

# Продвинутый SQL

Лекция 2.  
Архитектура и проектирование



# Николай Хащанов

Full-stack developer

Защитная зона  
для интеграции  
видео спикера

## О спикере:

- Разрабатываю и поддерживаю crm/erp системы
- Преподаю в Нетологии
- Окончил РГТЭУ по специальности Менеджмент
- Оптимизация и автоматизация бизнес-процессов

Я в Слаке:

 @Николай Хащанов



---

# Содержание

---

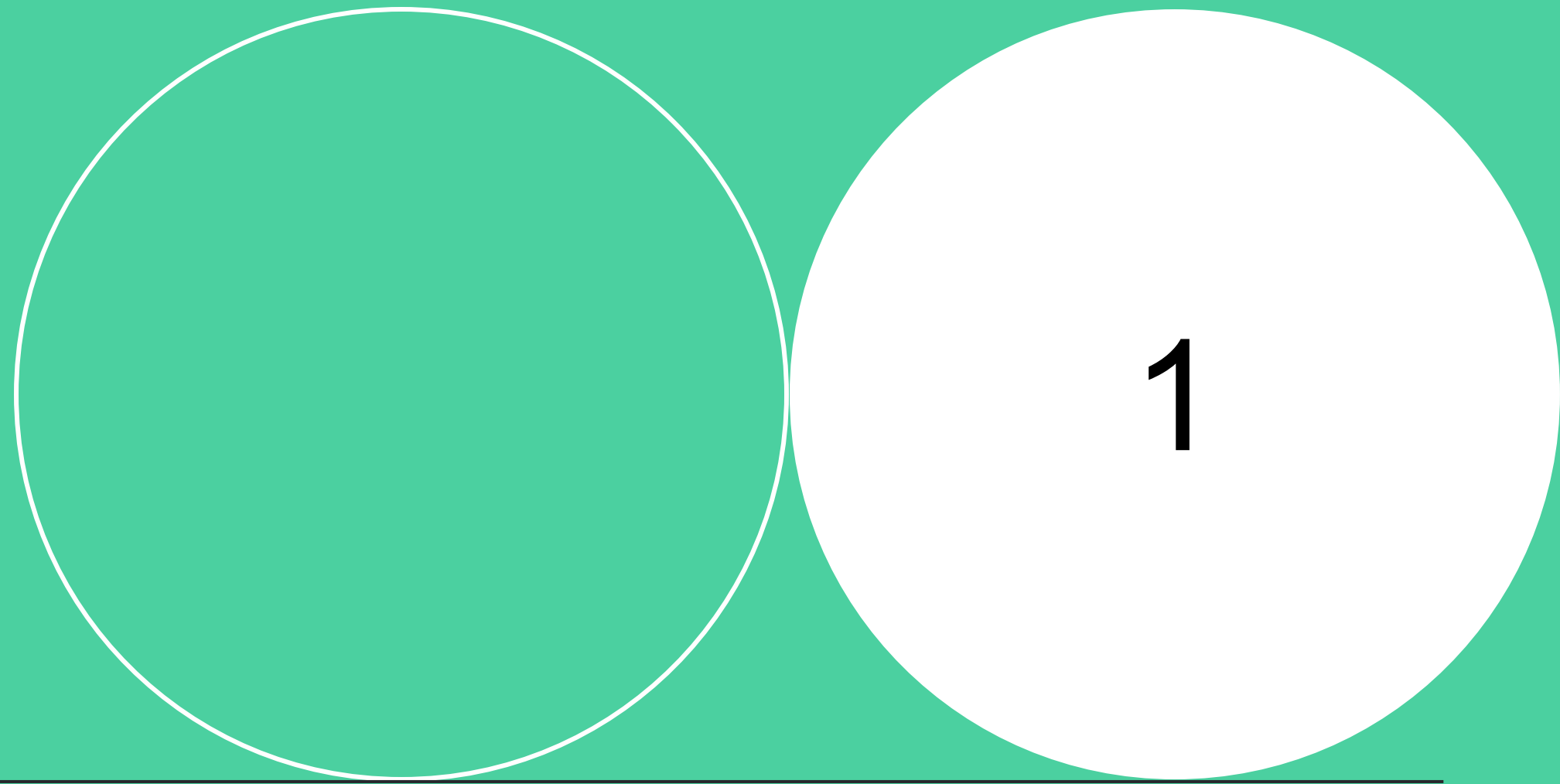
Защитная зона  
для интеграции  
видео спикера

- 
- 1 Архитектурные модели
  - 2 Основы проектирования
  - 3 Масштабирование
  - 4 Партиционирование



---

# Архитектурные модели



---

# Какие архитектурные модели бывают:

---

Защитная зона  
для интеграции  
видео спикера

- 1 Централизованная
- 2 Файл-сервер
- 3 Клиент – сервер



---

# «Сервер, не суетись под клиентом!»

---

Защитная зона  
для интеграции  
видео спикера

Продолжая разговор про РСУБД и PostgreSQL в частности, нужно разобраться в том, как оно вообще работает и из чего состоит.

В PostgreSQL используется архитектура «*клиент-сервер*» с распределением процессов между пользователями

Давайте более подробно разберемся, как же устроена архитектура.





# Из каких подсистем состоит:

- 1 Клиентская часть -

включает в себя клиентское приложение и библиотеку *LIBPQ*, реализующую интерфейс связи с сервером. Библиотека *LIBPQ* отвечает за установление соединения с сервером и передачу SQL-запросов.
- 2 Серверная часть -

включает в себя серверные процессы и контролирующий процесс-демон *Postmaster*, отвечающий за взаимодействие с клиентами. *Postmaster* авторизует и принимает запросы от клиентов и осуществляет обмен данными между клиентом и сервером. При получении запроса соединения от клиента *Postmaster* создаёт соответственный фоновый серверный процесс *postgres*, при этом используется связь один-к-одному. После того, как серверный процесс создан, клиент и сервер взаимодействуют напрямую.
- 3 Хранение и управление данными -

Несколько серверных процессов могут иметь одновременный доступ к информации из хранилища



# Процессы и память



После того, как клиент соединяется с сервером, процесс *postmaster* создает серверный процесс и дальше клиент работает уже с ним. На каждое соединение создается по серверному процессу, поэтому при большом числе соединений следует использовать пул (расширение *pgbouncer*).

*Postmaster* также запускает ряд служебных процессов, увидеть которые можно с помощью команды *ps fax*.

У экземпляра базы данных имеется общая для всех серверных процессов память.

Большую часть памяти занимает буферный кэш (*shared buffers*), необходимый для ускорения работы с данными на диске. PostgreSQL полностью полагается на операционную систему и сам не управляет устройствами. На пример, он считает, что вызов *fsync()* гарантирует попадание данных из памяти на диск.

Кроме буферного кэша в общей памяти находится информация о блокировках и многое другое, через нее же серверные процессы общаются друг с другом.

У каждого серверного процесса есть своя локальная память. В ней находится кэш каталога (часто используемая информация о базе данных), планы запросов, рабочее пространство для выполнения запросов и другое.





# Буферный кэш



Используется для оптимизации скорости работы памяти и дисков. Он состоит из массива буферов, которые содержат блоки данных и дополнительную информацию об этих блоках.

Размер блока обычно составляет 8 КБ, но может устанавливаться при сборке.

Буферный кэш, как и другие структуры в памяти, защищен блокировками от одновременного доступа. Блокировки организованы достаточно эффективно, чтобы не создавать большой конкуренции.

Любой блок, с которым работает СУБД, попадает в кэш. Часто используемые блоки остаются в кэше надолго, редко используемые — вытесняются и заменяются другими блоками.

Буфер, содержащий измененный блок, называется «грязным». Процесс *Background Writer* постепенно записывает такие блоки на диск в фоновом режиме — это позволяет снизить нагрузку на диски и увеличить производительность. Если *Background Writer* не успевает записать вытесняемый серверным процессом грязный буфер, то процесс записывает его сам. С точки зрения производительности этого лучше не допускать.



# Кластеры



Кластер баз данных представляет собой набор баз, управляемых одним экземпляром работающего сервера. После инициализации кластер будет содержать базу данных с именем `postgres`, предназначенную для использования по умолчанию утилитами, пользователями и сторонними приложениями. Сам сервер баз данных не требует наличия базы `postgres`, но многие внешние вспомогательные программы рассчитывают на её существование.

Хранение данных на диске организовано с помощью табличных пространств. Табличное пространство указывает расположение данных (каталог на файловой системе). Оно может использоваться несколькими базами данных. Например, можно организовать одно табличное пространство на быстрых дисках для активно используемых данных и другое — на медленных дисках для архивных данных.

Объекты (таблицы и индексы) хранятся в файлах. Каждый объект занимает один или несколько файлов внутри каталога табличного пространства. Кроме того, файлы разбиваются на части по 1 ГБ. Необходимо учитывать влияние потенциально большого количества файлов на используемую файловую систему



# Транзакции



Транзакции представляют собой последовательности операций, которые должны удовлетворять требованиям ACID: атомарность (*atomicity*), согласованность (*consistency*), изоляция (*isolation*) и долговечность (*durability*).

