



Кафедра ЦТ
Институт информационных технологий
РТУ МИРЭА



Дисциплина «Разработка баз данных»

Практическая работа №5.

Объекты базы данных: представления и хранимые процедуры



Постановка задачи: основываясь на индивидуальной схеме данных, составьте необходимые запросы:

Задание №1: создание модифицируемого представления

Создать простое **модифицируемое представление**, которое отбирает строки из одной таблицы по определенному критерию.

Задание №2: модификация данных через представление *(два запроса)*

Продemonстрировать возможность изменения данных в базовой таблице через представление из Задания №1.

Создать **два запроса** – **INSERT** и **DELETE**.

Задание №3: создание немодифицируемого аналитического представления

Создать единое **немодифицируемое представление** для аналитических целей.

Оно должно объединять данные (*минимум 2 таблицы*) и содержать **агрегирующие функции** (**COUNT**, **SUM** и т.д.) и **GROUP BY**.

(продолжение на следующих слайдах – 1/3)

Практическая работа №5.

Объекты базы данных: представления и хранимые процедуры



Постановка задачи: основываясь на индивидуальной схеме данных, составьте необходимые запросы:

Задание №4: использование аналитического представления в запросах

Написать **SELECT-запрос**, который использует созданное в **Задании №3** аналитическое представление в качестве источника данных для дальнейшей фильтрации или анализа.

Задание №5: Создание и обновление материализованного представления (**два запроса**)

1. **Создать материализованное представление** для ускорения выполнения ресурсоемкого аналитического запроса.
2. Продемонстрировать процесс **обновления данных** (**REFRESH MATERIALIZED VIEW**).

Задание №6: разработка пользовательской функции для аналитических вычислений (**два запроса**)

1. **Разработать пользовательскую функцию**, которая инкапсулирует комплексный аналитический расчет (например, принимает ID и возвращает скалярное значение, вычисленное на основе **JOIN** и **SUM**).
2. Продемонстрировать **вызов функции**.

(продолжение на следующем слайде, начало на предыдущем слайде – 2/3)

Практическая работа №5.

Объекты базы данных: представления и хранимые процедуры



Постановка задачи: основываясь на индивидуальной схеме данных, составьте необходимые запросы:

Задание №7: разработка хранимой процедуры для выполнения сложной операции

Разработать **хранимую процедуру**, выполняющую **безопасную** операцию по изменению данных (*например, принимает ID и количество*).

Внутри процедуры **должна быть проверка** (*например, достаточно ли остатков*).

- Если **да** – выполнить **DML-операции** (*например, UPDATE и INSERT*).
- Если **нет** – прерваться. Использовать **выходной параметр** для статуса.

Задание №8: демонстрация вызова хранимой процедуры (**два запроса**)

Привести **два примера** вызова процедуры из **Задания №7**:

- **Успешный**
- **Неудачный** (*демонстрирующий срабатывание проверки*).

(начало на предыдущих слайдах – 3/3)

Представления (Views)



Что такое представление?

- **Виртуальная таблица**, основанная на **сохраненном SELECT-запросе**.
- **Не хранит** данные физически (за исключением *материализованных представлений*).

Каждый раз, при выполнении **любого** запроса, например:

```
SELECT * FROM my_view;
```

СУБД в этот момент выполняет **сложный запрос**, **лежащий в основе** my_view.

Зачем нужны представления?

- ✓ **Упрощение**: спрятать сложный запрос с **JOIN** и аналитикой в **одно простое имя**.
- ✓ **Безопасность**: дать пользователю доступ только к **части данных**.
- ✓ **Независимость**: структура **базовых таблиц** может **меняться**, но представление **сохранит** прежний **интерфейс** для приложения.

Представления (Views) – типы представлений



Тип представления	Хранение данных	Актуальность данных	Основной сценарий использования
Немодифицируемое (View)	Не хранит (виртуальное)	Всегда актуальные (запрос выполняется «на лету»)	Упрощение сложных запросов, контроль доступа (безопасность).
Модифицируемое (Updatable View)	Не хранит (виртуальное)	Позволяет INSERT, UPDATE, DELETE, которые транслируются в базовую таблицу.	Предоставление безопасного и простого интерфейса для изменения подмножества данных.
Материализованное (Materialized View)	Хранит физически (копия результата на диске)	Данные могут быть неактуальными . Требует ручного обновления (REFRESH)	Ускорение выполнения «тяжёлых» аналитических запросов, не требующих данных в реальном времени.

