROM movie;
SELECT MIN(movie.rating) FROM movie;
SELECT COUNT(firstname) FROM people WHERE birth BETWEEN '1990-01-01' AND
'2000-01-01';
```

-- Сделайте выборку данных из связанных таблиц

```
SELECT movie.name , genres.genrename FROM movie, genres WHERE genrename = 'sci-
fi';
SELECT people.firstname , people.lastname FROM movie , people WHERE
movie.description LIKE '%war%';
```

-- С помощью оператора INSERT добавьте в каждую таблицу по одной записи

```
INSERT INTO movie (name, releaseyear, rating, movielength, description) VALUES
('dataBase', '2017', '10', '90', 'bla bla');
INSERT INTO people (firstname, lastname, birth) VALUES ('masha', 'volkova',
'1949-06-10');
INSERT INTO moviepeople (movieid, peopleid) VALUES ('1', '1');
INSERT INTO movieawards (movieid, awardid) VALUES ('1', '1');
INSERT INTO moviegenres (movieid, genreid) VALUES ('1', '1');
```

--измените значения нескольких полей у всех записей, отвечающих заданному условию

```
UPDATE people SET firstname = 'masha' WHERE firstname LIKE 'Bettye';
```

--удалите запись, имеющую максимальное/минимальное значение некоторой совокупной характеристики

```
DELETE FROM movieawards WHERE movieid = (SELECT MAX(movieid) FROM
movieawards);
DELETE FROM movieawards WHERE movieid = (SELECT MIN(movieid) FROM
movieawards);
```

--удалите записи в главной таблице, на которые не ссылается подчиненная таблица

```
DELETE FROM movie WHERE rating = '10';
DELETE FROM movie USING genres WHERE genres.genrename <> ‘Musical';
```

```sql
-- Вывести 10 человек, которые получали какие-либо премии за фильмы,
-- в которых участвовали в двух или более качествах
SELECT moviepeople.peopleid, firstname, lastname, count(moviepeople.professionid) as count_prof
FROM moviepeople
  JOIN people on (people.peopleid = moviepeople.peopleid)
  WHERE awardid is not NULL
GROUP BY moviepeople.movieid , moviepeople.peopleid , firstname,lastname
ORDER BY count_prof DESC
LIMIT 10;


-- Вывести 10 самых успешных деятелей кино (максимальное количество наград) за заданный период времени.
-- При одинаковом количестве наград вторым показателем успешности необходимо считать общее количество фильмов,
-- в которых принималось участие.
SELECT people.peopleid, people.firstname , people.lastname, count (moviepeople.awardid) as award_count ,
  count(moviepeople.movieid) as movie_count
FROM people
  JOIN moviepeople on (people.peopleid = moviepeople.peopleid)
  JOIN movie on (movie.movieid = moviepeople.movieid)
WHERE releaseyear BETWEEN 1917 and 2016
GROUP BY people.peopleid
  ORDER BY award_count DESC , movie_count DESC
LIMIT 10;
```

Рис. 2. Выборки по индивидуальному заданию

	firstname	lastname	count_prof
1	Darian	Kreiger	6
2	Antonia	Champlin	3
3	Audie	Fay	3
4	June	Ziemann	2
5	Mia	Breitenberg	1
6	Liliane	Gleason	1
7	Neal	Johnston	1
8	Kurtis	Kunze	1
9	Kyle	Jerde	1
10	Marcos	Willms	1

Рис. 3. Результат по первому скрипту

	peopleid	firstname	lastname	award_count	movie_count
1	388	Darian	Kreiger	12	12
2	600	Antonia	Champlin	8	8
3	593	Audie	Fay	6	6
4	127	Emmie	Hegmann	6	6
5	373	Rebecca	Cronin	6	6
6	192	Wava	Reinger	5	6
7	319	Kenya	Runolfsdottir	5	5
8	544	Yasmin	O'Connell	5	5
9	520	Mia	Wehner	5	5
10	341	Fatima	Runte	5	5

Рис. 4. Результат по второму скрипту

Проверка в виде триггера на запрет добавления одного и того же человека в один фильм в одном и том же качестве:

```
-- В виде триггера реализовать проверку: должно быть нельзя дважды добавить одного человека в один фильм в одном качестве.
CREATE OR REPLACE FUNCTION func() RETURNS TRIGGER AS $$
BEGIN
    IF (TG_OP = 'INSERT' OR TG_OP = 'UPDATE') THEN
        IF EXISTS (SELECT * FROM moviepeople AS mp WHERE mp.movieid = NEW.movieid AND mp.professionid  = NEW.professionid AND
              mp.peopleid = NEW.peopleid)
          THEN
          RAISE EXCEPTION '┐(￣ヘ￣;)┌';
        END IF;
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

