

САНКТ-ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
ПЕТРА ВЕЛИКОГО

Институт компьютерных наук и технологий
Кафедра компьютерных систем и программных технологий

Отчет по лабораторной работе № 4

Дисциплина: «Базы данных»

Тема: «Язык SQL-DML»

Выполнил студент гр. 43501/3

_____ А.Ю. Ламтев
(подпись)

Преподаватель

_____ А.В. Мяснов
(подпись)

«___» _____ 2019 г.

Санкт-Петербург
2019

Содержание

1	Цели работы	3
2	Программа работы	3
3	Стандартные запросы	3
3.1	Выборка данных из одной таблицы с использованием логических операций, LIKE, BETWEEN, IN	3
3.2	Запрос с вычисляемым полем	4
3.3	Выборка с сортировкой по нескольким полям	5
3.4	Запрос с вычислением совокупных характеристик	5
3.5	Выборка данных из связанных таблиц	6
3.6	Запрос с подзапросами	7
3.7	Запрос с ограничением результата группировки	7
3.8	Добавление записей в таблицы	7
3.9	Изменение значений полей записей, удовлетворяющих условиям	8
3.10	Удаление записей	8
4	Запросы в соответствии с индивидуальным заданием	9
4.1	Запрос 1	9
4.2	Запрос 2	10
5	Сохранение в БД выполненных запросов в виде представлений и ХП	11
6	Выводы	15

1. Цели работы

Познакомиться с языком создания запросов управления данными SQL DML.

2. Программа работы

1. Изучение SQL DML.
2. Выполнение всех запросов из списка стандартных запросов. Демонстрация результатов преподавателю.
3. Получение у преподавателя и реализация SQL-запросов в соответствии с индивидуальным заданием.
4. Демонстрация результатов преподавателю.
5. Сохранение в БД выполненных запросов **SELECT** в виде представлений, запросов **INSERT**, **UPDATE** или **DELETE** — в виде ХП.

3. Стандартные запросы

3.1. Выборка данных из одной таблицы с использованием логических операций, LIKE, BETWEEN, IN

В листинге 1 представлен запрос, формирующий выборку пользователей мужского пола, родившихся до 30 декабря 2001 года, и длина логина которых находится в диапазоне между 7 и 11.

```
1 SELECT *
2 FROM "user"
3 WHERE sex = '1'
4 AND length(login) BETWEEN 7 AND 11
5 AND birthday < '2001-12-30'
```

Листинг 1: like-between-in-1.sql

Выборка, сформированная данным запросом, представлена на рис. 3.1.

	id	login	password_hash	email	birthday	sex	first_name	last_name
1	6671	Dino.Von901	336491	Dino.Von901@email.com	1967-11-08	1	Dino	Von
2	7358	Olin.Toy549	228c58a1	Olin.Toy549@email.com	1957-03-18	1	Olin	Toy

Рис. 3.1: Выборка, сформированная like-between-in-1.sql

В листинге 2 представлен запрос, формирующий выборку пользователей женского пола, логин которых начинается с «Marie.».

```
1 SELECT id, login, email, birthday, sex, first_name, last_name
2 FROM "user"
3 WHERE sex = '0'
4 AND login LIKE 'Marie.%'
```

Листинг 2: like-between-in-2.sql

Выборка, сформированная данным запросом, представлена на рис. 3.2.

	id	login	email	birthday	sex	first_name	last_name
1	4501	Marie.Kovacek14983	Marie.Kovacek14983@email.com	1995-05-29	0	Marie	Kovacek
2	17162	Marie.Leannon2561	Marie.Leannon2561@email.com	2018-08-03	0	Marie	Leannon
3	12895	Marie.Reilly2214	Marie.Reilly2214@email.com	1990-04-23	0	Marie	Reilly
4	14934	Marie.Zemlak1779	Marie.Zemlak1779@email.com	1979-10-10	0	Marie	Zemlak

