Лабораторная работа 7
ОПЕРАТОРЫ ОПРЕДЕЛЕНИЯ ДАННЫХ

Создание пользователей и предоставление им прав

Все действия с базой данных, включая создание базы и ее объектов, осуществляются пользователями, поэтому на первом этапе необходимо создать пользователей и предоставить им необходимые права для работы с базой данных. Эту операцию должен выполнить администратор базы данных.

В СУБД PostgreSQL для реализации этого правила используются понятия роль и привилегия. Для каждой роли задается набор атрибутов, и предоставляются привилегии для работы с объектами базы данных.

При установке PostgreSQL автоматически создается роль postgres, которая имеет все возможные атрибуты и привилегии. Пользователь, который обладает ролью postgres, является администратором базы данных и должен создавать другие роли и предоставлять им привилегии

Создание роли

Для создания новой роли используется команда:

```
CREATE ROLE {имя роли}{список атрибутов}
```

Описание некоторых часто используемых атрибутов:

- **SUPERUSER** – определяет, будет ли эта роль суперпользователем, который может переопределить все ограничения доступа в базе данных. Создать нового суперпользователя может только суперпользователь. В отсутствие этих предложений по умолчанию подразумевается **NOSUPERUSER**.

- **CREATEDB** – определяет то, что эта сможет создавать базы данных. По умолчанию подразумевается **NOCREATEDB .**
- **CREATEROLE** – определяет, сможет ли роль создавать новые роли По умолчанию подразумевается **NOCREATEROLE**.
- **LOGIN** – определяет, разрешается ли роли вход на сервер; то есть, может ли эта роль стать начальным авторизованным именем при подключении клиента. По умолчанию подразумевается вариант **NOLOGIN**.
- **PASSWORD** 'пароль' – Задаёт пароль роли. Если проверка подлинности по паролю не будет использоваться, этот параметр можно опустить.

Пример команды создания роли суперпользователя с «положительными» значениями рассмотренных атрибутов.

```
CREATE ROLE user1 SUPERUSER CREATEDB CREATEROLE LOGIN
PASSWORD 'user1';
```

Предоставление привилегий

Роли определяются для всей базы данных, но для того, чтобы пользователь, обладающий определенной ролью, мог осуществлять операции с ее объектами ему должны быть предоставлены соответствующие привилегии. Примечание: если роль имеет атрибут **SUPERUSER,** то все привилегии предоставляются автоматически. Привилегии предоставляются для различных видов объектов базы данных.

Для работы с различными объектами базы данных могут быть предоставлены следующие привилегии:

- **SELECT** – позволяет оператору SELECT извлекать данные из столбцов таблицы или представления

- **INSERT** – позволяет добавлять новые строки в таблицу, может быть предоставлена для определенных столбцов таблицы.
- **DELETE** – разрешает удалять строки из таблицы.
- **TRUNCATE** – Разрешает осуществить очистку таблицы.
- **REFERENCES** – позволяет создавать ограничение внешнего ключа, для таблицы.
- **CREATE** – для баз данных позволяет создавать новые схемы базе данных, а для схем позволяет создавать новые объекты внутри схемы.
- **CONNECT** – позволяет подключаться к базе данных.
- **ALL [PRIVILEGES]** – предоставляет все привилегии, которые могут быть предоставлены для определенного типа объекта. Ключевое слово PRIVILEGES является необязательным в PostgreSQL, хотя в стандарте SQL оно требуется

В таблице 1.1. приведены рассмотренные привилегии и указаны типы объектов, которым они могут быть предоставлены.

Таблица 1.1. Привилегии и типы объектов

Привилегия	Применимые типы объектов
SELECT	Таблица, столбец таблицы, последовательность.
INSERT	Таблица, столбец таблицы.
UPDATE	Таблица, столбец таблицы, последовательность.
DELETE	Таблица
TRUNCATE	Таблица
REFERENCES	Таблица, столбец таблицы.
CREATE	База данных, схема.
CONNECT	База данных.

