Лабораторная работа № 2

Проектирование и создание базы данных на сервере Microsoft SQL Server

***Цель работы*:** получить навыки проектирования и создания баз данных с использованием утилиты SQL Server Management Studio (SSMS).

Лабораторное задание (вариант 25): спроектировать и создать БД для учета заявок, поступающих от слушателей, с просьбой передать музыкальные произведения в радиоэфире.

Ответы на контрольные вопросы:

1. **Из каких компонентов состоит инфологическая модель предметной области?**
   - *Сущность* – любой различимый объект (объект, который мы можем отличить от другого), информацию о котором необходимо хранить в базе данных. Сущностями могут быть люди, места, самолеты, рейсы, вкус, цвет и т.д.
   - *Атрибут* – поименованная характеристика сущности. Его наименование должно быть уникальным для конкретного типа сущности, но может быть одинаковым для различного типа сущностей/
   - *Ключ* – минимальный набор атрибутов, по значениям которых можно однозначно найти требуемый экземпляр сущности. Минимальность означает, что исключение из набора любого атрибута не позволяет идентифицировать сущность по оставшимся.
   - *Связь* – ассоциирование двух или более сущностей. Если бы назначением базы данных было только хранение отдельных, не связанных между собой данных, то ее структура могла бы быть очень простой. Однако одно из основных требований к организации базы данных – это обеспечение возможности отыскания одних сущностей по значениям других, для чего необходимо установить между ними определенные связи. А так как в реальных базах данных нередко содержатся сотни или даже тысячи сущностей, то теоретически между ними может быть установлено более миллиона связей. Наличие такого множества связей и определяет сложность инфологических моделей.

2. **Что представляет собой целостность базы данных и как она обеспечивается?**
   Целостность базы данных – соответствие имеющейся в базе данных информации её внутренней логике, структуре и всем явно заданным правилам. Каждое правило, налагающее некоторое ограничение на возможное состояние базы данных, называется ограничением целостности.
   Ограничение целостности – это специальные средства в СУБД, главное назначение которых — не допустить попадания в базу ошибочных данных, например — тридцатый день в феврале или восьмой день недели.

3. **Какие виды ограничений целостности существуют?**
   Все ограничения целостности можно разделить на четыре категории:
   - *Ограничение на значение столбцов:*

    - ограничение значения: например, мы можем ограничить длину значения [varchar(50)], добавить constraint UNIQUE, явно указав что каждой значение должно быть уникальным, либо явно указать список возможных значений для значения.
    - ограничение типа: множество значений, которые могут принадлежать к этому типа (т.е. если мы попытаемся записать в столбец с типом integer строку “miet”, база данных не даст нам этого сделать)
- *Ссылочная целостность* – обеспечивается системой первичных и внешних ключей. Этими средствами можно гарантировать, что у нас не будет ссылок на несуществующие объекты таблицы. Т.е. если вы попробуете создать запись с внешим ключом, которого не существует в другой таблице, произойдёт ошибка.
- *Доменная целостность* – отвечает за то, чтобы в соответствующих полях базы данных были соответствующие значения. Например, номер телефона, как правило, обозначается цифрами, а имя или фамилия — буквами. В базах данных такая целостность зачастую обеспечивается запретом пустых значений (NOT NULL), триггерами, ключами а так же хранимыми процедурами.
- *Целостность сущностей* – заключается в том, что любое отношение должно обладать первичным ключом или проверкой уникальности. Иными словами, главная задача целостности сущностей — сделать так, чтобы данные об одном объекте (сущности) не попали в базу данных дважды, так как при несоблюдении данного ограничения в базе данных может храниться противоречивая информация об одном объекте. Поддержание целостности сущностей осуществляется СУБД.

4. **Как обеспечить быстрый доступ к данным в проектируемой базе данных?**
   Способов несколько:
    - не денормализовать базу данных слишком сильно, чтобы не приходилось сканировать большое количество таблиц для получения информации об одной бизнес-сущности
    - использовать индексы для полей, по которым чаще всего происходит поиск: зачастую это уникальные идентификаторы и/или конкретные столбцы