Рис. 3.2: Выборка, сформированная like-between-in-2.sql

В листинге 3 представлен запрос, формирующий выборку фильмов, цена которых \$5 или \$7, при этом они не являются эпизодами сериалов, они были выпущены в октябре (не важно какого года), и их IMDB рейтинг лежит в диапазоне между 7.5 и 7.6. Поля выборки следующие: идентификатор, стоимость, возраст в годах и рейтинг IMDB.

```

1 SELECT id ,
2       price ,
3       date_part('years' , age(now() , release_date)) AS years_old ,
4       imdb_rating
5 FROM movie
6 WHERE price IN ('$5' , '$7')
7       AND imdb_rating BETWEEN 7.5 AND 7.6
8       AND series_season_id IS NULL
9       AND date_part('month' , release_date) = 10

```

Листинг 3: like-between-in-3.sql

Выборка, сформированная данным запросом, представлена на рис. 3.3.

	id	price	years_old	imdb_rating
1	435	\$7.00	26	7.560517
2	2520	\$7.00	33	7.5580893
3	2780	\$5.00	31	7.5655518
4	5028	\$5.00	5	7.5696015
5	5395	\$7.00	36	7.561081
6	7757	\$5.00	24	7.5056896

Рис. 3.3: Выборка, сформированная like-between-in-3.sql

3.2. Запрос с вычисляемым полем

В листинге 4 представлен запрос, формирующий выборку из 5-ти еще не закончившихся, автоматически возобновляемых подписок, длительность которых равна 30 дням. Поля у выборки следующие: идентификатор подписки, идентификатор пользователя, стоимость и число дней до завершения подписки. При этом последнее поле является вычисляемым.

```

1 SELECT id ,
2       user_id ,
3       payment ,
4       floor(extract(EPOCH FROM age(expires , now())) / 3600 / 24) AS expires_in_days
5 FROM subscription
6 WHERE date_part('days' , age(expires , started)) = 30
7       AND expires > now()
8       AND autorenewable
9 LIMIT 5

```

Листинг 4: calculated-field.sql

Выборка, сформированная данным запросом, представлена на рис. 3.4.

	id	user_id	payment	expires_in_days
1	74	19439	\$5.00	25
2	89	4999	\$12.00	15
3	274	18564	\$65.00	176
4	841	2426	\$45.00	176
5	867	18852	\$5.00	16

Рис. 3.4: Выборка, сформированная `calculated-field.sql`

Значения в столбце `expires_in_days`, превосходящие 30, объясняются тем, что эти подписки еще не вступили в силу.

3.3. Выборка с сортировкой по нескольким полям

В листинге 5 представлен запрос, формирующий выборку из 5-ти сериалов, при этом сериалы отсортированы по возрастанию числа сезонов и по убыванию стоимости.

```
1 SELECT *
2 FROM series
3 ORDER BY seasons ASC, price DESC
4 LIMIT 5
```

Листинг 5: `sorted.sql`

Выборка, сформированная данным запросом, представлена на рис. 3.5.

	id	seasons	price
1	472	1	\$35.00
2	454	1	\$35.00
3	459	1	\$35.00
4	452	1	\$35.00
5	473	1	\$35.00

Рис. 3.5: Выборка, сформированная `sorted.sql`

3.4. Запрос с вычислением совокупных характеристик

В листинге 6 представлен запрос, формирующий одну строку, содержащую общее число пользователей, максимальную длину имени, минимальную длину фамилии, средний возраст пользователей, а также число пользователей мужского пола.

```
1 SELECT count(*) AS users_count,
2        max(length(first_name)) AS longest_firstname,
3        min(length(last_name)) AS shortest_lastname,
4        round(avg(date_part('years', age(now(), birthday)))) AS avg_age,
5        sum(sex::INT) AS male_count
6 FROM "user"
```

Листинг 6: `aggregate.sql`

Выборка, сформированная данным запросом, представлена на рис. 3.6.

