

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

**"МИРЭА - Российский технологический университет"**

**РТУ МИРЭА**

Институт информационных технологий (ИИТ)

Кафедра цифровой трансформации

**ОТЧЕТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ №1**

**по дисциплине**

«Проектирование баз данных»

**Тема: «ОСНОВЫ DDL И ЗАПРОСЫ НА ВЫБОРКУ ДАННЫХ В POSTGRES PRO»**

|  |  |
| --- | --- |
| Выполнил студент группы ИНБО-12-23 | Душко И.А. |
| Принял преподаватель | Брайловский А.В. |

Москва 2025

**ПРАКТИЧЕСКАЯ РАБОТА №1. ОСНОВЫ DDL И ЗАПРОСЫ НА ВЫБОРКУ ДАННЫХ В POSTGRES PRO**

**Цель работы:**

Целью данной практической работы является формирование и закрепление у студентов фундаментальных навыков работы с реляционными базами данных на примере СУБД Postgres Pro. По завершении работы студент должен уметь:

• Сформировать практический навык определения структуры базы данных с использованием языка определения данных DDL (Data Definition Language).

• Научиться применять ограничения целостности (PRIMARY KEY, FOREIGN KEY, CHECK, UNIQUE, NOT NULL) для реализации бизнесправил и обеспечения консистентности (согласованности) данных, основываясь на теоретических положениях реляционной модели.

• Освоить составление SQL-запросов на выборку данных с использованием расширенного синтаксиса инструкции SELECT, включая выражения в списке выборки, псевдонимы и фильтрацию дубликатов с помощью DISTINCT.

• Развить умение применять разнообразные условия фильтрации записей в предложении WHERE, охватывая логические операции, проверку принадлежности диапазону и множеству, сравнение с шаблоном и корректную проверку на NULL.

• Получить базовые навыки агрегации данных с использованием GROUP BY и агрегатных функций, а также научиться корректно фильтровать агрегированные результаты с помощью предложения HAVING.

**Постановка задачи:**

Для выполнения практической работы необходимо последовательно выполнить следующие шаги, **основываясь на логической модели данных,** которая была спроектирована в рамках курса **«Проектирование баз данных»** в предыдущем семестре:

1. На основе логической модели данных, созданной в прошлом семестре, письменно описать 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 Postgres Pro, 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 оператора INSERT INTO.
4. Составить и выполнить не менее 6 SQL-запросов к таблицам, иллюстрирующих использование различных операторов SELECT и 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 предложений ORDER BY, GROUP BY и HAVING.
6. Каждый SQL-запрос сопроводить комментарием, объясняющим его назначение и логику работы.

**ХОД ВЫПОЛНЕНИЯ РАБОТЫ**

1. **Анализ и описание ограничений целостности**

Первый этап работы заключается в формализации бизнес-правил вашей предметной области в виде ограничений целостности данных. Ограничения — это правила, которые СУБД автоматически применяет к данным, чтобы гарантировать их точность, надежность и консистентность. Этот механизм является практической реализацией целостной части реляционной модели данных.

Для каждой таблицы из вашей логической модели необходимо составить описание ее атрибутов и применяемых к ним ограничений. Результат следует оформить в виде таблицы.

*Таблица 1. Описание ограничений для таблицы cargo\_details (детали груза)*

|  |  |  |  |
| --- | --- | --- | --- |
| **Название**  **столбца** | **Тип данных** | **Ограничение** | **Обоснование (Бизнес-правило)** |
| cargo\_details\_id | BIGSERIAL | PRIMARY KEY | Уникальный идентификатор записи о деталях груза, генерируется автоматически |
| cargo\_id | BIGINT | FOREGEN KEY, NOT NULL | Ссылка на таблицу cargo. Детали груза не могут существовать |
| weight | NUMERIC | NOT NULL, CHECK (weight > 0) | Вес груза является обязательным атрибутом и должен быть больше нуля. |
| amount | NUMERIC | NOT NULL, CHECK (amount > 0) | Количество груза обязательно должно быть строго больше нуля |
| description | TEXT | NOT NULL | Каждая запись должна иметь описание содержимого для идентификации |

1. **Создание структуры данных**

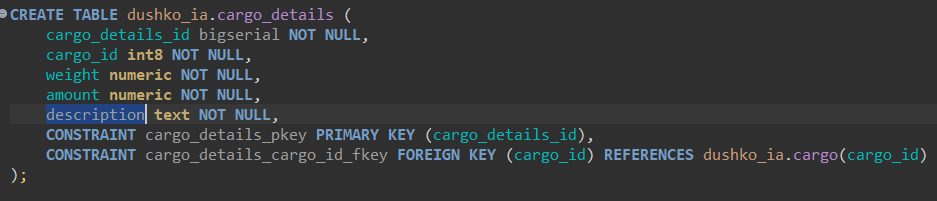
На этом этапе вы преобразуете спроектированные на Шаге 1 правила в SQL-код. Для создания таблиц используется оператор CREATE TABLE.

