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

РТУ МИРЭА

Институт информационных технологий

Кафедра цифровой трансформации

Презентации по лекционным материалам по дисциплине «Разработка баз данных»

Направления подготовки:

01.03.04 Прикладная математика
09.03.03 Прикладная информатика
09.03.04 Программная инженерия
09.03.01 Информатика и вычислительная техника

Уровень: бакалавриат

Форма обучения: очная

Лекция №5. Объекты базы данных (представления, хранимые процедуры и функции)

Лектор
Исаева Мария Владимировна
Кандидат технических наук,
доцент кафедры
цифровой трансформации

2025/2026 учебный год

СВЕДЕНИЯ О ДИСЦИПЛИНЕ

Лекции ведут:

- Исаева Мария Владимировна, к.т.н., доцент каф. Цифровой трансформации
- Дзгоев Алан Эдуардович, к.т.н., доцент каф. Цифровой трансформации (Dzgoev@mirea.ru / *При обращении в теме письма указывайте номер группы*)
- Миронов Антон Николаевич, Старший преподаватель каф. Цифровой трансформации
- Семыкина Наталья Александровна, д.т.н., профессор каф. Цифровой трансформации
- Резеньков Роман Николаевич, к.т.н., доцент каф. Цифровой трансформации

Объём **аудиторной** работы по дисциплине в 5 семестре:

- Лекции – 16 часов;
- Практические занятия – 48 часа;
- Аттестация – экзамен.

Допуск к экзамену:

- посещение лекций и практических работ;
- выполнение в установленный срок всех практических работ.

подробности в памятке по дисциплине (следующие слайды)

Выполнение практических заданий

1. Все материалы курса на https://online-edu.mirea.ru в разделе **Разработка баз данных**: *доступ к материалам курса открывается по расписанию занятий.*

2. Доступ с аккаунта МИРЭА.

3. Все создаваемые файлы прикладываются к заданию.

4. Учёт выполненных работ.

5. Общение с преподавателем через отзывы в заданиях.

...ков по кнопке "...вия"	Л4_% кликов по кнопке "контроль присутствия"	Л5_% кликов по кнопке "контроль присутствия"	Л6_04-05_% присутствия	Л7_18-05_% присутствия	Л8_01-06_% присутствия	%СРЗНАЧ участия в лекциях	Задание:Практика 1_Законодательная и нормативно-техническая база стандартизации в РФ	Задание:Практика 2_Нормативные документы по стандартизации ИТ	Задание:Практика 3_ч1_Классификаторы в области ИТ	Задание:Практика 3_ч2_Классификаторы в области ИТ	Задание:Практика 3_ч3_Классификаторы в области ИТ	Задание:Практика 4_ч1_Создание технического задания в соответствии с ГОСТ 34.602–2020
	0	50				6,3	-	-	-	-	-	-
	0	50		100	100	31,3	-	1	1	1	1	1
)	0	0				0	-	-	-	-	-	-
00	0	50	100		100	56,3	1	1	1	1	1	1
100	100	100	100	100	100	87,5	1	1	1	1	1	1
0	0	0				0	-	-	-	-	-	-
0	50	0				6,3	1	-	-	-	-	-
100	0	100		100		56,3	1	1	1	1	1	1
0	0	0				12,5	1	1	-	-	-	-
0	0	0				0	-	-	-	-	-	-
0	0	0				0	-	-	-	-	-	-
100	0	0				18,8	-	-	-	-	-	-
0	100	100	100	100	100	68,8	1	1	1	1	1	1
0	0	50				18,8	-	-	-	-	-	-
0	0	0	100	100	100	50	1	1	1	1	1	1
0	0	0				0	-	-	-	-	-	-
0	100	100	100	100	100	100	1	1	1	1	1	1
0	0	0	0			0	-	-	-	-	-	-
100	0	0	0			18,8	1	1	1	1	-	-
50	0	50	0			18,8	-	-	-	-	-	-
0	0	0	0			0	-	-	-	-	-	-
0	0	0	0			0	-	-	-	-	-	-
0	0	0	0		100	12,5	1	1	1	1	1	1
100	100	0	0			25	1	1	-	-	-	-
0	0	0	0			0	-	-	-	-	-	-
100	100	100	100	100	100	100	100	1	1	1	1	1	1

Рекомендуемая литература

1. Новиков Б.А. Основы технологий баз данных: учебное пособие / Б.А. Новиков, Е.А. Горшкова, Н.Г. Графеева; под.ред. Е.В. Рогова. – 2-е изд. – М.: ДМК Пресс, 2020. – 582 с. Режим доступа: https://postgrespro.ru//education/books/dbtech
2. Моргунов Е.П. PostgreSQL. Основы языка SQL: учеб. Пособие / Е.П. Моргунов; под. ред. Е.В. Рогова, П.В. Лузанова. – СПб.: БХВ-Петербург, 2018. – 336 с.: ил. Режим доступа: https://postgrespro.ru/education/books/sqlprimer
3. Комаров В.И. Путеводитель по базам данных. – М.: ДМК-Пресс, 2024. – 520 с. Режим доступа: https://edu.postgrespro.ru/dbguide.pdf
4. Смирнов М.В. Проектирование баз данных [Электронный ресурс]: Конспект лекций / Смирнов М.В. - М., МИРЭА – Российский технологический университет, 2020 – 1 электрон. Опт. диск (CD-ROM). Режим доступа: https://e.lanbook.com/book/163892/
5. Чистякова М.А. Проектирование и эксплуатация баз данных [Электронный ресурс]: Учебно-методическое пособие / Чистякова М.А., Иванова И.А., Котилевец И.Д. – М.: МИРЭА – Российский технологический университет, 2021. – 1 электрон. Опт. диск (CD-ROM). Режим доступа: https://e.lanbook.com/book/176572/
6. Братусь Н.В. Базы данных. Часть 1 [Электронный ресурс]: Практикум / Братусь Н.В.,Маличенко С.В., Матчин В.Т. — М.: МИРЭА – Российский технологический университет, 2024. — 1 электрон. опт. диск (CD-ROM) – Режим доступа: https://ibc.mirea.ru/books/SHARE/5859/
7. Моргунов Е. П. PostgreSQL. Профессиональный SQL : учеб. пособие / Е. П. Моргунов; под ред. Е. В. Рогова. — М.: ДМК Пресс, 2025. — 444 с. Режим доступа: https://postgrespro.ru/education/books/advancedsql
8. Рогов Е. В. PostgreSQL 17 изнутри. — М.: ДМК Пресс, 2025. — 668 с. Режим доступа: https://postgrespro.ru/education/books/internals
9. Введение в системы баз данных. : Пер. с англ. / К. Дж. Дейт .— М. : Изд. дом "Вильямс" , 2001 .— 1072 с. https://ibc.mirea.ru/books/search/?search_field=Дейт&page=3

4

Дисциплина «Разработка баз данных»

План-график занятий по неделям

Необходимо предусмотреть Три текущих теста и Две контрольные работы

Неделя	Лекции	Практические работы	Дедлайны, текущие и контрольные мероприятия
1	Лекция 1. Тема: Реляционная модель данных и базовые операции SQL. (2 часа)	Занятие 1. ПРАКТИЧЕСКАЯ РАБОТА №1. Создание БД и запросы на выборку данных (2 часа)	
2		Занятие 2. ПРАКТИЧЕСКАЯ РАБОТА №1. (2 часа)	Дедлайн по защите отчета по практике №1.
		Занятие 3. ПРАКТИЧЕСКАЯ РАБОТА №2. Многотабличные запросы, использование операций объединения, пересечения, разности (2 часа)	
3	Лекция 2. Тема: Многотабличные запросы и теоретико-множественные операции. (2 часа)	Занятие 4. ПРАКТИЧЕСКАЯ РАБОТА №2.(2 часа)	
4		Занятие 5. ПРАКТИЧЕСКАЯ РАБОТА №2. (2 часа)	Дедлайн по защите отчета по практике №2.
		Занятие 6 ПРАКТИЧЕСКАЯ РАБОТА №3 Использование подзапросов и CTE (2 часа)	
5	Лекция 3. Использование подзапросов и обобщетабличные выражения (2 часа)	Занятие 7. ПРАКТИЧЕСКАЯ РАБОТА №3 (2 часа)	Тест №1. (по лекциям №1-2).
6		Занятие 8. ПРАКТИЧЕСКАЯ РАБОТА №3 (2 часа).	Дедлайн по защите отчета по практике №3.
		Занятие 9. ПРАКТИЧЕСКАЯ РАБОТА №4 Использование оконных функций и функций ранжирования (2 часа)	
7	Лекция 4. Тема: SQL. Оконные функции и аналитические запросы (2 часа)	Занятие 10. ПРАКТИЧЕСКАЯ РАБОТА №4 (2 часа)	Дедлайн по защите отчета по практике №4.
8		Занятие 11. ПРАКТИЧЕСКАЯ РАБОТА №4 (2 часа)	Дедлайн по защите отчета по практике №4.
		Занятие 12. ПРАКТИЧЕСКАЯ РАБОТА №5 (2 часа)	
9	Лекция 5. Операторы модификации данных, объекты базы данных (2 часа)	Занятие 13. ПРАКТИЧЕСКАЯ РАБОТА №5 (2 часа)	Дедлайн по защите отчета по практике №5.