	users_count ÷	longest_firstname ÷	shortest_lastname ÷	avg_age ÷	male_count ÷
1	20001	11	3	49	9172

Рис. 3.6: Выборка, сформированная aggregate.sql

3.5. Выборка данных из связанных таблиц

В листинге 7 представлен запрос, в котором соединяются 2 таблицы — **series** (сериалы) и **series_translation** (переводы сериалов) для формирования названия сериала с наибольшим числом сезонов и наибольшей стоимостью.

```

1 SELECT st.name,
2       max(s.seasons) AS max_seasons,
3       max(s.price)   AS max_price
4 FROM series s
5       JOIN series_translation st ON s.id = st.series_id
6 GROUP BY st.name
7 ORDER BY max_seasons DESC, max_price DESC
8 LIMIT 1

```

Листинг 7: join1.sql

Выборка, сформированная данным запросом, представлена на рис. 3.7.

	name ÷	max_seasons ÷	max_price ÷
1	Fear and Trembling	5	\$35.00

Рис. 3.7: Выборка, сформированная join1.sql

В листинге 8 представлен запрос, в котором соединяются 3 таблицы — **movie_translation** (переводы фильмов), **language** (языки) и **user_movie** (пользователи – фильмы) для определения наиболее часто приобретаемого фильма, который не является эпизодом сериала. Выборка состоит из 2-х записей, содержащих идентификатор фильма, имя фильма, локаль и частоту его покупки, для 2-х локалей: русской и английской.

```

1 SELECT mt.movie_id,
2       mt.name,
3       l.name AS locale,
4       count(um.movie_id) AS frequency
5 FROM movie_translation mt
6       JOIN language l ON mt.language_id = l.id
7       JOIN user_movie um ON mt.movie_id = um.movie_id
8 WHERE mt.movie_id IN (SELECT id
9                       FROM movie
10                      WHERE id = mt.movie_id
11                      AND series_season_id IS NULL)
12 GROUP BY mt.movie_id, mt.name, locale
13 ORDER BY frequency DESC
14 LIMIT 2

```

Листинг 8: join2.sql

Выборка, сформированная данным запросом, представлена на рис. 3.8.

	movie_id ÷ name ÷	locale ÷	frequency ÷
1	6285 The Man Within	en-US	7
2	6285 This Lime Tree Bower	ru-RU	7

Рис. 3.8: Выборка, сформированная join2.sql

3.6. Запрос с подзапросами

В листинге 9 представлен запрос, который аналогично предыдущему запросу выводит название фильма, который наиболее часто покупается пользователями, но уже с помощью вложенных подзапросов.

```
1 SELECT mt.movie_id, mt.name, l.name AS locale
2 FROM movie_translation mt
3 JOIN language l ON mt.language_id = l.id
4 WHERE movie_id IN (SELECT movie_id
5                     FROM user_movie
6                     WHERE movie_id IN (SELECT id
7                                       FROM movie
8                                       WHERE id = movie_id
9                                       AND series_season_id IS NULL)
10                                GROUP BY movie_id
11                                ORDER BY count(movie_id) DESC
12                                LIMIT 1
13 )
```

Листинг 9: inner.sql

Выборка, сформированная данным запросом, представлена на рис. 3.9.

	movie_id	name	locale
1	6285	The Man Within	en-US
2	6285	This Lime Tree Bower	ru-RU

Рис. 3.9: Выборка, сформированная inner.sql

Выборка получилась такой же, как и при предыдущем запросе.