Выдать привилегии можно с помощью команды **GRANT**, которая имеет следующий формат:

```
GRANT <привилегии> ON <объект> TO <роль>;
```

Примеры предоставления привилегий

1. Предоставление привилегий **SELECT, UPDATE, INSERT** для всех таблиц в схеме **hr_poc1** роли **user1**.

```
GRANT SELECT, UPDATE, INSERT ON ALL TABLES IN SCHEMA hr_poc1 TO user1;
```

2. Предоставление **всех привилегий** для всех таблиц в схеме **hr_poc1** роли **user1**.

```
GRANT ALL PRIVILEGES ON ALL TABLES IN SCHEMA hr_poc1 TO user1;
```

Отзыв привилегий

Для отзыва предоставленных привилегий используется команда:

```
REVOKE <привилегии> ON <объект> FROM <роль>;
```

Примеры отзыва привилегий.

1. Отзыв привилегий **INSERT для всех таблиц** в схеме **hr_poc1** у роли **user1**.

```
REVOKE INSERT, ON ALL TABLES IN SCHEMA hr_poc1 FROM user1;
```

2. Отзыв привилегий **DELETE, TRUNCATE для одной таблицы** у роли user1.

```
REVOKE DELETE, TRUNCATE ON TABLE customers FROM user1;
```

3. Отзыв привилегий **UPDATE** для столбца **credit_limit** таблицы **customers** у роли **user1**.

```
REVOKE UPDATE (credit_limit) ON TABLE customers FROM user1;
```

Создание новой базы данных

Для создания и редактирования объектов базы данных используются операторы определения данных – **DDL** (Data Definition Language,).

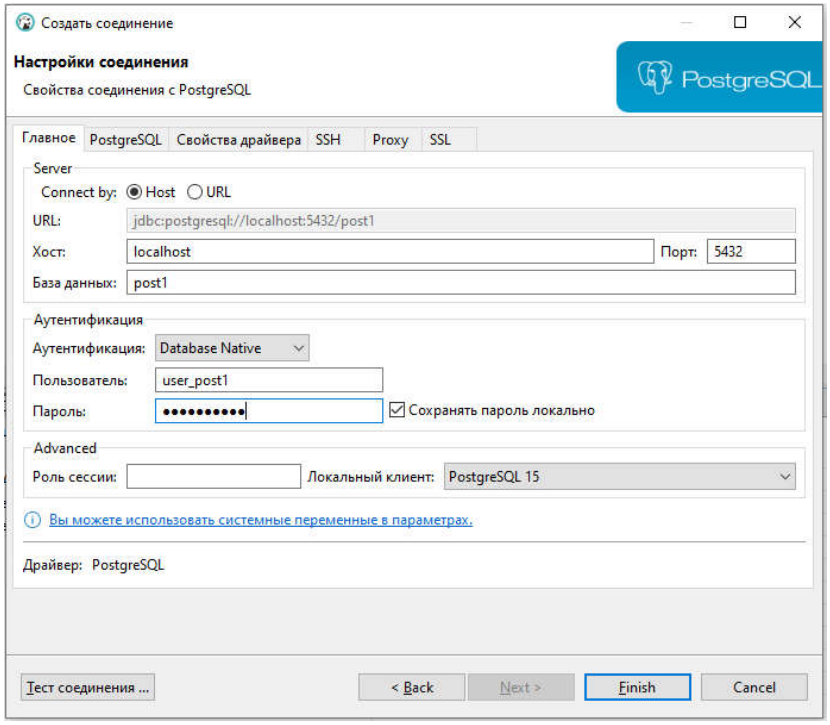
Операторами определения данных являются:

- **CREATE** – используется для создания объектов базы данных;
- **ALTER** – используется для редактирования объектов базы данных;
- **DROP** – используется для удаления объектов базы данных.

Создание базы данных **POST1** и пользователя **user_post1**, который будет обладать правами разработчика.

```
CREATE DATABASE POST1;
CREATE ROLE user_post1 SUPERUSER CREATEDB LOGIN PASSWORD 'user_post1';
GRANT ALL PRIVILEGES ON DATABASE POST1 TO user_post1;
```

Создадим новое соединения, для базы данных **POST1**



Создание соединения с базой данных POST1

После подключения, создадим схему **POC**.

```
CREATE SCHEMA POC;
```

В этой схеме создадим некоторые таблицы из схемы HR_POC.

Создание таблиц базы данных

Для того чтобы создать таблицу нужно задать ее имя и определить столбцы. Каждому столбцу нужно присвоить имя, определить тип и размер. Типом столбца могут быть встроенные типы данных и большинство типов созданных пользователями. Для столбца можно задать значение по умолчанию и ограничения, которым он должен удовлетворять.