- *Атомарность* означает, что при фиксации выполняются все операции, составляющие транзакцию, при откате — не выполняется ни одна.
- *Согласованность* — поддержание целостности данных. Транзакция начинает работу в согласованном (целостном) состоянии и по окончании своей работы также оставляет данные согласованными.
- *Изоляция* означает, что на результат работы транзакции не оказывают влияния другие одновременно с ней выполняющиеся транзакции. По стандарту изоляция может иметь несколько различных уровней, в различной степени защищающих транзакции от внешних воздействий.
- *Долговечность* подразумевает возможность восстановить базу данных после сбоя в согласованном состоянии.



---

# Уровни описания данных

---

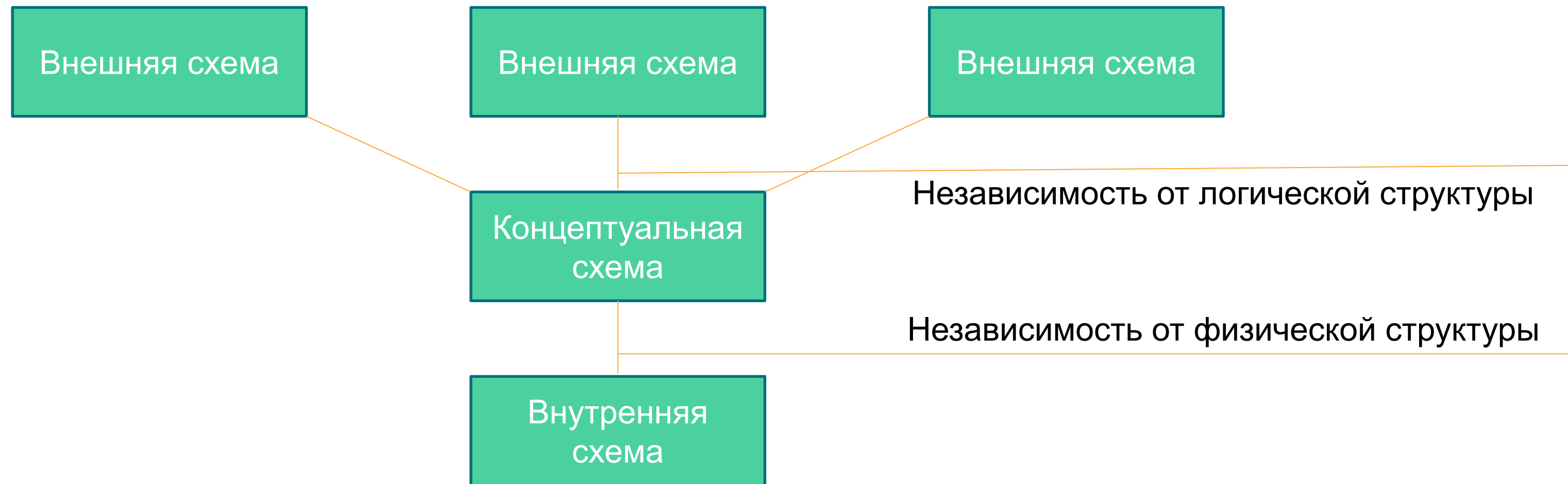


- *Внешний уровень*, на котором пользователи воспринимают данные, где отдельные группы пользователей имеют свое представление на базу данных
- *Внутренний уровень*, на котором СУБД и операционная система воспринимают данные
- *Концептуальный уровень*, предназначен для отображения внешнего уровня на внутренний уровень, а также для обеспечения необходимой их независимости друг от друга. Он связан с обобщенным представлением пользователей



# Три уровня архитектуры

Защитная зона  
для интеграции  
видео спикера



# Независимость данных

Защитная зона  
для интеграции  
видео спикера

*Логическая независимость* от данных означает полную защищенность внешних схем от изменений, вносимых в концептуальную схему. Такие изменения концептуальной схемы, как добавление или удаление новых сущностей, атрибутов или связей, должны осуществляться без необходимости внесения изменений в уже существующие внешние схемы для других групп пользователей.

*Физическая независимость* от данных означает защищенность концептуальной схемы от изменений, вносимых во внутреннюю схему. Такие изменения внутренней схемы, как использование различных файловых систем или структур хранения, разных устройств хранения, модификация индексов или хеширование, должны осуществляться без необходимости внесения изменений в концептуальную или внешнюю схемы.

*Описание структуры данных на любом уровне называется - **схемой***



# Внешний уровень

— это пользовательский уровень.

Пользователем может быть разработчик, или конечный пользователь, или администратор базы данных. Представление базы данных с точки зрения пользователей называется внешним представлением. Каждая группа пользователей выделяет в моделируемой предметной области, общей для всей организации, те сущности, атрибуты и связи, которые ей интересны. Эти частичные или переопределенные описания БД для отдельных групп пользователей или ориентированные на отдельные аспекты предметной области называют подсхемой.





# Концептуальный уровень



- является промежуточным уровнем в трехуровневой архитектуре и обеспечивает представление всей информации базы данных в абстрактной форме. Описание базы данных на этом уровне называется концептуальной схемой, которая является результатом концептуального проектирования.

Концептуальное проектирование базы данных включает анализ информационных потребностей пользователей и определение нужных им элементов данных. Таким образом, концептуальная схема — это единое логическое описание всех элементов данных и отношений между ними, логическая структура всей базы данных. Для каждой базы данных имеется только одна концептуальная схема.

Концептуальная схема должна содержать:

- сущности и их атрибуты;
- связи между сущностями;
- ограничения, накладываемые на данные;
- семантическую информацию о данных;
- обеспечение безопасности и поддержки целостности данных





# Внутренний уровень



- является третьим уровнем архитектуры БД.

Внутреннее представление не связано с физическим уровнем, так как физический уровень хранения информации обладает значительной индивидуальностью для каждой системы.

На нижнем уровне находится внутренняя схема, которая является полным описанием внутренней модели данных. Для каждой базы данных существует только одна внутренняя схема.

Внутренняя схема описывает физическую реализацию базы данных и предназначена для достижения оптимальной производительности и обеспечения экономного использования дискового пространства. На внутреннем уровне хранится следующая информация:

- распределение дискового пространства для хранения данных и индексов
- описание подробностей сохранения записей (с указанием реальных размеров сохраняемых элементов данных)
- сведения о размещении записей
- сведения о сжатии данных и выбранных методах их шифрования



# Архитектура – хорошо, а как же данные?

Защитная зона  
для интеграции  
видео спикера



---

# Какие бывают модели данных:

---

Защитная зона  
для интеграции  
видео спикера

- 1 Объектные модели данных
- 2 Модели данных на основе записей
- 3 Физические модели данных



# Объектные модели данных



Объектно-ориентированные базы данных позволяют представлять сложные объекты более непосредственным образом, нежели реляционные системы.

При построении объектных моделей данных используются такие понятия как сущности, атрибуты и связи:

- сущность - это отдельный элемент предметной области, который должен быть представлен в базе данных.
- атрибут - это свойство, которое описывает некоторый аспект объекта и значение которого следует зафиксировать
- связь является ассоциативным отношением между сущностями

ER-модель (Entity-Relationship model) стала одним из основных методов концептуального проектирования баз данных. Объектно-ориентированная модель расширяет определение сущности с целью включения в него не только атрибутов, которые описывают состояние объекта, но и действий, которые с ним связаны.



# Модели данных на основе записей



В модели данных на основе записей база данных состоит из нескольких записей фиксированного формата, которые могут иметь разные типы.

В большинстве случаев используются следующие два типа такого рода моделей данных:

- теоретико-графовые
- теоретико-множественные

К теоретико-графовым моделям относятся две разновидности:

- сетевые модели
- иерархические модели

В таких моделях данных предусматриваются характерные для подобного рода структур операции навигации и манипулирования данными. Аппарат навигации служит для установки тех объектов данных, к которым будет применяться очередная операция манипулирования данными.

Теоретико-множественные модели используют математический аппарат, реляционную алгебру (знаковая обработка множеств), реляционное исчисление. К моделям данного типа относятся реляционные модели.

В соответствии с реляционной моделью данных БД представляется в виде совокупности таблиц, над которыми могут выполняться операции, формируемые в терминах реляционной алгебры и реляционного исчисления.



# Физические модели данных



Физическая модель – логическая модель базы данных, выраженная в терминах языка описания данных конкретной СУБД

Физическая модель базы данных содержит все детали, необходимые конкретной СУБД для создания базы: наименования таблиц и столбцов, типы данных, определения первичных и внешних ключей и т.д.

Физическая модель строится на основе логической с учетом ограничений, накладываемых возможностями выбранной СУБД.

Физические модели данных описывают то, как данные хранятся в компьютере, представляя информацию о структуре записей, их упорядоченности и существующих путях доступа. Физических моделей данных не так много, как логических, а самыми популярными среди них являются обобщающая модель (unifying model) и модель памяти кадров (frame memory).