План – график лекций и практических занятий

10		Занятие 14. ПРАКТИЧЕСКАЯ РАБОТА №5 (2 часа)	Дедлайн по защите отчета по практике №5.
		Занятие 15. ПРАКТИЧЕСКАЯ РАБОТА №6 (2 часа)	
11	Лекция 6. Управление данными: триггеры, курсоры (2 часа)	**Занятие 16.** 2 часа	**Контрольная работа №1.**
12		Занятие 17. ПРАКТИЧЕСКАЯ РАБОТА №6 (2 часа)	Дедлайн по защите отчета по практике №6.
		Занятие 18. ПРАКТИЧЕСКАЯ РАБОТА №7 Оптимизация запросов (2 часа)	**Тест №2. (по лекциям №3-5).**
13	Лекция 7. Оптимизация запросов (2 часа)	Занятие 19. ПРАКТИЧЕСКАЯ РАБОТА №7 (2 часа)	
14		Занятие 20. ПРАКТИЧЕСКАЯ РАБОТА №7 (2 часа)	Дедлайн по защите отчета по практике №7.
		Занятие 21. ПРАКТИЧЕСКАЯ РАБОТА №8 Хранение неструктурированных данных (2 часа)	
15	Лекция 8. Администрирование баз данных и хранение неструктурированных данных (2 часа) Тест №3. (по лекциям №6-7).	Занятие 22. ПРАКТИЧЕСКАЯ РАБОТА №8 (2 часа)	
16		Занятие 23. ПРАКТИЧЕСКАЯ РАБОТА №8 (2 часа)	Дедлайн Практики №8.
		Занятие 24. 2 часа	**Контрольная работа №2.**
Итого	**16 часов**	**48 часов**	

ПАМЯТКА

о правилах обучения дисциплины

«Разработка баз данных» (2025-2026, I семестр)

№	Наименование	Формат	Период проведения	Баллы (макс.)
1	Защита практических работ (**8 практик**)	Очно	Сентябрь 2025 – Декабрь 2025	48 (6 баллов за 1 практику)
2	Контрольный тест №1 (по лекциям №1 и №2)	Очно, СДО	Октябрь 2025	5
3	Контрольный тест №2 (по лекциям №3 - №5)	Очно, СДО	Ноябрь 2025	5
4	Контрольная работа №1	Очно, СДО	Ноябрь 2025	10
5	Контрольный тест №3 (по лекциям №6 - №8)	Очно, СДО	Декабрь 2025	6
6	Контрольная работа №2	Очно, СДО	Декабрь 2025	10
7	Посещение (лекции 8 занятий, практики 24 занятия)	Очно	Сентябрь– Декабрь 2025	16 б. 0,5 баллов за посещения 1 лекции и практики (0,5·32 пар)
Максимальная сумма баллов за активность по дисциплине в течение учебного семестра				**100**

ДИСЦИПЛИНА	**Разработка баз данных**
	(укажите полное наименование дисциплины без сокращений)
ИНСТИТУТ	информационных технологий
	(укажите название учебного института)
КАФЕДРА	Цифровой трансформации
	(укажите полное наименование кафедры)
Уровень обучения	Бакалавриат
Аудиторные часы	16/0/48
	(укажите количество часов лекционных, лабораторных и практ

Лекции 16 часов.

Обязательное посещение всех лекций.

В тестах в рамках мероприятий текущего контроля вопросы из лекций

ПАМЯТКА

о правилах обучения дисциплины

«Разработка баз данных» (2025-2026, I семестр)

№	Наименование	Формат	Период проведения	Баллы (макс.)
1	Защита практических работ (**8 практик**)	Очно	Сентябрь 2025 – Декабрь 2025	48 (6 баллов за 1 практику)
2	Контрольный тест №1 (по лекциям №1 и №2)	Очно, СДО	Октябрь 2025	5
3	Контрольный тест №2 (по лекциям №3 - №5)	Очно, СДО	Ноябрь 2025	5
4	Контрольная работа №1	Очно, СДО	Ноябрь 2025	10
5	Контрольный тест №3 (по лекциям №6 - №8)	Очно, СДО	Декабрь 2025	6
6	Контрольная работа №2	Очно, СДО	Декабрь 2025	10
7	Посещение (лекции 8 занятий, практики 24 занятия)	Очно	Сентябрь– Декабрь 2025	16 б. 0,5 баллов за посещения 1 лекции и практики (0,5·32 пар)
Максимальная сумма баллов за активность по дисциплине в течение учебного семестра				**100**

Практические работы (8 работ)

Обязательное посещение всех практических занятий.

Допуск к экзамену – очная защита всех практических работ в течение семестра. Защита отчёта до дедлайна

Защита практической работы после дедлайна – 0 баллов, но работа засчитывается.

Если есть справка по перенесённой болезни, заверенная в учебном отделе, то эту справку необходимо принести и показать преподавателю по практике. В таком случае назначается пересдача пропущенной практической работы в день консультаций (важно: студент защищает именно ту практическую работу, которую он пропустил по болезни, согласно датам в справке). Если студент пропустил практические работы без уважительной причины и, соответственно, работу не защитил, то на экзамен он не допускается.

Если студент загрузил отчет до дедлайна, но не защитил, то такой отчет не засчитывается студенту, т.е. не сдал.

Дополнительного времени на защиту практических работ не выделяется.

Выполненные работы студент загружает в СДО в формате pdf.

Фон области построения модели – белый (в отчёте).

Если студент не загрузил отчет до дедлайна, то загрузка после дедлайна станет недоступна.

Контрольные тесты

Тесты студенты выполняют в СДО на паре.

Баллы минусуются, если по базовым вопросам студент отвечает не правильно.

Проходного балла нет, сколько студент набрал столько и остается.

Контрольный тест №1. (Лекции 1, 2) проводится на 5-м практическом занятии в начале пары (20 минут).

Контрольный тест №2. (Лекции 3, 4, 5) проводится на 16-м практическом занятии в начале пары (20 минут)..

Контрольный тест №3 (Лекции №6-7) проводится на 8-й лекции в начале пары (20 минут).

Контрольные работы

Выполняются на 15 и 24 практических занятиях в аудитории.

Студент получает задание на паре.

Время выполнения контрольной работы – 1 час.

Экзамен

Допуск к экзамену – защита всех практических работ в течение семестра очно. В случае, если студент не сдал какие-либо практики, у него есть возможность сдать их во время экзамена, если успеет. В противном случае, отправляется на пересдачу (необходимость сдачи работ сохраняется).

Если студент набрал меньше 50 % от общей суммы баллов, то экзамен будет проходить по билету (2теоретических вопроса + 1 задача).

Если студент набрал больше 50% от суммы максимальной суммы баллов, то сдаёт тестирование на экзамене. + добавляются баллы за активность.

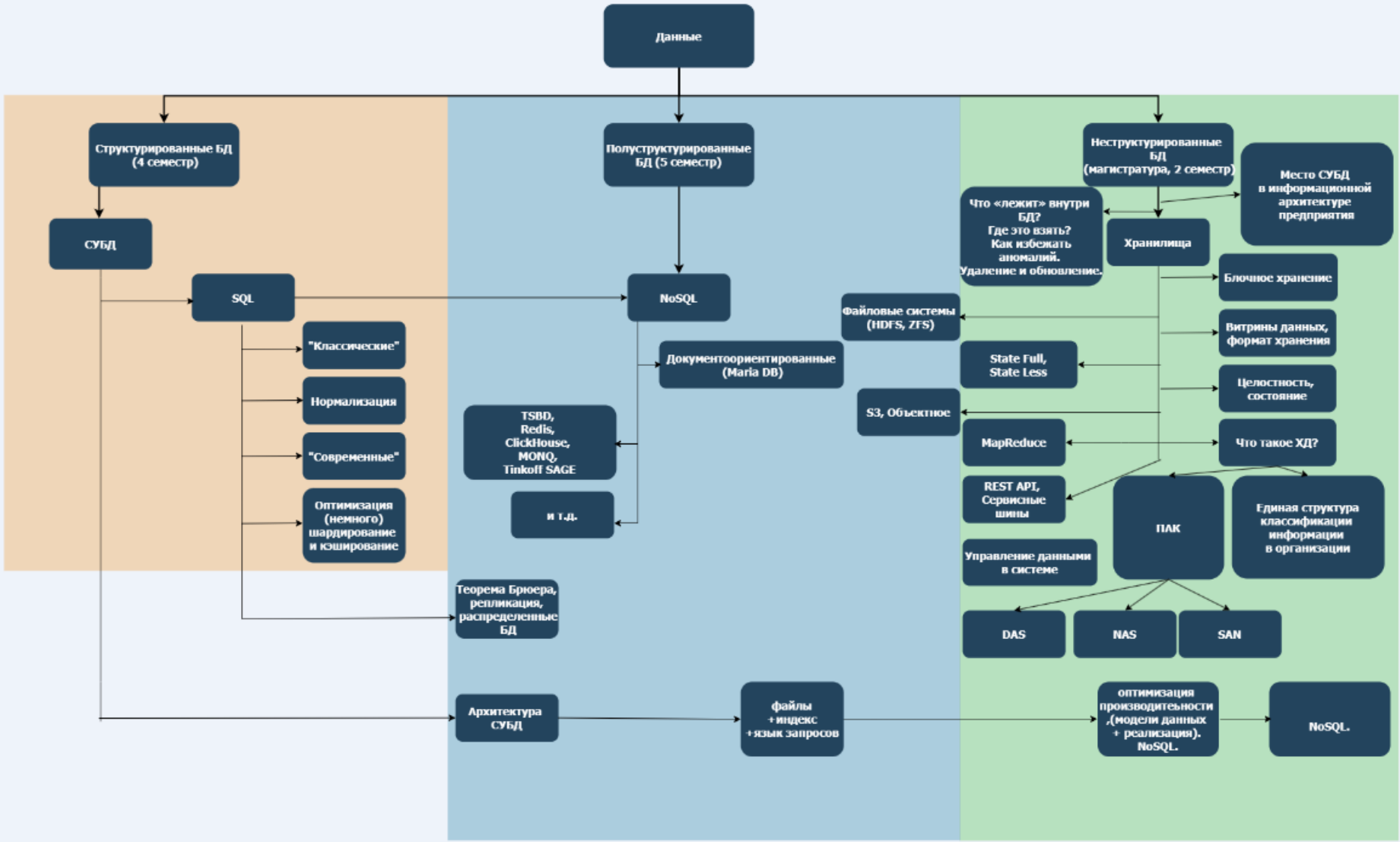
За тест студент может набрать 20 баллов, за которые можно получить:

