Министерство науки и высшего образования РФ

федеральное государственное автономное  
образовательное учреждение высшего образования

«Омский государственный технический университет»

Факультет информационных технологий и компьютерных систем

Кафедра «Прикладная математика и фундаментальная информатика»

|  |
| --- |
| **Лабораторная работа №5** |
| по дисциплине **Системы управления базами данных**  **Тема: СОЗДАНИЕ ЗАПРОСОВ НА ВЫБОРКУ** |

|  |  |  |  |
| --- | --- | --- | --- |
| Студента | Кузиной Варвары Михайловны | | |
| Курс | 3 | Группа | МО-231 |
| Направление | 02.03.03 Математическое обеспечение и администрирование информационных систем | | |
| Руководитель | доц., к.н. | | |
|  | Моисеева Н.А. | | |
| Выполнил | 16.10 | | |
| Проверил |  | | |

Омск 2025

**Цель работы:**

Выполнить и продемонстрировать в БД *rgr\_store* типовые запросы SELECT: все типы соединений, DISTINCT, BETWEEN, LIMIT/OFFSET, LIKE/NOT LIKE, POSIX-регулярные выражения, агрегаты, подзапрос, CASE, группировка с сортировкой и с HAVING. Результаты подтвердить нумерованными скриншотами.

**Исходные данные:**

БД *rgr\_store*:employees, suppliers, product\_catalog, orders, order\_composition, receipts, check\_composition, invoices, invoice\_composition, а также product\_media (массивы) и receipt\_payments (JSONB). Связи реализованы внешними ключами с каскадами; заданы CHECK/NOT NULL/PK.

**2.1. Запрос с использованием всех типов соединений**

**2.1a) INNER JOIN**

**Смысл:**

взять только совпадающие строки из связанных таблиц.  
**SQL:**

SELECT o.order\_number, o.order\_date, p.product\_name, oc.product\_quantity

FROM public.orders o

JOIN public.order\_composition oc ON oc.order\_number = o.order\_number

JOIN public.product\_catalog p ON p.product\_code = oc.product\_code

ORDER BY o.order\_number, p.product\_name;

**Пояснение:**

соединяем заявки с их строками и товарами — видим только те товары, которые реально присутствуют в составе заявок (рисунок 1).

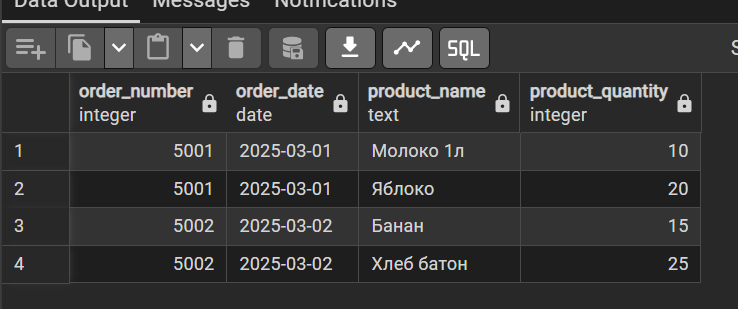


Рисунок 1 — Результат запроса с INNER JOIN

**2.1b) LEFT JOIN**

**Смысл:**

взять все строки **слева**, даже если справа нет пары.  
**SQL:**

SELECT r.receipt\_number, r.receipt\_date, e.emp\_name

FROM public.receipts r

LEFT JOIN public.employees e ON e.employee\_id = r.employee\_id

ORDER BY r.receipt\_number;

**Пояснение:**

все чеки показываются всегда; если кассир не найден — emp\_name будет NULL (рисунок 2).

