

ПРАКТИЧЕСКАЯ РАБОТА №2.

МНОГОТАБЛИЧНЫЕ ЗАПРОСЫ И ТЕОРЕТИКО-МНОЖЕСТВЕННЫЕ ОПЕРАЦИИ В POSTGRES PRO

Цель работы:

Научиться извлекать и комбинировать данные из нескольких связанных таблиц с помощью соединений (JOIN) и теоретико-множественных операторов (UNION, INTERSECT, EXCEPT), а также освоить продвинутые паттерны, такие как «само-соединение» и «анти-соединение».

По завершении работы студент должен уметь:

- Сформировать глубокое концептуальное понимание и практические навыки применения различных типов соединений таблиц (INNER, LEFT, RIGHT, FULLJOIN) для извлечения связанных данных из нескольких таблиц.
- Освоить технику написания сложных многотабличных запросов (с соединением 3-4 и более таблиц), используя псевдонимы таблиц для повышения читаемости кода и разрешения неоднозначности имен столбцов.
- Научиться применять теоретико-множественные операторы (UNION, UNIONALL, INTERSECT, EXCEPT) для комбинирования и сравнения результатов нескольких независимых запросов, соблюдая правила их использования.
- Развить аналитические навыки для декомпозиции сложных бизнес-вопросов в последовательность логических шагов, реализуемых с помощью SQL-запросов.

- Освоить и применять специфические паттерны SQL, такие как «анти-соединение» (anti-join) для поиска несоответствий и «само-соединение» (self-join) для работы с иерархическими данными в рамках одной таблицы.

Постановка задачи:

Задание 1: демонстрация различных типов соединений.

На основе индивидуальной схемы данных, составить и выполнить пять аналитических запросов, демонстрирующих различные типы соединений.

Каждый запрос должен решать осмысленную задачу в рамках вашей предметной области.

1. В начале отчёта должны быть приложены скриншоты всех используемых таблиц индивидуальной схемы данных.
2. Запрос с **INNER JOIN**: подсчитайте количество связанных записей между таблицами (*например, «сколько лекарств у каждого производителя?»*)
3. Запрос с **LEFT JOIN**: проанализируйте наличие или отсутствие связей (*например, «сколько лекарств у каждого производителя, включая тех, у кого лекарств нет?»*)
4. Запрос с **RIGHT JOIN** и **WHERE... IS NULL** (паттерн «анти-соединение»): найдите и подсчитайте записи без связей (*например, «сколько лекарств не имеют производителя в базе?»*)
5. Запрос с **FULL JOIN**: получите общую статистику – сколько всего связанных записей, и сколько записей без связей.
6. Запрос с **CROSS JOIN**: сформировать декартово произведение всех записей одной таблицы со всеми записями другой, создав тем самым все возможные комбинации строк между ними.

Задание 2: применение теоретико-множественных операторов.

На основе индивидуальной схемы данных составить и выполнить три запроса, демонстрирующих практическое применение операторов **UNION**, **INTERSECT** и **EXCEPT**.

1. **UNION**: составить единый список из данных двух разных таблиц (столбцы должны быть совместимы по типу).
2. **INTERSECT**: найти общие записи, которые удовлетворяют двум разным условиям или находятся в двух разных наборах данных.
3. **EXCEPT**: найти записи, которые присутствуют в одном наборе данных, но отсутствуют в другом.

ХОД ВЫПОЛНЕНИЯ РАБОТЫ

Введение: от одной таблицы к реляционной модели

В предыдущей работе акцент делался на создании структуры базы данных и выполнении запросов к отдельным таблицам. Однако сила реляционной модели заключается в способности эффективно хранить данные без избыточности и связывать их между собой.

Процесс нормализации, который вы изучали ранее, приводит к разделению данных на несколько логических сущностей (таблиц).

Операторы **JOIN** являются фундаментальным механизмом SQL, который позволяет "собирать" эти разделенные данные обратно в единое, осмысленное представление для анализа и формирования отчетов.

Ниже приводятся таблицы, используемые для построения запросов.

