



Кафедра ЦТ  
Институт информационных технологий  
РТУ МИРЭА



# Дисциплина «Разработка баз данных»

# Критерии активности



Наименование		Формат	Баллы (макс.)
<b>Защита практических работ</b> (8 практик, 6 баллов за 1 практику)	Сентябрь – Декабрь 2025	Очно	48, [0-6] / практика
<b>Контрольный тест №1</b> (Лекции №1-2)	Октябрь 2025	Очно, СДО	5
<b>Контрольный тест №2</b> (Лекции №3-5)	Ноябрь 2025	Очно, СДО	5
<b>Контрольная работа №1</b>	Ноябрь 2025	Очно, СДО	10
<b>Контрольный тест №3</b> (Лекции №6-7)	Декабрь 2025	Очно, СДО	6
<b>Контрольная работа №2</b>	Декабрь 2025	Очно, СДО	10
<b>Посещение</b> (лекции 8 занятий, практики 24 занятия)	Сентябрь – Декабрь 2025	Очно	16, 0.5 / занятие
Максимальная сумма баллов			100

# Экзамен



- **Допуск к экзамену** – защита всех практических работ, тестов и контрольных в течение семестра очно.
- Если студент набрал меньше 50 % от общей суммы баллов, то экзамен будет проходить по **билету** (3 теоретических вопроса + 1 задача).
- Если студент набрал больше 50% от максимальной суммы баллов, то студент сдаёт **тестирование на экзамене**. + **добавляются баллы за активность** по следующим правилам:
  - Набрано 55-64 балла за семестр – 1 доп.
  - Набрано 65-74 балла за семестр – 2 доп.
  - Набрано 75-84 балла за семестр – 3 доп.
  - Набрано 85-94 балла за семестр – 4 доп.
  - Набрано 95-100 балла за семестр – 5 доп.

# Практическая работа №1.

## Основы ddl и запросы на выборку данных.



**Постановка задачи:** основываясь на логической модели данных, спроектированной в рамках курса «Проектирование баз данных» в предыдущем семестре, выполните следующие шаги:

1. Письменно опишите не менее **5 различных бизнес-правил** и не менее **3 ограничений целостности** для таблиц. Выбор бизнес-правил и ограничений целостности производится на усмотрение студента.  
**Результаты представить в виде таблицы.**
2. С использованием DDL-оператора CREATE TABLE создать **все необходимые таблицы** (согласно созданной в прошлом семестре логической модели данных) в СУБД PostgreSQL, корректно реализовав все описанные **ограничения целостности**.
3. Заполнить созданные таблицы **согласованными тестовыми данными** (не менее 5-7 записей на таблицу, где это применимо) с помощью DML-оператора INSERT INTO.

*(продолжение на следующем слайде)*

# Практическая работа №1.

## Основы ddl и запросы на выборку данных.



**Постановка задачи:** основываясь на логической модели данных, спроектированной в рамках курса «Проектирование баз данных» в предыдущем семестре, выполните следующие шаги:

4. Составить и выполнить не менее **6 SQL-запросов** к таблицам, иллюстрирующих использование различных элементов списка выборки и условных выражений WHERE, **согласно перечню, указанному в задании** (см. *Ход выполнения работы*).  
В составленных запросах должны быть **использованы все приведённые примеры**.
5. Составить и выполнить **по два SQL-запроса** к таблицам для демонстрации работы ORDER BY, GROUP BY и HAVING.
6. Каждый SQL-запрос **сопроводить комментарием**, объясняющим его **назначение и логику работы**.

*(начало на предыдущем слайде)*

# 1. Анализ и описание ограничений целостности



**Ограничения целостности** – это правила, которые СУБД автоматически применяет к данным, чтобы гарантировать их точность, надежность и консистентность (согласованность).

Это – основа реляционной модели данных, которая не позволяет поместить в базу некорректную информацию.

# 1. Анализ и описание ограничений целостности



➤ **PRIMARY KEY** (*первичный ключ*) – уникально **идентифицирует каждую запись** в таблице.

В таблице может быть только **один** первичный ключ.

Фактически, это **комбинация** ограничений *UNIQUE* и *NOT NULL*

➤ **UNIQUE** (*уникальность*) – гарантирует, что **все значения** в столбце или группе столбцов **уникальны**.

Отличие от PRIMARY KEY: **допускает** *NULL* (в том числе несколько *NULL* в столбце).

➤ **NOT NULL** (*непустое значение*) – запрещает столбцу содержать *NULL*-значения.

**Каждая** запись должна иметь **конкретное значение** в этом поле.

# 1. Анализ и описание ограничений целостности



- **FOREIGN KEY** (внешний ключ) – **связывает** две таблицы, обеспечивая **ссылочную целостность**.

