ЦИФРОВАЯ КУЛЬТУРА
УНИВЕРСИТЕТ ИТМО

Cassandra

Высшая Школа Цифровой Культуры
Университет ИТМО
dc@itmo.ru

Содержание

1 Введение в Cassandra

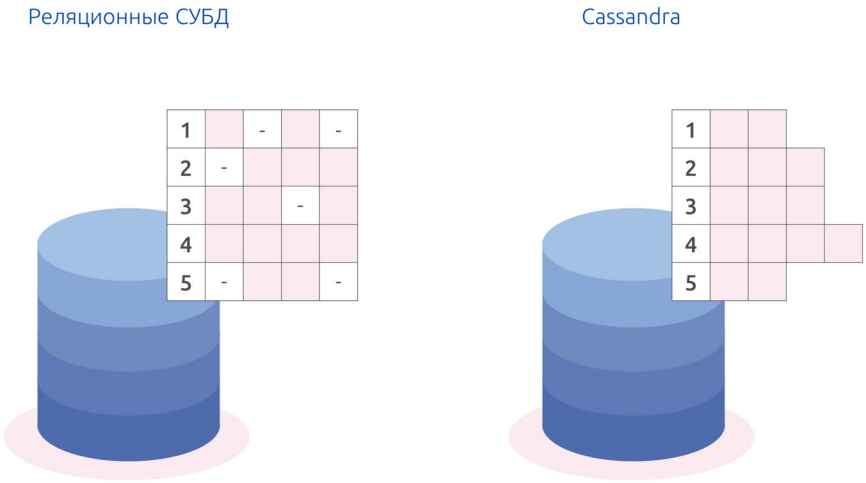
В этом и последующих фрагментах данной лекции мы обсудим модель данных в СУБД Cassandra и язык запросов Cassandra Query Language (CQL). Почему именно Cassandra? Да просто потому, что это хранилище данных

считают наиболее ярким представителем группы колоночных хранилищ.

CQL во многих отношениях напоминает SQL, но, тем не менее, имеет ряд важных отличий.

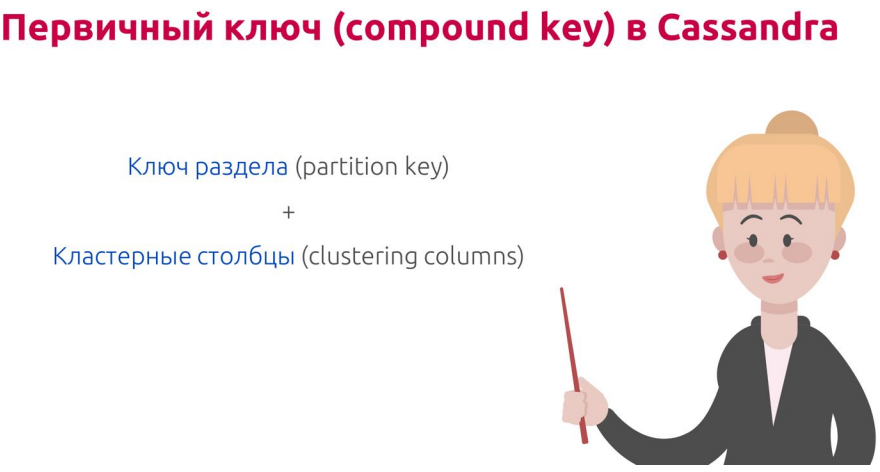
Для начала познакомимся с терминологией, используемой в СУБД Cassandra. На первый взгляд, Cassandra кажется похожей на обычные реляционные СУБД. Например, в Cassandra тоже есть таблицы. Однако, в Cassandra таблицей называется специальная логическая структура, объединяющая похожие данные. Например, могут быть таблицы, представляющие группу пользователей, студентов, отелей и т.п. В этом смысле таблица Cassandra аналогична реляционной таблице. Принципиальное отличие состоит в том, что в Cassandra нет необходимости хранить значения каждого столбца (или поля) при сохранении новой сущности. Возможно, значения некоторых столбцов неизвестны. Например, у одних людей есть второй номер