Таблица 1. Таблица *manufacturers* (Производители)

<small>123</small> <small>↕</small> manufacturer_id	<small>A-Z</small> manufacturer_name	<small>A-Z</small> country
1	ООО "Фармстандарт"	Россия
2	Bayer AG	Германия
3	Sanofi	Франция

Таблица 2. Таблица *medicines* (Лекарства)

<small>123</small> <small>↕</small> id	<small>A-Z</small> name	<small>123</small> quantity_in_stock	<small>123</small> price	<small>🕒</small> production_date	<small>🕒</small> expiration_date	<small>123</small> <small>↕</small> manufacturer_id
1	Парацетамол	200	50,5	2025-07-10	2028-07-10	1
2	Аспирин	150	120	2025-07-12	2027-07-12	2
3	Ибупрофен	100	85	2025-07-11	2028-07-11	1
4	Витамин С	300	250	2025-07-10	2027-07-10	2
5	Активированный уголь	500	25	2025-07-10	2030-07-10	1

Таблица 3. Таблица *customers* (Покупатели)

<small>123</small> <small>↕</small> customer_id	<small>A-Z</small> first_name	<small>A-Z</small> last_name	<small>A-Z</small> email	<small>A-Z</small> phone_number
1	Иван	Иванов	ivan@example.com	+79001234567
2	Петр	Петров	petr@example.com	[NULL]

Таблица 4. Таблица *pharmacists* (Фармацевты)

123 ↗ pharmacist_id ▼	A-Z first_name ▼	A-Z last_name ▼	A-Z phone_number ▼	123 ↗ manager_id ▼
1	Иван	Сорокин	+7(905)123-44-56	[NULL]
2	Алексей	Петров	+7(905)123-44-56	1
3	Марина	Иванова	+7(905)123-44-56	1
4	Евгений	Смирнов	+7(905)123-44-56	2
5	Александр	Орлов	+7(905)123-44-56	3

Таблица 5. Таблица *sales* (Продажи)

123 ↗ sale_id ▼	123 ↗ customer_id ▼	🕒 sale_date ▼	123 ↗ total_amount ▼
1	1	2025-01-22	221
2	[NULL]	2025-01-23	1 555,5

Таблица 6. Таблица *sale_items* (Позиции продаж)

123 ↗ sale_item_id ▼	123 ↗ sale_id ▼	123 ↗ medicine_id ▼	123 ↗ quantity ▼	123 ↗ unit_price ▼
1	1	1	2	50,5
2	1	2	1	120
3	2	3	3	85
4	2	1	1	50,5
5	2	4	5	250

1. Основы соединения таблиц (JOIN)

1.1 Псевдонимы (aliases) – необходимость, а не просто удобство

Псевдонимы (или алиасы) – это временные имена, которые присваиваются таблицам или столбцам внутри одного SQL-запроса. Хотя в простых запросах с двумя таблицами их польза может быть неочевидна, по мере усложнения запросов они становятся мощным инструментом, решающим три ключевые задачи.

1. Повышение читаемости и сокращение кода

В запросах, соединяющих 3 и более таблиц, многократное использование полных имен делает код громоздким и трудным для восприятия.

Псевдонимы позволяют сделать его значительно короче и нагляднее.

2. Краткость при разрешении неоднозначности имен столбцов (Ambiguity)

В реальных базах данных таблицы часто имеют одинаковые имена столбцов (например, id, name). При соединении таких таблиц СУБД выдаст ошибку `ambiguous column name`, так как не сможет определить, столбец какой именно таблицы имеется в виду. Хотя эту проблему можно решить, указывая полное имя (`medicines.name`), общепринятой практикой является использование псевдонимов (`m.name`), так как это сохраняет запрос лаконичным.

3. Синтаксическая необходимость в само-соединениях (SELF-JOIN)

Это случай, где псевдонимы **абсолютно необходимы**. При соединении таблицы с **ней же самой** (например, для поиска руководителя для каждого сотрудника в таблице `pharmacists`), СУБД должна иметь возможность различать два экземпляра одной и той же таблицы.