Значение в столбце дочерней таблицы *должно* существовать в первичном ключе родительской таблицы.

- **CHECK** (проверка) – **гарантирует**, что значение в столбце удовлетворяет определённому **логическому условию**.

Примеры: цена > 0 или дата\_окончания > дата\_начала

- **DEFAULT** (значение по умолчанию) – задаёт **значение**, которое будет **автоматически вставлено** в столбец, если оно **не указано** явно.



# 1. Анализ и описание ограничений целостности



Название столбца	Тип данных	Ограничение	Обоснование (бизнес-правило)
<b>ID</b>	SERIAL	PRIMARY KEY	Уникальный <b>идентификатор</b> лекарства, генерируется автоматически.
<b>price</b>	DECIMAL(10, 2)	<ul style="list-style-type: none"><li>• NOT NULL</li><li>• CHECK (price &gt; 0)</li></ul>	Цена является <b>обязательной</b> и должна быть <b>положительным</b> числом.
<b>manufacturer_id</b>	INTEGER	FOREIGN KEY (manufacturers)	Ссылка на производителя. Лекарство <b>не может существовать</b> без связи с компанией-производителем.

## 2. Создание структуры данных



Для **создания таблиц** в базе данных используется **DDL**-оператор **CREATE TABLE**.

**DDL** означает **Data Definition Language** (*язык описания данных*).

**Критически** важен **порядок создания таблиц**:

- 1. В начале** создаются «**родительские**» таблицы (те, на которые будут ссылаться).
- 2. Затем** создаются «**дочерние**» таблицы (те, которые содержат внешние ключи).

Попытка создать дочернюю таблицу *раньше* родительской приведёт к **ошибке**, так как она будет ссылаться на **несуществующий объект**.

## 2. Создание структуры данных



При определении внешнего ключа (**FOREIGN KEY**), можно определить действие **ON DELETE** которое указывает, что должно произойти с **зависимыми** (дочерними) записями при попытке **удалить основную** (родительскую) запись, на которую они ссылаются.

- **ON DELETE RESTRICT** (или **NO ACTION**) – **действие по умолчанию**.

**Запрещает удаление родительской** записи, если на неё есть ссылки. Это – самый **безопасный** вариант.

- **ON DELETE CASCADE** – при удалении родительской записи автоматически **удаляются все связанные дочерние записи**.

Может вызвать «**эффект домино**» – очень **опасный** вариант, используйте только при необходимости!!!

- **ON DELETE SET NULL** – при удалении родительской записи, в дочерних записях **значение внешнего ключа заменяется на NULL**.

Требует, чтобы столбец внешнего ключа не имел ограничения NOT NULL.

# 3. Заполнение таблиц данными



Для **добавления данных в таблицы** используется **DML**-оператор **INSERT INTO**.

**DML** означает **Data Manipulation Language** (*язык манипулирования данными*).

**Критически** важен **порядок вставки данных**:

- 1. В начале** заполняются «**родительские**» таблицы (те, на которые будут ссылаться).
- 2. Затем** заполняются «**дочерние**» таблицы, используя для внешних ключей идентификаторы уже существующих родительских записей.

Попытка заполнить дочернюю таблицу *раньше* родительской приведёт к **ошибке**, так как **нельзя** сделать **ссылку** на запись, если её **ещё нет**!

## 4. Составление запросов на выборку (часть 1)



Для **извлечения данных** из таблиц используется оператор **SELECT**.

**Общий синтаксис:**

**SELECT**

column1, column2, ...

**FROM**

table\_name

[**WHERE** condition]

[**GROUP BY** column1, column2, ...]

[**HAVING** condition]

[**ORDER BY**

column1 [**ASC** | **DESC**],

column2 [**ASC** | **DESC**],

...]

**Краткое описание:**

**SELECT** – указывает один или несколько столбцов, которые нужно выбрать. Для выбора всех столбцов используется символ \*.