- От 10 до 14 баллов — оценка «3»;
- От 15 до 19 баллов — оценка «4»;
- 20 баллов — оценка «5».

К баллам за тест прибавляются доп. баллы, которые студент получает в течение семестра по данным критериям:

- Набрано 55-64 балла за семестр — 1 доп.;
- Набрано 65-74 балла за семестр — 2 доп.;
- Набрано 75-84 балла за семестр — 3 доп.;
- Набрано 85-94 балла за семестр — 4 доп.;
- Набрано 95-100 балла за семестр — 5 доп.

В случае, если студент сдал тест на «2», то отправляется на пересдачу без учета доп. баллов.



Заметки:
Бакалавриат:

Проектирование БД (2 курс): Классические структурированные БД.

СУБД = приложение / БП (фактические информационные сущности).

Бакалавриат:

Разработка БД (3 курс) Упор на стройку СУБД в конечное приложение:

1) Место СУБД в архитектуре современных **приложений**.
2) Нестандартные СУБД (NoSQL, S3 Хранилища…).

3) Распределенные БД (шардирование, кэширование, проблема удаленного узла).

Управление данными в приложении:

1) Физический. Блочное ХД.

2) Файловая система.

3) Объект.
4) СУБД (или набор СУБД). 5) Приложение.

Магистратура:

Проектирование
и разработка баз и хранилищ данных (магистратура 1 курс):

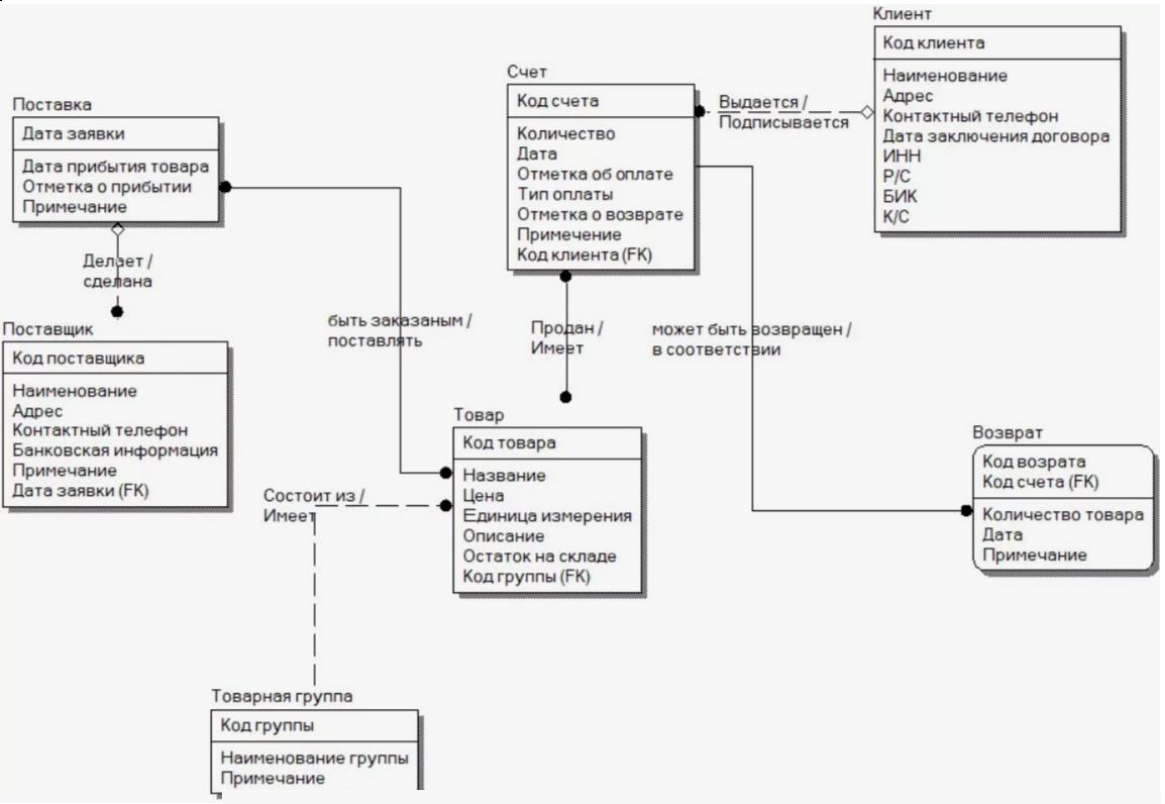
1) Место СУБД в информационной архитектуре **предприятия.**

2) Онтологическая модель. Что «лежит» внутри БД? Где это взять?

Как избежать аномалий. Удаление и обновление.

3) Единая структура классификации.

MapReduce. NoSQL, State Less, State Full, состояние , целостность данных.



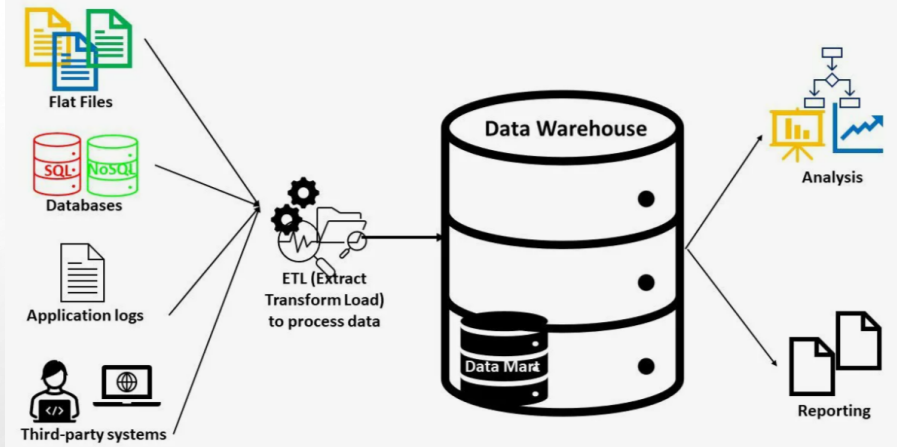
2 курс.

Проектирование баз данных

3 курс.

Разработка баз данных

PostgreSQL



Магистратура

Проектирование и разработка баз и хранилищ данных

Тема 5_ Объекты базы данных. Представления, хранимые процедуры и функции

5.1_Операторы модификации данных 12

5.2. Представления 25

Материализованное представление 33

5.3. Хранимые процедуры и функции 36

5.4. Операторы управления потоком 51

Операторы модификации данных (DML)

Язык манипуляции данными (DML — Data Manipulation Language) помимо оператора SELECT, осуществляющего извлечение информации из базы данных, включает операторы, изменяющие состояние данных.

Оператор	Назначение
INSERT	Добавление записей (строк) в таблицу БД
UPDATE	Обновление данных в столбце таблицы БД
DELETE	Удаление записей из таблицы БД
MERGE	Выполняет разные операции модификации целевой таблицы в зависимости от результатов соединения с целевой таблицей

Добавление данных

Для добавления строк используется оператор INSERT:

Синтаксис:

INSERT INTO <имя таблицы>[(<имя столбца>,...)]