3.7. Запрос с ограничением результата группировки

В листинге 10 представлен запрос, находящий пользователей, у которых больше 42 подписок, с помощью ограничения результата группировки по идентификатору пользователя.

```
1 SELECT user_id
2 FROM subscription
3 GROUP BY user_id
4 HAVING count(user_id) > 42
```

Листинг 10: group.sql

Выборка, сформированная данным запросом, представлена на рис. 3.10.

	user_id
1	18339
2	19345
3	18411
4	10309

Рис. 3.10: Выборка, сформированная group.sql

3.8. Добавление записей в таблицы

В листинге 10 представлен запрос, добавляющий записи во все таблицы.

```

1 INSERT INTO language (name)
2 VALUES ('sp-Ar');
3
4 INSERT INTO "user" (login, password_hash, email, birthday, sex, first_name, last_name)
5 VALUES ('null', 'ABCDEFGHIJKLMNOPQRSTUVWXYZ123456', 'email@email.email', '1984-11-12', '0', '
    Firstname', 'Lastname');
6
7 INSERT INTO series (seasons, price)
8 VALUES (1, 25);
9
10 INSERT INTO series_season (series_id, number)
11 VALUES (1, 1);
12
13 INSERT INTO series_translation (series_id, language_id, name, director, description)
14 VALUES (1, 3, 'Silicon Valley', 'Some Director', 'Cool series');
15
16 INSERT INTO series_season_translation (series_season_id, language_id, name, description)
17 VALUES (1, 3, 'First season', 'Very cool season');
18
19 INSERT INTO movie (price, release_date, imdb_rating, series_season_id)
20 VALUES (5, '2018-03-18', 9.0, 1);
21
22 INSERT INTO movie_translation (movie_id, language_id, name, director, description, file_url)
23 VALUES (1, 3, 'Episode 1', 'Some Director', 'First episode', 'https://url.to.video.com/file1')
24 ;
25
26 INSERT INTO user_movie (user_id, movie_id, payment)
27 VALUES (1, 1, 5);
28
29 INSERT INTO subscription (user_id, started, expires, autorenewable, payment)
30 VALUES (1, '2018-12-22', '2019-12-22', FALSE, 15);
31
32 INSERT INTO subscription_movie (subscription_id, movie_id)
33 VALUES (1, 1);
34
35 INSERT INTO subscription_series_season (subscription_id, series_season_id)
36 VALUES (1, 1);
37
38 INSERT INTO category (name, supercategory_id)
39 VALUES ('somegenre', NULL);
40
41 INSERT INTO movie_category (movie_id, category_id)
42 VALUES (1, 1);
43
44 INSERT INTO category_translation (category_id, language_id, translation)
45 VALUES (1, 3, 'somegenre');

```

Листинг 11: insert.sql

3.9. Изменение значений полей записей, удовлетворяющих условиям

В листинге 12 представлен запрос, который в таблице "user" изменяет логин, равный 'null', на значение 'notnull'.

```

1 UPDATE "user"
2 SET login = 'notnull'
3 WHERE login = 'null';

```

Листинг 12: update.sql

3.10. Удаление записей

В листинге 13 представлен запрос, который в таблице movie удаляет все записи, у которых максимальная цена или максимальный рейтинг.


```

1 DELETE
2 FROM movie
3 WHERE price = (SELECT max(price) FROM movie)
4 OR imdb_rating = (SELECT max(imdb_rating) FROM movie)

```

Листинг 13: delete.sql

В листинге 14 представлен запрос, который в таблице `movie` удаляет все записи, для которых в таблице `movie_translation` нет переводов на языки с идентификаторами 1 и 2.

```

1 DELETE
2 FROM movie
3 WHERE id IN (SELECT movie_id
4             FROM movie_translation
5             WHERE language_id NOT IN (1, 2))