телефона, а у других - нет. Или в форме на странице сайта, реализованного с использованием Cassandra, одни поля обязательны, а другие - нет. Это нормально. Вместо того чтобы хранить значение null в качестве признака от-



сутствия значения и расходовать на это место, Cassandra вообще не хранит соответствующее значение столбца для такой строки. При проектировании таблицы в традиционной реляционной базе данных мы обычно имеем дело с «сущностями», т.е. наборами атрибутов, описывающих некоторый объект. Мы не задумываемся о размере строк, потому что после того, как решение о том, как объект представляется в виде таблицы, размер строки уже не подлежит обсуждению. Но при работе с Cassandra размер строки фиксирован не так жестко: строка может быть широкой или узкой в зависимости от количества присутствующих в ней столбцов.

В Cassandra для представления широких строк, называемых также разделами (partition), используется специальный первичный ключ. Его иногда называют составным ключом (compound key). Составной ключ состоит из ключа раздела (partition key) и необязательного набора кластерных столбцов (clustering column). Ключ раздела (partition key) служит для определения уз-

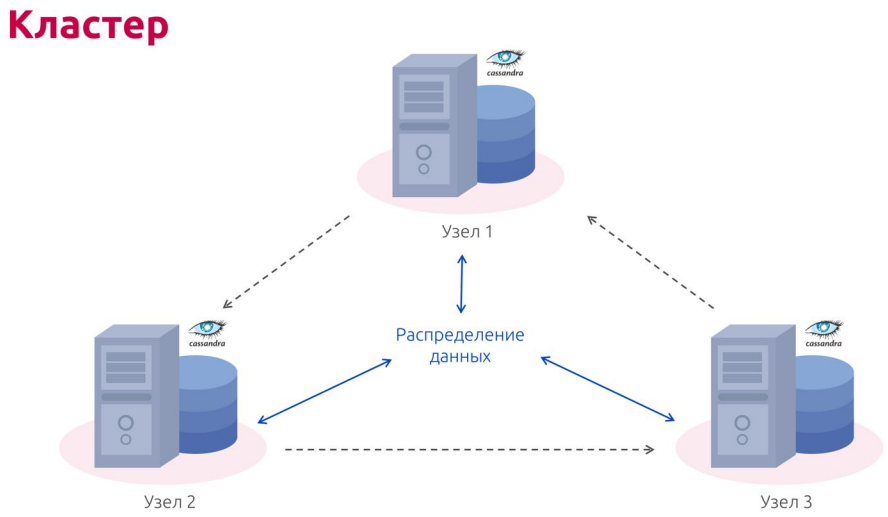


лов распределенной системы, на которых хранятся строки. Ключ раздела может состоять из нескольких столбцов. Кластерные же столбцы используются для управления сортировкой данных при хранении внутри каждого раздела.

Основными структурами данных в Cassandra являются: столбец, строка, таблица, пространство ключей и кластер. Познакомимся с ними поближе. Итак, столбец в понимании Cassandra – это пара ключ-значение; строка – это контейнер столбцов, на который можно сослаться по первичному ключу; таблица – контейнер строк; пространство ключей – контейнер таблиц, а кластер – контейнер пространства ключей, расположенных в одном или нескольких узлах сети.

Это взгляд снизу вверх на модель данных Cassandra. Познакомившись с основными терминами, перейдем к более детальному изучению каждой структуры.

База данных Cassandra изначально спроектирована для распределения данных между несколькими серверами, которые обрабатываются совместно и представляются пользователю единым целым. Поэтому самая внеш-

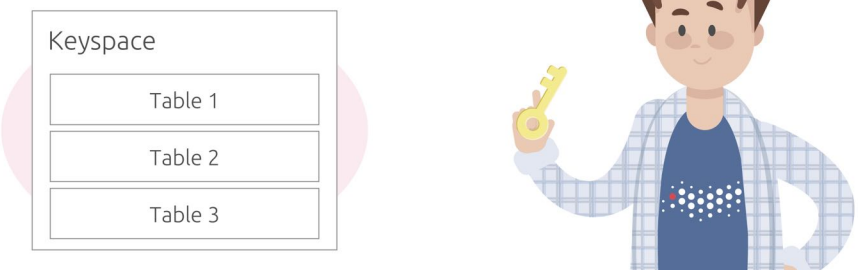


няя структура в Cassandra называется кластером, или кольцом, потому что Cassandra распределяет данные между узлами кластера, считая их закольцованными.