{VALUES (<значение столбца>,…)}

| <выражение запроса>

| {DEFAULT VALUES}

INSERT INTO products VALUES (1, 'Cheese', 9.99);

Можно явно перечислить столбцы:

INSERT INTO products (product_no, name, price) VALUES (1, 'Cheese', 9.99);

Добавление данных

Если значения определяются не для всех столбцов, лишние столбцы можно опустить. В таком случае эти столбцы получат значения по умолчанию:

```
INSERT INTO products (product_no, name) VALUES (1, 'Cheese');
```

В примере ниже заполняются столбцы слева по числу переданных значений, а все остальные столбцы принимают значения по умолчанию:

```
INSERT INTO products VALUES (1, 'Cheese');
```

Можно явно указать значения по умолчанию:

```
INSERT INTO products (product_no, name, price) VALUES (1, 'Cheese',
DEFAULT);
INSERT INTO products DEFAULT VALUES;
```

Добавление данных

Одна команда может вставить сразу несколько строк:

```
INSERT INTO products (product_no, name, price) VALUES
    (1, 'Cheese', 9.99),
    (2, 'Bread', 1.99),
    (3, 'Milk', 2.99);
```

Или вставить результат запроса:

```
INSERT INTO products (product_no, name, price)
  SELECT product_no, name, price FROM new_products
    WHERE release_date = 'today';
```

Изменение данных

Для изменения данных в существующих строках используется команда UPDATE:

UPDATE <имя таблицы>

SET {<имя столбца> = {<выражение для вычисления значения столбца>

| NULL

| DEFAULT},...}

[{WHERE <предикат>}]

Увеличиваем цену всех товаров с 5 до 10:

UPDATE products SET price = 10 WHERE price = 5;

Увеличиваем цену на 10%: **UPDATE products SET price = price * 1.10;**

Удаление данных

Для удаления строк используется команда DELETE:

DELETE FROM <имя таблицы > [WHERE <предикат>];

Удаляем все строки из таблицы с товарами, имеющими цену 10:

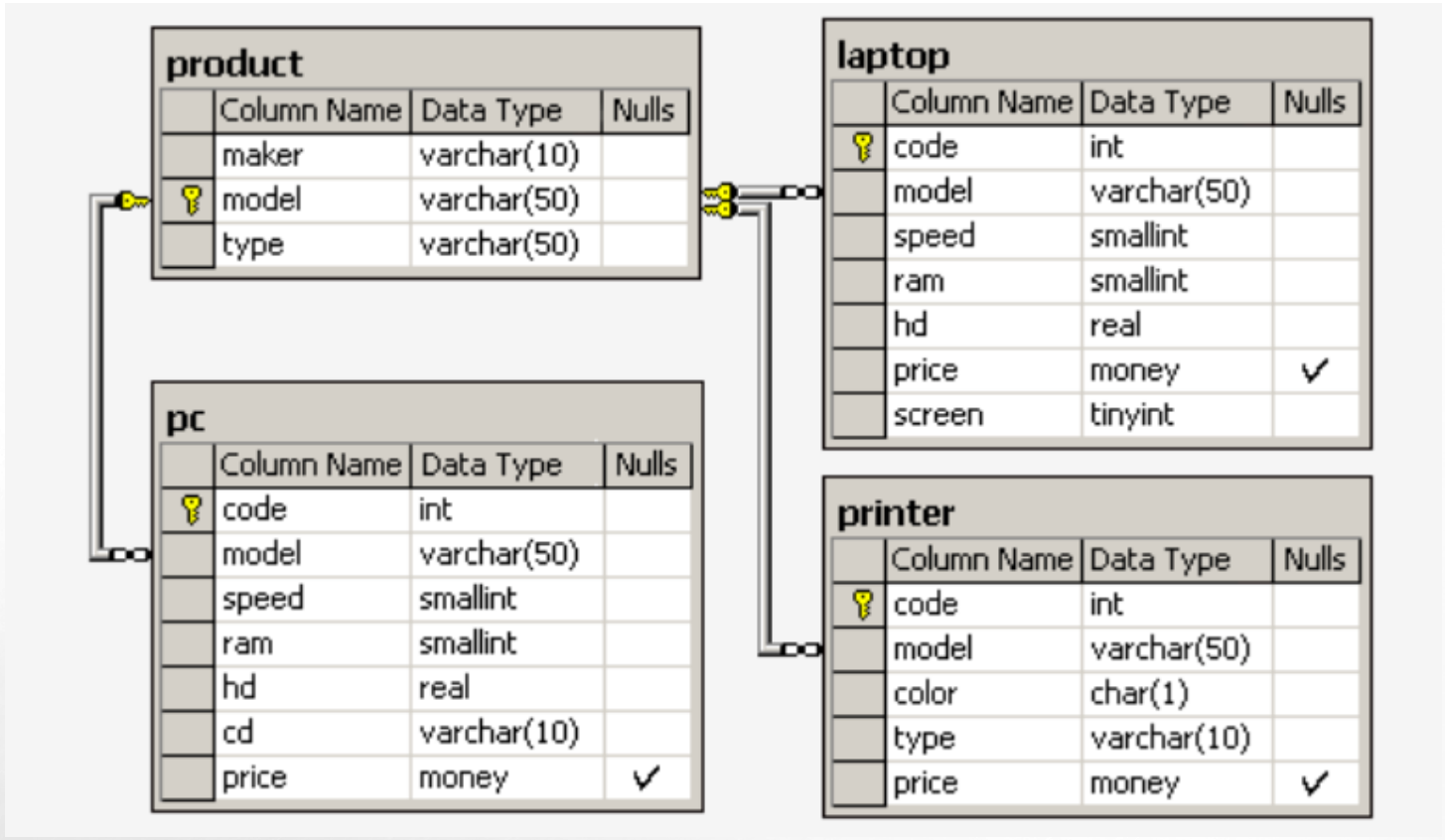
DELETE FROM products WHERE price = 10;

Удалим все строки:

DELETE FROM products;

Удаление данных

Удалить из таблицы PC компьютеры, имеющие минимальный объем диска или памяти.




```
DELETE FROM PC
WHERE code IN (
    SELECT code
    FROM PC
    WHERE hd =
    (SELECT MIN(hd) FROM
PC) OR ram = (SELECT
MIN(ram) FROM PC) )
```

Оператор MERGE

Позволяет **объединить операции** INSERT, UPDATE и DELETE **в одном запросе**, автоматически выбирая нужное действие в зависимости от того, существует ли соответствующая запись в целевой таблице. Был введен в стандарте SQL:2003:

```
MERGE INTO target_table AS t

USING source_table AS s

ON <condition>

WHEN MATCHED THEN

    <update_statement>

WHEN NOT MATCHED THEN

    <insert_statement>
```

USING — указывает на **источник данных,** которые будут использоваться для слияния.
ON <condition> — это условие, на основании которого происходит **сравнение строк** из источника USING и целевой таблицы.
WHEN MATCHED — указывается, что делать с данными, если условие ON обнаружило соответствие между строками в источнике и целевой таблице.
WHEN NOT MATCHED — описывает действия, которые должны быть выполнены, если строка из источника не нашла соответствия в целевой таблице.

Оператор MERGE

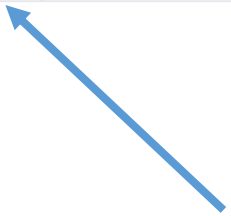
```
MERGE INTO customers AS c  USING leads AS l ON c.email = l.email
WHEN NOT MATCHED THEN
        INSERT (email, name, created_at) VALUES (l.email, l.name, DEFAULT);
```

Исходная таблица Customers:

	id [PK] integer	email text	name text	created_at timestamp without time zone
1	1	j.brown@gmail.com	Jonh Brown	2025-08-05 10:40:51.419886
2	2	sarah_white@yahoo.com	Sarah White	2025-08-05 10:40:51.419886

Исходная таблица Leads:

	id integer	email text	name text	created_at timestamp without time zone
1	1	j.brown@gmail.com	Jonh Brown ml.	2025-08-05 10:56:30.777236
2	2	mf@mns.com	Mike Finkel	2025-08-05 10:56:30.777236



Результат Merge:

	id [PK] integer	email text	name text	created_at timestamp without time zone
1	1	j.brown@gmail.com	Jonh Brown	2025-08-05 10:56:30.777236
2	2	sarah_white@yahoo.com	Sarah White	2025-08-05 10:56:30.777236
3	3	mf@mns.com	Mike Finkel	2025-08-05 10:57:44.000247

Оператор MERGE

Добавим новые записи в таблицу customers с новыми электронными адресами и обновим имя для уже существующих записей.

```
MERGE INTO customers AS c
USING leads AS l
ON c.email = l.email
WHEN NOT MATCHED THEN
        INSERT (email, name, created_at)
        VALUES (l.email, l.name, DEFAULT)
WHEN  MATCHED THEN
        UPDATE    SET name = l.name;
```

Результат Merge:

	id [PK] integer	email text	name text	created_at timestamp without time zone
1	2	sarah_white@yahoo.com	Sarah White	2025-08-05 10:59:01.560975
2	1	j.brown@gmail.com	Jonh Brown ml.	2025-08-05 10:59:01.560975
3	3	mf@mns.com	Mike Finkel	2025-08-05 10:59:01.560975

Оператор MERGE

Добавим записи в таблицу customers с отсутствующими электронными адресами, обновим имя, если дата создания клиента меньше, чем соответствующая дата в leads, и удалим другие записи

```
MERGE INTO customers AS c  USING leads AS l ON c.email = l.email
WHEN NOT MATCHED THEN
        INSERT (email, name, created_at)  VALUES (l.email, l.name, DEFAULT)
WHEN MATCHED AND c.created_at < l.created_at THEN
        UPDATE    SET name = l.name
WHEN MATCHED THEN
        DELETE;
```

	id [PK] integer	email text	name text	created_at timestamp without time zone
1	2	sarah_white@yahoo.com	Sarah White	2025-08-05 11:21:22.315611
2	3	mf@mns.com	Mike Finkel	2025-08-05 11:21:22.315611

Объекты базы данных

Объекты базы данных - это компоненты, которые используются для хранения, организации, управления и доступа к данным в реляционной базе данных.