Оператор создания таблицы может быть представлен следующим образом:

```
CREATE TABLE [{имя схемы}].{Имя таблицы}
({Имя столбца 1} {Тип столбца_1}
*[CONSTRAINT {имя_ограничения_c1}
  код_ограничения_c1]
[DEFAULT значение_1],
....
({Имя столбца n} {Тип столбца_n}
[CONSTRAINT имя_ограничения_cn
  код_ограничения_cn]
[DEFAULT значение_n],

**[CONSTRAINT {имя_ограничения_t1}
    код_ограничения_t1,]
....
   [CONSTRAINT {имя_ограничения_tn}
    код_ограничения_tn]);
```

Оператор **CREATE TABLE** создаст новую таблицу, не содержащую строк. Пользователь, который ввел этот оператор, будет обладать всеми привилегиями для выполнения операций с этой таблицей.

Если задано имя схемы, то таблица создается в указанной схеме. В противном случае она создается в текущей схеме.

Оператор создания таблицы Customers.

```
CREATE TABLE Customers
(    customer_id int4 NOT NULL,
     c_name varchar(255) NOT NULL,
     address varchar(255),
     credit_limit numeric(10,2),
     CONSTRAINT customers_id_pk PRIMARY KEY (customer_id)
);
```

Значения по умолчанию

Для столбца можно задать значение по умолчанию. Если оно не указано, то этому столбцу, в добавляемой строке, будет присвоено значение NULL.
Присваиваемое значение может быть константой или выражением. В качестве примера приведем присвоение столбцу **order_date** имеющему тип DATE значения текущей даты.

```
odrer_date DATE DEFAULT CURRENT_DATE,
```

Ограничения

Важным элементом при создании таблиц является задание ограничений, которые позволяют отслеживать правильность модификации имеющихся данных или вставляемых в таблицу новых данных. Ограничения могут быть определены на уровне столбца (*) или на уровне таблицы (**).

В общем виде, определение ограничения выглядит следующим образом:

```
CONSTRAINT {имя_ограничения} {определение_ограничения}
```

Ограничение NOT NULL

Данное ограничение не позволяет присваивать неопределенные значения (NULL) столбцу при добавлении новых строк в таблицу. Данное ограничение задается на уровне столбца без использования служебного слова CONSTRAINT.

Пример:

```
c_name VARCHAR(255) NOT NULL,
```

Ограничение UNIQUE

Данное ограничение заключается в том, что все значения в столбце должны встречаться только по одному разу. Ограничение UNIQUE допускает значения NULL.

Используются два варианта назначения данного ограничения:

В виде ограничения на уровне столбца:

```
{Имя столбца} {Тип столбца} UNIQUE
```

например

```
name_p VARCHAR(20) UNIQUE,
```

и в виде ограничения на таблицу:

```
CONSTRAINT <имя ограничения> UNIQUE(<имя столбца>),
```

Например

```
CONSTRAINT products_name_p_un UNIQUE(name_p),
```

Ограничение CHECK

Данное ограничение определяет диапазон значений, которые могут быть присвоены элементам столбца. Ограничение задается с путем определения логического выражения, которое должно иметь значение true для всех элементов столбца. Если при добавлении новой строки логическое выражение будет иметь значение false то СУБД выдаст сообщение об ошибке и строка добавлена не будет.

В общем виде

```
CONSTRAINT <имя ограничения>
CHECK (<логическое выражение>),
```

Создание ограничения CHECK.

```
CONSTRAINT products_rating_p_ch
CHECK rating_p BETWEEN 1 AND 5;
```

Ограничение первичного ключа (PRIMARY KEY)

При создании таблицы, как правило, определяется первичный ключ – один или несколько столбцов однозначно идентифицирующих строки этой таблицы. Первичный ключ надо обязательно определять в том случае если данная таблица является главной при установлении связи с подчиненной таблицей. Одна строка главной таблицы может быть связана с несколькими строками подчиненной таблицы.

Описание ограничения первичного ключа, в общем виде, может быть представлено следующим образом:

```
CONSTRAINT <имя ограничения> PRIMARY KEY (<Список столбцов>)
```

Ограничения первичного ключа гарантирует, что все значения ключа не пустые (NOT NULL) и уникальны (UNIQUE).