---

# Реляционная модель

---

Защитная зона  
для интеграции  
видео спикера

Реляционная модель данных - созданная Эдгаром Коддом логическая модель данных, описывающая:

- структуры данных в виде наборов отношений
- теоретико-множественные операции над данными: объединение, пересечение разность и декартово произведение
- специальные реляционные операции: селекция, проекция, соединение и деление
- специальные правила, обеспечивающие целостность данных



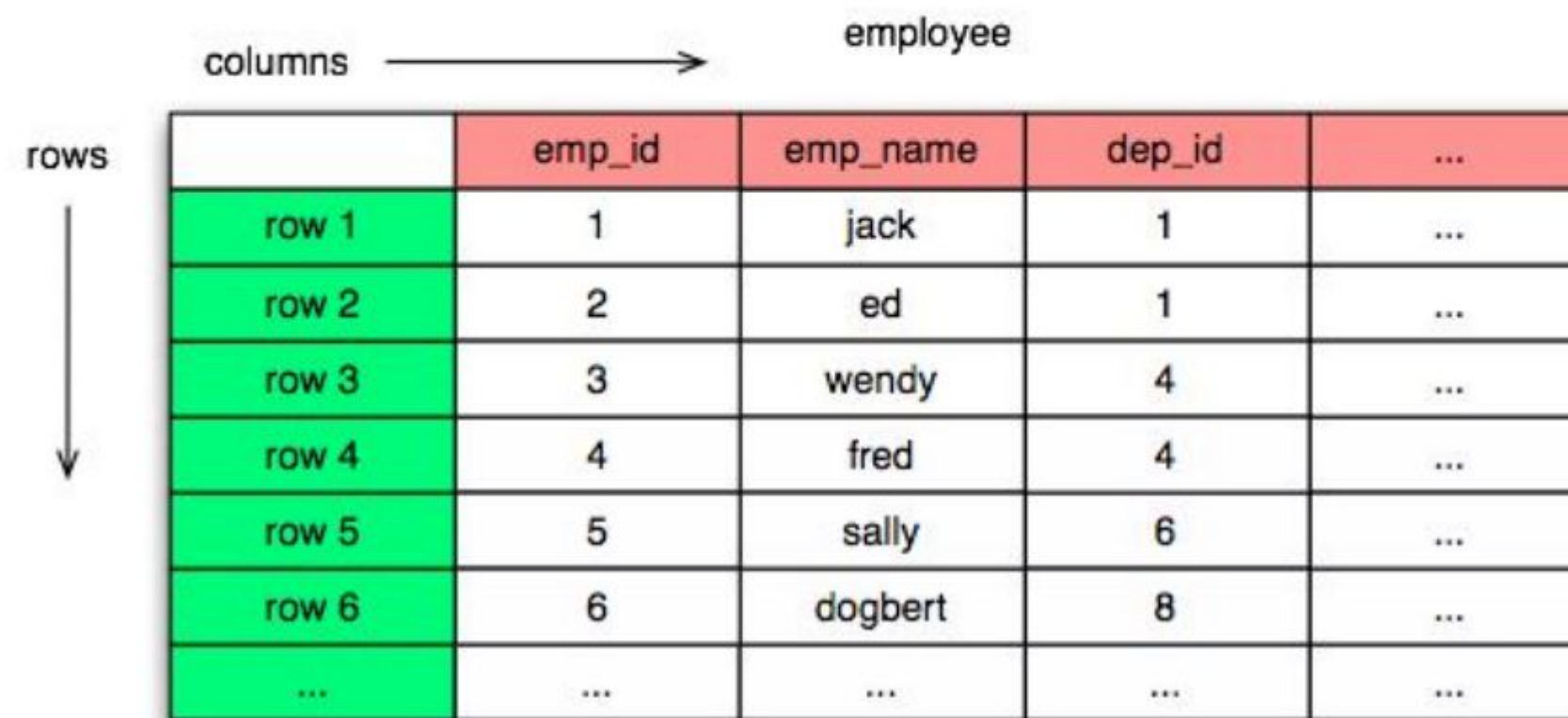


# Где-то это было...

Защитная зона  
для интеграции  
видео спикера

## Таблица

Таблица - это базовая единица данных. В реляционной алгебре она называется «отношение» (relation). Состоит из столбцов (columns), которые определяют конкретные типы данных. Данные в таблице организованы в строки (rows), которые содержат множества значений столбцов.



The diagram illustrates a table structure. A horizontal arrow labeled 'columns' points to the top row of the table. A vertical arrow labeled 'rows' points to the first column of the table. The table itself is titled 'employee' and contains the following data:

	emp_id	emp_name	dep_id	...
row 1	1	jack	1	...
row 2	2	ed	1	...
row 3	3	wendy	4	...
row 4	4	fred	4	...
row 5	5	sally	6	...
row 6	6	dogbert	8	...
...	...	...	...	...





# Формально – не формально



Что такое реляционная модель и как она устроена вы должны знать, давайте освежим из каких элементов состоит структура и какие названия они имеют:

Формальное название	Не формальное название
отношение	таблица
атрибут	столбец
кортеж	строка
степень	количество столбцов
кардинальное число	количество строк
первичный ключ	уникальный идентификатор
домен	общая совокупность допустимых значений



---

# ОСНОВЫ проектирования



2

---

Прежде чем создавать,  
нужно понять, что  
создавать.

Защитная зона  
для интеграции  
видео спикера



---

# С чего начать:

---

Защитная зона  
для интеграции  
видео спикера


- 1 Сбор и анализ требований заказчика и пользователей
- 2 Нужно сформулировать вопросы, на которые мы захотим получить ответы
- 3 Нужно выяснить, какие есть технические возможности для реализации БД
- 4 Поняв, что мы хотим, выбрать СУБД под наши нужды



---

# Когда определились:

---



Защитная зона  
для интеграции  
видео спикера

- 1 Проектируем
- 2 Реализуем
- 3 Тестируем
- 4 Эксплуатируем



# Магазин цветов

Давайте рассмотрим все этапы на примере создания базы данных для магазина по продаже цветов.

Мы предполагаем, что предприятие состоит из:

- один генеральный директор
- один бухгалтер (по совместительству кладовщик и закупщик)
- два продавца
- одно помещение в аренде
- один курьер
- три контрагента



# Сбор и анализ требований заказчика и пользователей

- Система должна быть отказоустойчивой и работать 24/7
- Должен быть удаленный доступ
- Возможность настраивать права доступа
- Интуитивно понятный интерфейс, так как курьер не дружит с ПК
- Возможность масштабирования
- Легкость эксплуатации, что бы один из продавцов мог вносить правки в структуру



# Вопросы, на которые мы захотим получить ответы



- Генеральный директор должен иметь кнопку с отчетом: «how much of my money»
- Бухгалтер должен видеть и иметь возможность:
  - видеть движение денег по товару
  - вносить информацию по налогам
  - вносить информацию по аренде
  - видеть наличие товара
  - вносить информацию по заработной плате
  - вносить информацию по закупкам
  - иметь возможность выгружать любые отчеты
- Продавцы должны видеть наличие товара и вносить данные по продажам
- Курьер должен видеть информацию по заказам и наличие товара и вносить данные по выполненным доставкам





# Технические возможности для реализации БД

- ПК «Спектрум» Vu-File
- «средняя машина»
- мобильный доступ в интернет
- ОС Win8 64 бит
- быстрообучаемый продавец

Нужно докупить:

- SSD диск
- планку DDR

или

Предусмотреть возможность миграции на облачный сервер, так как мы заранее подразумеваем, что заказчик захочет интернет магазин, а наших ресурсов не хватит.



# Какую СУБД выбрать?



Учитывая пожелания, возможности и наши умения, мы решаем остановиться на PostgreSQL.

\* Когда захотим создать простой интернет магазин, мы можем на бэкенде использовать PHP и для соединения с нашей БД использовать драйвер PDO\_PGSQL

\*\* Конечно мы понимаем, что было бы не плохо использовать MySQL на NDB движке, что бы один кластер работал на оффлайн магазин, второй на онлайн и третий хранил информацию по товару и налогам. Но при таком выборе затруднится разработка, эксплуатация и потребуются дополнительные финансовые вложения.



---

# Этапы проектирования базы данных:

---

Защитная зона  
для интеграции  
видео спикера

- 1 Концептуальный
- 2 Логический
- 3 Физический



# Концептуальный



Если мы говорим про проектирование сложных баз данных с большим количеством атрибутов, то необходимо использовать нисходящий подход.

Этот подход начинается с разработки моделей данных, содержащих высокоуровневую сущности и связи, затем прорабатываются нисходящие уточнения низкоуровневых сущностей, связей и относящихся к ним атрибутов.

Нисходящий подход демонстрируется в концепции модели «сущность - связь».

При разработке концептуальной модели данных выделяют ряд этапов:

- Выделение локальных представлений, соответствующих обычно относительно независимым данным. Каждое такое представление проектируется как подзадача
- Формулирование сущностей, описывающих локальную предметную область проектируемой БД, и описание атрибутов, составляющих структуру каждой сущности
- Выделение ключевых атрибутов
- Спецификация связей между сущностями. Удаление избыточных связей
- Анализ и добавление не ключевых атрибутов
- Объединение локальных представлений



# Логический



Второй этап проектирования заключается в создании логической модели данных для исследуемой части организации, где будет использоваться база данных.

Логическая модель, отражающая особенности представления о функционировании организации одновременно нескольких типов пользователей, называется глобальной логической моделью данных.