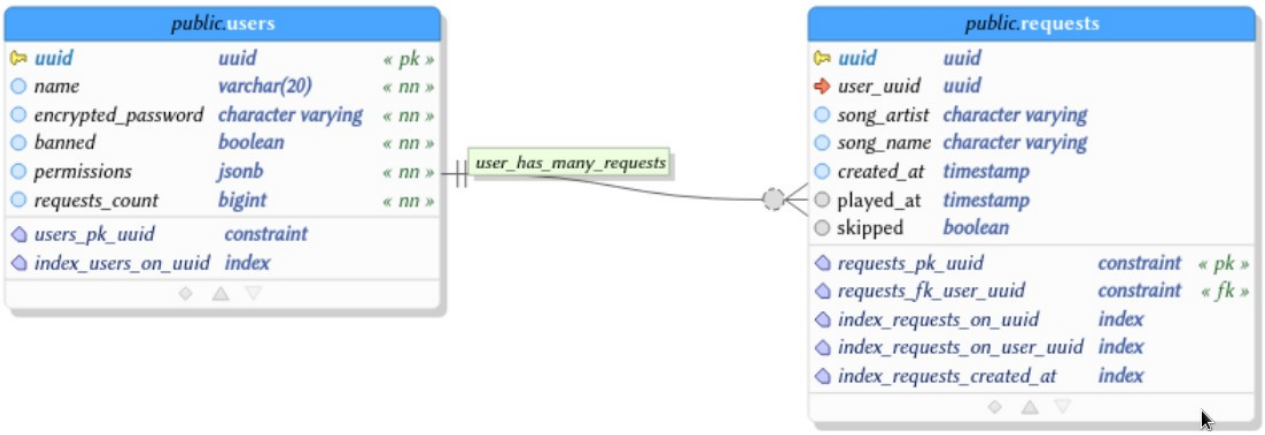
5. **Что представляет собой индексный ключ?**
   Столбец или столбцы которыйые используются для формирования индекса.

6. **Какие виды индексов существуют?**
   (тут специфично для PostgreSQL):
   Классический btree индекс подходит для большинства полей, hash индекс хорошо подходит для индексирования строковых столбцов, gist индекс хорошо подходит для индексирования, например, координат, gin или rum индекс – для полнотекстового поиска

7. **Структура каких таблиц описывается в первую очередь при создании базы данных?**
   Главных бизнес-сущностей: сначала мы должны примерно описать, какие данные должны быть в них, а уже затем проектировать вспомогательные таблицы. Т.е. требуется оперирование некими абстракциями (этакий SICP-like подход, когда мы используем ещё не существующий сущности для решения текущей задачи). В данной лабораторной работе я сначала спроектировал таблицу **requests** (представив, что какая-то табличка **users** уже существует), а уже потом спроектировал таблицу **users**.

Выполнение работы:

1. Ссылка на папку с исходным кодом: https://github.com/osovv/miet-dbms/tree/main/lab_2
2. Спроектированная в **pgModeler** модель базы данных (не знаю, нужно ли было писать туториал по тому, как этим пользоваться [надеюсь, что нет])
   Подробное описание каждой из таблиц есть в пункте 4.



3. Примерный сценарий использования:
   - есть пользователи, у пользователей есть права, хранящиеся в jsonb поле permissions (например `{ “create_requests”: true, “play_requests”: false }` для обычного пользователя и `{ “create_requests”: true, “play_requests”: true }` для ведущего
   - на фронтенде в зависимости от пришедших прав рендерятся страницы: например, если у пользователя нет прав на воспроизведение, он сможет только послушать то, что играет сейчас, либо создать новую заявку, для ведущего появится новая вкладка с возможность выбора какую заявку играть сейчас (или, допустим, поставить в очередь)

   С такой архитектурой ведущий сам должен искать песню, которая стоит следующей в очереди. Я постарался сделать так, чтобы в случае чего эту архитектуру можно было легко расширить (допустим, добавить таблицу **tracks** со всеми песнями, которые можно заказать, и **track_uuid** в **requests**, а пользователю позволять заказывать только их)
4. Выполнение команд в среде **psql**

```
al=# create database dbms_lab_2;
CREATE DATABASE
Time: 71.171 ms
al=# \c dbms_lab_2;
You are now connected to database "dbms_lab_2" as user "al".
dbms_lab_2=# \i db.sql;
CREATE DATABASE
Time: 75.203 ms
CREATE TABLE
Time: 4.810 ms
ALTER TABLE
Time: 0.227 ms
CREATE INDEX
Time: 1.064 ms
CREATE TABLE
Time: 2.754 ms
ALTER TABLE
Time: 0.287 ms
CREATE INDEX
Time: 1.148 ms
CREATE INDEX
Time: 1.045 ms
CREATE INDEX
Time: 1.228 ms
```

```
ALTER TABLE
```

Schema	Name	Type	Owner
public	requests	table	postgres
public	users	table	postgres

```
(2 rows)

dbms_lab_2=*# \d requests;
                        Table "public.requests"
```

Column	Type	Collation	Nullable	Default
uuid	uuid		not null	
user_uuid	uuid		not null	
song_artist	character varying		not null	
song_name	character varying		not null	
created_at	timestamp without time zone		not null	
played_at	timestamp without time zone			
skipped	boolean			false

```
Indexes:
    "requests_pk_uuid" PRIMARY KEY, btree (uuid)
    "index_requests_created_at" btree (created_at)
    "index_requests_on_user_uuid" btree (user_uuid)
    "index_requests_on_uuid" btree (uuid)
Foreign-key constraints:
    "requests_fk_user_uuid" FOREIGN KEY (user_uuid) REFERENCES users(uuid)

dbms_lab_2=*# \d users;
                          Table "public.users"
```