Псевдонимы предоставляют уникальные имена для каждого экземпляра в рамках запроса.

Листинг 1.

```
-- Этот код синтаксически невозможен без псевдонимов 'e' и 'm'
SELECT
    ...
FROM
    -- Таблица в роли "сотрудников"
    pharmacists AS e
LEFT JOIN
    -- Та же таблица в роли "менеджеров"
    pharmacists AS m ON e.manager_id = m.pharmacist_id;
```

Таким образом, псевдонимы эволюционируют от простого удобства в коротких запросах до критически важного инструмента для обеспечения читаемости, ясности и функциональности в сложном аналитическом SQL-коде.

1.2 INNER JOIN (*внутреннее соединение*)

INNER JOIN является наиболее распространенным типом соединения.

Он возвращает только те строки, для которых было найдено совпадение в **обеих соединяемых таблицах** на основе указанного условия (**ON**).

Если для строки в одной таблице нет соответствующей строки в другой, обе эти строки в результат не попадают.

Листинг 2. Пример INNER JOIN

```
-- Вывести количество лекарств, выпущенных каждым производителем
-- (только для производителей, у которых есть лекарства)
SELECT
    man.name AS manufacturer_name,
    COUNT(med.manufacturer_id) AS medicines_count
FROM
    manufacturers AS man
INNER JOIN
    medicines AS med
ON
    man.manufacturer_id = med.manufacturer_id
GROUP BY
    man.manufacturer_id;
```

Этот запрос сначала находит все пары "производитель-лекарство", для которых есть совпадение по manufacturer_id.

Производители без лекарств и лекарства без производителей в этот промежуточный результат не попадают.

Затем оператор **GROUP BY** объединяет все строки с одинаковым названием производителя, а функция **COUNT(med.id)** подсчитывает, сколько лекарств (med.id) оказалось в каждой такой группе.

	A-Z manufacturer_name	123 medicines_count
1	Bayer AG	2
2	ООО "Фармстандарт"	3

Рисунок 1 – Результат запроса INNER JOIN

1.3 LEFT OUTER JOIN (левое внешнее соединение)

LEFT [OUTER] JOIN возвращает все строки из левой таблицы и только совпадающие строки из правой таблицы. Если для строки из левой таблицы не найдено совпадения в правой, то столбцы, относящиеся к правой таблице, будут заполнены значениями **NULL**.

«**Левой**» считается таблица, указанная сразу после **FROM**.

«Правой» – таблица, указанная после **LEFT JOIN**. Этот тип соединения полезен, когда нужно получить полный список чего-либо и дополнить его связанной информацией, если она есть.

Листинг 3. Пример LEFT JOIN

```
-- Вывести всех производителей и количество лекарств у каждого
-- (включая тех, у кого нет лекарств)
SELECT
    man.name AS manufacturer_name,
    COUNT(med.manufacturer_id) AS medicines_count
FROM
    manufacturers AS man
LEFT JOIN
    medicines AS med
ON
    man.manufacturer_id = med.manufacturer_id
GROUP BY
    man.manufacturer_id
ORDER BY
    medicines_count DESC;
```

Запрос берет всех производителей из «левой» таблицы manufacturers. Для каждого из них он ищет соответствия в medicines.

Если у производителя (например, у «Sanofi») нет лекарств, то для него не будет найдено ни одной совпадающей строки, и при подсчете **COUNT**(med.id) вернет 0 (так как функция **COUNT** не считает **NULL** значения, которые подставляются для отсутствующих лекарств).

Таким образом, мы получаем полный список производителей с количеством их лекарств, даже если это количество равно нулю.

	A-Z manufacturer_name	123 medicines_count
1	ООО "Фармстандарт"	3
2	Bayer AG	2
3	Sanofi	0

Рисунок 2 – Результат запроса **LEFT JOIN**

1.4 RIGHT OUTER JOIN (*правое внешнее соединение*)

RIGHT [OUTER] JOIN является зеркальным отражением **LEFT JOIN**. Он возвращает все строки из **правой** таблицы и совпадающие из **левой**. Если совпадения нет, **NULL** будут подставлены в столбцы левой таблицы.