Процесс проектирования базы данных должен опираться на определенную модель данных (реляционная, сетевая, иерархическая), которая определяется типом предполагаемой для реализации информационной системы СУБД.

Концептуальное и логическое проектирование — это итеративные процессы, которые включают в себя ряд уточнений, продолжающиеся до тех пор, пока не будет получен наиболее соответствующий структуре организации продукт.



# Физический



Целью проектирования на данном этапе является создание описания СУБД ориентированной модели БД.

Для каждой модели данный этап реализуется по разному. В рамках реляционной модели данных мы подразумеваем следующие действия:

- создание описания набора реляционных таблиц и ограничений для них на основе информации, представленной в глобальной логической модели данных
- определение конкретных структур хранения данных и методов доступа к ним, обеспечивающих оптимальную производительность системы с базой данных
- разработка средств защиты создаваемой системы



# Пришло время рисовать!

Есть различные продукты для моделирования баз данных, на платные не выделен бюджет, на бесплатные нет времени. Используем самые оптимальные инструменты – бумагу и карандаш!

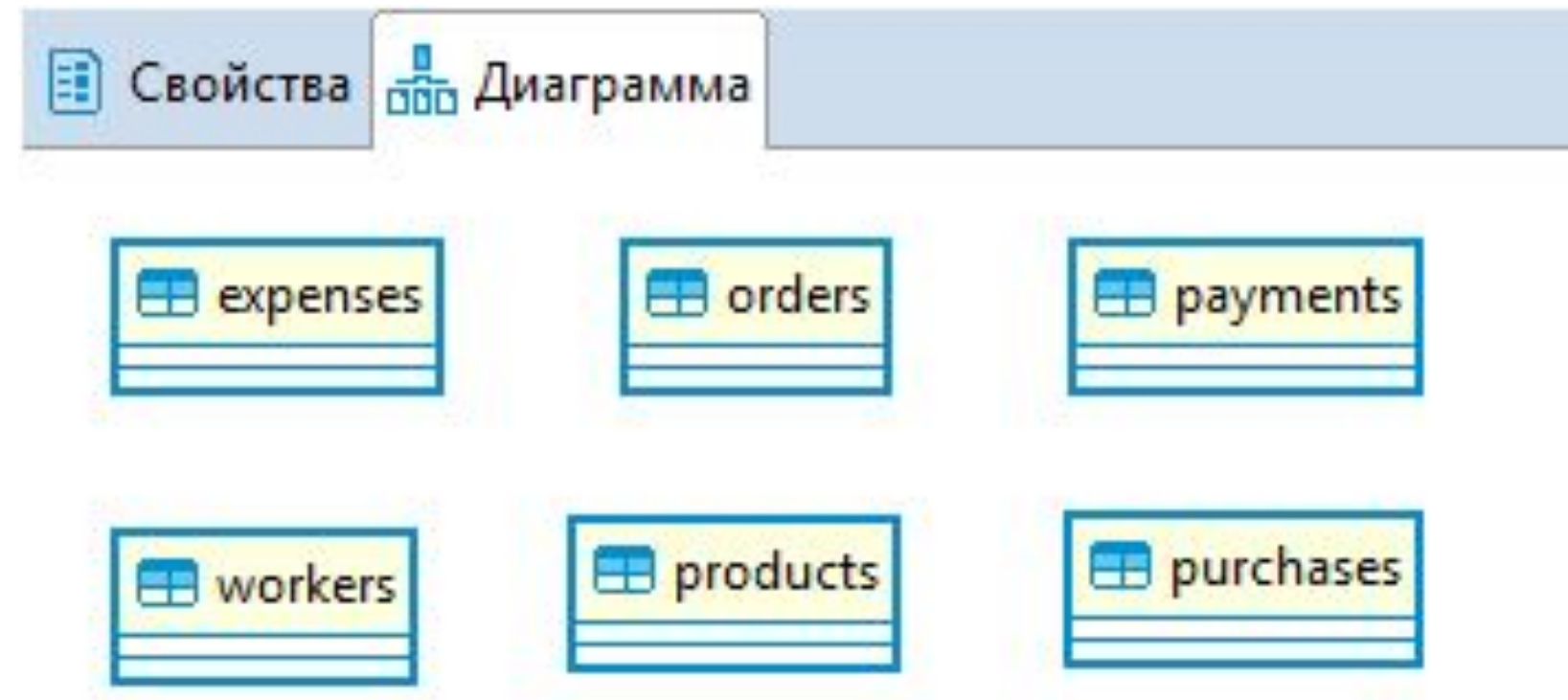
Наша база будет состоять из следующих таблиц:

- products (информация о товарах)
- workers (информация по сотрудникам)
- expenses (информация о затратах: налоги, аренда, заработная плата)
- purchases (информация о закупках)
- orders (заказы на доставку)
- payments (информация о продажах)



# Практика 1:

Отношения есть, теперь нам нужно их заполнить атрибутами.  
Составьте список атрибутов для каждой таблицы.



Защитная зона  
для интеграции  
видео спикера

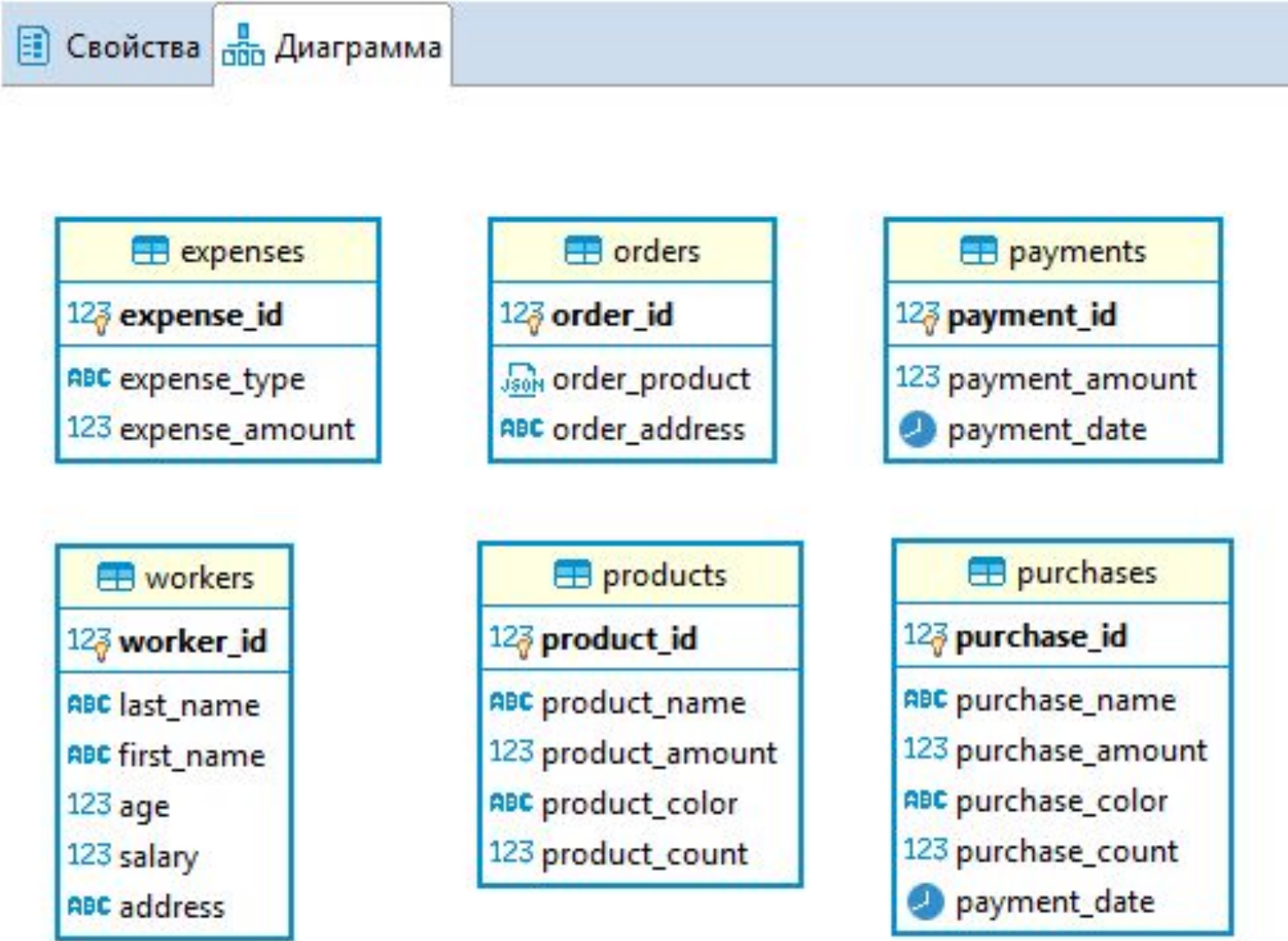




# Практика 2:



Мы предполагаем, что следующих атрибутов будет достаточно для решения всех поставленных задач. Какое ваше мнение? Стоит ли что-то изменить и почему?



# Избыточность данных



Избыточность вызывается прежде всего дублированием данных в разных кортежах одного отношения или в разных отношениях одной базы данных.

