



МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«МИРЭА – Российский технологический университет»

РТУ МИРЭА „

**Институт информационных технологий (ИИТ)
Кафедра цифровой трансформации (ЦТ)**

ОТЧЕТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ
по дисциплине «Разработка баз данных»

Практическое занятие № 4

Студенты группы *ИКБО-66-23 Смирнов А.Ю.*

(подпись)

Ассистент *Копылова Я.А.*

(подпись)

Отчет представлен «___» _____ 2025 г.

Москва 2025 г.

Постановка задачи:

Для выполнения практической работы необходимо последовательно выполнить четыре задачи, используя собственную базу данных. Все примеры в данном документе основаны на демонстрационной базе данных «Аптека», содержащей таблицы manufacturers (производители), medicines (лекарства) и sales (продажи).

Ваша задача — адаптировать каждую из поставленных задач к логической структуре и предметной области вашей базы данных. Приведенные ниже формулировки и последующие примеры кода служат шаблоном для понимания, какой тип аналитического запроса требуется составить.

Задание №1: использование ранжирующих функций

Для каждой основной «родительской» сущности в вашей БД (например, производитель, категория товара, автор) определить три наиболее значимых по некоторому числовому признаку дочерних сущности (например, три самых дорогих товара, три самые популярные книги по количеству продаж). В результирующей таблице должны быть указаны идентификатор группы, идентификатор дочерней сущности, её числовой признак и ранг. Для расчёта ранга использовать функцию RANK() или DENSE_RANK().

Задание №2: использование агрегатных оконных функций

Для ключевой сущности, имеющей транзакции по времени (например, товар, услуга), рассчитать нарастающий итог (кумулятивную сумму) по некоторому показателю (например, объем продаж, количество заказов) с разбивкой по временным периодам (месяцам или годам). Отчёт должен содержать идентификатор сущности (id/название/...), временной период, сумму за период и кумулятивную сумму.

Задание №3: использование функции смещения

Провести сравнительный анализ общих показателей по периодам. Для каждого периода (например, месяца), начиная со второго, необходимо вывести

общий показатель за текущий период и аналогичный показатель за предыдущий период в одной строке. Это позволит наглядно оценить динамику. Необходимо использовать функцию LAG().

Задание №4: построение сводной таблицы

Создать сводный отчет, который агрегирует некоторый числовой показатель для основной сущности по категориям, представленным в виде столбцов.

Решение:

Задание №1: использование ранжирующих функций

- Запрос с DENSE_RANK для трех сотрудника с самой высокой зарплатой

The screenshot shows the Oracle SQL Developer interface. In the top-left pane, there are two tabs: 'Script1' and 'Script2'. The 'Script1' tab contains the following SQL code:

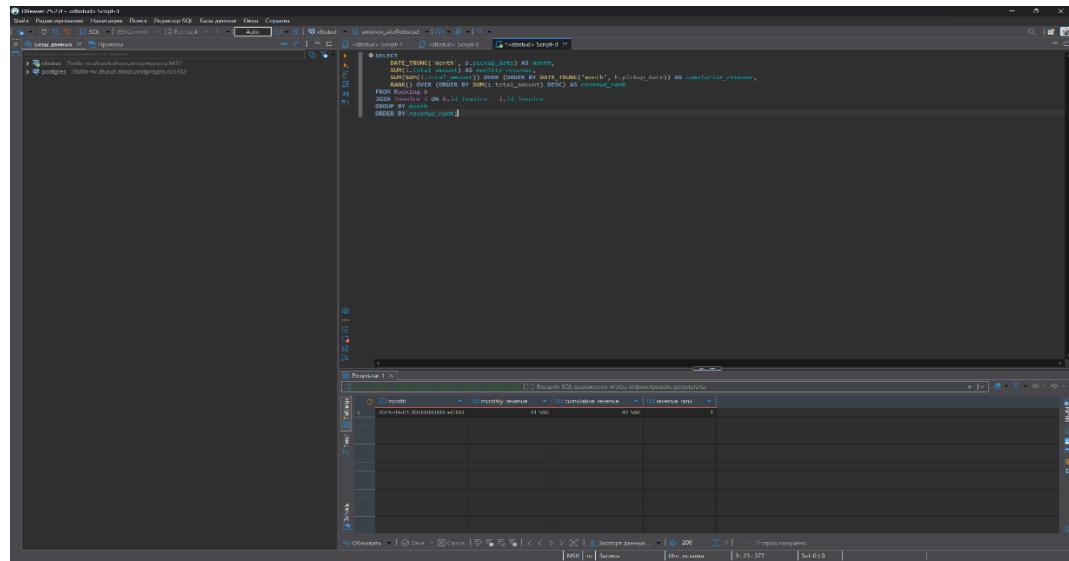
```
SELECT *
  FROM (
    SELECT
      c.id,
      c.location,
      c.address,
      c.city,
      c.id_employee,
      e.first_name,
      e.last_name,
      e.position,
      e.salary,
      DENSE_RANK() OVER (PARTITION BY c.id_location ORDER BY e.salary DESC) AS rank
    FROM Employee e
    JOIN location l ON e.id_location = l.id_location
  ) ranked_employees
 WHERE ranked_employees.rank <= 3
 ORDER BY id_employee, rank;
```