```

Листинг 14: delete-inner-select.sql

4. Запросы в соответствии с индивидуальным заданием

4.1. Запрос 1

Вывести пользователей, которые смотрят сериалы больше, чем фильмы.

```

1 SELECT u.id, u.first_name, u.last_name, episodes_count, movie_count
2 FROM (SELECT um.user_id,
3            count(DISTINCT um.movie_id) AS movie_count
4 FROM (SELECT user_id, movie_id
5       FROM user_movie
6       WHERE (SELECT series_season_id
7             FROM movie
8             WHERE id = movie_id) IS NULL
9
10      UNION
11      SELECT s.user_id, sm.movie_id
12      FROM subscription s
13           JOIN subscription_movie sm
14             ON s.id = sm.subscription_id
15      ) AS um
16  GROUP BY um.user_id
17 ) AS user_movie_count
18 FULL JOIN
19 (SELECT ue.user_id, count(DISTINCT ue.series_season_episode_id) AS episodes_count
20 FROM (SELECT us.user_id AS user_id,
21        m.id AS series_season_episode_id
22       FROM user_series us
23           JOIN series_season ss
24             ON us.series_id = ss.series_id
25           JOIN movie m
26             ON ss.id = m.series_season_id
27      UNION
28      SELECT s.user_id, sm.movie_id
29      FROM subscription s
30           JOIN subscription_movie sm
31             ON s.id = sm.subscription_id
32      ) AS ue
33  GROUP BY ue.user_id
34 ) AS user_episode_count
35 ON user_movie_count.user_id = user_episode_count.user_id
36 JOIN "user" u
37   ON user_movie_count.user_id = u.id OR user_episode_count.user_id = u.id
38 GROUP BY u.id, episodes_count, movie_count
39 HAVING episodes_count > movie_count

```

Листинг 15: series-lovers.sql

Запрос логически состоит из 3 частей:

1. формирование выборки (`user_id`, `movie_count`), в которой каждому пользователю сопоставляется общее количество фильмов, которые он купил, либо которые входят в приобретенные им подписки
2. формирование выборки (`user_id`, `episodes_count`), в которой каждому пользователю сопоставляется общее число эпизодов сериалов, которые пользователь купил, или которые входят в состав его подписок
3. соединение двух сформированных выборок и таблицы `user`, сравнение для каждого пользователя значений `movie_count` и `episodes_count`, и на основе этого формирование новой выборки для пользователей, у которых значение `movie_count` < `episodes_count`

Первые 5 записей сформированной выборки представлены на рис. 4.1.

	id	first_name	last_name	episodes_count	movie_count
1	1	Shayne	Bailey	115	81
2	4	Jules	Christiansen	185	50
3	6	Mitchel	Dach	245	45
4	8	Travis	Wolf	370	120
5	9	Esperanza	Jenkins	104	98

Рис. 4.1: Выборка, сформированная `series-lovers.sql`

4.2. Запрос 2

Вывести сериалы, которые от сезона к сезону увеличивали аудиторию.

```

1 SELECT series_increasing_audience.series_id, st.name
2 FROM (WITH series_season_audience AS (SELECT s.id AS series_id,
3 ss.id AS series_season_id,
4 ss.number AS season_number,
5 count(DISTINCT sb.user_id) AS audience
6 FROM series_season ss
7 JOIN series s
8 ON ss.series_id = s.id
9 JOIN subscription_series_season sss
10 ON ss.id = sss.series_season_id
11 JOIN subscription sb
12 ON sss.subscription_id = sb.id
13 GROUP BY s.id, ss.id
14 )
15 SELECT DISTINCT ssa.series_id
16 FROM series_season_audience ssa
17 JOIN (SELECT current_series_id,
18 count(diff) AS positive_diffs_count
19 FROM (SELECT current_series_id AS current_series_id,
20 (next.audience - current.audience) AS diff
21 FROM series_season_audience current
22 JOIN series_season_audience next
23 ON current.series_id = next.series_id
24 AND current.season_number = next.season_number - 1
25 GROUP BY current_series_id, diff
26 ) AS diffs
27 WHERE diff > 0
28 GROUP BY current_series_id) as sd
29 ON ssa.series_id = sd.current_series_id
30 JOIN series
31 ON ssa.series_id = series.id
32 WHERE sd.positive_diffs_count = series.seasons - 1) AS series_increasing_audience

```

```

33 JOIN series_translation st
34 ON series_increasing_audience.series_id = st.series_id
35 WHERE st.language_id = 1