Представления (Views) – создание



Для создания **виртуальной таблицы**, основанной на **сохраненном SELECT-запросе**, используется **CREATE VIEW**.

Общий синтаксис:

CREATE [OR REPLACE] VIEW

view_name [(column1, column2, ...)] **AS**

SELECT

table_column1, table_column2, ...

FROM

имя_исходной_таблицы

[WHERE условия]

[WITH CHECK OPTION];

Краткое описание:

CREATE VIEW view_name – создает новое представление с именем.

[OR REPLACE] – если этой опции нет, попытка создать представление с тем же именем вызовет ошибку.

view_name [(column1, column2, ...)] – список **псевдонимов** для столбцов представления.

AS – ключевое слово, **отделяющее определение** представления от **SELECT**.

SELECT ... FROM ... (обязательно) – сам **SELECT-запрос**, определяющий содержимое столбцов представления.

[WITH CHECK OPTION] (необязательно) применяется к **модифицируемым представлениям**. Запрещает выполнять операции **INSERT** или **UPDATE**, если новая или измененная строка перестанет соответствовать условию **WHERE** этого представления.

Представления (Views) – концепция работы



Представление работает **как посредник** между **пользователем** и **сложной** структурой данных.

Схема Потока:

1. Приложение/Пользователь **отправляет простой запрос** к представлению:

```
SELECT * FROM v_report;
```

2. Представление (**View**) **перехватывает этот запрос** и выполняет **свой**, заранее **сохраненный, сложный запрос** к базовым таблицам:

```
SELECT ... FROM table1 JOIN table2 ...
```

3. Базовые таблицы (*реально хранящиеся в базе*) **обрабатывают** сложный запрос и **возвращают данные** представлению.

4. Представление (**View**) **форматирует** полученные данные и **передает их** как итоговый **результат**.

5. Приложение/Пользователь **получает результат**, как если бы он **обращался к простой таблице**.

Представления (Views) – модифицируемые представления



Представление является **модифицируемым** (позволяет **INSERT, UPDATE, DELETE**), только если оно «простое».

НЕдопустимо (становится **НЕ** модифицируемым):

- Запрос **содержит GROUP BY, HAVING**.
- Запрос **содержит WITH (CTE), DISTINCT, LIMIT** или **OFFSET**.
- **SELECT** **содержит агрегатные** или **оконные функции**.
- **SELECT** **содержит вычисляемые столбцы** ($price * 1.1$ **AS** pr).
- Запрос **содержит** теоретико-множественные **операции** (**UNION, INTERSECT, EXCEPT**).
- **SELECT** **НЕ содержит id** (не сработают **UPDATE, DELETE**)

Допустимо:

- В **FROM** указана **одна таблица** (без **JOIN**).
- Используется **WHERE**.
- Переименовываются столбцы.
- Задана сортировку (**ORDER BY**).

Вывод: **СУБД** должна быть **способна сама написать** (переписать) ваш запрос под реальные таблицы, **иначе** представление **автоматически** становится **немодифицируемым** (только для чтения).

Представления (Views) – пример



Пример использования представлений можно увидеть в **DBeaver**.

Окно с результатами каждого вашего **SELECT** ведёт себя как **временное представление** – это **виртуальная таблица**, основанная на вашем **SELECT**-запросе.

DBeaver **пытается** сделать это «представление» **модифицируемым**, чтобы вы могли редактировать данные прямо в таблице.

SELECT * FROM sales;

SELECT * FROM sales Введите SQL выражение чтобы отфильтровать результаты				
Таблица	123 sale_id	123 customer_id	sale_date	123 total_amount
1	2	[NULL]	2025-01-23	1 555,5
2	1	1	2025-01-22	221

SELECT sale_date FROM sales;

SELECT sale_date FROM sales Введите SQL	
Таблица	sale_date
1	2025-01-23
2	2025-01-22

ВАЖНО: это – **не полноценные представления**, но тут можно быстро и наглядно «увидеть» их аналог визуально.

Представления (Views) – пример модифицируемого представления




Модифицируемый результат (**SELECT** * **FROM** sales;)

Что видит **DBeaver**? Он видит все столбцы, включая первичный ключ (sale_id).