Есть схемы, в которых нельзя устранить избыточность, но надо стремиться к минимальной избыточности. Данные могут группироваться в таблицы разными способами.

При проектировании БД в качестве отправной точки может использоваться одно универсальное отношение, в которое включаются все необходимые атрибуты. Оно может содержать все данные, которые предполагается размещать в БД.

Как раз в этом случае и возникают проблемы избыточности и аномалий.

Проблема избыточности решается декомпозицией на несколько отношений.



# Аномалии:



**Включения:** представьте, что решили сделать классификацию цветов и у каждой группы есть свой идентификатор по которому мы можем ориентироваться. После закупки новых цветов, не попадающих под классификацию, идентификатор будет равен null и значит работать с ним будет проблематично.

**Удаления:** к примеру, мы создали отношение, в котором хранится информация и по цветам и по классификаторам, если мы удалим все товары, относящиеся к какому-то классификатору, то тем самым мы удалим любое упоминание об этом классификаторе.

**Модификации:** в таблице с товаром решили хранить информацию по поставщику, в какой-то момент у ряда позиций поставщик поменялся, часть данных мы изменили, а про какую-то забыли, соответственно данные стали ошибочны



# Что бы было удобно:

Защитная зона  
для интеграции  
видео спикера

```
create table products(  
    product_id serial primary key,  
    product_name varchar(50) not null unique,  
    product_amount decimal(10, 2) not null,  
    product_color varchar(30) not null,  
    product_count integer not null  
)
```

```
create table workers(  
    worker_id serial primary key,  
    last_name varchar(50) not null,  
    first_name varchar(30) not null,  
    age integer not null,  
    salary decimal(10, 2) not null,  
    address text not null  
)
```

```
create table expenses(  
    expense_id serial primary key,  
    expense_type varchar(50) not null,  
    expense_amount decimal(10, 2) not null  
)
```

```
create table purchases(  
    purchase_id serial primary key,  
    purchase_name varchar(50) not null unique,  
    purchase_amount decimal(10, 2) not null,  
    purchase_color varchar(30) not null,  
    purchase_count integer not null,  
    payment_date timestamp  
)
```



# Что бы было удобно 2:

Защитная зона  
для интеграции  
видео спикера

```
create table orders(  
  order_id serial primary key,  
  order_product json not null, --{product_id: count}  
  order_address text not null  
)
```

```
create table payments(  
  payment_id serial primary key,  
  payment_amount decimal(10, 2) not null,  
  payment_date timestamp default now()  
)
```

\* Данные запросы пригодятся при выполнении домашнего задания



# Связи, триггеры и интерфейс



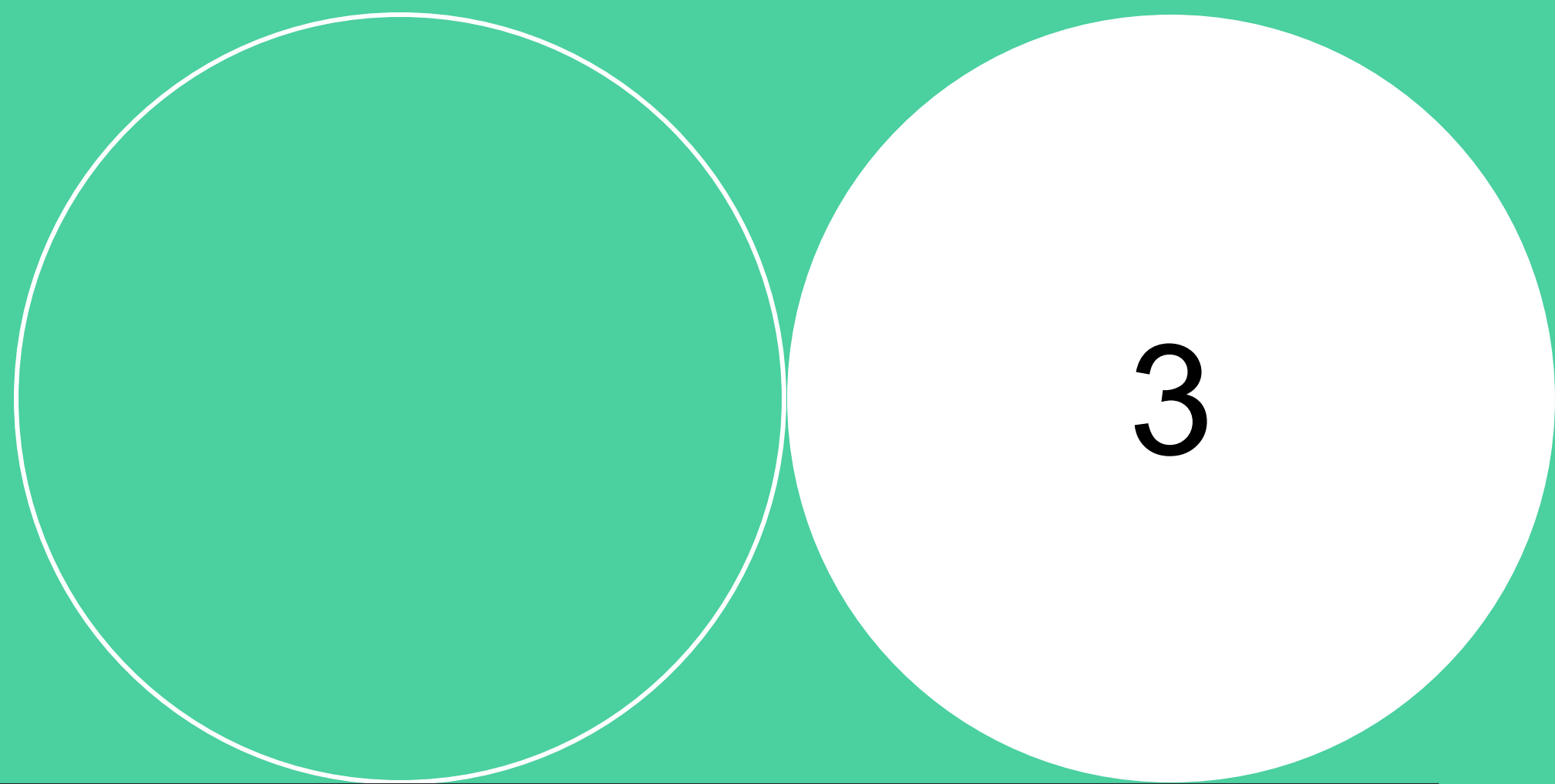
Связи и триггеры нужно будет создать при выполнении домашнего задания. При указании внешних ключей не должно возникнуть сложностей.

Так же при проектировании базы данных нужно понимать, что пользователь будет работать с базой через интерфейс, соответственно при разработке нужно вести документацию, на основании которой будет разрабатываться бэкенд для интерфейса.



---

# Масштабирование



---

Система не  
справляется — думать  
надо было раньше.

Защитная зона  
для интеграции  
видео спикера





---

# Что предпринять:

---

Защитная зона  
для интеграции  
видео спикера

- 1 Оптимизировать запросы
- 2 Апгрейдить машину
- 3 Читать документацию по Memcached (сервис кэширования данных в оперативной памяти на основе хеш-таблицы)
- 4 Сформировать несколько серверов и распределить нагрузку



# Оптимизировали. Апгрейдили. Кэшировали. Покупаем еще один сервер.



Шардирование — разделение данных на уровне ресурсов. Концепция шардинга заключается в логическом разделении данных по различным ресурсам исходя из требований к нагрузке.

Можно разделить таблицы по разным серверам, тем самым использовать вертикальное масштабирование.

Можно одну таблицу разбить на несколько таблиц с одинаковой структурой и распределить их между серверами, тем самым использовать горизонтальное масштабирование.

Так же стоит упомянуть про репликацию — это процесс, под которым понимается копирование данных из одного источника на множество других и наоборот. При репликации изменения, сделанные в одной копии объекта, могут быть распространены в другие копии. Репликация может быть синхронной или асинхронной.



# PL/Proxu – еще один процедурный язык?

Защитная зона  
для интеграции  
видео спикера



# Доступ к данным



PL/Proxy - прокси-язык для удаленного вызова процедур и партиционирования данных между разными базами. Основная идея его использования заключается в том, что появляется возможность вызывать функции, расположенные в удаленных базах, а также свободно работать с кластером баз данных (например, вызвать функцию на всех узлах кластера, или на случайном узле, или на каком-то одном определенном).

Таким образом пользователь делает обычный запрос к базе данных, но вызывает оно не чистый SQL-код, а заранее написанную функцию.

Далее база данных определяет на каком из серверов расположены требуемые данные и перенаправляет запрос на нужный сервер.

Запрос выполняется и возвращается на главный сервер после чего данные возвращаются в приложение.

Важно запомнить, что вся работа с использованием PL/Proxy идет через функции.



# А можно как-то проще?



Можно использовать расширение **postgres\_fdw**.