Кластер – это контейнер пространства ключей. Пространство ключей (keyspace) – самый внешний контейнер данных в Cassandra, во многом аналогичный реляционной базе данных. Как база данных является контейнером таблиц в реляционной модели, так пространство ключей – контейнер таблиц в модели данных Cassandra. Как и у реляционной базы данных, у пространства ключей есть имя и набор атрибутов, определяющих его поведение в целом.

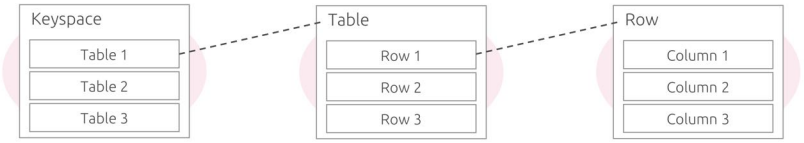
Таблица – это контейнер для упорядоченной коллекции строк, каждая из которых, в свою очередь, представляет собой упорядоченную коллекцию столбцов. Порядок определяется столбцами, назначенными в качестве

Пространство ключей



ключей. В Cassandra могут существовать и дополнительные ключи, помимо первичного. При записи данных в таблицу задаются значения одного или нескольких столбцов. Эта коллекция значений называется строкой. Хотя бы одно из заданных значений должно быть первичным ключом, который играет роль уникального идентификатора строки.

Пространство ключей, таблицы и строки



Как проектируются модели данных в Cassandra? Прежде, чем начнем проектировать модель данных для Cassandra, обратим внимание на ключевые особенности проектирования. В Cassandra нет операций соединения. Соединение таблиц, если оно необходимо, приходиться производить на стороне клиента, либо создавать дополнительную (т.е. денормализованную) таблицу для хранения результатов виртуального соединения. При создании моделей данных для Cassandra предпочтителен второй вариант. Соединение на стороне клиента применяется очень редко, лучше продублировать (т.е. денормализовать) данные.

Несмотря на поддержку облегченных транзакций и пакетов совместно выполняемых команд, в Cassandra нет понятия ссылочной целостности таблиц. В реляционной базе данных мы можем определить в таблице внешний ключ для ссылки на первичный ключ в другой таблице. В Cassandra такого механизма нет.

При проектировании реляционной базы данных чрезвычайно важным фактором считается нормализация таблиц (т.е. чтобы данные в таблицах не повторялись многократно и т.п.). Но при работе с Cassandra это вовсе не считается преимуществом, потому что оптимальная производительность в Cassandra достигается тогда, когда модель данных денормализована. Порою

Различия в проектировании для РСУБД и Cassandra

- Отсутствие соединений
- Отсутствие ссылочной целостности
- Допустима денормализация данных
- Методика проектирования от запроса
- Оптимизация хранения на этапе проектирования
- Решение о сортировке принимается при проектировании

и в реляционной базе приходится идти на денормализацию, и тому есть две причины.

Первая – производительность. Разработчику программного обеспечения попросту не удается добиться нужной производительности, если необходимо соединять накопившиеся за много лет данные, поэтому он производит денормализацию, подстраиваясь под нужды известных запросов. Этот подход работает, но идет вразрез с идеологией реляционной модели, так, что, в конце концов, возникает вопрос, а является ли реляционная база оптимальным решением в данных условиях. *Вторая* причина намеренной денормализации –