**Создание таблицы** dushko\_ia.cargo\_details.

**Определяются столбцы**: cargo\_details\_id - первичный ключ (bigserial, NOT NULL), cargo\_id – внешний ключ на таблицу cargo (int8, NOT NULL), weight – числовое значение веса, обязательно (numeric NOT NULL), amount – количество, обязательно (numeric NOT NULL) description – текстовое описание, обязательно (text NOT NULL).

**Устанавливаются ограничения**: cargo\_details\_pkey - первичный ключ по cargo\_details\_id. cargo\_details\_cargo\_id\_fkey - внешний ключ на cargo(cargo\_id), обеспечивающий ссылочную целостность.

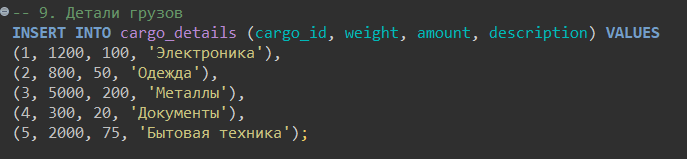
В результате создаётся таблица для хранения подробной информации о грузе, связанной с основной записью о грузе через внешний ключ.



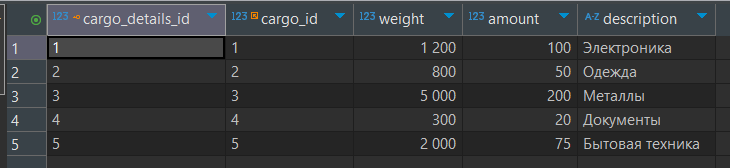
**Рисунок 1 – Создание таблицы cargo\_details с помощью оператора CREATE TABLE**

1. **Заполнение таблиц данными (DML – Data Manipulation Language)**

После создания структуры, базу данных необходимо наполнить тестовыми данными с помощью оператора INSERT INTO. При работе с таблицами, связанными внешними ключами, порядок вставки данных также имеет критическое значение.



**Рисунок 2 – Скрипт для заполнения таблицы cargo\_details с помощью оператора INSERT INTO**

****

**Рисунок 3 – Заполненная таблица cargo\_details**

1. **Составление запросов на выборку (часть 1)**

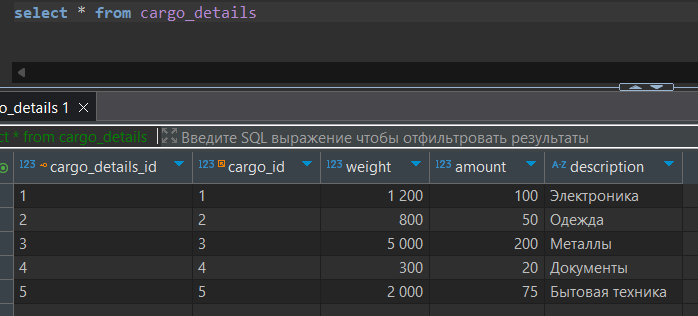
Этот раздел посвящен выполнению четвертого пункта задания, в рамках которого необходимо составить не менее шести запросов, демонстрируя различные возможности SELECT и WHERE.

**4.1 Элементы списка выборки – SELECT**

**4.1.1 Выбор всех столбцов (\*)**

Оператор \* является сокращением, которое позволяет выбрать все столбцы из таблицы без их явного перечисления. Это удобно для быстрой проверки содержимого таблицы, однако при разработке крайне рекомендуется перечислять все нужные столбцы.

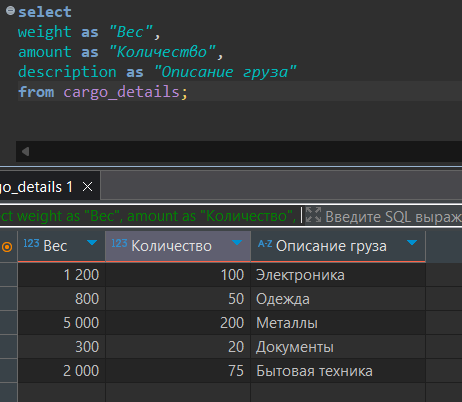
Выведем все столбцы.