Как он действует? Он точно знает, какую строку в базовой таблице нужно обновить, и может создать необходимый запрос (например, **UPDATE** sales **SET** sale_date = '2025-01-25' **WHERE** sale_id = 2).

Результат: редактирование успешно выполняется.

	123 sale_id ▼	123 customer_id ▼	sale_date ▼	123 total_amount ▼
1	2	[NULL]	2025-01-25	1 555,5
2	1	1	2025-01-22	221



	123 sale_id ▼	123 customer_id ▼	sale_date ▼	123 total_amount ▼
1	2	[NULL]	2025-01-25	1 555,5
2	1	1	2025-01-22	221

Представления (Views) – пример Немодифицируемого представления



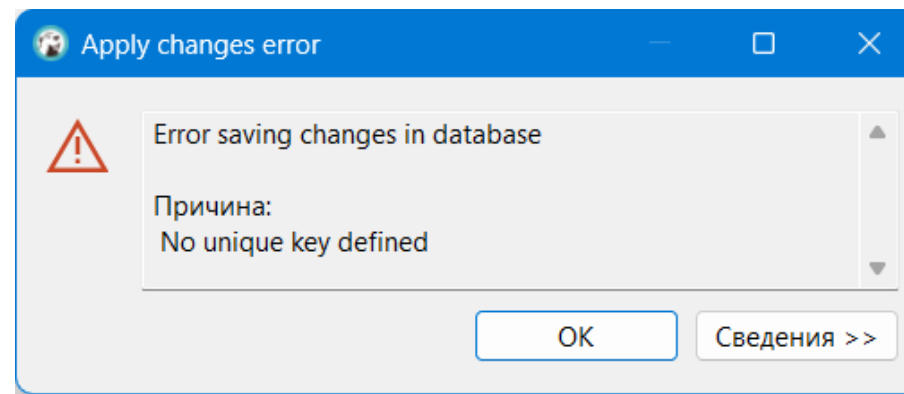
Немодифицируемый результат (`SELECT sale_date FROM sales;`)

Что видит DBeaver? Он видит только столбец **sale_date**. Первичный ключ **sale_id** отсутствует.

Как он действует? При попытке изменить дату DBeaver не может однозначно определить, какую строку в базовой таблице нужно обновить.

Результат: он выдает ошибку* «No unique key» (нет уникального ключа) и запрещает изменение.

sale_id	sale_date
1	2025-01-23
2	2025-01-25



* В начале он выдаёт предупреждение и предложение создать индекс, если все значения в столбце будут уникальны, он сможет создать запрос и внести изменения. Однако с настоящими **VIEW** это **не работает**.

Представления (Views) – материализованные представления



Материализованное представление (MV) – это заранее **вычисленный** и **сохранённый** на диск **результат запроса**.

Оно ускоряет чтение «дорогих» запросов, но требует осознанной стратегии обновления, а также имеет значительную проблему – **контролируемую неактуальность** данных между обновлениями.

Создание материализованного представления:

CREATE MATERIALIZED VIEW *название_представления* **AS** **<SELECT** *запрос* **>;**

Уже созданное представление требуется **актуализировать**, т.к. оно содержит данные, актуальные **на момент создания**.

Актуальность поддерживается (по расписанию/событию/вручную) **операцией** :

REFRESH MATERIALIZED VIEW *название_представления* **;**

Однако такое **обновление** временно **блокирует чтение** данных из представления.

Для **минимизации блокировок** при обновлении нужен **уникальный индекс**:

CREATE UNIQUE INDEX *имя_индекса* **ON** *название_представления* (*колонка_с_уникальными_значениями*) **;**

Запрос на обновление с учётом индекса:

REFRESH MATERIALIZED VIEW CONCURRENTLY *название_представления* **;**

Функции (UDF) и Хранимые процедуры (SP)



Функции и **Хранимые процедуры** – это **именованные блоки кода** на языке PL/pgSQL, которые хранятся и выполняются непосредственно на сервере базы данных.

Зачем они нужны?

- ✓ **Инкапсуляция**: «прячут» сложную бизнес-логику **внутри одной команды**.
- ✓ **Переиспользование**: логика **вызывается по имени**, а не копируется в код приложения.
- ✓ **Производительность**: код **выполняется на сервере**, уменьшая сетевой трафик.
- ✓ **Безопасность**: приложение **вызывает процедуру/функцию**, не имея **прямого доступа** к таблицам, которые она изменяет.