```

Листинг 16: series-increasing-audience.sql

Определение того, что от сезона к сезону происходило увеличение аудитории у сериала, сделано следующим образом:

1. формируем выборку (`series_id`, `series_season_id`, `season_number`, `audience`) — идентификатор сериала, идентификатор сезона сериала, номер сезона и аудитория
2. на основе предыдущей выборки с помощью `SELF JOIN` формируем выборку, в которой для каждого сезона сериала, номер которого больше 1, содержится разность аудитории по сравнению с предыдущим сезоном
3. для каждого сериала считаем количество положительных разностей
4. если это количество = количеству сезонов в сериале минус 1 (т.к. для первого сезона нет предыдущего сезона), то этот сериал нам подходит

Первые 5 записей сформированной выборки представлены на рис. 4.2.

series_id	name
1	2 Edna O'Brien
2	17 Cover Her Face
3	39 Far From the Madding Crowd
4	44 The Last Temptation
5	48 As I Lay Dying

Рис. 4.2: Выборка, сформированная `series-increasing-audience.sql`

5. Сохранение в БД выполненных запросов в виде представлений и ХП

В листинге 17 представлены все ранее написанные `SELECT`-запросы в виде представлений.

```

1 CREATE VIEW like_between_in_1 AS
2 SELECT *
3 FROM "user"
4 WHERE sex = '1'
5 AND length(login) BETWEEN 7 AND 11
6 AND birthday < '2001-12-30';
7
8 CREATE VIEW like_between_in_2 AS
9 SELECT id, login, email, birthday, sex, first_name, last_name
10 FROM "user"
11 WHERE sex = '0'
12 AND login LIKE 'Marie.%';
13
14 CREATE VIEW like_between_in_2 AS
15 SELECT id,
16 price,
17 date_part('years', age(now(), release_date)) AS years_old,
18 imdb_rating

```

```

19 FROM movie
20 WHERE price IN ( '$5', '$7' )
21 AND imdb_rating BETWEEN 7.5 AND 7.6
22 AND series_season_id IS NULL
23 AND date_part('month', release_date) = 10;
24
25
26 CREATE VIEW calculated_fields AS
27 SELECT id,
28        user_id,
29        payment,
30        floor(extract(EPOCH FROM age(expires, now()))) / 3600 / 24) AS expires_in_days
31 FROM subscription
32 WHERE date_part('days', age(expires, started)) = 30
33 AND expires > now()
34 AND autorenewable
35 LIMIT 5;
36
37
38 CREATE VIEW sorted AS
39 SELECT *
40 FROM series
41 ORDER BY seasons ASC, price DESC
42 LIMIT 5;
43
44
45 CREATE VIEW aggregated AS
46 SELECT count(*) AS users_count,
47        max(length(first_name)) AS longest_firstname,
48        min(length(last_name)) AS shortest_lastname,
49        round(avg(date_part('years', age(now(), birthday)))) AS avg_age,
50        sum(sex::INT) AS male_count
51 FROM "user";
52
53
54 CREATE VIEW join1 AS
55 SELECT st.name,
56        max(s.seasons) AS max_seasons,
57        max(s.price) AS max_price
58 FROM series s
59 JOIN series_translation st ON s.id = st.series_id
60 GROUP BY st.name
61 ORDER BY max_seasons DESC, max_price DESC
62 LIMIT 1;
63
64
65 CREATE VIEW join2 AS
66 SELECT mt.movie_id,
67        mt.name,
68        l.name AS locale,
69        count(um.movie_id) AS frequency
70 FROM movie_translation mt
71 JOIN language l ON mt.language_id = l.id
72 JOIN user_movie um ON mt.movie_id = um.movie_id
73 WHERE mt.movie_id in (SELECT id
74                       FROM movie
75                       WHERE id = mt.movie_id
76                       AND series_season_id IS NULL)
77 GROUP BY mt.movie_id, mt.name, locale
78 ORDER BY frequency DESC
79 LIMIT 2;
80
81
82 CREATE VIEW inner_sel AS
83 SELECT mt.movie_id, mt.name, l.name AS locale
84 FROM movie_translation mt
85 JOIN language l ON mt.language_id = l.id
86 WHERE movie_id in (SELECT movie_id
87                   FROM user_movie
88                   WHERE movie_id in (SELECT id
89                                     FROM movie
90                                     WHERE id = movie_id
91                                     AND series_season_id IS NULL)
92                   GROUP BY movie_id
93                   ORDER BY count(movie_id) DESC
94                   LIMIT 1