Таблицы - основной объект для хранения данных в базе.
Представления - виртуальные таблицы, основанные на запросах к одной или нескольким реальным таблицам.
Индексы - ускоряют поиск данных в таблицах, позволяя СУБД быстро находить требуемые записи.
Хранимые процедуры и функции - блоки кода, которые можно многократно вызывать для выполнения определенных задач.
Триггеры - специальные процедуры, которые автоматически выполняются при определенных событиях в базе данных (например, при вставке, обновлении или удалении данных).
Последовательности - объекты, генерирующие уникальные числовые значения, часто используемые для автоматической нумерации записей.
Схемы - иерархическая организация объектов базы данных, позволяющая логически группировать связанные объекты и управлять правами доступа.
Ограничения - правила, накладываемые на данные в таблицах для обеспечения целостности и корректности данных.
Роли и пользователи - отвечают за управление доступом к базе данных и разграничение прав пользователей.

Объекты базы данных

Роль объектов базы данных

- **Организация данных –** объекты базы данных позволяют структурировать данные в таблицы, представления и другие формы, делая их более удобными для поиска, анализа и обработки.
- **Обеспечение целостности данных -** ограничения и триггеры помогают поддерживать целостность данных, предотвращая вставку некорректных или несовместимых данных.
- **Оптимизация доступа к данным -** индексы и хранимые процедуры ускоряют выполнение запросов и операций с данными.
- **Автоматизация действий -** триггеры и хранимые процедуры позволяют автоматизировать выполнение определенных действий при изменении данных или в других ситуациях, повышая эффективность работы базы данных.
- **Управление доступом к данным -** объекты, такие как схемы и разрешения, позволяют управлять доступом к данным и обеспечивать безопасность базы данных.

Представления

Представления (VIEW) являются объектами базы данных, информация в которых формируется динамически при обращении к ним.

- Содержимое представлений выбирается из других таблиц с помощью выполнения запроса, если данные в основной таблице меняются, пользователь получает актуальные данные при обращении к представлению, использующему эту таблицу.
- Для хранения представления используется только оперативная память.
- Представление не занимает дисковой памяти за исключением памяти, необходимой для хранения определения самого представления.

Представления

Синтаксис:

CREATE [OR REPLACE] VIEW имя_представления [(имена_полей_представления)]

AS select_выражение

Пример:

```
CREATE VIEW View1 AS
SELECT company_id, company_name
FROM  companies;
```

Использование:

1. К представлению можно строить запрос:

```
SELECT * FROM View1;
```

2. Соединять с другими таблицами:

```
SELECT Products.name, company_name
FROM Products
INNER JOIN View1
USING (company_id);
```

3. Модифицировать данные

Изменяемые представления

Представление будет автоматически изменяемым, если оно удовлетворяют одновременно всем следующим условиям:

- Список FROM в запросе, определяющем представлении, должен содержать ровно один элемент, и это должна быть таблица или другое изменяемое представление.
- Определение представления не должно содержать предложения WITH, DISTINCT, GROUP BY, HAVING, LIMIT и OFFSET на верхнем уровне запроса,
- Не использует константы или вычисляемые выражения,
- Включен каждый столбец таблицы, имеющий атрибут NOT NULL;
- Определение представления не должно содержать операции с множествами (UNION, INTERSECT и EXCEPT) на верхнем уровне запроса.
- Список выборки в запросе не должен содержать агрегатные и оконные функции, а также функции, возвращающие множества.

Изменяемые представления

```
INSERT INTO View1
VALUES (DEFAULT, 'ООО "Инарктика"');
SELECT * FROM companies;
```

company_id [PK] integer	company_name character varying (255)	country character varying (100)
1	ООО "Молочная река"	Россия
2	ИП "Хлебный рай"	Россия
3	ООО "Космол"	Россия
4	ООО "Инарктика"	[null]

```
DELETE FROM View1 WHERE
company_name LIKE '%Инарктика%';
SELECT * FROM companies;
```

company_id [PK] integer	company_name character varying (255)	country character varying (100)
1	ООО "Молочная река"	Россия
2	ИП "Хлебный рай"	Россия
3	ООО "Космол"	Россия

Неизменяемые представления

- предназначены только для чтения;
- позволяют формировать сложные запросы на основе данных базовых таблиц;
- результаты этих запросов могут использоваться в других запросах, что позволяет избежать сложных формулировок и снизить вероятность ошибочных действий.

Неизменяемое представление с данными из разных таблиц (Клиенты, не совершавшие покупок):

```
CREATE VIEW View2 AS
        SELECT customers.first_name, customers.last_name
        FROM customers
        LEFT JOIN orders
        ON customers.customer_id=orders.customer_id
        WHERE orders.customer_id IS NULL;
```

Неизменяемые представления

Неизменяемое представление с группировкой и итоговыми функциями (остатки товаров):

```
CREATE VIEW View3 (Товар, Остаток) AS  SELECT name, Sum(quantity_in_stock)
FROM Products  GROUP BY name;
```

Неизменяемое представления с использованием выражения (Сумма налога с продаж по каждому товару):

```
CREATE VIEW View4 (Код, Количество, Цена, Сумма, Налог)  AS
SELECT          product_id, quantity, unit_price,
                quantity * unit_price, (quantity * unit_price) * 0.15
FROM order_items;
```

Неизменяемые представления

Пример. Необходимо частично скрыть электронные адреса в таблице покупателей для сохранения конфиденциальности информации о пользователях.

```
CREATE VIEW View5 AS
        SELECT      customer_id, first_name,   last_name,
                    CONCAT(SUBSTR(email, 1, 2), '****', RIGHT(email, 4)) AS email
        FROM customers;
```

```
SELECT * FROM View5;
```

	customer_id integer	first_name character varying (255)	last_name character varying (255)	email text
1	1	Иван	Иванов	iv****.com
2	2	Петр	Петров	pe****.com

Преимущества и недостатки представлений

- **Независимость от данных**
- **Актуальность**
- **Повышение защищенности данных**

- **Ограниченные возможности обновления**
- **Структурные ограничения**
- **Снижение производительности**

Материализованное представление

Материализованные представления - физически хранят результаты запроса и периодически обновляются, что может значительно улучшить производительность для сложных запросов.

Синтаксис:

```
CREATE MATERIALIZED VIEW
[ IF NOT EXISTS ]
имя_представления
 [(имена_полей_представления)]  AS
select_выражение
```

```
CREATE MATERIALIZED VIEW MView1 AS
SELECT
c.customer_id,  c.first_name,
c.last_name, SUM(o.total_amount) AS total_sales
FROM  customers c JOIN    orders o
ON c.customer_id = o.customer_id
GROUP BY c.customer_id, c.first_name, c.last_name;
```

Материализованное представление

Чтобы обновить данные в материализованном представлении, необходимо выполнить команду

REFRESH MATERIALIZED VIEW [CONCURRENTLY] имя

[WITH [NO] DATA]

Параметры:

[NO] DATA – оставлять ли старые данные при обновлении MV для доступа на чтение.

CONCURRENTLY – обновить материализованное представление, не блокируя параллельные выборки из него.

Сравнение представлений

```
CREATE VIEW View1 AS
    SELECT company_id, company_name
    FROM  companies;
```

```
CREATE MATERIALIZED VIEW MView2 AS
    SELECT company_id, company_name
    FROM  companies;
```

	company_id integer	company_name character varying (255)
1	1	ООО "Молочная река"
2	2	ИП "Хлебный рай"
3	3	ООО "Космол"

```
INSERT INTO companies (company_name, country) VALUES ('ООО "РусАгро"', 'Россия');
```

```
SELECT * FROM View1;
```

company_id integer	company_name character varying (255)
1	ООО "Молочная река"
2	ИП "Хлебный рай"
3	ООО "Космол"
4	ООО "РусАгро"

```
REFRESH MATERIALIZED
VIEW MView2;
```

```
SELECT * FROM MView2;
```

	company_id integer	company_name character varying (255)
1	1	ООО "Молочная река"
2	2	ИП "Хлебный рай"
3	3	ООО "Космол"

Хранимые процедуры и функции

Хранимый процедуры и функции - это набор команд на языке SQL, сохранённый на сервере базы данных и выполняемый по запросу. Они создаются один раз и хранятся на сервере, что позволяет вызывать их многократно, обеспечивая переиспользование кода и повышая производительность базы данных.