Ограничение PRIMARY KEY может быть определено как на уровне столбца, так и на уровне таблицы. Но если первичный ключ состоит из нескольких столбцов, то он должен быть определен на уровне таблицы.

Создание ограничения PRIMARY KEY на уровне столбца:

```
customer_id  NUMBER(4) PRIMARY KEY
```

Создание ограничения PRIMARY KEY на уровне таблицы:

```
CONSTRAINT cust_id_pk PRIMARY KEY (customer_id)
```

Для определения первичного ключа, состоящего из нескольких столбцов, необходимо определить ограничение на уровне таблицы, например:

```
CONSTRAINT Ordit_ Id_It_Pk PRIMARY KEY (order_id, item_id)
```

Ограничение внешнего ключа (FOREIGN KEY)

Связь между таблицами реализуется путем объявления первичного ключа в главной таблице, и объявления внешнего ключа в подчиненной таблице, Значения внешнего ключа могут повторяться, но обязательно должны совпадать с одним из значений первичного ключа в главной таблице, или иметь значение NULL. Это свойство называется **свойством ссылочной целостности**. За его выполнением будет следить СУБД.

Сначала, нужно определить ограничение PRIMARY KEY в главной таблице, потом определить ограничение **FOREIGN KEY** в подчиненной таблице.

Данное ограничение, как правило, определяется на уровне таблицы и может быть представлено следующим образом:

CONSTRAINT <имя ограничения> FOREIGN KEY (<Список столбцов>) REFERENCES <Имя родительской таблицы> (<Список столбцов первичного ключа родительской таблицы>) [<Правила поддержания целостности связи>];

В отличие от ограничения первичного ключа, таблица может иметь несколько ограничений внешнего ключа. При определении внешнего ключа можно указать, какие правила поддержания целостности необходимо использовать при удалении строк главной таблицы.

- ON DELETE CASCADE — каскадное удаление строк подчиненной таблицы. При удалении строки главной таблицы удаляются связанные с ней строки подчиненной таблицы;
- ON DELETE SET NULL — присвоение значения NULL столбцам внешнего ключа. При удалении строки главной таблицы, в связанных с ней строках подчиненной таблицы, столбцам внешнего ключа присваивается значение NULL.

Для того чтобы, создаваемая база данных, соответствовала правилам предметной области нужно, при определении ограничения FOREIGN KEY, правильно определять правила поддержания целостности. Эти правила выбираются исходя из ограничений, существующих в предметной области и определять их должен клиент, а не программист.

Схема **Poc** будет содержать следующие таблицы: Customers, Products, Orders, Order_Items. Таблицу Customers мы уже создали, рассмотрим операторы создания остальных таблиц схемы. Таблицы нужно создавать в определенной последовательности. Таблица, содержащая внешний ключ для связи с некоторой таблицей, должна создаваться после этой таблицы.

Запрос 8.2. Оператор создания таблицы Products.

```
CREATE TABLE Products
(    product_id   INTEGER PRIMARY KEY,
     product_name VARCHAR( 255 ) NOT NULL,
     rating_p     INTEGER,
     price        NUMERIC(10,2),
     CONSTRAINT product_r CHECK((rating_p>0) AND (rating_p<=5))
);
```

В этом запросе первичный ключ определяется на уровне столбца, а на уровне таблицы определяется ограничение CHECK для столбца rating_p (0<=rating_p<=5).

Запрос 8.3. Оператор создания таблицы Orders

```
CREATE TABLE Orders
(    order_id    INTEGER PRIMARY KEY,
     customer_id INTEGER NOT NULL,
     status      VARCHAR(20) NOT NULL,
     salesman_id INTEGER,
     order_date  DATE NOT NULL,
     CONSTRAINT fk_orders_customers FOREIGN KEY (customer_id)
     REFERENCES Customers (customer_id) ON DELETE CASCADE
);
```

В этом запросе на уровне таблицы определяется внешний ключ (customer_id), для создания связи с таблицей **Customers**. Свойства этой связи будет рассмотрено позже.