На практике **RIGHT JOIN** используется редко, так как любой такой запрос можно переписать с помощью **LEFT JOIN**, просто поменяв таблицы местами, что часто делает логику более интуитивной.

Листинг 4. Пример RIGHT JOIN

```
-- Найти лекарства, у которых нет указанного производителя в базе
SELECT
    med.name AS medicine_name,
    med.manufacturer_id
FROM
    manufacturers AS man
RIGHT JOIN
    medicines AS med
ON
    man.manufacturer_id = med.manufacturer_id
WHERE
    man.manufacturer_id IS NULL;
```

Этот запрос демонстрирует паттерн «анти-соединение».

Сначала **RIGHT JOIN** берет все лекарства из «правой» таблицы `medicines` и пытается присоединить к ним производителей. Для лекарств без производителя (как 'Плацебо') поля из таблицы `manufacturers` (включая `man.manufacturer_id`) будут **NULL**.

Затем условие **WHERE man.manufacturer_id IS NULL** отфильтровывает все строки, где производитель нашелся, оставляя только те лекарства, которые остались «без пары».

	A-Z medicine_name	123 manufacturer_id
1	Плацебо	[NULL]

Рисунок 3 – Результат запроса **RIGHT JOIN**

1.5 FULL OUTER JOIN (*полное внешнее соединение*)

FULL [OUTER] JOIN объединяет в себе логику **LEFT** и **RIGHT JOIN**. Он возвращает все строки из обеих таблиц.

Если для строки из одной таблицы нет совпадения в другой, недостающие столбцы заполняются **NULL**.

Этот тип соединения можно представить как объединение результатов **LEFT JOIN** и **RIGHT JOIN**.

Листинг 5. Пример FULL JOIN

```
-- Проанализировать полную картину по всем производителям и лекарствам
SELECT
    COALESCE(man.name, 'Не указан') AS manufacturer_name,
    COUNT(med.id) AS medicines_count
FROM
    manufacturers AS man
FULL JOIN
    medicines AS med ON man.manufacturer_id = med.manufacturer_id
GROUP BY
    man.manufacturer_id, man.name
ORDER BY
    medicines_count DESC;
```

FULL JOIN гарантирует, что в промежуточный результат попадут и производители без лекарств (как 'Sanofi'), и лекарства без производителей (как 'Плацебо'). Затем оператор **GROUP BY** собирает все эти строки в группы по названию производителя.

Для лекарств без производителя (у которых `man.manufacturer_name` будет **NULL**) создается отдельная группа. Функция **COALESCE** используется для красивого вывода: она заменяет этот **NULL** на понятную строку 'Производитель не указан'.

Наконец, **COUNT(med.id)** подсчитывает количество лекарств в каждой группе, давая полную картину по всему ассортименту.

	A-Z manufacturer_name ▼	123 medicines_count ▼
1	ООО "Фармстандарт"	3
2	Bayer AG	2
3	Не указан	1
4	Sanofi	0

Рисунок 4 – Результат запроса FULL JOIN

1.6 CROSS JOIN (декартово произведение)

CROSS JOIN создает декартово произведение двух таблиц: каждая строка из первой таблицы соединяется с каждой строкой из второй.

В результате получается набор всех возможных комбинаций строк. Этот тип соединения **не требует условия «ON»**.

ВАЖНО: использовать CROSS JOIN следует с большой осторожностью, так как количество строк в результате равно произведению количества строк в исходных таблицах. На больших таблицах это может привести к огромному результату и **исчерпанию ресурсов сервера**.

Листинг 6. Пример CROSS JOIN

```
-- Вывести все возможные пары "покупатель-производитель"
SELECT
    c.last_name,
    m.name
FROM
    customers AS c
CROSS JOIN
    manufacturers AS m;
```

Запрос вернет таблицу со всеми возможными комбинациями покупателей и производителей. Если у нас 2 покупателя и 3 производителя, итоговая таблица будет содержать $2 * 3 = 6$ строк.