2 причины денормализации данных

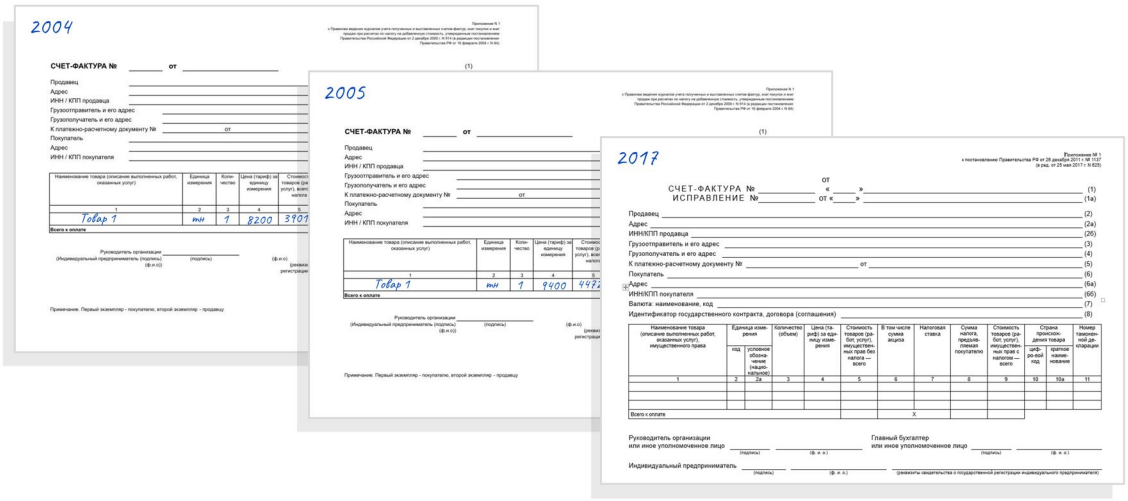
- Производительность
- Структура документа, нуждающегося в длительном хранении

структура документа, нуждающегося в длительном хранении. То есть имеется основная таблица, которая ссылается на много внешних таблиц, данные в которых со временем изменяются, однако необходимо сохранить для истории документ в том виде, в котором он существовал в момент создания.

Типичный пример – счета-фактуры. У нас уже есть таблицы заказчиков и продуктов, и, казалось бы, счет-фактура может просто сослаться на эти таблицы. Но на практике так никогда не делается. Информация о заказчике

или ценах может измениться, да сама форма и количество полей в счете фактуры может измениться и тогда будет нарушена целостность счета-фактуры в том виде, в котором она существовала на момент создания. А это приведет

2 причины денормализации данных



к ошибкам при аудите, в отчетах, к нарушению законов и прочим проблемам. В реляционном мире денормализация нарушает классические принципы Кодда, и, поэтому, этого стараются избегать. Но в Cassandra денормализация – типичный прием и считается совершенно нормальным делом. Если модель простая, то денормализовывать ее не обязательно. Но и бояться этого не надо.

Но, вернемся к особенностям проектирования моделей данных в Cassandra. Еще одно важное отличие состоит в методике проектирования от запроса. Говоря простыми словами, реляционное моделирование означает, что мы начинаем с концептуальной модели предметной области, а затем представляем сущности, оказавшиеся в этой модели, таблицами. После этого назначаются первичные и внешние ключи для моделирования связей. Связи многие-ко- многим представляются связующими таблицами. В реальном мире связующим таблицам ничего не соответствует, это просто необходимый побочный эффект построения реляционных моделей. Определившись с таблицами, мы начинаем писать запросы, в которых данные из разных таблиц связываются с помощью ключей. В реляционном мире запросы – вещь вторичная. Предполагается, что если таблицы спроектированы правильно, то уж данные мы как-нибудь достанем. И обычно это – правда, даже если для решения задачи приходится прибегать к сложным подзапросам или соединениям. Напротив, в Cassandra мы начинаем не с модели данных, а с модели запросов. Вместо того, чтобы сначала построить модель данных, а затем писать запросы, при работе с Cassandra мы сначала продумываем запросы, а уже вокруг

Методика проектирования от запроса

Реляционные СУБД

- Представление сущностей в виде таблиц
- Назначение первичных и внешних ключей
- Проектирование запросов