Запрос 8.4. Оператор создания таблицы Order_Items

```
CREATE TABLE Order_Items
(    order_id   integer, -- fk, pk
     item_id    integer,-- pk
```

```
    product_id integer NOT NULL, -- fk
    quantity   integer NOT NULL,
    unit_price numeric(10,2)NOT NULL,
    PRIMARY KEY(order_id,item_id),
    FOREIGN KEY(product_id) REFERENCES Products(product_id),
    FOREIGN KEY(order_id) REFERENCES Orders(order_id)
    ON DELETE CASCADE
);
```

Таблица **Order_Items** имеет составной первичный ключ (**order_id, item_id)**, поэтому он определен на уровне таблицы. При этом столбец **order_id** одновременно является внешним ключом, для установления связи с таблицей **Orders**. Такие связи называются **идентифицирующими**. Строка таблицы **Order_Items** не может существовать, если она не связана со строкой таблицы **Orders.** Также для этой таблицы определен внешний ключ **(product_id)**, для связи с таблицей **Products.**

Рассмотрим свойства связей между созданными таблицами и проанализируем правила поддержания ссылочной целостности заданные для этих связей.

1. Связь между таблицами **Products** и **Order_Items**.

```
CONSTRAINT Ord_It_Pr_fk FOREIGN KEY (product_id)
REFERENCES Products(product_id);
```

2. Связь между таблицами **Customers** и **Orders**.

```
CONSTRAINT Ord_fk FOREIGN KEY (customer_id)
REFERENCES Customers(customer_id)
ON DELETE SET NULL;
```

3. Связь между таблицами **Orders** и **Order_Items**.

```
CONSTRAINT ord_it_or_fk FOREIGN KEY (order_id)
REFERENCES Orders(order_id)
ON DELETE CASCADE;
```

В этих таблицах использованы разные правила поддержания ссылочной целостности, определяющие действия которые должны быть выполнены со строками подчиненной таблицы при удалении связанной с ними строки главной таблицы.

Для первой связи это правило не задано. Это означает, что нельзя удалить товар, который был продан. Обосновать это можно следующим образом: данные о продаже теряют смысл, если не будет известно, какой товар был продан.

Для второй связи задано правило **ON DELETE SET NULL**. Это означает, что при удалении данных о клиенте, у всех заказов, которые сделал этот клиент, значение столбца **customer_id** в таблице **Orders** будет иметь значение **NULL**. Обосновать это можно следующим образом: данные о заказе не теряют смысла, даже в том случае если неизвестно какой клиент сделал этот заказ.

Для третьей связи задано правило **ON DELETE CASCADE**. Это означает, что при удалении данных о заказе будут удалены данные о содержимом этого заказа. Целесообразность использования этого правила очевидна.

На рис.1. представлена E-R диаграмма схемы **POC**.

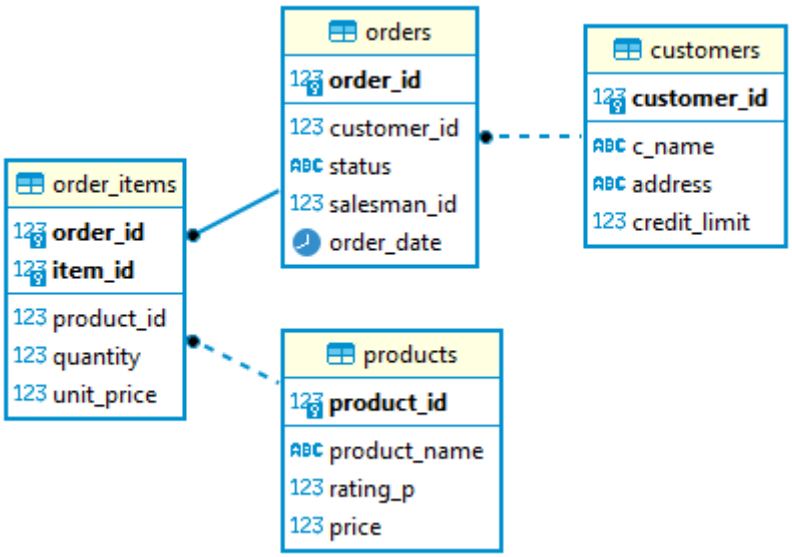


Рис.1. E-R диаграмма схемы POC

Редактирование таблиц

В процессе работы с базой данных возникает необходимость вносить изменения в существующие таблицы. Такими изменениями могут быть:

- добавление или удаление столбцов;
- изменение имени, типа и значения по умолчанию;
- добавление и удаление ограничений;
- удаление таблиц.