Предположим, что на удаленном сервере уже существует нужная нам база flowers, которая содержит часть данных магазина цветов.

```
create extension postgres_fdw; -- подключаем расширение

create server remote_flowers -- создаем удаленный сервер
foreign data wrapper postgres_fdw
options (host 'remote_host', port 'remote_port', dbname 'flowers');

create user mapping for postgres -- создаем маппинг для авторизации основного сервера к удаленному
server remote_flowers
options (user 'postgres', password 'postgres');
```



# Что мы получаем:

К примеру, на удаленном сервере хранится финансовая документация. Создадим на основном сервере таблицу идентичную удаленной.

```
create foreign table finance(  
    id int not null,  
    operation_name varchar(50) not null,  
    amount decimal(10,2) not null,  
    date timestamp  
)  
server remote_flowers  
options (schema_name 'public', table_name 'black_finance')
```

Для того, что бы работать с данными, создадим представление, через которое будем обращаться к удаленной таблице:

```
create view black_finance as  
select * from finance
```

Защитная зона  
для интеграции  
видео спикера





# Модификация данных:



Для того, что бы модифицировать данные, нам необходимо создавать правила.

```
create rule insert_to_finance as on insert to black_finance
-- если предположим, что несколько шардов (серверов),
-- то здесь нужно прописать условие для выбора нужного сервера
do instead insert into finance values (new.*)
```

По аналогии создаются различные правила для select / update / delete.

*\* Заметьте, что в настоящее время в postgres\_fdw не поддерживаются операторы INSERT с предложением ON CONFLICT DO UPDATE. Однако предложение ON CONFLICT DO NOTHING поддерживается, при отсутствии указания для выбора уникального индекса. Заметьте также, что postgres\_fdw поддерживает перемещение строк, вызванное командами UPDATE, выполняемыми для секционированных таблиц. Однако в настоящее время невозможно выполнить изменение, при котором удалённая секция, выбранная для добавления перемещаемой строки, также является целевой секцией для UPDATE и должна модифицироваться позже той же командой.*



# Краткий итог:



Существуют различные инструменты для работы с несколькими серверами. Какие-то инструменты встроены в PostgreSQL, какие-то идут отдельными приложениями и т.д.

У каждого инструмента есть положительные и отрицательные стороны: удобство использования, скорость обработки данных, нагрузка на систему и т.д.

Как уже говорили: нет идеального решения – есть оптимальное решение.

Если говорить про оптимальное решение для высоконагруженных систем, то это основной сервер, несколько серверов с горизонтальным шардингом, несколько с вертикальным и для каждого шарда по паре серверов с репликой для дублирования информации и повышения отказоустойчивости.

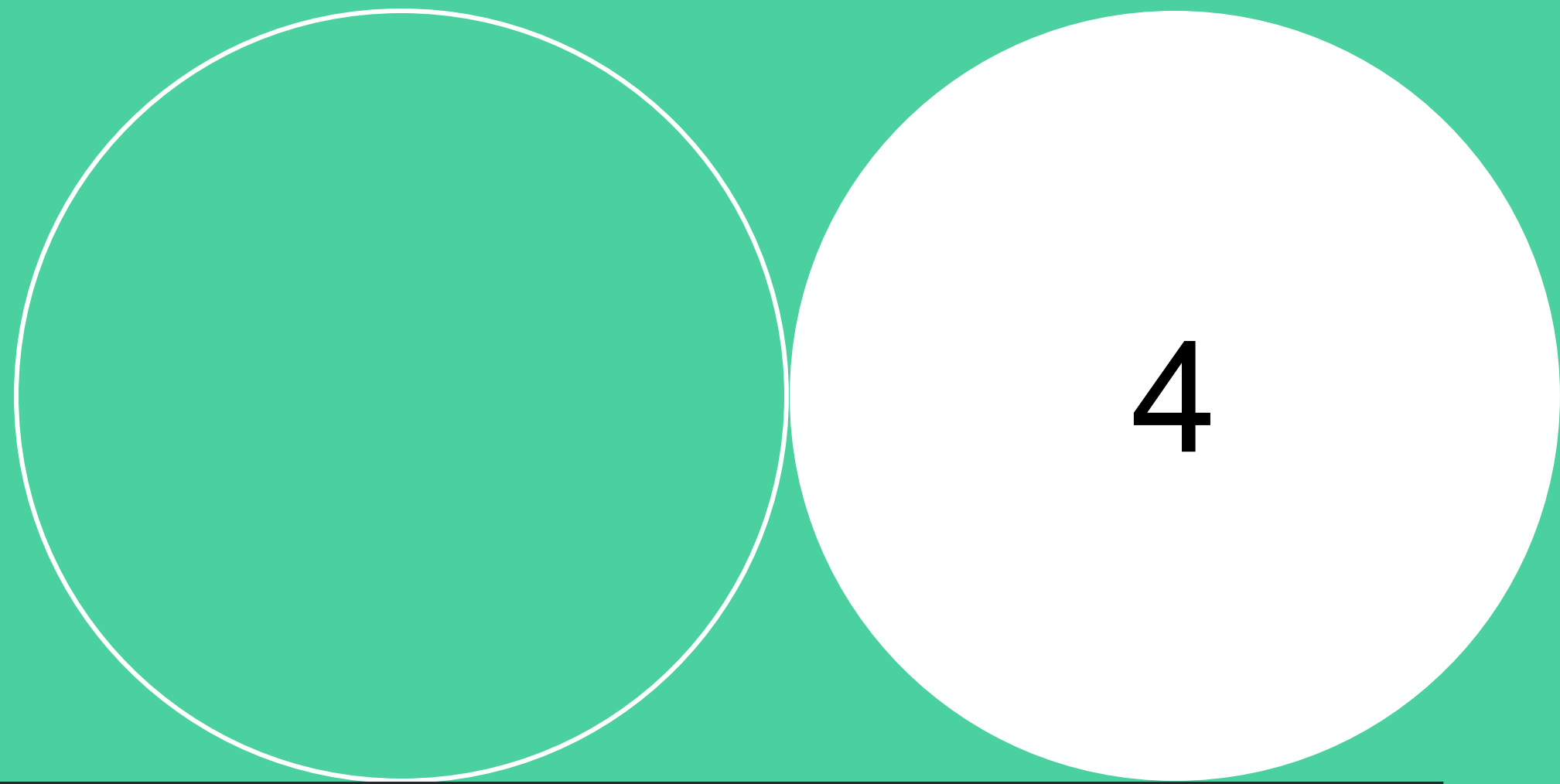
А далее давайте поговорим о горизонтальном масштабировании, которое бывает как в рамках одного сервера, так и нескольких.





---

# Партиционирование



# PostgreSQL - это объектно-реляционная база данных.

Защитная зона  
для интеграции  
видео спикера



# Что это такое?



Партиционирование (partitioning) — это разбиение больших таблиц на логические части по выбранным критериям. Партиционированные или секционированные таблицы призваны улучшить производительность и управляемость базами данных.

- А при чем тут *объектно*-реляционная база данных?
- А при том, что она знает о «наследовании»!

Партиционирование в PostgreSQL использует принципы наследования.

```
create table products_1 () inherits (products)
```

Таблица products\_1 будет наследовать, все методы и свойства родительской таблицы products

```
create table products_1 ( like products including all );
```

Таблица products\_1 будет содержать все индексы, ограничения и т.д. таблицы products, но без учета наследования, но наследование мы можем добавить позже используя запрос:

```
alter table products_1 inherit products
```



# Практика:



Давайте возьмем нашу базу магазина цветов, для таблицы products добавим индекс

```
create index on products(product_name)
```

Посмотрим, какие есть ограничения у таблицы: первичный ключ и индекс.

	ABC table_name	ABC constraint_name
1	products	products_pkey
2	products	products_product_name_key

Теперь посмотрим значение нашего счетчика

```
select * from products_product_id_seq
```

	123 last_value	123 log_cnt	<input checked="" type="checkbox"/> is_called
1	7	26	<input checked="" type="checkbox"/>

Наша таблица уже содержит данные:

	123 product_id	ABC product_name	123 product_amount	ABC product_color	123 product_count
1	1	фиалка	500	фиалковый	100
2	2	роза	1 000	розовый	250
3	3	сирень	320	сиреневый	37
4	4	гладиолус	705	гладиолусовый	80
5	5	астра	80	астровый	90
6	6	пион	50	пионовый	588
7	7	тюльпан	70	тюльпановый	255



# Практика :



Давайте создадим таблицу, которая будет наследовать таблицу products

```
create table products_temp () inherits (products);
```

Мы получили новую пустую таблицу

	123 product_id	ABC product_name	123 product_amount	ABC product_color	123 product_count

Название	#	Тип данных	Д...	То...	М...	А.	Прави...	Not Null	По умолчанию
123 product_id	1	int4		10				<input checked="" type="checkbox"/>	nextval('flowers.products_product_id_seq'::regclass)
ABC product_name	2	varchar	50	50			default	<input checked="" type="checkbox"/>	
123 product_amount	3	numeric		10	2			<input checked="" type="checkbox"/>	
ABC product_color	4	varchar	30	30			default	<input checked="" type="checkbox"/>	
123 product_count	5	int4		10				<input checked="" type="checkbox"/>	

И если обратить внимание, то счетчик привязан к родительской таблице





# Практика :

А что будет, если посмотреть план запроса?

```
explain analyze  
select * from products p
```