In the bottom-right pane, there is a table named 'employee(+)1' with the following data:

id	id_location	address	city	id_employee	first_name	last_name	position	salary	rank
1	LOC001	ул. Ленина, 122	Москва	EMP001	Анна	Иванова	Менеджер	80 000	1
2	LOC001	ул. Ленина, 123	Москва	EMP002	Игорь	Смирнов	Администратор	60 000	2
3	LOC002	пр. Мира, 45	Санкт-Петербург	EMP003	Мария	Петрова	Агент	50 000	1
4	LOC003	ул. Сакко и Ванцетти, 67	Казань	EMP004	Дмитрий	Коликов	Менеджер	75 000	1

Рисунок 1 — Запрос DENSE_RANK

Задание №2: использование агрегатных оконных функций



The screenshot shows the Oracle SQL Developer interface. In the top-left pane, there's a tree view of database objects. The main area contains a SQL editor window with the following query:

```
SELECT
    DATE_TRUNC('month', p.actual_date) AS month,
    SUM(p.monthly_revenue) OVER (ORDER BY DATE_TRUNC('month', p.actual_date)) AS cumulative_revenue,
    RANK() OVER (ORDER BY monthly_revenue DESC) AS revenue_rank
FROM Bookings p
WHERE p.actual_date >= DATEADD(MONTH, -12, GETDATE())
GROUP BY month
ORDER BY cumulative_revenue;
```

Below the editor is a results grid titled "Показатель 1" (Indicator 1). It has three columns: "month", "monthly_revenue", and "cumulative_revenue". The first row shows data for January 2024.

month	monthly_revenue	cumulative_revenue
2024-01-01 00:00:00.000	41 740	41 740
2024-02-01 00:00:00.000	41 740	41 740
2024-03-01 00:00:00.000	41 740	41 740
2024-04-01 00:00:00.000	41 740	41 740
2024-05-01 00:00:00.000	41 740	41 740
2024-06-01 00:00:00.000	41 740	41 740
2024-07-01 00:00:00.000	41 740	41 740
2024-08-01 00:00:00.000	41 740	41 740
2024-09-01 00:00:00.000	41 740	41 740
2024-10-01 00:00:00.000	41 740	41 740
2024-11-01 00:00:00.000	41 740	41 740
2024-12-01 00:00:00.000	41 740	41 740

Рисунок 2 — Общая кумулятивная выручка по месяцам

Задание №3: использование функции смещения Lag.

The screenshot shows the DBVisualizer interface with a dark theme. In the top-left corner, there's a tree view of databases: 'Базы данных' (dbsuid, postgres) and 'Проекты'. Below it, a connection status bar shows 'dbsuid' and 'postgres'. The main area has three tabs: 'Script 1', 'Script 2', and 'Script 3'. 'Script 3' contains the following SQL code:

```
-- Сравнение ежемесячной выручки с предыдущим месяцем
WITH monthly_revenue AS (
  SELECT
    DATE_TRUNC('month', i.issue_date) AS month,
    SUM(i.total_amount) AS monthly_total
  FROM Invoice i
  GROUP BY DATE_TRUNC('month', i.issue_date)
)
SELECT
  month,
  monthly_total,
  LAG(monthly_total, 1) OVER (ORDER BY month) AS previous_month_total,
  monthly_total - LAG(monthly_total, 1) OVER (ORDER BY month) AS difference
FROM monthly_revenue
ORDER BY month;
```

Below the code is a results pane titled 'Результат 1' (Result 1). It displays a single row of data:

month	monthly_total	previous_month_total	difference
2024-05-01 00:00:00.000 +0300	41 580	[NULL]	[NULL]

The status bar at the bottom right shows 'DBeaver Уведомление' with the message 'Datasource was invalidated', 'Live connection count: 5/5', and system icons for battery, signal, and time.

Рисунок 3 — Таблица Сравнение ежемесячной выручки с предыдущим месяцем

Задание №4: построение сводной таблицы

The screenshot shows the DBeaver interface with a SQL script editor and a results grid.