Для осуществления этих операций служит команда

```
ALTER TABLE {имя таблицы}
{код редактирования}
```

Добавление и удаление столбцов

Для добавления нового столбца используется команда

```
ALTER TABLE {имя таблицы} ADD COLUMN
({имя столбца} {тип столбца};
```

Пример:

```
ALTER TABLE Customers ADD COLUMN
rating int4;
```

Для удаления существующих столбцов служит команда

```
ALTER TABLE {имя таблицы} DROP COLUMN {имя столбца};
```

При применении этого оператора следует соблюдать следующие правила:

- одним оператором можно удалить только один столбец,
- нельзя удалить все столбцы в таблице.

Изменение столбцов

Можно изменить **имя столбца**, используя команду

```
ALTER TABLE {имя таблицы}
RENAME COLUMN {старое имя} TO {новое имя};
```

Пример:

```
ALTER TABLE Customers
RENAME COLUMN rating TO rating_c;
```

Для изменения **типа столбца** служит команда

```
ALTER TABLE {имя таблицы}
ALTER COLUMN {имя столбца} TYPE {новый тип};
```

Пример

```
ALTER TABLE Customers
ALTER COLUMN rating_c TYPE smallint
```

При использовании этого оператора следует соблюдать следующие правила: увеличивать число разрядов, числового столбца и ширину строки символов можно всегда, а уменьшать можно только до наибольшего значения, содержащегося в столбце.

Изменить **значение по умолчанию** можно командой

```
ALTER TABLE {имя таблицы}
ALTER COLUMN {имя столбца} SET DEFAULT {значение};
```

Пример

```
ALTER TABLE Customers
ALTER COLUMN rating_c SET DEFAULT 1;
```

Изменение ограничений

Добавление ограничения:

```
ALTER TABLE {имя таблицы} ADD CONSTRAINT
 {имя ограничения}{текст ограничения}
```

Пример

```
ALTER TABLE Customers ADD CONSTRAINT Customers_rating_ch
CHECK((rating_c>0) AND (rating_c<=5));
```

Удаление таблицы

Для удаления таблицы используется команда DROP TABLE. В результате выполнения этой команды из таблицы удаляются все данные, и описание таблицы удаляется из словаря данных.

Синтаксис

```
DROP TABLE <Имя таблицы>;
```

Если удаляемая таблица связана с другими таблицами, в которых определены внешние ключи, ссылающиеся на столбцы удаляемой таблицы, то эта команда не будет выполнена. Для удаления таблиц с ограничениями целостности следует использовать команду:

DROP TABLE <Имя таблицы> CASCADE;

Пример

```
DROP TABLE Products CASCADE
```

При выполнении этой команды, в связанных таблицах удаляются ограничения внешнего ключа.

Представления

Представлением называют сохраненный запрос, которому присваивается имя. Это имя можно использоваться в качестве источника данных в других запросах. Имя представления не должно совпадать с именем таблиц базы данных. Если в качестве источника данных указано имя представления, то СУБД выполняет содержащийся в нем запрос и возвращает результат его выполнения. Имена представлений можно указывать там, где можно указывать имена таблиц.
Для создания представлений используется оператор **CREATE VIEW**, который имеет следующий синтаксис:

```
CREATE [ OR REPLACE ] [ TEMP ] [ RECURSIVE ]
VIEW {имя представления} [ {список столбцов} ]
AS {текст запроса}
[ WITH [ CASCADED | LOCAL ] CHECK OPTION ]
```

Рассмотрим параметры этого оператора.

- **OR REPLACE** – если этот параметр указан, то при повторном выполнении, ранее созданное представление будет перезаписано;
- **TEMP** – при наличии этого параметра будет создано временное представление. Временные представления автоматически удаляются в конце текущего сеанса. Существующие постоянные представления с тем же именем не видны текущему сеансу, пока существует временное представление;
- **RECURSIVE** – будет создано рекурсивное представление;
- **WITH [CASCADED | LOCAL] CHECK OPTION** – этот параметр управляет поведением представлений используемых для изменения данных. Если этот параметр указан, то при выполнении операторов NSERT и UPDATE будет осуществляться проверка: новые строки должны удовлетворять условиям, заданным в представлении. Если эти условия не

будут выполнены, хотя бы для одной строки, то изменение будет отменено для всех строк.

Параметр CHECK OPTION нельзя использовать в рекурсивных представлениях.

Представления позволяют:

- Упросить создание сложных запросов, которые следует разделить на части и реализовать каждую часть в виде представления.
- Ограничить доступ пользователей к данным, создавая представления, которые содержат только те столбцы, доступ к которым разрешен.