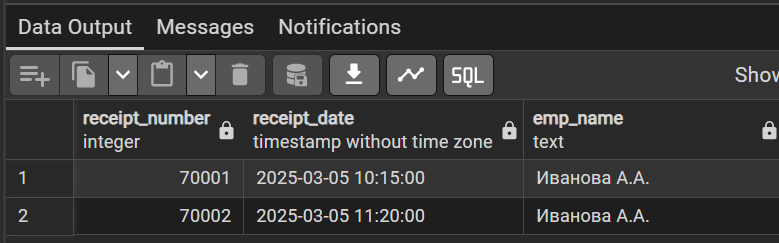


Рисунок 2 — Результат запроса с LEFT JOIN

**2.1c) RIGHT JOIN**

**Смысл:**

взять все строки **справа**, даже если слева нет пары.  
**SQL:**

SELECT ic.invoice\_number, p.product\_code, p.product\_name

FROM public.invoice\_composition ic

RIGHT JOIN public.product\_catalog p ON p.product\_code = ic.product\_code

ORDER BY p.product\_code;

**Пояснение:**

все товары выводятся даже при отсутствии строк в накладных (рисунок 3).

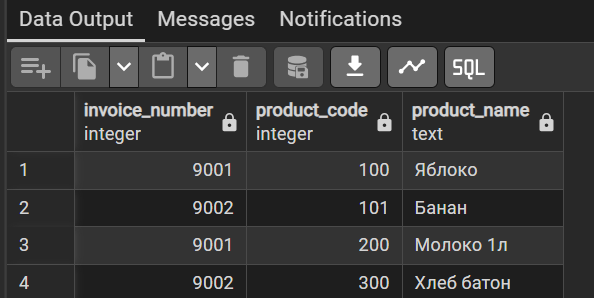


Рисунок 3 — Результат запроса с RIGHT JOIN

**2.1d) FULL OUTER JOIN**

**Смысл:**

взять всё слева и всё справа, заполняя пропуски NULL.  
**SQL:**

SELECT s.supplier\_code, s.supplier\_name, i.invoice\_number, i.invoice\_date

FROM public.suppliers s

FULL JOIN public.invoices i ON i.supplier\_code = s.supplier\_code

ORDER BY COALESCE(s.supplier\_code, -1), COALESCE(i.invoice\_number, -1);

**Пояснение:**

видны поставщики без накладных и накладные без поставщиков (если такие есть) (рисунок 4).

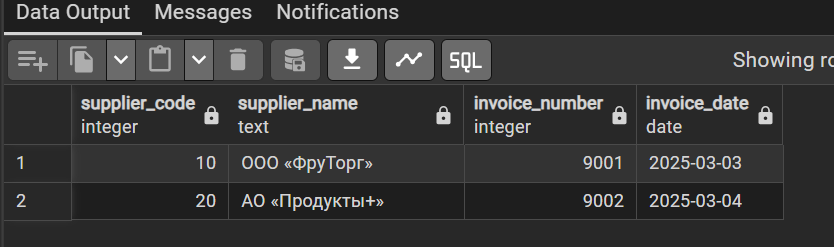


Рисунок 4 — Результат запроса с FULL OUTER JOIN

**2.1e) CROSS JOIN**

**Смысл:**

декартово произведение (каждый с каждым).  
**SQL:**

SELECT e.emp\_name, p.product\_name

FROM (SELECT \* FROM public.employees ORDER BY employee\_id LIMIT 2) e

CROSS JOIN (SELECT \* FROM public.product\_catalog ORDER BY product\_code LIMIT 2) p;

**Пояснение:**

2×2 = 4 комбинации — демонстрация работы CROSS JOIN на малом наборе (рисунок 5).

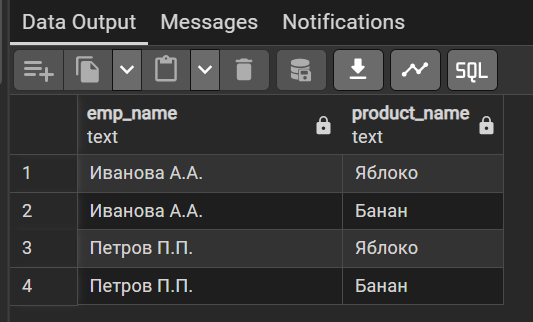
****

Рисунок 5 — Результат запроса с CROSS JOIN

**2.2.** **DISTINCT**

**Смысл:**

убрать дубликаты значений.  
**SQL:**

SELECT DISTINCT product\_category

FROM public.product\_catalog

ORDER BY product\_category;

**Пояснение:**

получаем список уникальных категорий товаров (рисунок 6).