Column	Type	Collation	Nullable	Default
uuid	uuid		not null	
name	character varying(20)		not null	
encrypted_password	character varying		not null	
banned	boolean		not null	false
permissions	jsonb		not null	
requests_count	bigint		not null	

```
Indexes:
    "users_pk_uuid" PRIMARY KEY, btree (uuid)
    "index_users_on_uuid" btree (uuid)
Referenced by:
    TABLE "requests" CONSTRAINT "requests_fk_user_uuid" FOREIGN KEY (user_uuid) REFERENCES

dbms_lab_2=*# \i fill.sql;
INSERT 0 5
Time: 1.255 ms
UPDATE 1
Time: 0.586 ms
INSERT 0 8
Time: 2.791 ms
dbms_lab_2=*# select * from users;
```

uuid	name	encrypted_password	banned	permissions	requests_count
af2ba6b0-88d9-11ec-8683-d7a850d73638	aleksey	fakeuser	f	{"play_requests": false, "create_requests": true}	3
af2baa20-88d9-11ec-8684-275c1c46bef4	dima	fakeuser	f	{"play_requests": false, "create_requests": true}	2
af2baa52-88d9-11ec-8685-47c3c6d6e3a2	dmitriy	fakeuser	f	{"play_requests": true, "create_requests": true}	0
af2baaac-88d9-11ec-8687-e311aff7a11d	tolya	fakeuser	f	{"play_requests": false, "create_requests": true}	2
af2baa7a-88d9-11ec-8686-bf2f74771574	vadim	fakeuser	t	{"play_requests": false, "create_requests": true}	1

```
(5 rows)

Time: 0.687 ms
dbms_lab_2=*# select * from requests;
```

uuid	user_uuid	song_artist	song_name	created_at	played_at	skipped
61e3a4d8-88da-11ec-8c65-f31660403032	af2baa7a-88d9-11ec-8686-bf2f74771574	bad	song	2021-02-08 15:29:25		

```
ø          | t       |
| 61e3a5c8-88da-11ec-8c66-3fbbd018f8b8 | af2baaac-88d9-11ec-8687-e311aff7a11d | БРАВО       | Чудесная страна   | 2021-02-08 15:31:14 |
ø          | f       |
| 61e3a5f0-88da-11ec-8c67-1b52878f7f56 | af2baaac-88d9-11ec-8687-e311aff7a11d | Clairo      | Pretty Girl       | 2021-02-08 15:32:15 |
ø          | f       |
| 61e3a60e-88da-11ec-8c68-7ff0042a20fd | af2ba6b0-88d9-11ec-8683-d7a850d73638 | Morandi     | Angels            | 2021-02-08 15:43:12 |
ø          | f       |
| 61e3a62c-88da-11ec-8c69-fb153e44133e | af2ba6b0-88d9-11ec-8683-d7a850d73638 | Air         | Playground Love   | 2021-02-08 15:45:31 |
ø          | f       |
| 61e3a64a-88da-11ec-8c6a-4b95919023e4 | af2ba6b0-88d9-11ec-8683-d7a850d73638 | Ray Charles | Hit the Road Jack | 2021-02-08 15:47:33 |
ø          | f       |
| 61e3a668-88da-11ec-8c6b-c7b8b46a3de4 | af2baa20-88d9-11ec-8684-275c1c46bef4 | Versake     | Outside           | 2021-02-08 15:55:01 |
ø          | f       |
| 61e3a686-88da-11ec-8c6c-1f72cb0e1e64 | af2baa20-88d9-11ec-8684-275c1c46bef4 | Rodriguez   | Sugar Man         | 2021-02-08 15:56:19 |
ø          | f       |
└──────────────────────────────────────┴──────────────────────────────────────┴─────────────┴───────────────────┴─────────────────────┴──
──────────┴─────────┘
(8 rows)

Time: 0.675 ms
dbms_lab_2=⋆#
```

Тут немного слетело форматирование, к сожалению, лучше посмотреть в репозитории: https://github.com/osovv/miet-dbms/tree/main/lab_2/output.txt

Использовал **uuid**, а не **id**, потому что а) на данный момент это стандарт в индустрии; б) с этим проще работать, т.к. не нужна дополнительная табличка для отслеживания существующих id (т.к. uuid гарантированно уникальный); в) мне так привычнее.

Использовал jsonb для users.permissions, т.к. точной архитектуры прав доступа ещё не придумал.

Есть возможность забанить пользователя (**users.banned**) и пропустить какую-то заявку (**requests.skipped**)

Вывод: приобрёл навыки проектирования и создания баз данных с помощью утилиты pgModeler. Спроектировал и создал базу данных для заданной предметной области.