**Рисунок 4 – Оператор (\*)**

* + 1. **Выбор конкретных полей и использование псевдонима (AS)**

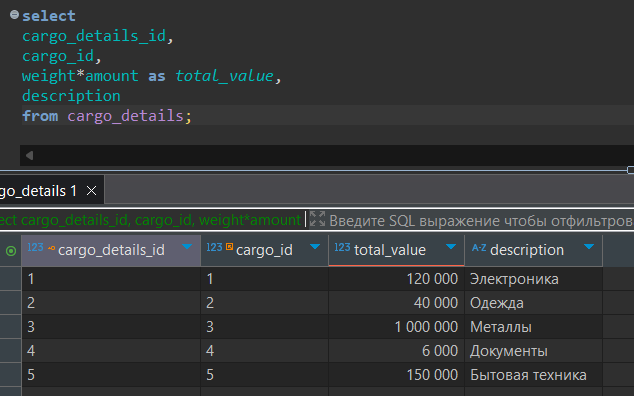
Для повышения читаемости отчетов и результатов запроса можно выбирать только необходимые столбцы и переименовывать их с помощью ключевого слова AS. Псевдонимы, заключенные в двойные кавычки (например, "Цена, руб."), могут содержать пробелы и становятся регистро-зависимыми.



**Рисунок 5 – Оператор (AS)**

* + 1. **Выражение в списке выборки**

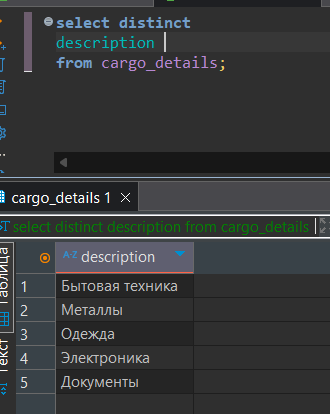
Прямо в списке полей в SELECT можно выполнять различные операции над данными. Это позволяет переложить часть логики обработки данных на СУБД, что часто бывает эффективнее, чем выполнять те же вычисления в коде приложения. В данном примере для каждой строки вычисляется общая стоимость запаса лекарства (total\_value).



**Рисунок 6 – Выражение в списке выборки**

**4.1.3 Удаление дубликатов (DISTINCT)**

Ключевое слово DISTINCT анализирует весь результирующий набор и исключает из него полностью идентичные строки. Этот запрос вернет только уникальные названия стран. Следует помнить, что операция DISTINCT требует от СУБД дополнительных ресурсов на сортировку или группировку данных, что может замедлить выполнение запроса на очень больших объемах данных.



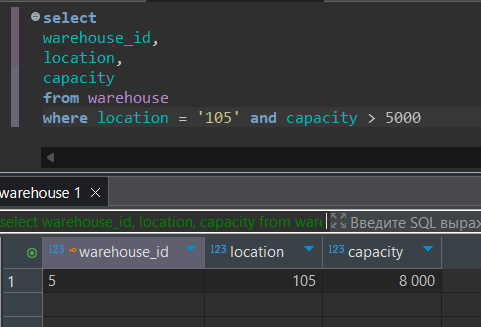
**Рисунок 7 – Удаление дубликатов (DISTINCT)**

**4.2 Условия фильтрации – WHERE**

**4.2.1 Простое условие и логические связки (AND, OR)**

Предложение WHERE фильтрует строки до того, как они попадут в результирующий набор.

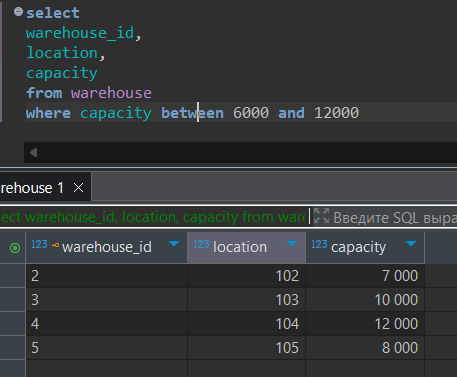
Учтите, что логический оператор AND имеет более высокий приоритет, чем OR, поэтому при сложных условиях рекомендуется использовать скобки () для явного указания порядка вычислений. В данном случае в результат попадут только те лекарства, которые удовлетворяют обоим условиям одновременно