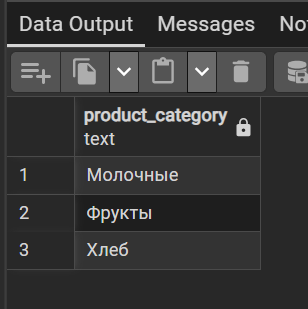


Рисунок 6 — Результат запроса с DISTINCT

**2.3. BETWEEN**

**Смысл:**

отбор по диапазону.  
**SQL:**

SELECT receipt\_number, receipt\_date

FROM public.receipts

WHERE receipt\_date BETWEEN TIMESTAMP '2025-03-05 00:00:00'

AND TIMESTAMP '2025-03-05 23:59:59'

ORDER BY receipt\_number;

**Пояснение:**

чеки за конкретные сутки (при необходимости подставь свою дату) (рисунок 7).

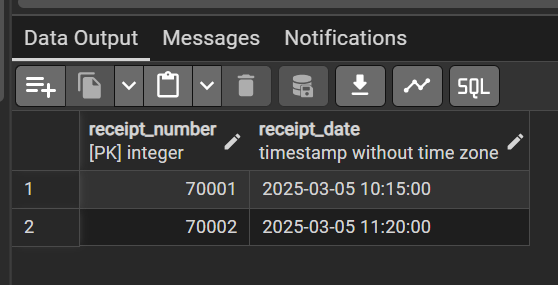


Рисунок 7 — Результат запроса с BETWEEN

**2.4. LIMIT**

**Смысл:**

ограничить количество строк результата.  
**SQL:**

SELECT product\_code, product\_name, product\_price

FROM public.product\_catalog

ORDER BY product\_price DESC

LIMIT 3;

**Пояснение:**

выводим «топ-3» по цене, чтобы точно было ≥1 строка (рисунок 8).

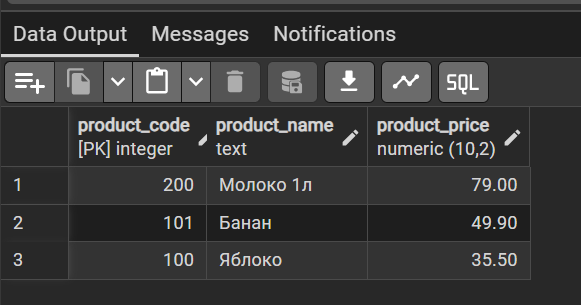


Рисунок 8 — Результат запроса с LIMIT

**2.5. OFFSET**

**Смысл:**

пропустить первые N строк и вывести следующие M.  
**SQL:**

SELECT product\_code, product\_name, product\_price

FROM public.product\_catalog

ORDER BY product\_code

LIMIT 2 OFFSET 1;

**Пояснение:**

пропускаем первую запись, показываем последующие две (рисунок 9).

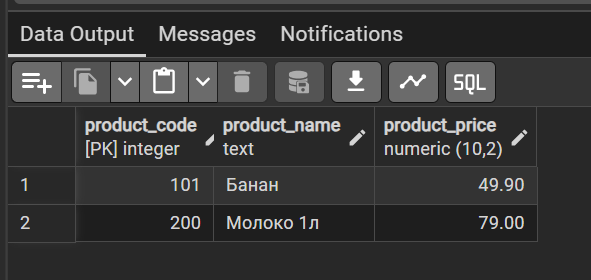


Рисунок 9 — Результат запроса с OFFSET

**2.6. LIKE / NOT LIKE**

**Смысл:**

шаблоны поиска по подстроке.  
**SQL:**

-- содержит «ло» (например, «Молоко»)

SELECT product\_code, product\_name

FROM public.product\_catalog

WHERE product\_name LIKE '%ло%';

-- исключаем «батон»

SELECT product\_code, product\_name

FROM public.product\_catalog

WHERE product\_name NOT LIKE '%батон%';

**Пояснение:**

% — «любой набор символов». Для регистронезависимого поиска можно использовать ILIKE (рисунок 10-11).