Функции (UDF) и Хранимые процедуры (SP) – создание процедуры – основы языка PL/pgSQL



Создание процедуры

1. Блок заголовка
(сигнатура процедуры)
2. Блок объявления
локальных переменных
3. Блок исполняемой логики
(главный код)
4. Генерация ошибки и выход
5. (Опционально)
Блок обработки ошибок

```
CREATE [OR REPLACE] PROCEDURE process_sale(  
    [IN | OUT | INOUT] p_medicine_id INT,  
    [IN | OUT | INOUT] p_quantity_sold INT  
)  
LANGUAGE plpgsql AS $$  
DECLARE  
    current_stock INT;  
BEGIN  
    SELECT quantity_in_stock INTO current_stock  
    FROM medicines WHERE id = p_medicine_id;  
  
    IF current_stock < p_quantity_sold THEN  
        RAISE EXCEPTION 'Недостаточно товара: %', current_stock;  
    END IF;  
EXCEPTION  
    WHEN OTHERS THEN  
        RAISE NOTICE 'Произошла неизвестная ошибка!'; RAISE;  
END;  
$$;  
CALL process_sale(42, 3);
```

Вызов процедуры

Функции (UDF) и Хранимые процедуры (SP) – параметры в процедурах



Процедуры используют **режимы параметров** для обмена данными с вызывающим кодом.

- **IN** (режим по умолчанию) – входной параметр.

Служит для передачи информации **ВНУТРЬ** процедуры.

Пример: **IN p_medicine_id INT** или **p_medicine_id INT** (без IN, т.к. это значение по умолчанию)

- **OUT** – выходной параметр.

Служит для возврата результата **ИЗ** процедуры.

Пример: **OUT p_success BOOLEAN**

Внутри процедуры ему присваивается значение: **p_success := TRUE;**

- **INOUT** – входной и выходной параметр.

Передаёт значение **ВНУТРЬ**, процедура его **изменяет** и **ВОЗВРАЩАЕТ**.

Пример: **INOUT p_counter INT**

Функции (UDF) и Хранимые процедуры (SP) – создание функции – основы языка PL/pgSQL



Создание функции

1. Блок заголовка
(сигнатура функции)
2. Блок объявления
локальных переменных
3. Блок исполняемой логики
(главный код)
4. Генерация ошибки и выход
5. Возвращаем итоговое значение
6. (Опционально)
Блок обработки ошибок

```
CREATE [OR REPLACE] FUNCTION check_stock(  
    f_medicine_id INT,  
    f_quantity_sold INT  
) RETURNS INT  
LANGUAGE plpgsql AS $$  
DECLARE  
    current_stock INT;  
BEGIN  
    SELECT quantity_in_stock INTO current_stock  
    FROM medicines WHERE id = f_medicine_id;  
    IF current_stock < f_quantity_sold THEN  
        RAISE EXCEPTION 'Недостаточно товара: %', current_stock;  
    END IF;  
    RETURN current_stock;  
EXCEPTION  
    WHEN OTHERS THEN  
        RAISE NOTICE 'Произошла неизвестная ошибка!'; RAISE;  
END;  
$$;  
SELECT check_stock(42, 3);
```

Вызов функции

Функции (UDF) и Хранимые процедуры (SP) – ключевые отличия



Признак	Пользовательская функция (UDF)	Хранимая процедура (SP)
Основное назначение	Вычисления. Инкапсуляция логики выборки данных .	Действия. Инкапсуляция бизнес-процессов .
Возвращаемое значение	Обязана возвращать значение (<i>скаляр или таблицу</i>) через RETURN .	Не может использовать RETURN . Использует OUT / INOUT параметры.
Побочные эффекты (DML)	Как правило, не должна изменять состояние БД (<i>нет INSERT, UPDATE</i>).	Создана для изменения состояния БД (INSERT, UPDATE, DELETE).
Управление транзакциями *	Не может управлять.	Может управлять транзакциями (<i>атомарные операции</i>).
Способ вызова	Как часть SELECT -запроса: <i>SELECT get_revenue(1);</i>	Отдельной командой CALL : <i>CALL process_sale(1, 1, 10);</i>

* Транзакции будут разбираться в одной из следующих практик.



Кафедра ЦТ
Институт информационных технологий
РТУ МИРЭА



Спасибо за внимание