Мы увидим, что сканируется две таблицы

	ABC QUERY PLAN
1	Append (cost=0.00..14.27 rows=327 width=220) (actual time=0.013..0.019 rows=7 loops=1)
2	-> Seq Scan on products p (cost=0.00..1.07 rows=7 width=220) (actual time=0.012..0.014 rows=7 loops=1)
3	-> Seq Scan on products_temp p_1 (cost=0.00..13.20 rows=320 width=220) (actual time=0.003..0.003 rows=0 loops=1)
4	Planning time: 0.298 ms
5	Execution time: 0.049 ms

Это хорошо, но давайте еще раз.

```
drop table products_temp
```

Защитная зона  
для интеграции  
видео спикера



# Практика :



Давайте выполним два следующих запроса:

```
create table products_temp (like products including all);
alter table products_temp inherit products;
```

Что с индексом и первичным ключом?

```
select ccu.table_name, ccu.constraint_name
from information_schema.constraint_column_usage ccu
where ccu.table_name = 'products_temp'
```

	ABC table_name	ABC constraint_name
1	products_temp	products_temp_pkey
2	products_temp	products_temp_product_name_key

Теперь у нас есть таблица (партиция), которая имеет все те же ограничения, связи и параметры, что и родительская таблица



# И снова триггерные функции



Если обратиться к родителю, то сканироваться будет родительская и дочерние таблицы.  
Если обратиться к дочерней таблице, то сканироваться будет только дочерняя таблица.  
Если хотим обратиться только к родительской таблице без сканирования дочерних:

```
select * from ONLY products
```

select / update / delete при обращении к родителю сканируют и дочерние таблицы, то есть работа идет со всеми партициями.

Для работы с insert применяем триггерные функции.

Сперва нужно определить, по какому критерию будет происходить партиционирование: идентификатор, год, первая буква сущности и т.д.

Далее создаем необходимое количество партиций и пишем логику распределения новых записей по партициям с помощью триггерных функций.

Так как мы должны как-то ограничивать данные в партициях, нужно для каждой такой таблицы создать условие, к примеру, если разбиваем по идентификатору, то:

```
alter table products_temp add check (product_id > 1000);
```

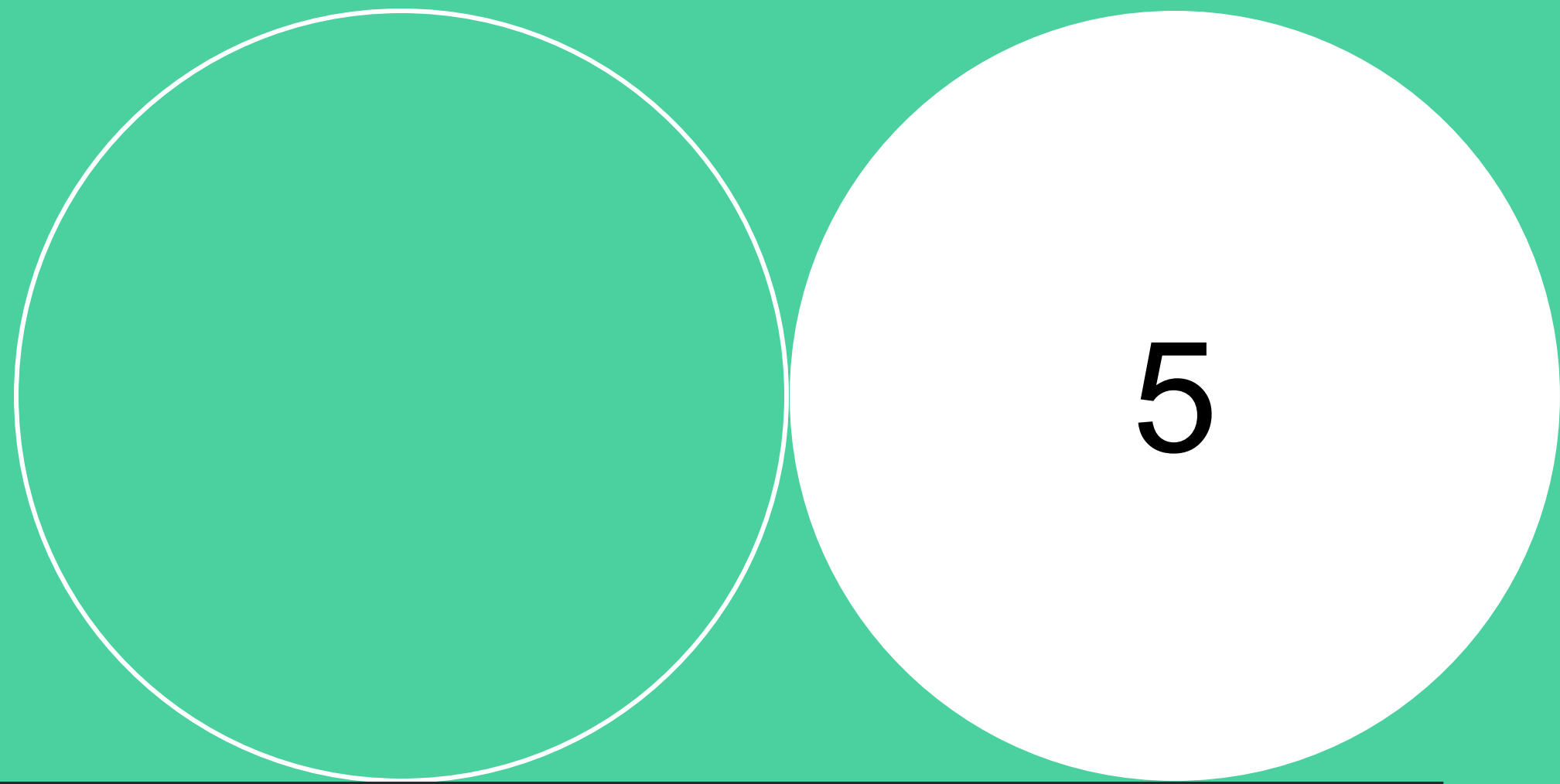
После мы знаем, что если product\_id будет менее либо равен 1000, то данные добавятся в родительскую таблицу, в противном случае в дочернюю.





---

# Итоги



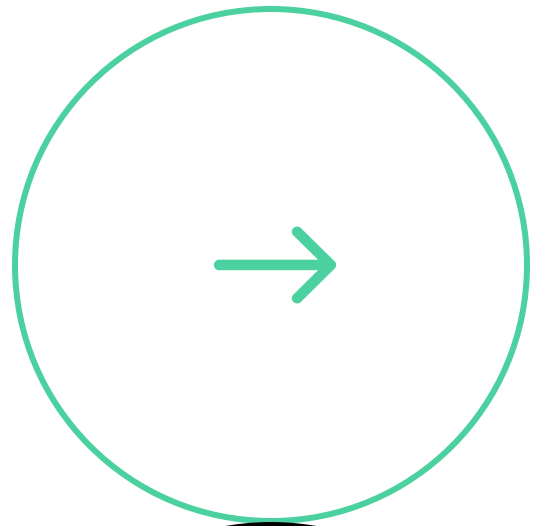
---

Николай Хащанов

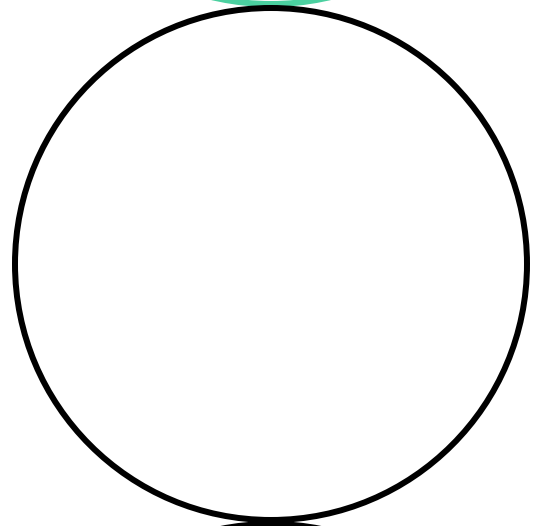
---

Продвинутый SQL

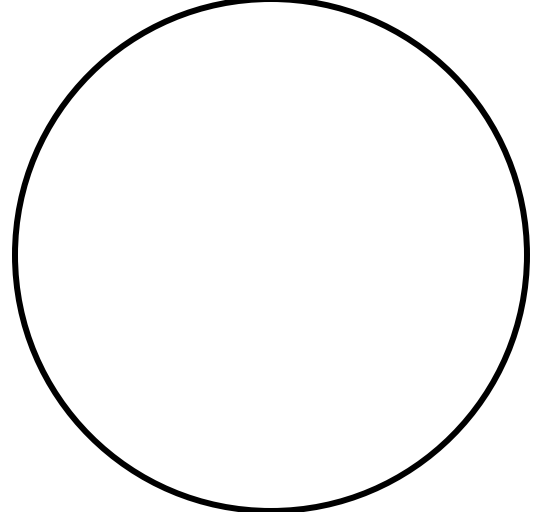
# Подведем итог:



Рассмотрели архитектурные модели



Разобрались в основах проектирования



Углубились в масштабирование и  
поговорили о партиционировании



---

# Домашнее задание



6

## Задание 1.

- Используя запросы из презентации создайте базу данных магазина по продаже цветов
- Проанализируйте, все ли отношения и атрибуты удовлетворяют требованиям
- Добавьте все необходимые связи между отношениями / атрибутами



*И тут позвонил Заказчик. В разработку запустили интернет магазин.*

## Задание 2.

- Масштабируйте существующую базу используя вертикальный шардинг, добавив возможность хранения информации с интернет магазина:
  - создайте необходимые отношения (пользователь, заказ)
  - создайте необходимые атрибуты
  - создайте необходимые связи
  - создайте подключение к удаленному серверу используя postgres\_fdw и сделайте возможность работы с данными на основном сервере (три запроса на получение данных и 2 запроса на модификацию данных).