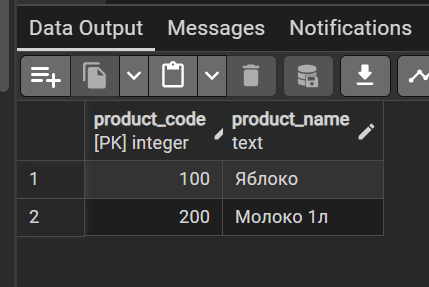


Рисунок 10 — Результат запроса с LIKE

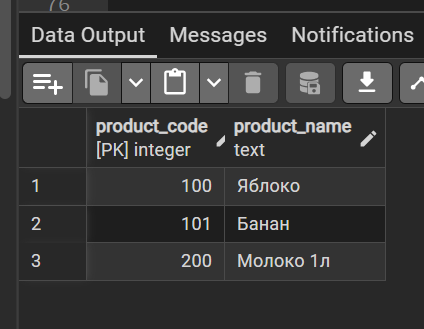


Рисунок 11 — Результат запроса с NOT LIKE

**2.7. POSIX-регулярные выражения**

**Смысл:**

поиск по шаблонам сложнее, чем LIKE.  
**SQL:**

-- начинаются с «Яблоко» или «Банан»

SELECT product\_code, product\_name

FROM public.product\_catalog

WHERE product\_name ~ '^(Яблоко|Банан)';

-- нечувствительно к регистру: фамилии на «…ова»

SELECT employee\_id, emp\_name

FROM public.employees

WHERE emp\_name ~\* 'ова\b';

**Пояснение:**

~ — с учётом регистра, ~\* — без учёта. ^ — начало строки, | — «или», \b — граница слова (рисунок 12 - 13).

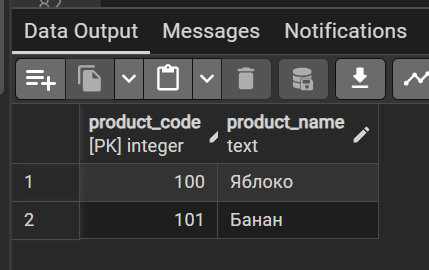


Рисунок 12 — Результат запроса с регулярным выражением ~ '^(Яблоко|Банан)'

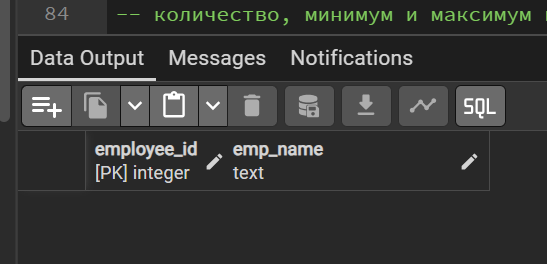


Рисунок 13 — Результат запроса с регулярным выражением ~\* 'ова\b' (не чувствительным к регистру)

**2.8. Агрегированные данные:**

**Смысл:**

сводные значения по набору строк.  
**SQL:**

-- количество, минимум и максимум цен

SELECT COUNT(\*) AS product\_cnt,

MIN(product\_price) AS min\_price,

MAX(product\_price) AS max\_price

FROM public.product\_catalog;

-- средняя цена по категориям

SELECT product\_category, ROUND(AVG(product\_price), 2) AS avg\_price

FROM public.product\_catalog

GROUP BY product\_category

ORDER BY product\_category;

**Пояснение:**

первые три — общие статистики, затем — средняя по группам (рисунок 14 – 15).