DROP TRIGGER trggr_1 ON moviepeople;
CREATE TRIGGER trggr_1
BEFORE INSERT OR UPDATE ON moviepeople
FOR EACH ROW EXECUTE PROCEDURE func ();
```

Рис. 5. Код триггера

498	273	1041	197	1	2
499	399	885	195	2	3
500	615	1112	195	1	2
501	<default>	1112	195	1	<null>

[P0001] ERROR: !ERORR! ┐(￣ヘ￣;)┌
Где: PL/pgSQL function func() line 7 at RAISE

Рис. 6. Пример работы триггера

Триггер на автоматическое присущдение фильму награды «лучший», при условии, что фильм получил больше 5ти наград.

```
-- При суммарном количестве наград более заданного присуждать фильму премию за лучший фильм.
CREATE OR REPLACE FUNCTION func_2() RETURNS TRIGGER AS $$
BEGIN
    IF EXISTS (SELECT * from movieawards WHERE movieid = new.movieid and movieawards.awardid = 12) THEN RETURN new; END IF ;

      IF ((SELECT COUNT(ma.awardid) FROM movieawards as ma WHERE ma.movieid = NEW.movieid ) > 3)
  THEN
      INSERT INTO movieawards (movieid , awardid) VALUES (new.movieid , 12);
 END IF;
      RETURN NEW;
END;
$$ LANGUAGE plpgsql;

DROP TRIGGER trggr_2 ON movieawards;
CREATE TRIGGER trggr_2
AFTER INSERT OR UPDATE ON movieawards
FOR EACH ROW EXECUTE PROCEDURE func_2 ();
```

Рис. 7. Код триггера

	movieawardsid	movieid ▲1	awardid
1	127	773	3
2	239	773	8
3	167	773	1

Рис. 8. Таблица movieAward

Добавляем четвертую награду фильму с id = 773:

	movieawardsid	movieid ▲1	awardid
1	10029	773	12
2	127	773	3
3	167	773	1
4	239	773	8
5	309	773	4

Рис. 8. Таблица movieAward после срабатывания триггера

Триггер работает корректно.

5. Вывод

В ходе работы, я ознакомилась с основами проектирования баз данных, созданием SQL-диаграмм. База данных приведена к нормальной форме Бойса-Кодда, что дает следующие преимущества:

1. Поддержка целостности, при изменениях базы данных.
2. Экономия пространства, занимаемого базой данных.

Из недостатков нормальной формы Бойса-Кодда выделяют уменьшение производительности запросов из-за необходимости соединения нескольких таблиц.

Учитывая небольшой размер базы данных дальнейшая нормализация не имеет смысла.

Также я ознакомился с языком определения данных SQL-DDL. С его помощью можно определять, изменять и удалять структуры данных. При создании таблиц и заполнением их данными важно делать это в правильном порядке. Это связано со связями между таблицами, которые накладывают некоторые ограничения.

Также я познакомился с языком SQL-DML, который позволяет производить операции над данными, хранящимися в базу.

Операции языка SQL-DML, которые не влияют на данные в таблицах:

1. SELECT – выборка данных из таблицы.
2. JOIN – объединение таблиц для выборки.
3. WHERE – наложение условия на выборку.
4. ORDER BY – сортировка данных.
5. GROUP BY – группировка данных.

Операции языка SQL-DML, которые влияют на данные в таблицах:

1. INSERT – добавление записи в таблицу.
2. UPDATE – обновление данных, находящихся в таблице.
3. DELETE – удаление данных из таблицы.

Использование SQL-DML открывает большие возможности по созданию запросов, а именно объединение таблиц, группировку данных, вложенные запросы, операторы условий и т. д.

Хранимые процедуры позволяют сохранить часто используемые однотипные операции сложной выборки из базы данных.

Использование хранимых процедур позволяет повысить безопасность путем предоставления пользователю доступа только к ним, а не непосредственно к таблицам базы данных. Также стоит отметить уменьшение запросов к серверу, что позволяет экономить сетевой трафик и тратить дополнительное время на передачу по сети.

Триггер можно считать автоматической процедурой, срабатывающей на серверной стороне в результате некоторого события (insert, update, delete). Триггер может сработать до наступления события или после.

Главным преимуществом триггеров является контроль целостности базы данных любой сложности. Также упрощается логика приложения, так как часть логики выполняется на сервере.

Из недостатков триггеров можно выделить: уменьшение производительности системы при большом количестве триггеров, а также рекурсивную модификацию таблиц при неаккуратной реализации.