Основное применение этого типа запросов – генерация тестовых данных или сложных комбинаторных наборов.

	A-Z last_name ▼	A-Z name ▼
1	Иванов	ООО "Фармстандарт"
2	Петров	ООО "Фармстандарт"
3	Иванов	Bayer AG
4	Петров	Bayer AG
5	Иванов	Sanofi
6	Петров	Sanofi

Рисунок 5 – Результат запроса CROSS JOIN

2. Продвинутые техники и паттерны соединений

2.1 Соединение трех и более таблиц

Для получения комплексных отчетов часто требуется соединять более двух таблиц.

Логика здесь проста: СУБД последовательно выполняет соединения.

Сначала соединяются первые две таблицы, и их результат рассматривается как одна временная, виртуальная таблица.

Затем эта виртуальная таблица соединяется со следующей, и так далее, пока все соединения не будут выполнены.

Листинг 7.

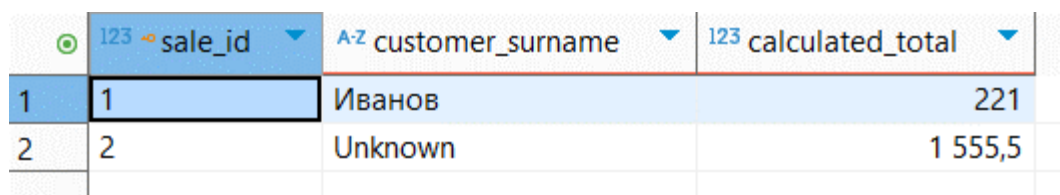
```
-- Вывести список всех продаж, включая фамилию покупателя и
-- рассчитанную общую стоимость
SELECT
    s.sale_id,
    COALESCE(c.last_name, 'Unknown') AS customer_surname,
    SUM(si.quantity * si.unit_price) AS calculated_total
FROM
    sales AS s
LEFT JOIN
    customers AS c ON s.customer_id = c.customer_id
JOIN
    sale_items AS si ON s.sale_id = si.sale_id
GROUP BY s.sale_id, c.last_name
ORDER BY s.sale_id;
```

1. **sales LEFT JOIN customers:** берутся все продажи и к ним присоединяются покупатели. Если продажа была "анонимной" (`customer_id = NULL`), поля покупателя будут **NULL**.
2. **(результат) JOIN sale_items:** далее к полученному результату присоединяются позиции продаж. Это означает, что если у продажи по какой-то причине нет позиций в `sale_items`, такая продажа не попадёт в итоговый результат.
Обратите внимание, что **JOIN** без указания типа по умолчанию является **INNER JOIN**
3. **GROUP BY** и **SUM:** результаты группируются по ID продажи, и для каждой группы вычисляется итоговая сумма.

В этом запросе функция **COALESCE**(`c.last_name`, 'Unknown') используется для обработки "анонимных" продаж, у которых `customer_id` равен **NULL**.

Она проверяет значение `c.last_name`. Если оно не **NULL**, функция возвращает его.

Если же оно **NULL**, функция возвращает второе указанное значение, то есть строку 'Unknown', делая отчет более читаемым.



	123 sale_id	A-Z customer_surname	123 calculated_total
1	1	Иванов	221
2	2	Unknown	1 555,5

Рисунок 6 – Результат запроса

2.2 Паттерн «Анти-соединение» (*anti-Join*)

Одной из самых частых задач является поиск записей, у которых отсутствуют связи (например, клиенты без заказов или товары, которые никогда не продавались).

Эту задачу эффективно решает паттерн «анти-соединение», который является комбинацией **LEFT JOIN** и условия **WHERE... IS NULL**.