Преимущества хранимых процедур и функций:

1. Хранимые процедуры и функции хранятся и выполняются непосредственно на стороне базы данных, что снижает вероятность задержек в работе.
2. Благодаря поддержке параметров и SQL-конструкций, таких, как циклы и условия, они позволяют организовать сложную обработку данных, сводя работу к вызову одной команды.
3. Один раз созданную процедуру можно многократно вызывать для разных наборов данных, что сокращает дублирование кода и делает приложение более надёжным.
4. Процедуры компилируются при создании, поэтому при каждом вызове не требуется повторная компиляция — это ускоряет выполнение.

Хранимые процедуры и функции

Функции	Хранимые процедуры
Функция обязана возвращать значение (одно или несколько).	Хранимая процедура не имеет возвращаемого типа, но имеет выходные аргументы
Разрешен только оператор SELECT для извлечения данных.	Могут содержать любые операторы SQL, включая DML, DDL и другие.
Функции часто используются для вычислений, преобразования данных, создания новых значений.	Процедуры больше подходят для выполнения последовательности действий, операций ввода-вывода, работы с данными.
Обычно не поддерживают транзакции, обработку ошибок и динамический SQL.	Могут включать транзакции, обработку ошибок и динамический SQL.
Функции могут использоваться как часть SQL-запросов, например, в SELECT, WHERE, JOIN.	Процедуры вызываются отдельно командой CALL

Функции

CREATE FUNCTION имя_функции(параметры) RETURNS тип_результата

AS 'тело функции'

LANGUAGE plpgsql;

Тело функции представляет собой просто текстовую строку, заключенную в $.

Текст тела функции должен быть *блоком*:

[<<метка>>]

[DECLARE

  объявления]

BEGIN

  операторы

END [метка];

Функции. Пример вложенных блоков

```
CREATE OR REPLACE FUNCTION some_func() RETURNS integer AS $$
<< outerblock >>
DECLARE
   quantity integer := 30;
BEGIN
   RAISE NOTICE 'Сейчас quantity = %', quantity;
   quantity := 50;
   -- Вложенный блок
   DECLARE
      quantity integer := 80;
   BEGIN
      RAISE NOTICE 'Сейчас quantity = %', quantity;
      RAISE NOTICE 'Во внешнем блоке quantity = %', outerblock.quantity;
   END;
   RAISE NOTICE 'Сейчас quantity = %', quantity;
   RETURN quantity;
END;
$$ LANGUAGE plpgsql;
SELECT  some_func();
```

```
NOTICE:  Сейчас quantity = 30
NOTICE:  Сейчас quantity = 80
NOTICE:  Во внешнем блоке quantity = 50
NOTICE:  Сейчас quantity = 50

Successfully run. Total query runtime: 65 msec.
1 rows affected.
```

Функции. Области видимости переменных

- Переменные, объявленные в блоке DECLARE, видны только внутри этого блока и его вложенных блоков (если они не перекрыты переменной с тем же именем).

- Если во вложенном блоке объявляется переменная с тем же именем, что и переменная во внешнем блоке, то переменная во вложенном блоке перекрывает переменную во внешнем блоке.

- Внутри вложенного блока доступна только локальная переменная. Метки блоков позволяют явно ссылаться на переменные, объявленные во внешних блоках, даже если они перекрыты локальными переменными.

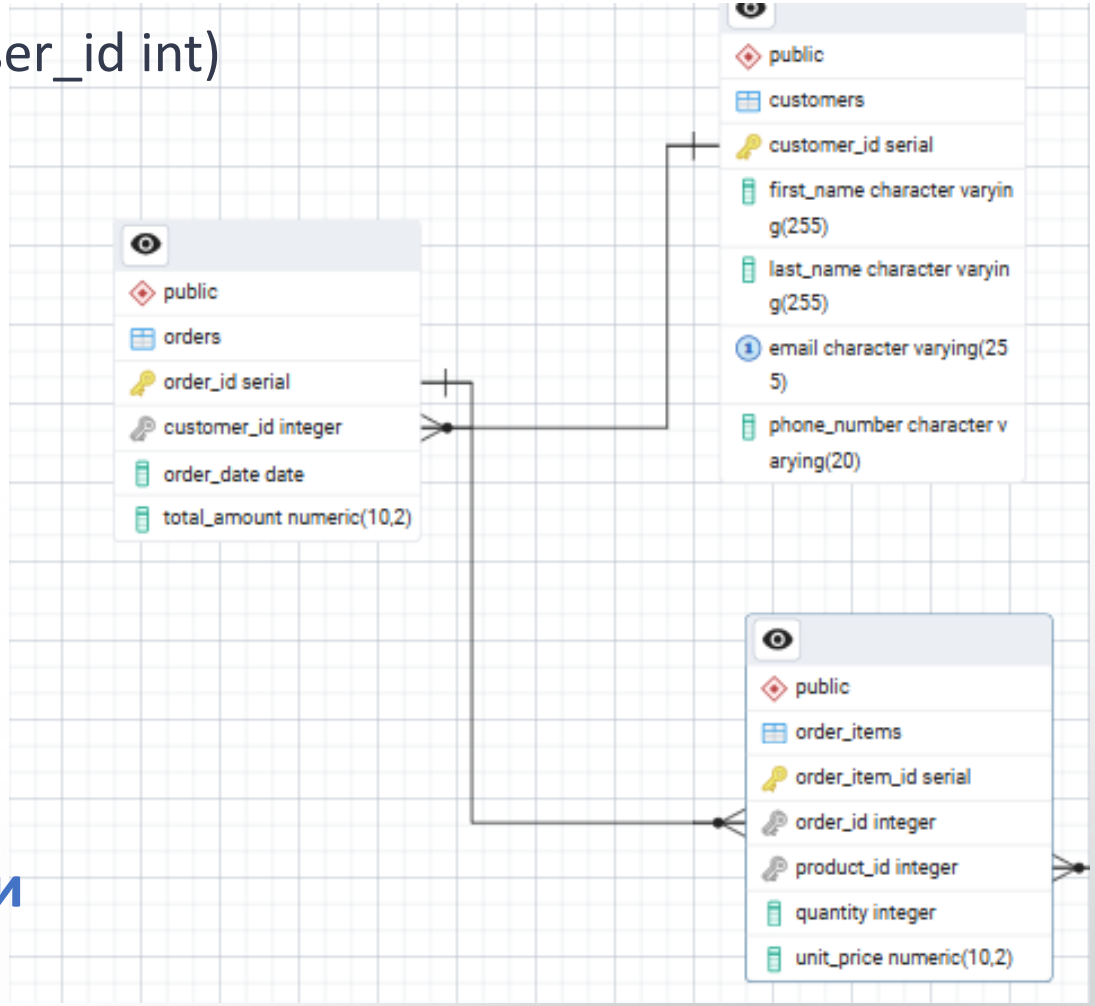
Функции. Примеры

Вернуть максимальную стоимость заказа пользователя с указанным идентификатором

```
CREATE OR REPLACE FUNCTION find_most_order(user_id int)
RETURNS numeric(10, 2) AS
$$
DECLARE
   item_cost numeric(10, 2);
begin
   SELECT MAX(total_amount)
   INTO item_cost
   FROM orders
   WHERE customer_id = user_id;
   RETURN item_cost;
end;
$$LANGUAGE plpgsql;
```