**FROM** – указывает таблицу-источник этих столбцов.

**WHERE** (опциональный) – условие фильтрации данных.

**GROUP BY** – условие группировки данных.

**HAVING** – фильтрует группы (применяется после GROUP BY).

**ORDER BY** – сортирует результат по указанным столбцам (по возрастанию/убыванию).

## 4. Составление запросов на выборку (часть 1)



Ключевые операторы **SELECT**:

- Оператор «**\***» – выбрать все столбцы из таблицы.  
**SELECT \* FROM** medicines;
- Оператор «**AS**» – переименование столбца в итоговой таблице (не меняет исходную).  
**SELECT** price **AS** "Цена, руб." **FROM** medicines;
- Оператор «**DISTINCT**» – удалить дубликаты в итоговой таблице.  
**SELECT DISTINCT** country **FROM** manufacturers;
- **Математические выражения** – позволяют выполнять вычисления прямо в запросе.  
**SELECT** name, price \* quantity\_in\_stock **AS** total\_value **FROM** medicines;

## 4. Составление запросов на выборку (часть 1)



Ключевые операторы **WHERE**:

- Логические операторы «**AND**», «**OR**», скобки «**(**» и «**)**»  
**WHERE** man.country = 'Россия' **AND** med.price < 50;
- Оператор «**BETWEEN**» – проверка диапазона.  
**WHERE** price **BETWEEN** 30 **AND** 100;
- Оператор «**IN**» – проверка вхождения в множества.  
**WHERE** name **IN** ('Аспирин', 'Цитрамон', 'Валидол');
- Оператор «**LIKE**» – для поиска по строкам, «**%**» – любое кол-во любых символов, «**\_**» – 1 любой символ.  
**WHERE** name **LIKE** '%ол%';
- Оператор «**IS [NOT] NULL**» – проверка на *NULL*. Сравнение с *NULL* (= или <>) **всегда UNKNOWN**.  
**WHERE** expiration\_date **IS NULL**;

## 5. Составление запросов на выборку (часть 2)



**ORDER BY** сортирует **финальный результирующий набор данных**.

Направление сортировки:

- ✓ **ASC** – по возрастанию (используется по умолчанию).
- ✓ **DESC** – по убыванию.

Примеры:

- Сортировка **по одному столбцу**, в порядке убывания  
**SELECT** name, price **FROM** medicines **ORDER BY** price **DESC**;
- Сортировка **по нескольким столбцам**:  
**SELECT** country, manufacturer\_name **FROM** manufacturers **ORDER BY** country **ASC**,  
manufacturer\_name **ASC**;



## 5. Составление запросов на выборку (часть 2)



**GROUP BY** «схлопывает» несколько строк с **одинаковыми значениями** в одну **сводную строку**.

Почти всегда используется вместе с **агрегатными функциями**:

- ✓ **COUNT()** – подсчёт количества
- ✓ **SUM()** – сумма значений
- ✓ **AVG()** – среднее значение
- ✓ **MIN()** / **MAX()** – минимальное / максимальное значение

Пример:

```
SELECT manufacturer_id, COUNT(*) AS number_of_medicines  
FROM medicines GROUP BY manufacturer_id;
```

## 5. Составление запросов на выборку (часть 2)



### «Золотое правило» **GROUP BY**:

Любой столбец, указанный в **SELECT**, должен либо быть частью **GROUP BY**, либо использоваться внутри агрегатной функции.

### Почему?

Если сгруппировать лекарства по производителю и попытаться выбрать **name**, СУБД не будет знать, название **какого** из десятков лекарств этого производителя нужно отобразить.

**Агрегатные функции** (COUNT, MIN и т.д.) **решают** эту неоднозначность, сводя множество значений к **одному**.

## 5. Составление запросов на выборку (часть 2)



Фильтрация групп – **HAVING**.

**HAVING** очень похоже на **WHERE**, но они работают на разных этапах.

➤ **WHERE** фильтрует отдельные строки **ДО группировки**.

Поэтому в **WHERE** **нельзя** использовать **агрегатные функции**, так как групп ещё не существует.

➤ **HAVING** фильтрует целые группы **ПОСЛЕ их формирования**.

Поэтому в **HAVING** **можно** и **нужно** использовать **агрегатные функции**.

## 5. Составление запросов на выборку (часть 2)



Фильтрация групп – **HAVING** – пример запроса:

**SELECT**

manufacturer\_id,  
COUNT(\*) **AS** number\_of\_medicines

**FROM**

medicines

**GROUP BY**

manufacturer\_id

**HAVING**

COUNT(\*) > 3; -- Фильтруем уже посчитанные группы

# ВАЖНО: Порядок выполнения запроса



SQL-запросы **выполняются не в том порядке**, в котором они **написаны**.

Понимание этой последовательности – **ключ к написанию сложных запросов**.

1. **FROM** – определение и соединение таблиц.
2. **WHERE** – фильтрация отдельных строк.
3. **GROUP BY** – группировка строк.
4. **HAVING** – фильтрация целых групп.
5. **SELECT** – вычисление и выборка столбцов (здесь создаются псевдонимы).
6. **ORDER BY** – сортировка финального результата.



Кафедра ЦТ  
Институт информационных технологий  
РТУ МИРЭА



**Спасибо за внимание**