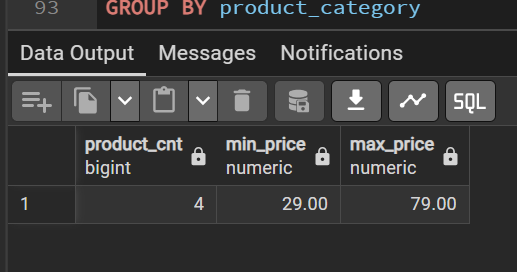


Рисунок 14 — Количество, максимум и минимум цен

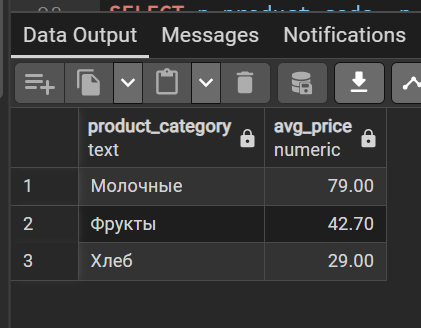


Рисунок 15 — Средняя цена по категориям

**2.9. Вложенный запрос**

**Смысл:**

использовать результат подзапроса для отбора.  
**SQL:**

-- товары, попадавшие в чеки

SELECT p.product\_code, p.product\_name

FROM public.product\_catalog p

WHERE EXISTS (

SELECT 1

FROM public.check\_composition cc

WHERE cc.product\_code = p.product\_code

)

ORDER BY p.product\_code;

-- товары, НЕ попадавшие в чеки

SELECT p.product\_code, p.product\_name

FROM public.product\_catalog p

WHERE NOT EXISTS (

SELECT 1

FROM public.check\_composition cc

WHERE cc.product\_code = p.product\_code

)

ORDER BY p.product\_code;

**Пояснение:**

EXISTS/NOT EXISTS проверяют факт наличия/отсутствия связанной строки (рисунок 16 – 17).

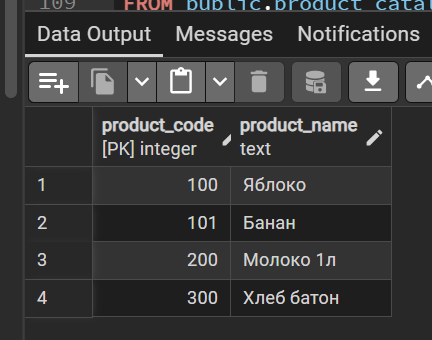


Рисунок 16 — Товары попавшие в чек

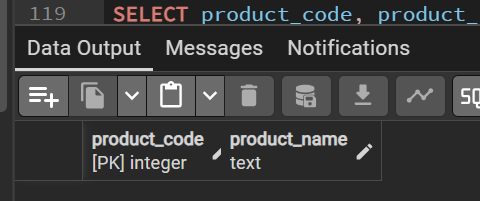


Рисунок 17 — Товары не попавшие в чек

**2.10. CASE**

**Смысл:**

вывести вычисляемую категорию по условию.  
**SQL:**

SELECT product\_code, product\_name, product\_price,

CASE

WHEN product\_price >= 100 THEN 'дорогой'

WHEN product\_price >= 50 THEN 'средний'

ELSE 'дешевый'

END AS price\_band

FROM public.product\_catalog

ORDER BY product\_price DESC;

**Пояснение:**

условное выражение присваивает «ярлык» товару по цене (рисунок 18).

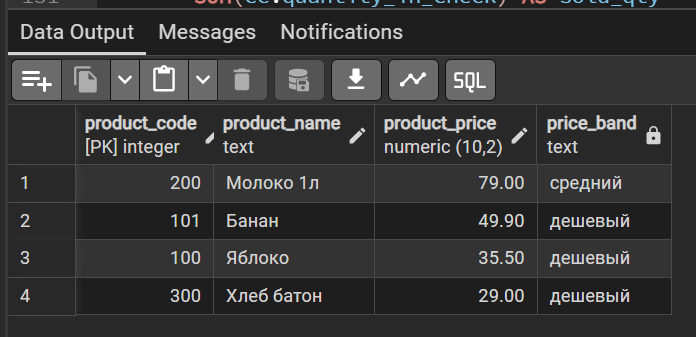


Рисунок 18 — Результат запроса с CASE

**2.11. Группировка и простая сортировка**

**Смысл:**

агрегирование по товарам с сортировкой.  
**SQL:**

SELECT p.product\_code, p.product\_name,

SUM(cc.quantity\_in\_check) AS sold\_qty

FROM public.product\_catalog p

JOIN public.check\_composition cc ON cc.product\_code = p.product\_code

GROUP BY p.product\_code, p.product\_name

ORDER BY sold\_qty DESC;

**Пояснение:**

сумма проданных штук по каждому товару; сортировка — по убыванию (рисунок 19).