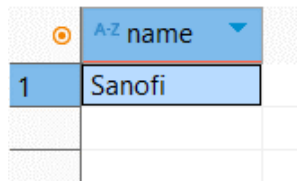
Листинг 8.

```
-- Найти производителей, у которых нет ни одного лекарства
SELECT man.name
FROM
    manufacturers AS man
LEFT JOIN
    medicines AS med
ON man.manufacturer_id = med.manufacturer_id
-- Оставляем только тех производителей,
-- для которых не нашлось ни одного лекарства
WHERE med.id IS NULL;
```

Запрос сначала с помощью **LEFT JOIN** пытается найти пару каждому производителю из таблицы `medicines`.

Для тех производителей, кому пара не нашлась, в столбцах таблицы `medicines` будет **NULL**.

Затем условие **WHERE med.id IS NULL** отфильтровывает все успешные совпадения, оставляя в результате только тех производителей, у которых нет лекарств.



	A-Z name
1	Sanofi

Рисунок 7 – Результат запроса

2.3 Само-соединение (Self-Join)

Иногда возникает необходимость соединить таблицу с ней же самой.

Это может потребоваться для работы с иерархическими данными, хранящимися в одной таблице, где одна запись ссылается на другую в той же таблице (например, сотрудник и его руководитель).

Листинг 9.

```
-- Вывести для каждого фармацевта имя его руководителя
SELECT
    e.first_name || ' ' || e.last_name AS employee,
    m.first_name || ' ' || m.last_name AS manager
FROM
    pharmacists AS e
LEFT JOIN
    pharmacists AS m ON e.manager_id = m.pharmacist_id;
```

123 pharmacist_id	A-Z first_name	A-Z last_name	A-Z phone_number	123 manager_id
1	Иван	Сорокин	+7(905)123-44-56	[NULL]
2	Алексей	Петров	+7(905)123-44-56	1
3	Марина	Иванова	+7(905)123-44-56	1
4	Евгений	Смирнов	+7(905)123-44-56	2
5	Александр	Орлов	+7(905)123-44-56	3

Рисунок 8 – Исходная таблица

Запрос соединяет таблицу саму с собой по полю `manager_id`, находя для каждого сотрудника (**e**) соответствующую запись его руководителя (**m**).

LEFT JOIN используется для того, чтобы в результат попали и сотрудники верхнего уровня, у которых нет руководителя.

	A-Z employee	A-Z manager
1	Иван Сорокин	[NULL]
2	Алексей Петров	Иван Сорокин
3	Марина Иванова	Иван Сорокин
4	Евгений Смирнов	Алексей Петров
5	Александр Орлов	Марина Иванова

Рисунок 9 – Результат запроса

3. Теоретико-множественные операторы

В отличие от **JOIN**, которые «склеивают» таблицы по **горизонтали**, добавляя новые столбцы, **теоретико-множественные операторы** работают с наборами строк и «склеивают» их по **вертикали**, добавляя новые строки.

Главное правило: все запросы, объединяемые этими операторами, должны возвращать одинаковое количество столбцов, и типы данных в этих столбцах должны быть совместимы.

3.1 UNION

Оператор **UNION** объединяет результаты двух или более запросов в один набор данных, удаляя все дублирующиеся строки.

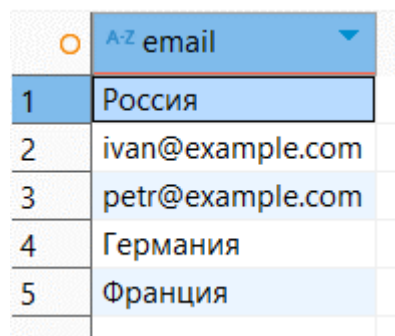
Если удаление дубликатов не требуется, можно использовать **UNION ALL**, который работает быстрее.

Листинг 10.

```
-- Получить единый список email-адресов покупателей
-- и названий стран производителей
SELECT
    email
FROM
    customers
UNION
SELECT
    country
FROM
    manufacturers;
```

Этот запрос выполняет две независимые выборки (email-адреса и страны) и объединяет их в один общий список.

Результатом будет одна колонка, содержащая и email, и названия стран.