```

```

95 );
96
97
98 CREATE VIEW grouped AS
99 SELECT user_id
100 FROM subscription
101 GROUP BY user_id
102 HAVING count(user_id) > 42;
103
104
105 CREATE VIEW series_lovers AS
106 SELECT u.id, u.first_name, u.last_name, episodes_count, movie_count
107 FROM (SELECT um.user_id,
108        count(DISTINCT um.movie_id) AS movie_count
109 FROM (SELECT user_id, movie_id
110 FROM user_movie
111 WHERE (SELECT series_season_id
112 FROM movie
113 WHERE id = movie_id) IS NULL
114 UNION
115 SELECT s.user_id, sm.movie_id
116 FROM subscription s
117 JOIN subscription_movie sm
118 ON s.id = sm.subscription_id
119 ) AS um
120 GROUP BY um.user_id
121 ) AS user_movie_count
122 FULL JOIN (SELECT ue.user_id AS user_id,
123        count(DISTINCT ue.series_season_episode_id) AS episodes_count
124 FROM (SELECT us.user_id AS user_id,
125        m.id AS series_season_episode_id
126 FROM user_series us
127 JOIN series_season ss
128 ON us.series_id = ss.series_id
129 JOIN movie m
130 ON ss.id = m.series_season_id
131 UNION
132 SELECT s.user_id, sm.movie_id
133 FROM subscription s
134 JOIN subscription_movie sm
135 ON s.id = sm.subscription_id
136 ) AS ue
137 GROUP BY ue.user_id) AS user_episode_count
138 ON user_movie_count.user_id = user_episode_count.user_id
139 JOIN "user" u
140 ON user_movie_count.user_id = u.id OR user_episode_count.user_id = u.id
141 GROUP BY u.id, episodes_count, movie_count
142 HAVING episodes_count > movie_count;
143
144
145 CREATE VIEW series_increasing_audience AS
146 SELECT series_increasing_audience.series_id, st.name
147 FROM (WITH series_season_audience AS (SELECT s.id AS series_id,
148        ss.id AS series_season_id,
149        ss.number AS season_number,
150        count(DISTINCT sb.user_id) AS audience
151 FROM series_season ss
152 JOIN series s
153 ON ss.series_id = s.id
154 JOIN subscription_series_season sss
155 ON ss.id = sss.series_season_id
156 JOIN subscription sb
157 ON sss.subscription_id = sb.id
158 GROUP BY s.id, ss.id
159 )
160 SELECT DISTINCT ssa.series_id
161 FROM series_season_audience ssa
162 JOIN (SELECT current_series_id,
163        count(diff) AS positive_diffs_count
164 FROM (SELECT current.series_id AS current_series_id,
165        (next.audience - current.audience) AS diff
166 FROM series_season_audience current
167 JOIN series_season_audience next
168 ON current.series_id = next.series_id
169 AND current.season_number = next.season_number - 1
170 GROUP BY current_series_id, diff

```

```

171         ) AS diffs
172         WHERE diff > 0
173         GROUP BY current_series_id) as sd
174     ON ssa.series_id = sd.current_series_id
175 JOIN series
176     ON ssa.series_id = series.id
177 WHERE sd.positive_diffs_count = series.seasons - 1) AS series_increasing_audience
178 JOIN series_translation st
179     ON series_increasing_audience.series_id = st.series_id
180 WHERE st.language_id = 1;