**Рисунок 8 – Простое условие (AND)**

**4.2.2 Проверка принадлежности диапазону (BETWEEN)**

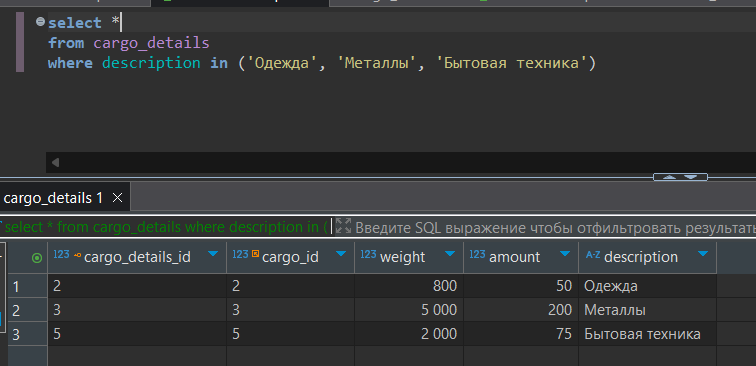
Оператор BETWEEN – это удобный и более читаемый способ проверить, попадает ли значение в диапазон [], т.е. включающий крайние элементы. Он работает не только с числами, но и с датами, и со строками (в лексикографическом порядке).



**Рисунок 9 – Проверка принадлежности диапазону (BETWEEN)**

**4.2.3 Проверка вхождения в множество (IN)**

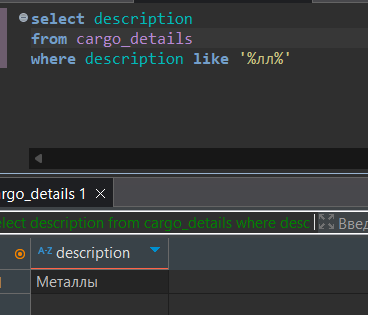
Оператор IN является синтаксически более удобной и часто более производительной альтернативой нескольким условиям OR. Он проверяет, совпадает ли значение столбца с любым из значений в предоставленном списке. Для обратной операции существует оператор NOT IN.



**Рисунок 10 – Проверка вхождения в множество (IN)**

**4.2.4 Сравнение с шаблоном (LIKE)**

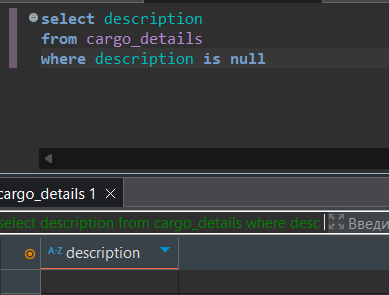
Оператор LIKE используется для гибкого поиска по строкам. Кроме символа % (любое количество любых символов), существует также символ \_ (ровно один любой символ). Важно помнить, что поиск по шаблону, который начинается с % (например, «%ол»), как правило, не может использовать индекс по этому столбцу, что приводит к полному сканированию таблицы и может быть очень медленным.



**Рисунок 11 – Сравнение с шаблоном (LIKE)**

**4.2.5 Проверка на NULL**

В SQL действует трехзначная логика (TRUE, FALSE, UNKNOWN). Любая операция сравнения со значением NULL (например, = NULL или <> NULL) дает результат UNKNOWN. Предложение WHERE отбирает только те строки, для которых условие истинно (TRUE), поэтому строки с результатом UNKNOWN отбрасываются. Именно поэтому для корректной проверки на отсутствие значения всегда следует использовать конструкции IS NULL или IS NOT NULL.



**Рисунок 12 – Проверка на null**

1. **Составление запросов на выборку (часть 2)**

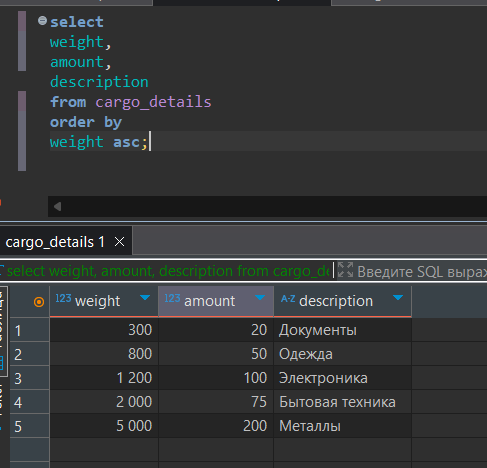
Этот раздел посвящен выполнению пятого пункта задания и охватывает более сложные концепции: сортировку, агрегацию и фильтрацию групп.