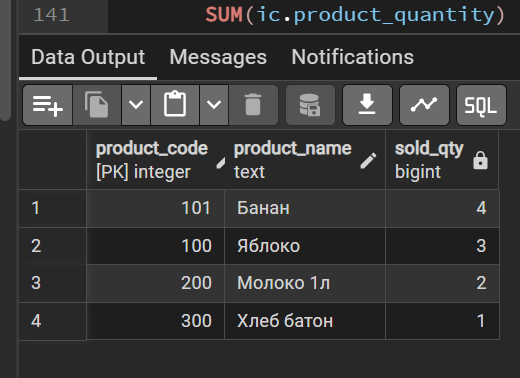


Рисунок 19 — Результат группировки и простой сортировки

**2.12. Группировка с условием и многоуровневой сортировкой**

**Смысл:**

отфильтровать группы по условию (HAVING) и отсортировать по нескольким полям.  
**SQL:**

SELECT s.supplier\_name,

p.product\_category,

SUM(ic.product\_quantity) AS total\_qty

FROM public.invoice\_composition ic

JOIN public.invoices i ON i.invoice\_number = ic.invoice\_number

JOIN public.suppliers s ON s.supplier\_code = i.supplier\_code

JOIN public.product\_catalog p ON p.product\_code = ic.product\_code

GROUP BY s.supplier\_name, p.product\_category

HAVING SUM(ic.product\_quantity) > 50

ORDER BY s.supplier\_name ASC, total\_qty DESC;

**Пояснение:**

считаем по поставщику и категории, оставляем только группы с суммой > 50; сортируем по имени поставщика и затем по убыванию суммы (рисунок 20).

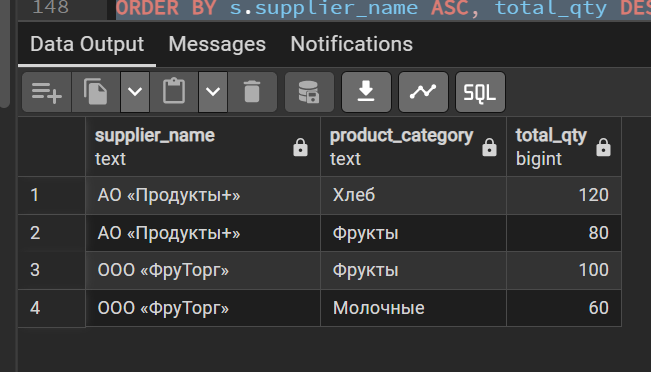


Рисунок 20 — Результат группировки с условием и многоуровневой сортировки

**Заключение:**

В рамках ЛР-6 для базы *rgr\_store* выполнен полный набор запросов согласно методичке: продемонстрированы все типы соединений (INNER/LEFT/RIGHT/FULL/CROSS), фильтрация DISTINCT/BETWEEN/LIKE/регулярные выражения, ограничение и смещение результатов (LIMIT/OFFSET), агрегирование (COUNT/MIN/MAX и AVG по группам), подзапросы (EXISTS/NOT EXISTS), условные выражения CASE, группировка с простой сортировкой и группировка с HAVING и многоуровневой сортировкой. Каждый запрос успешно отработал в pgAdmin; результаты подтверждены нумерованными скриншотами экрана.

Работа показала практическое использование ключевых возможностей языка SQL в PostgreSQL на учебной предметной области магазина с заказами, чеками, накладными и справочниками.