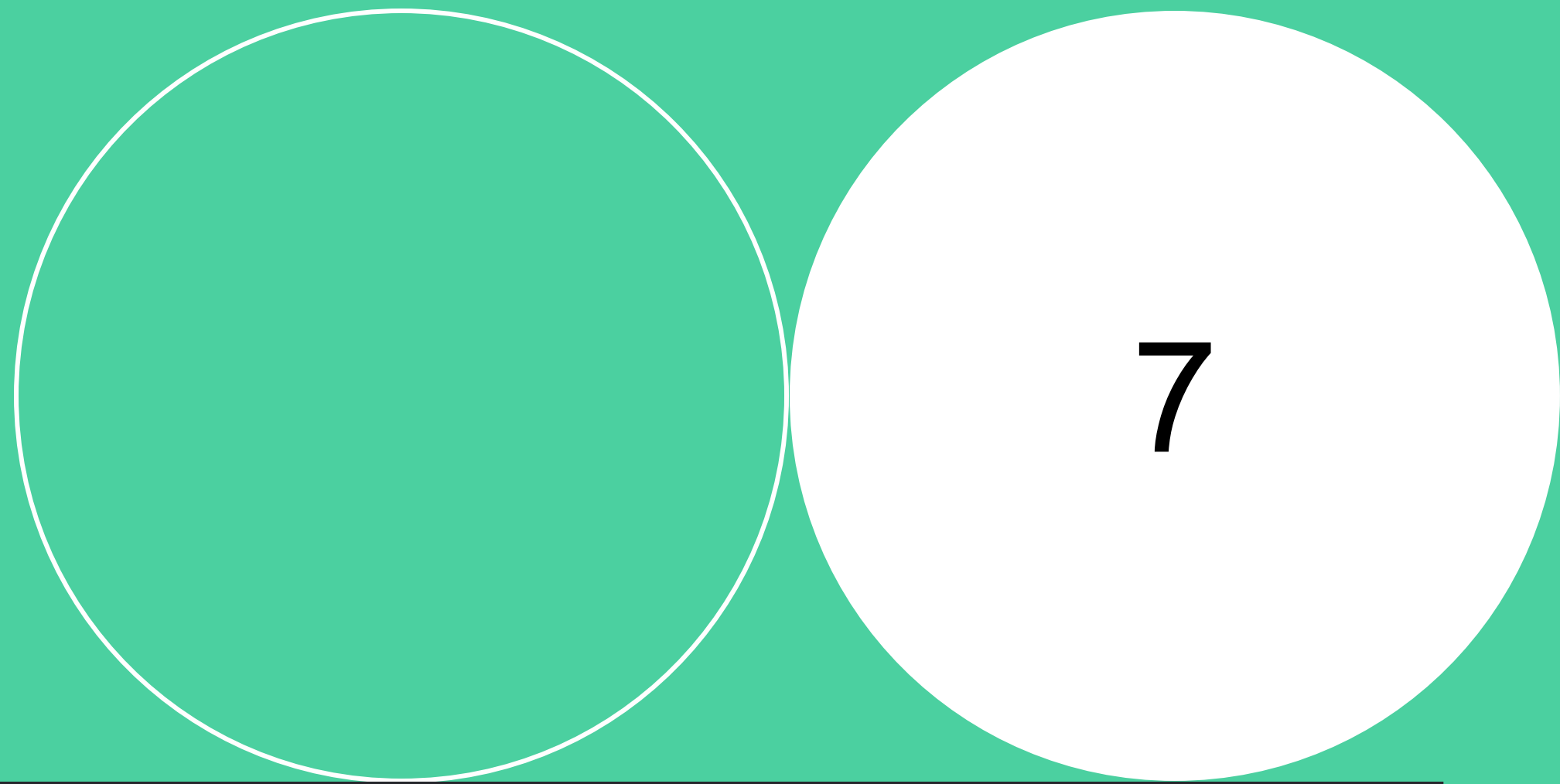
## Дополнительное задание:

- Напишите необходимые функции и триггеры для автоматизации работы с данными (творческое



---

# Полезные ссылки



Партиционирование:

<https://habr.com/ru/post/273933/>

Масштабирование:

<https://knasys.ru/wp-content/uploads/2014/10/postgresql.pdf>

Модель «сущность — связь»:

<https://nsu.ru/xmlui/bitstream/handle/nsu/9026/Chen1995.pdf>

Размещение файлов базы данных:

<https://postgrespro.ru/docs/postgresql/12/storage-file-layout>

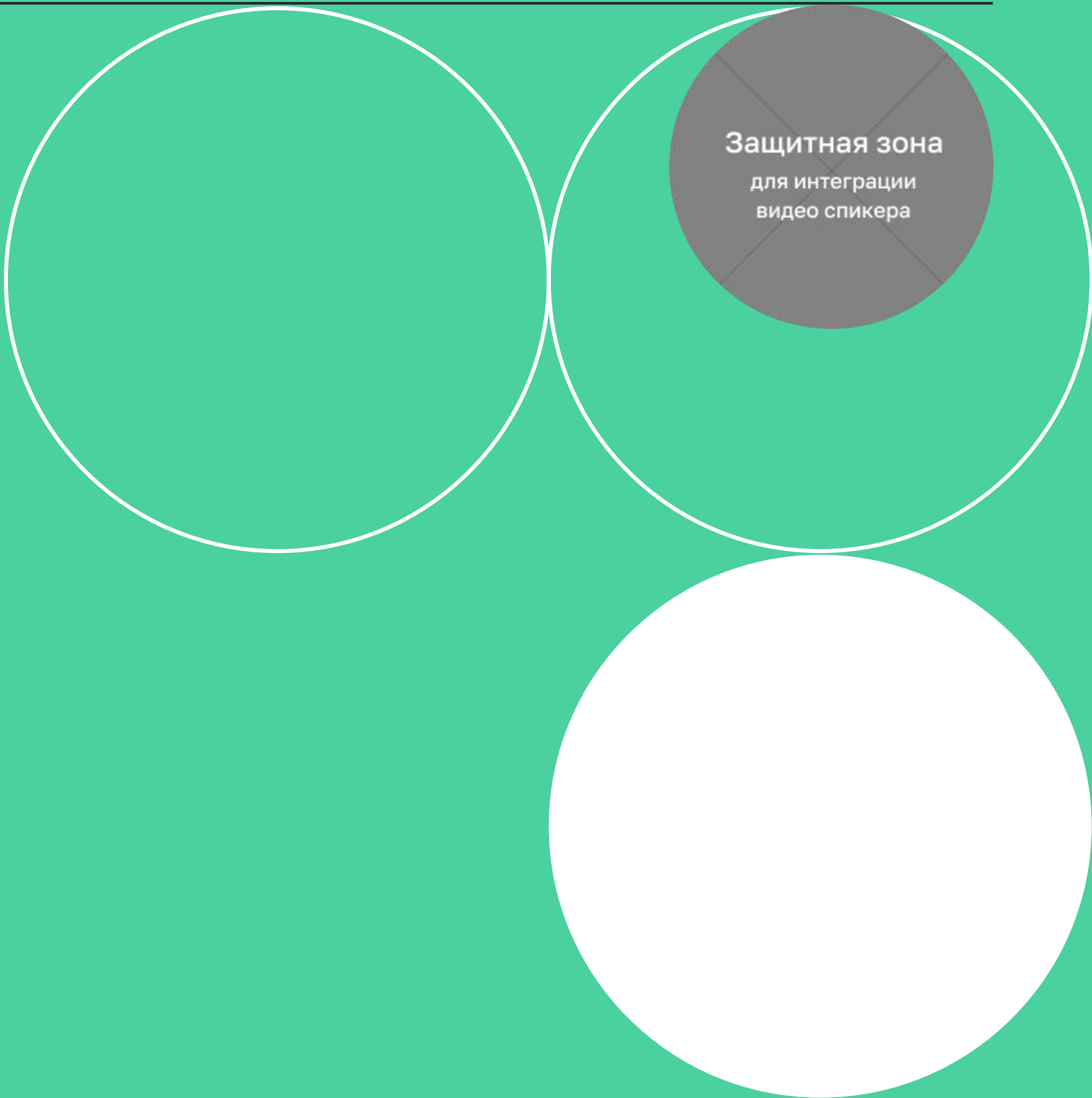




# Спасибо за внимание!

---

Николай Хащанов



Защитная зона  
для интеграции  
видео спикера