**5.1. Сортировка результатов (ORDER BY)**

Предложение ORDER BY применяется к финальному набору данных, полученному после всех фильтраций и группировок, и упорядочивает его строки. Это одна из последних операций в логическом конвейере выполнения запроса. Сортировка может быть ресурсоемкой операцией, особенно на больших объемах данных, если для столбцов сортировки отсутствуют подходящие индексы.

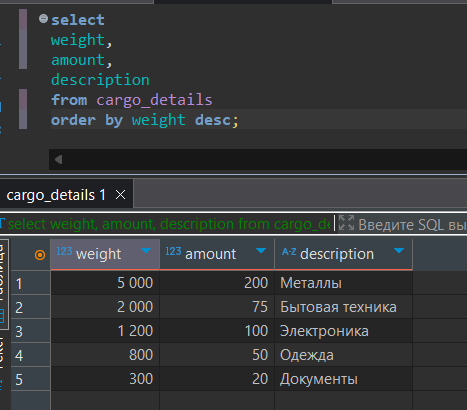
Есть два способа сортировки данных:

• ASC - по возрастанию (значение по умолчанию)



**Рисунок 13 – Сортировка по возрастанию**

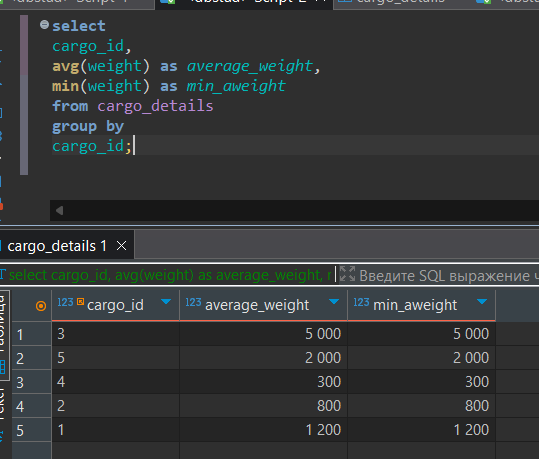
• DESC - по убыванию.



**Рисунок 14 – Сортировка по убыванию**

**5.2 Группировка и агрегатные функции (GROUP BY)**

Предложение GROUP BY «схлопывает» несколько строк с одинаковыми значениями в указанных столбцах в одну сводную строку. Оно практически всегда используется в паре с агрегатными функциями (COUNT, SUM, AVG, MIN, MAX), которые выполняют вычисления для каждой такой группы. «Золотое правило» GROUP BY: любой столбец, указанный в SELECT, должен либо быть частью GROUP BY, либо использоваться внутри агрегатной функции. Это логическая необходимость: если сгруппировать лекарства по производителю и попытаться выбрать name (название лекарства), СУБД не будет знать, название какого из десятков лекарств этого производителя следует отобразить. Агрегатные функции решают эту неоднозначность, сводя множество значений к одному (например, к их количеству COUNT(name)).



**Рисунок 15 – Группировка и агрегатные функции (GROUP BY)**

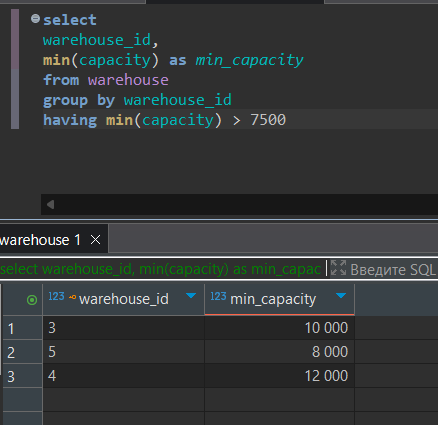
**5.3 Фильтрация групп (HAVING)**

Предложение HAVING очень похоже на WHERE, но с одним ключевым отличием в логике работы:

• WHERE фильтрует отдельные строки до того, как они будут сгруппированы.

• HAVING фильтрует целые группы после того, как они были сформированы и для них были вычислены агрегатные значения.

Именно поэтому в HAVING можно использовать агрегатные функции (такие как COUNT(\*) или AVG(price)), а в WHERE — нельзя, так как на момент работы WHERE никаких групп еще не существует.



**Рисунок 16 – Группировка и агрегатные функции (GROUP BY)**

**Вывод**

В ходе практической работы были успешно освоены основы языка DDL и запросов на выборку данных в Postgres Pro.