SQL Script:

```
-- Суммы платежей по способам оплаты по кварталам
SELECT
    p.payment_method_name,
    SUM(CASE WHEN EXTRACT(QUARTER FROM p.payment_date) = 1 THEN p.amount ELSE 0 END) AS q1,
    SUM(CASE WHEN EXTRACT(QUARTER FROM p.payment_date) = 2 THEN p.amount ELSE 0 END) AS q2,
    SUM(CASE WHEN EXTRACT(QUARTER FROM p.payment_date) = 3 THEN p.amount ELSE 0 END) AS q3,
    SUM(CASE WHEN EXTRACT(QUARTER FROM p.payment_date) = 4 THEN p.amount ELSE 0 END) AS q4,
    SUM(p.amount) AS total_payments
FROM Payment p
JOIN Payment_Method pm ON p.id_payment_method = pm.id_payment_method
GROUP BY p.payment_method_name
ORDER BY pm.method_name;
```

Results Grid:

method_name	q1	q2	q3	q4	total_payments
Дебетовая карта	0	0	11 088	0	11 088
Кредитная карта	0	0	13 860	0	13 860

Рисунок 4 — Количество бронирований по локациям (SUM + CASE)

The screenshot shows the DBeaver interface with a SQL script editor and a results grid.

SQL Script:

```
CREATE EXTENSION IF NOT EXISTS tablefunc;
SELECT
    p.payment_method_name,
    EXTRACT(QUARTER FROM p.payment_date) AS quarter,
    SUM(p.amount) AS amount
FROM Payment p
JOIN Payment_Method pm ON p.id_payment_method = pm.id_payment_method
CROSS TABULATE p.payment_date AS month, EXTRACT(QUARTER FROM p.payment_date)
ORDER BY 1, 2;
```

Results Grid:

method_name	q1	q2	q3	q4
Дебетовая карта	0	0	11 088	0
Кредитная карта	0	0	13 860	0

Рисунок 4 — Метод с crosstab()

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Объясните принципиальное различие в результате работы запроса с $\text{SUM}(\dots) \text{ OVER } (\text{PARTITION BY} \dots)$ и запроса с $\text{SUM}(\dots)$ и GROUP BY . В какой ситуации каждый из них предпочтителен?

Принципиальное различие между $\text{SUM}(\dots) \text{ OVER } (\text{PARTITION BY} \dots)$ и $\text{SUM}(\dots) \text{ ... GROUP BY}$ заключается в том, что GROUP BY агрегирует строки и возвращает одну строку на каждую группу, тогда как оконная функция $\text{SUM}()$ OVER вычисляет результат для каждой строки, не сворачивая данные. То есть GROUP BY сокращает количество строк, а оконная функция просто добавляет вычисленное значение в каждую из них. Использование GROUP BY предпочтительно, когда нужен итог по группам, а оконная функция — когда важно сохранить детали каждой записи.

2. В чем разница между функциями $\text{RANK}()$ и $\text{DENSE_RANK}()$ при наличии одинаковых значений в столбце, по которому идет сортировка? Приведите бизнес-сценарий, в котором использование $\text{DENSE_RANK}()$ будет более корректным.

Разница между $\text{RANK}()$ и $\text{DENSE_RANK}()$ проявляется при наличии одинаковых значений в сортируемом столбце. Функция $\text{RANK}()$ присваивает одинаковые ранги для одинаковых значений, но пропускает следующий номер (например, 1, 2, 2, 4). $\text{DENSE_RANK}()$ также присваивает одинаковый ранг, но не делает пропуска (1, 2, 2, 3). В бизнес-сценариях, где важно иметь последовательную нумерацию без пропусков, например при формировании списка лучших трёх товаров по продажам, $\text{DENSE_RANK}()$ будет более корректным.

3. Почему нельзя использовать оконную функцию непосредственно в предложении WHERE (например, WHERE price > AVG(price) OVER (...))?
Опишите, как можно обойти это ограничение, используя CTE или подзапрос.

Оконные функции нельзя использовать непосредственно в предложении WHERE, потому что WHERE выполняется **до** вычисления оконных выражений. На момент фильтрации окно ещё не рассчитано, и СУБД не знает его значение. Чтобы обойти это ограничение, можно вынести вычисление оконной функции в подзапрос или обобщённое табличное выражение (CTE), а затем применить WHERE уже к результату этого подзапроса, где значение функции доступно.

4. Какие два основных SQL-запроса требуются в качестве аргументов для функции crosstab? Каково назначение каждого из них?

Функция crosstab требует два запроса в качестве аргументов. Первый запрос возвращает тройки значений — категорию строк (например, сотрудник), категорию столбцов (например, квартал) и числовой показатель (например, сумма продаж). Второй запрос задаёт возможные категории для столбцов (например, 1, 2, 3, 4 для кварталов). Первый формирует данные, второй определяет структуру итоговой таблицы.