	A-Z email
1	Россия
2	ivan@example.com
3	petr@example.com
4	Германия
5	Франция

Рисунок 10 – Результат запроса

3.2 INTERSECT

Оператор **INTERSECT** возвращает только те строки, которые присутствуют в результатах всех объединенных запросов.

Это – операция нахождения **пересечения множеств**.

Листинг 11.

```
-- Найти лекарства, которые есть в ассортименте и которые были проданы
SELECT
    name
FROM
```



```

medicines
INTERSECT
SELECT
    m.name
FROM
    medicines AS m
INNER JOIN
    sale_items AS si
ON m.id = si.medicine_id;

```

Запрос находит пересечение двух множеств: (1) полного списка названий лекарств и (2) списка названий только тех лекарств, которые были проданы.

В результате мы получаем уникальный список названий **проданных** лекарств.

	A-Z name
1	Витамин С
2	Парацетамол
3	Ибупрофен
4	Аспирин

Рисунок 11 – Результат запроса

3.3 EXCEPT

Оператор **EXCEPT** возвращает строки из первого запроса, которых нет в результате второго запроса.

Это операция нахождения **разности множеств**, где важен **порядок запросов**.

Листинг 12.

```

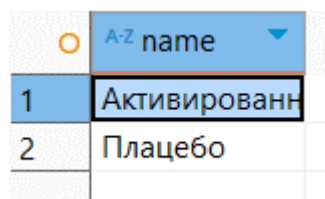
-- Найти лекарства, которые есть в ассортименте, но которые ни разу не
заказывали
SELECT
    name
FROM
    medicines
EXCEPT
SELECT
    m.name
FROM

```

```
medicines AS m
INNER JOIN
sale_items AS si
ON m.id = si.medicine_id;
```

Запрос «вычитает» из полного списка названий лекарств список названий проданных лекарств.

В результате мы получаем список лекарств, которые есть на складе, но еще ни разу не были проданы.



	A-Z name
1	Активированн
2	Плацебо

Рисунок 12 – Результат запроса

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Опишите бизнес-задачу, для решения которой оптимально подойдет **LEFT JOIN**, но будет некорректен **INNER JOIN**. Приведите пример SQL-запроса для вашей схемы данных.
2. В чем принципиальное отличие условия в предложении **ON** от условия в предложении **WHERE** при использовании **OUTER JOIN**?
Продemonстрируйте на примере, как перенос условия из **ON** в **WHERE** может кардинально поменять результат запроса (превратив **LEFT JOIN** в аналог **INNER JOIN**).
3. Объясните, почему для само-соединения (**SELF-JOIN**) использование псевдонимов таблицы является синтаксически обязательным. Что произойдет, если их не использовать?
4. Сравните операторы **UNION** и **UNION ALL**. В какой ситуации **UNION ALL** будет значительно производительнее и почему?
5. Можно ли получить результат, аналогичный **INTERSECT**, используя только операторы **JOIN**? Если да, то как? Если нет, то почему?

КРАТКИЙ СПРАВОЧНЫЙ МАТЕРИАЛ

1. Виды соединений (SQL JOINS)

Соединения (**JOIN**) – это основной инструмент для извлечения связанных данных из нескольких таблиц путем их объединения по горизонтали. Выбор типа соединения определяет, какие строки попадут в итоговый результат.

Ключевое слово **ON** указывает условие, по которому СУБД будет сопоставлять строки из двух таблиц

Например, «*ON medicines.manufacturer_id = manufacturers.id*».

Синтаксические сокращения:

- Слово **JOIN** без указания типа по умолчанию означает **INNER JOIN**.
- Ключевое слово **OUTER** в LEFT OUTER JOIN, RIGHT OUTER JOIN и FULL OUTER JOIN является **необязательным** и может быть **опущено** (LEFT JOIN, RIGHT JOIN, FULL JOIN).