```

Листинг 17: views.sql

В листинге 18 представлены все ранее написанные запросы по добавлению, удалению или обновлению данных в таблицах в виде хранимых процедур.

```

1 CREATE PROCEDURE insert_one_value_to_all_tables()
2     LANGUAGE SQL
3 AS
4 $$
5 INSERT INTO language (name)
6 VALUES ('sp-Ar');
7 INSERT INTO "user" (login, password_hash, email, birthday, sex, first_name, last_name)
8 VALUES ('null', 'ABCDEFGHJKLMNOPQRSTUVWXYZ123456', 'email@email.email', '1984-11-12', '0', '
    Firstname', 'Lastname');
9 INSERT INTO series (seasons, price)
10 VALUES (1, 25);
11 INSERT INTO series_season (series_id, number)
12 VALUES (1, 1);
13 INSERT INTO series_translation (series_id, language_id, name, director, description)
14 VALUES (1, 3, 'Silicon Valley', 'Some Director', 'Cool series');
15 INSERT INTO series_season_translation (series_season_id, language_id, name, description)
16 VALUES (1, 3, 'First season', 'Very cool season');
17 INSERT INTO movie (price, release_date, imdb_rating, series_season_id)
18 VALUES (5, '2018-03-18', 9.0, 1);
19 INSERT INTO movie_translation (movie_id, language_id, name, director, description, file_url)
20 VALUES (1, 3, 'Episode 1', 'Some Director', 'First episode', 'https://url.to.video.com/file1')
21 ;
22 INSERT INTO user_movie (user_id, movie_id, payment)
23 VALUES (1, 1, 5);
24 INSERT INTO subscription (user_id, started, expires, autorenewable, payment)
25 VALUES (1, '2018-12-22', '2019-12-22', FALSE, 15);
26 INSERT INTO subscription_movie (subscription_id, movie_id)
27 VALUES (1, 1);
28 INSERT INTO subscription_series_season (subscription_id, series_season_id)
29 VALUES (1, 1);
30 INSERT INTO category (name, supercategory_id)
31 VALUES ('somegenre', NULL);
32 INSERT INTO movie_category (movie_id, category_id)
33 VALUES (1, 1);
34 INSERT INTO category_translation (category_id, language_id, translation)
35 VALUES (1, 3, 'somegenre');
36 $$;
37 CALL insert_one_value_to_all_tables();
38
39 CREATE PROCEDURE update_login_to_new_login(old_login VARCHAR(50), new_login VARCHAR(50))
40     LANGUAGE SQL
41 AS
42 $$
43 UPDATE "user"
44 SET login = old_login
45 WHERE login = new_login
46 $$;
47 CALL insert_one_value_to_all_tables('null', 'notnull');
48
49
50 CREATE PROCEDURE delete_movie_with_max_price_or_max_rating()
51     LANGUAGE SQL
52 AS
53 $$
54 DELETE
55 FROM movie

```

```

56 WHERE price = (SELECT max(price) FROM movie)
57 OR imdb_rating = (SELECT max(imdb_rating) FROM movie)
58 $$;
59 CALL delete_movie_with_max_price_or_max_rating();
60
61 CREATE PROCEDURE delete_movie_if_it_has_not_got_translations(tr1 INTEGER, tr2 INTEGER)
62 LANGUAGE SQL
63 AS
64 $$
65 DELETE
66 FROM movie
67 WHERE id IN (SELECT movie_id
68              FROM movie_translation
69              WHERE language_id NOT IN (tr1, tr2))
70 $$;
71 CALL delete_movie_if_it_has_not_got_translations(1, 2);

```

Листинг 18: stored-procedures.sql

6. Выводы

В результате работы был изучен синтаксис языка SQL DML, позволяющий формировать запросы выборки данных из таблиц, формировать запросы по вставке, удалению или обновлению записей в таблицах, создавать на основе запросов представления, которые являются виртуальными таблицами, и создавать хранимые процедуры.