Вызов функции

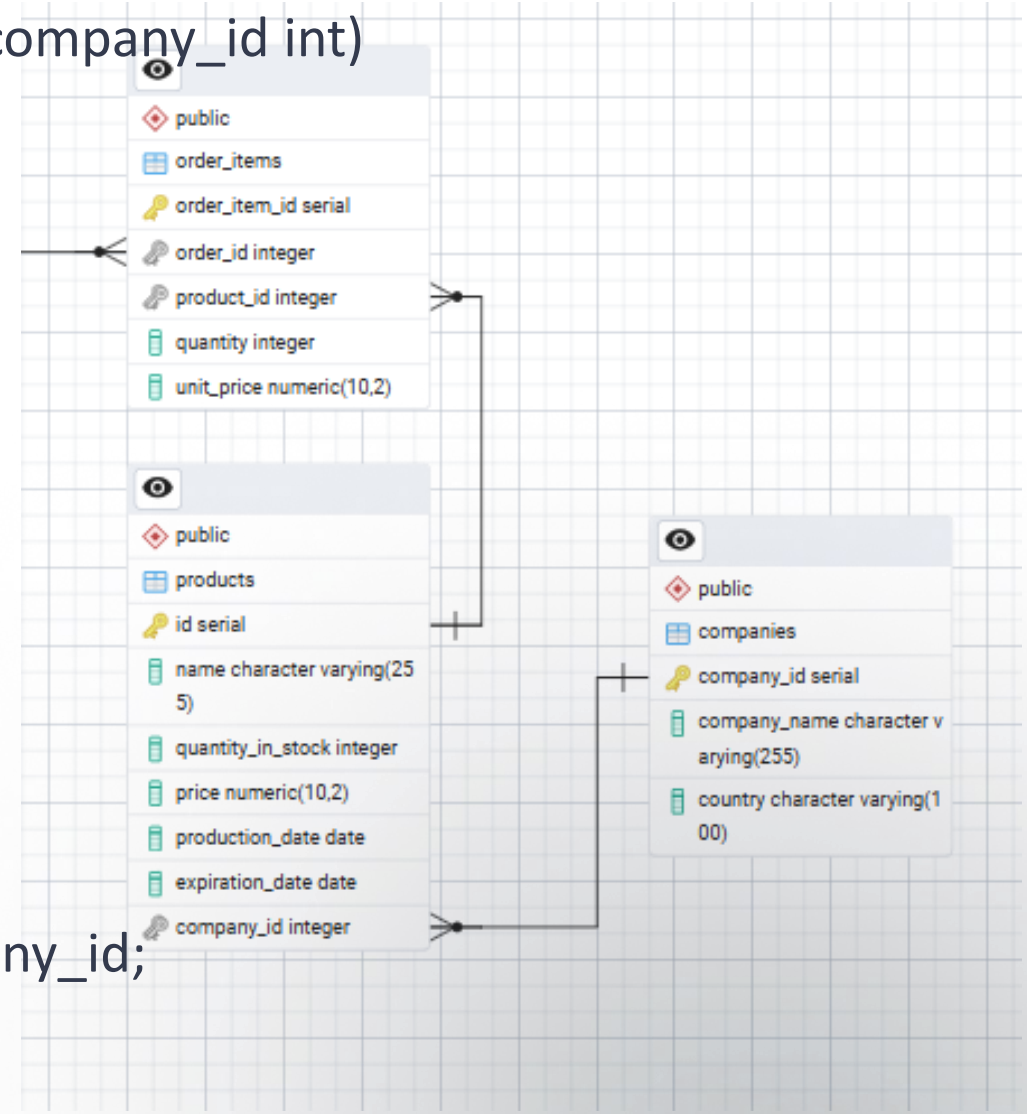
```
SELECT find_most_order(1);
```



Функции. Примеры

Подсчитать сумму продажи товаров указанного поставщика

```
CREATE OR REPLACE FUNCTION calc_company_profit(company_id int)
RETURNS numeric(20, 2) AS
$$
DECLARE
    company_profit numeric(20, 2);
begin
    SELECT SUM(quantity * unit_price)
    INTO company_profit
    FROM companies AS c
        JOIN products AS p
        ON c.company_id = p.company_id
        JOIN order_items AS o
        ON p.id = o.product_id
    WHERE c.company_id = calc_company_profit.company_id;
    RETURN company_profit;
end; $$LANGUAGE plpgsql;
```



Хранимые процедуры

CREATE PROCEDURE

имя_процедуры (параметры)

AS 'тело процедуры'

LANGUAGE plpgsql;

- **Входные параметры (IN)** позволяют передавать значения в процедуру, чтобы использовать их в логике выполнения.
- **Выходные параметры (OUT)** позволяют процедуре возвращать значения после выполнения.
- **INOUT параметры** могут быть как входными, так и выходными, позволяя передавать и изменять данные.

Функция, процедура или блок DO могут вызвать процедуру, используя оператор CALL.

Каждому параметру OUT или INOUT для процедуры должна соответствовать переменная в операторе CALL, и этой переменной по завершении процедуры будет присвоено возвращаемое процедурой значение.

Хранимые процедуры. Примеры

Процедура, подсчитывающая количество заказов для определенного клиента в таблице orders. Результат выводится на экран.

```
CREATE PROCEDURE count_customer_orders(IN c_id INT)
AS $$
DECLARE
  total_orders INT;
BEGIN
  SELECT COUNT(*) INTO total_orders FROM orders   WHERE orders.customer_id = c_id;
  RAISE NOTICE 'Total Orders: %', total_orders;
END;
$$ LANGUAGE plpgsql ;
CALL count_customer_orders(2);
```

Вызов процедуры

Хранимые процедуры. Примеры

В процедуру передаются идентификатор продукта и процент, на который требуется увеличить его цену. Процедура изменяет стоимость товара.

```
CREATE PROCEDURE add_percent_price(IN product_id INT, p INT) AS
$$
BEGIN
   UPDATE products SET price = price + price * p/100 WHERE id = product_id;
END;
$$ LANGUAGE plpgsql;

SELECT * FROM products;
CALL add_percent_price(5, 10);
SELECT * FROM products;
```

По умолчанию IN

2	2	Хлеб ржаной	50	35.00	2025-07-12	[null]
3	3	Хлеб ржаной	50	55.00	2025-07-11	[null]
4	4	Яйца куриные	200	120.00	2025-07-10	2025-07-20
5	5	Вода питьевая	300	25.00	2025-07-10	[null]

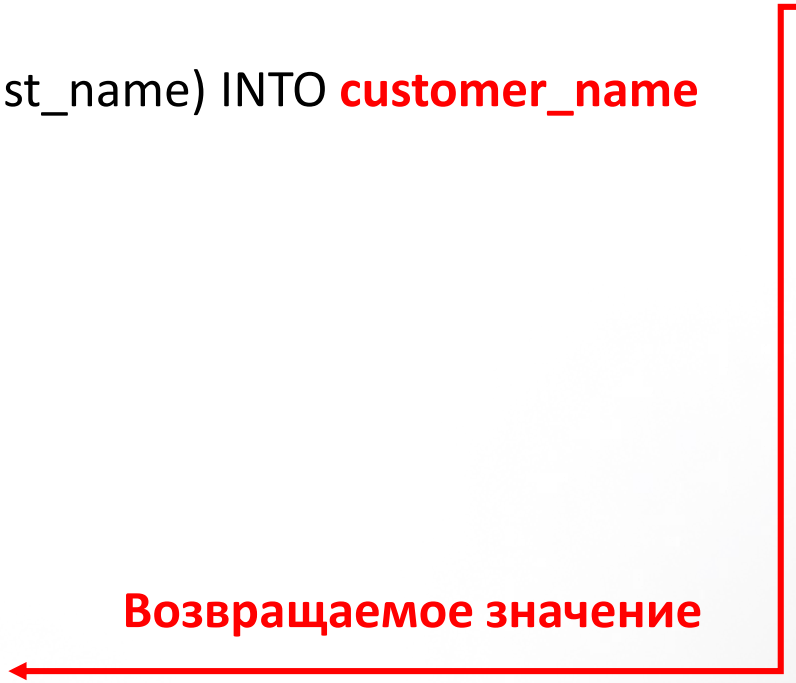
10	Греча	10	95.00	2025-06-01	[null]
5	Вода питьевая	300	27.50	2025-07-10	[null]

Хранимые процедуры. Примеры

Возвращаем имя покупателя по его идентификатору:

```
CREATE OR REPLACE PROCEDURE get_customer_name(id INT, OUT customer_name VARCHAR(100)) AS $$
BEGIN
   SELECT CONCAT(first_name, ' ', last_name) INTO customer_name
   FROM customers
   WHERE customer_id = id;
END;
$$ LANGUAGE plpgsql;

DO $$
DECLARE fio VARCHAR(100);
BEGIN
  CALL get_customer_name(2, fio);
  RAISE NOTICE 'myvar = %', fio;
END;
$$;
```



Хранимые процедуры. Примеры

Демонстрация INOUT параметров

```
CREATE PROCEDURE triple(INOUT x int) AS
$$
BEGIN
   x := x * 3;
END;
$$ LANGUAGE plpgsql;
```

Переменная должна быть инициализирована перед передачей в процедуру

```
DO $$
DECLARE a int := 5;
BEGIN
  CALL triple(a);
  RAISE NOTICE 'a = %', a;
END;
$$;
```

Хранимые процедуры. Примеры

Разработать хранимую процедуру, которая принимает на вход идентификатор продукта (product_id) и его количество (quantity) и выполняет следующие действия:

1. Проверяет, достаточно ли товара в магазине

2. Если товара достаточно - обновляет количество товара, возвращает значение TRUE и новое значение quantity_in_stock.