Таблица 1. Сравнение типов JOIN

Тип соединения	Краткое описание	Основной сценарий использования
INNER JOIN	Возвращает только те строки, для которых найдено совпадение в обеих таблицах.	Получение строго связанных данных (<i>например, заказы и их клиенты</i>).
LEFT JOIN	Возвращает все строки из левой таблицы и только совпадающие из правой.	Получение полного списка чего-либо с дополнением (<i>например, все клиенты и их заказы, даже если заказов нет</i>).
RIGHT JOIN	Возвращает все строки из правой таблицы и только совпадающие из левой.	Когда фокус запроса на "правой" таблице. Например, "показать ВСЕ лекарства и добавить информацию об их производителях, если она есть". Гарантирует, что ни одно лекарство не будет потеряно, даже если у него нет

		производителя. Чаще всего такой запрос переписывают в более интуитивный LEFT JOIN, меняя таблицы местами.
FULL JOIN	Возвращает все строки из обеих таблиц, соединяя их там, где это возможно.	Полное сведение двух списков без потери данных (например, все сотрудники и все отделы, даже если в отделе нет сотрудников).
CROSS JOIN	Возвращает декартово произведение – все возможные комбинации строк из обеих таблиц. Внимание: может привести к "комбинаторному взрыву" на больших таблицах.	Генерация тестовых данных или комбинаторных наборов (например, все размеры для всех цветов товара).

2. Теоретико-множественные операторы

Эти операторы объединяют результаты двух или более запросов по вертикали, работая с наборами строк.

Главное условие – количество и типы столбцов в объединяемых запросах должны совпадать.

Таблица 2. Сравнение теоретико-множественных операторов

Оператор	Действие	Работа с дубликатами
UNION	Объединение: возвращает все строки из обоих запросов.	Удаляет дубликаты (<i>требует дополнительных ресурсов на проверку уникальности</i>).
UNION ALL	Объединение: возвращает абсолютно все строки из обоих запросов.	Сохраняет дубликаты (<i>работает значительно быстрее, предпочтителен, если дубликаты не мешают</i>).
INTERSECT	Пересечение: возвращает только те строки, которые есть в обоих запросах. Порядок запросов не имеет значения.	По определению возвращает уникальные строки.

EXCЕРТ	Разность: возвращает строки из первого запроса, которых нет во втором. Порядок запросов критически важен.	По определению возвращает уникальные строки.
---------------	---	--

3. Продвинутые паттерны и функции

3.1 Псевдонимы (Aliases)

Позволяет задать временные имена для таблиц и столбцов.

Где используются:

1. Повышения читаемости и сокращения кода в сложных запросах;
2. Разрешения неоднозначности имен столбцов;
3. являются синтаксически обязательными для само-соединений.

Листинг 1. Синтаксис

```
...
FROM
    table_name AS alias_name
-- ключевое слово AS является необязательным
```

3.2 Само-соединение (Self-Join)

Позволяет соединить таблицу с ней же самой для анализа иерархических связей внутри нее.

ВАЖНО: использование псевдонимов для каждого экземпляра таблицы является обязательным.

Листинг 2. Пример синтаксиса:

```
FROM
    pharmacists AS e
LEFT JOIN
    pharmacists AS m
ON e.manager_id = m.pharmacist_id.
```

3.3 Анти-соединение (Anti-Join)

Это эффективный паттерн для поиска записей в одной таблице, не имеющих связанных записей в другой.

Листинг 3. Пример использования

```
...  
FROM  
    table_A AS a  
LEFT JOIN  
    table_B AS b  
ON  
    a.key = b.key  
WHERE  
    b.key IS NULL
```

Принцип работы:

LEFT JOIN пытается найти совпадения. Там, где совпадений нет, поля правой таблицы (table_B) становятся NULL. Условие WHERE отфильтровывает все успешные совпадения, оставляя только "одиночные" записи из table_A.

3.4 Функция COALESCE

Возвращает первое **не-NULL** значение из списка переданных аргументов.

Часто используется для замены **NULL** на более понятные значения по умолчанию в отчетах.

Листинг 4. Пример использования

```
-- если last_name равен NULL, вернется строка 'Unknown'  
SELECT  
    COALESCE(c.last_name, 'Unknown')  
FROM  
    ...
```