Есть два типа представлений: простые и сложные.

- **Простое представление** – это представление, которое: использует данные только из одной таблицы, не содержит функций или группировку данных.
- **Сложное представление** – это представление, которое: использует данные из нескольких таблиц или содержит функции или группировку данных

В операторах изменения данных: NSERT, UPDATE, MERGE, DELETE можно использовать только простые представления.

Запрос 1. Создание представления, которое содержит данные о сотрудниках, зарплата которых больше 8000 но меньше 10000.

```
CREATE OR REPLACE VIEW View_Salary_10000 AS
SELECT department_id, employee_id, first_name, last_name, salary
FROM Employees
WHERE salary between 8000 and 10000
WITH CHECK OPTION;
```

Запрос 2. Увеличить на 10% зарплату сотрудников используя представление View_Salary_10000.

```
UPDATE View_Salary_10000
SET salary = salary*1.1
returning *;
```

```
SQL Error [44000]: ОШИБКА: новая строка нарушает ограничение-проверку для
представления "view_salary_10000"
  Подробности: Ошибочная строка содержит (151, David, Bernstein, DBERNSTE,
011.44.1344.345268, 1997-03-24, SA_REP, 10450.00, 0.250, 145, 80, 3).
```

Ошибка возникла вследствие того что при выполнении этого оператора зарплата одного или нескольких сотрудников превысила значение 10000.

Запрос 3. Создание сложного представления Order_Summa, которое возвращает order_id, order_date и общую сумму заказа.

```
CREATE OR REPLACE  VIEW Order_Summa (order_id,order_date,summa)
AS
SELECT orders.order_id, order_date, SUM(quantity*unit_price)
FROM  Orders JOIN Order_Items
ON(Orders.order_id=Order_Items.order_id)
GROUP BY orders.order_id, orders.order_date;
```

Запрос 4. Используя представление Order_Summa вывести заказы оформленные '27.05.2017, общая сумма которых превышает 100000.

```
SELECT order_id,order_date, TO_CHAR(summa,'999G999D99') As SUMMA
FROM Order_Summa
WHERE  order_date ='27.05.2017' AND SUMMA > 100000
```

```
order_id|order_date|summa      |
--------+----------+-----------+
      20|2017-05-27| 577 500,00|
```

Задание

Задача 1. Создать схему базы данных, E-R диаграмма которой представлена на рисунке.

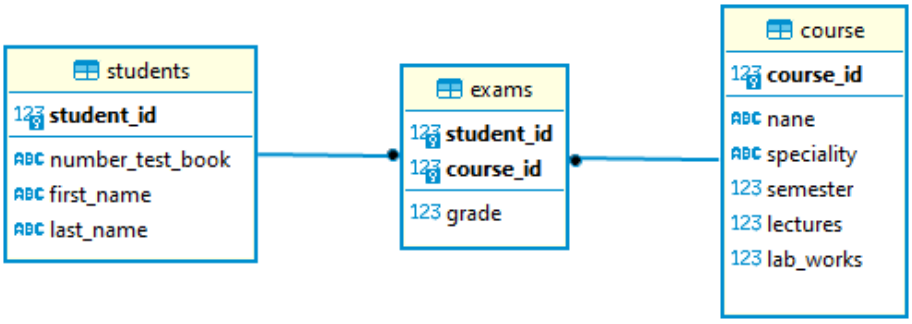


Рис. E-R диаграмма схемы EDU.

Задача 2. Добавить в схему EDU таблицу **Teachers.** Установить связь между таблицами **Course** и **Teachers**, которая должна обеспечивать выполнение следующего правила: каждый преподаватель может вести занятия по нескольким дисциплинам, а занятия по каждой дисциплине ведет только один преподаватель.

Задача 3. Схема **EDU** предполагает, что каждый студент по каждой дисциплине сдает экзамен один раз. Внести в схему изменения, которые позволят хранить для каждого студента данные о нескольких экзаменах по каждой дисциплине. После этих изменений создать E-R диаграмму схемы

Задача 4. Создать представление, которое возвращает данные о студентах и среднем бале каждого студента.

Задача 5. Создать представление, которое возвращает значения столбцов first_name, last_name преподавателя и среднее значение отметок по каждой дисциплине, которые он вел.