3. Если товара недостаточно - не изменяет количество товара, возвращает значение FALSE и текущее значение quantity_in_stock.

```
CREATE OR REPLACE PROCEDURE update_product_quantity(IN product_id INT,  INOUT quantity INT,
    OUT success BOOLEAN ) AS
$$
DECLARE
    current_quantity INT;
BEGIN
            SELECT quantity_in_stock INTO current_quantity FROM products   WHERE id = product_id;
```

Хранимые процедуры. Примеры

```
IF current_quantity >= quantity THEN      -- Если товара достаточно
        UPDATE products SET quantity_in_stock = quantity_in_stock - quantity
        WHERE id = product_id;
        quantity := current_quantity - quantity;
        success := TRUE;
ELSE
        quantity := current_quantity;
        success := FALSE;
END IF;
END;
$$ LANGUAGE plpgsql ;
```

Исходная таблица:

id [PK] integer	name character varying (255)	quantity_in_stock integer
1	Молоко 3.2%	100
2	Хлеб ржаной	50
3	Хлеб ржаной	50
4	Яйца куриные	200
6	Рис длинозерный	100
7	Рис круглозерный	15
9	Геркулес	13
11	Геркулес	130
8	Греча	10
10	Греча	10
5	Вода питьевая	300

Хранимые процедуры. Примеры

```
-- Товара достаточно
DO $$
DECLARE
   count_product INT := 20;  flag_success BOOLEAN;
BEGIN
   CALL update_product_quantity(6, count_product, flag_success);
   RAISE NOTICE 'quantity: %, success: %', count_product, flag_success;
END $$;
```

id [PK] integer	name character varying (255)	quantity_in_stock integer
6	Рис длинозерный	80

```
-- Недостаточно товара
DO $$
DECLARE
   count_product INT := 90;   flag_success BOOLEAN;
BEGIN
   CALL update_product_quantity(6, count_product, flag_success);
   RAISE NOTICE 'quantity: %, success: %', count_product, flag_success;
END $$;
```

id [PK] integer	name character varying (255)	quantity_in_stock integer
6	Рис длинозерный	80

Операторы управления потоком (Flow Control Language)

Условные операторы:

IF: Выполняет блок кода, если условие истинно:

```
IF <условие> THEN
    <Код для выполнения, если условие истинно>
ELSEIF <другое условие> THEN
    <Код для выполнения, если другое условие истинно>
ELSE
    <Код для выполнения, если ни одно из условий не истинно>
END IF;
```

Операторы управления потоком (Flow Control Language)

Условные операторы:

CASE: Выполняет блок кода, соответствующий первому совпавшему значению или условию:

```
CASE expression
        WHEN value1 THEN
                <Код для выполнения, если expression = value1>
        WHEN value2 THEN
                <Код для выполнения, если expression = value2>
        ELSE
              <Код для выполнения, если ни одно из значений не совпало>
END CASE;
```

Пример:

```
CASE
    WHEN x BETWEEN 0 AND 10 THEN
      msg := 'значение в диапазоне между 0 и 10';
    WHEN x BETWEEN 11 AND 20 THEN
      msg := 'значение в диапазоне между 11 и 20';
  END CASE;
```

Операторы управления потоком (Flow Control Language)

Циклы:

LOOP: организует бесконечный цикл, который продолжается до тех пор, пока не будет прерван операторами EXIT или RETURN. Этот цикл может быть вложенным, и для управления вложенными циклами можно использовать в операторах EXIT и CONTINUE, чтобы указать, к какому циклу они относятся.

```
[ <<метка>> ]
LOOP
    [ EXIT [ метка ]; ]
    [ CONTINUE [ метка ]; ]
END LOOP [ метка ];
```

```
NOTICE:  i = 1
NOTICE:  i = 2
NOTICE:  i = 3
NOTICE:  i = 4
NOTICE:  i = 5
NOTICE:  Выход из цикла!
```

```
DO $$
DECLARE
   i INTEGER := 1;
BEGIN
   <<new_loop>>
   LOOP
      RAISE NOTICE 'i = %', i;
      IF i = 5 THEN
         RAISE NOTICE 'Выход из цикла!';
         EXIT new_loop;
      END IF;
      i := i + 1;
   END LOOP new_loop;
END $$;
```

Операторы управления потоком (Flow Control Language)

Циклы:

WHILE: Выполняет блок кода, пока условие истинно.

[<<label>>]

WHILE условие LOOP

операторы

END LOOP [label];

```
NOTICE:  Текущее значение счетчика: 1
NOTICE:  Текущее значение счетчика: 2
NOTICE:  Текущее значение счетчика: 3
NOTICE:  Текущее значение счетчика: 4
NOTICE:  Текущее значение счетчика: 5
```

```
DO $$
DECLARE
   i INTEGER := 1;
BEGIN
   WHILE i <= 5 LOOP
      RAISE NOTICE
         'Текущее значение счетчика: %', i;
      i := i + 1;
   END LOOP;
END $$;
```

Операторы управления потоком (Flow Control Language)

Циклы: FOR: Перебирает значения в диапазоне или коллекции, выполняя блок кода для каждого значения.

```
<<[метка]>>
FOR target IN [ REVERSE ]
expression .. expression [ BY step ]
LOOP

   -- Операторы

END LOOP [метка];
```

```
DO $$
BEGIN
   FOR i IN 1..5 LOOP
      RAISE NOTICE 'Значение i: %', i;
   END LOOP ;
END $$;
```

```
NOTICE:  Значение i: 1
NOTICE:  Значение i: 2
NOTICE:  Значение i: 3
NOTICE:  Значение i: 4
NOTICE:  Значение i: 5
```

Операторы управления потоком (Flow Control Language)

Циклы: FOR: Перебирает значения в диапазоне или коллекции, выполняя блок кода для каждого значения.

```
DO $$
BEGIN
   FOR i IN REVERSE 5..1 LOOP
      RAISE NOTICE 'Значение i: %', i;
   END LOOP ;
END $$;
```

```
NOTICE:  Значение i: 5
NOTICE:  Значение i: 4
NOTICE:  Значение i: 3
NOTICE:  Значение i: 2
NOTICE:  Значение i: 1
```

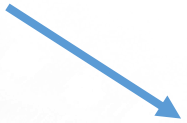
```
DO $$
BEGIN
   FOR i IN REVERSE 10..1 BY 2 LOOP
      RAISE NOTICE 'Значение i: %', i;
   END LOOP ;
END $$;
```

```
NOTICE:  Значение i: 10
NOTICE:  Значение i: 8
NOTICE:  Значение i: 6
NOTICE:  Значение i: 4
NOTICE:  Значение i: 2
```

Операторы управления потоком (Flow Control Language)

Циклы: В качестве запроса в этом типе оператора FOR может задаваться любая команда SQL, возвращающая строки

```
DO $$
DECLARE rec RECORD;
BEGIN
<<row_loop>>
   FOR rec IN
          SELECT name || '-' || CAST(price AS TEXT) AS new_name
          FROM products WHERE price > 100 LOOP
      RAISE NOTICE 'Продукт: %', rec.new_name;
   END LOOP row_loop;
END $$;
```



```
NOTICE:  Продукт: Яйца куриные-120.00
NOTICE:  Продукт: Рис круглозерный-105.00
```

Операторы управления потоком (Flow Control Language)

EXCEPTION: Позволяет перехватывать и обрабатывать исключения, возникающие в блоке кода.

Синтаксис:

```
BEGIN
    -- Код, который может вызвать исключение
EXCEPTION
    WHEN condition1 THEN
        -- Обработка исключения condition1
    WHEN condition2 THEN
        -- Обработка исключения condition2
    WHEN OTHERS THEN
        -- Обработка всех остальных исключений
END;
```

```
DO $$
DECLARE
  a integer:= 5;
  b integer:= 0;
  result  integer;
BEGIN
            IF b = 0 THEN RAISE division_by_zero;
            END IF;
            result := a / b;
            RAISE NOTICE 'Результат = %', result;
            EXCEPTION
            WHEN division_by_zero THEN
            RAISE NOTICE 'Деление на ноль';
            WHEN OTHERS THEN
            RAISE NOTICE 'Произошла другая ошибка';
END$$;
```

Какие данные включает представление? Является ли оно модифицируемым?

```
CREATE VIEW top_products AS
        SELECT
                product_name, price, rating
        FROM
                 products
        WHERE
                price < 50
        ORDER BY
                rating DESC, price ASC;
```

Какие данные включает представление? Является ли оно модифицируемым?

```
CREATE VIEW top_products AS
        SELECT
                product_name, price, rating
        FROM
                 products
        WHERE
                price < 50
        ORDER BY
                rating DESC, price ASC;
```

Данное представление не модифицируемое:

- В SELECT не включен первичный ключ таблицы products

Какое значение вернет вызов calculate_discount(200, 15)?

```
CREATE OR REPLACE FUNCTION calculate_discount(
        price DECIMAL(10,2),
        discount_percent INTEGER
)
RETURNS DECIMAL(10,2) AS $$
BEGIN
   RETURN price * (1 - discount_percent/100.0);
END;
$$ LANGUAGE plpgsql;
```

Какое значение вернет вызов calculate_discount(200, 15)?

```
CREATE OR REPLACE FUNCTION calculate_discount(
        price DECIMAL(10,2),
        discount_percent INTEGER
)
RETURNS DECIMAL(10,2) AS $$
BEGIN
   RETURN price * (1 - discount_percent/100.0);
END;
$$ LANGUAGE plpgsql;
```

Ответ: 170

Спасибо за внимание!