Cassandra

- Проектирование запросов
- Создание таблиц

них организуем данные. Решите, какие запросы чаще всего будут предъявляться в приложении, а затем создавайте таблицы, поддерживающие их эффективное выполнение. Критики высказывают мнение, что проектирование от запросов чрезмерно ограничивает область применимости приложения, не говоря уже о моделировании базы данных. Но разве не разумно предположить, что мы обязаны продумать запросы приложения не менее тщательно, чем обдумываем особенности предметной области. Допустив ошибку, мы получим проблемы и в том, и в другом случае. Со временем запросы могут измениться, и тогда придется изменить структуру данных. Но это ничем не отличается от неправильного определения таблиц или необходимости добавления новых таблиц в РСУБД.

В реляционных базах структура хранения таблиц на диске обычно прозрачна для разработчика, и в контексте моделирования данных редко можно встретить рекомендации, касающиеся физического хранения таблиц. Но в Cassandra это важный аспект. Поскольку каждая таблица хранится в отдельном файле, важно, чтобы взаимосвязанные столбцы были определены вместе в одной и той же таблице. При проектировании модели данных в Cassandra важно минимизировать количество разделов, которые нужно просматривать при обработке запроса. Поскольку каждый раздел целиком расположен только на одном узле, быстрее всего выполняются запросы, которым нужно просматривать только один раздел.

Еще одна важная особенность в проектировании – учет сортировки данных в запросах. В РСУБД порядок, в котором возвращаются записи, легко изменить, включив в запрос фразу ORDER BY. Порядок сортировки по умолчанию не задается, и в отсутствии других указаний записи возвращаются в том порядке, в котором добавлялись. Чтобы изменить порядок сортировки, нужно всего лишь модифицировать запрос, причем сортировать можно по любому набору столбцов. Но в Cassandra к сортировке отношение другое: решение о ней принимается на этапе проектирования. Порядок сортировки в

запросе фиксирован и определяется, так называемыми, кластерными столбцами, заданными в команде `CREATE TABLE`. Команда `SELECT` (которая также есть в Cassandra) поддерживает семантику `ORDER BY`, но только в порядке, определяемом кластерными столбцами.

В следующих фрагментах данной лекции познакомимся с приемами моделирования базы данных Cassandra и языком запросов CQL(Cassandra Query Language).

2 Проектирование модели данных

Для того, чтобы поближе познакомится с упомянутыми в предыдущем фрагменте принципами, попробуем построить модель данных для какого- нибудь конкретного приложения. Для начала опишем простую модель предметной области, вполне понятную в реляционном мире, а затем посмотрим, как перейти от реляционной модели к модели, принятой в Cassandra. Для этого примера нам понадобится нечто такое, что позволило бы продемонстрировать различные структуры данных и паттерны проектирования, но не настолько сложное, чтобы погрязнуть в деталях. К тому же предметная область должна быть всем знакома, чтобы сосредоточиться на особенностях Cassandra, а не на описании того, что мы пытаемся смоделировать. Подходящим примером

Предметная область <Top-100 Cartoon>

можно считать создание модели для приложения, предоставляющего различную информацию о некой базе данных с сотней лучших мультфильмов.

В концептуальной модели данных будут присутствовать названия мультфильмов, годы выпуска, страна происхождения, продолжительность, количе-

ство просмотров и режиссеры, принимавшие участие в создании мультфильмов. Мы уже упоминали эту предметную область ранее, но, тем не менее,

продемонстрируем структуру данных соответствующей предметной области в виде ER-модели, как это принято при проектировании реляционных баз данных.

Теперь попробуем применить подход проектирования от запроса к этим же данным. Структура проектируемого пользовательского интерфейса часто помогает, когда мы начинаем продумывать, какие необходимы запросы. Предположим, что мы переговорили со всеми заинтересованными сторонами, и дизайнеры предложили набросок интерфейса для основных выявленных сценариев использования.

Потом мы подготовим примерно такой список запросов, который вы видите на слайде. Какие именно это запросы:

1. Выдать названия мультфильмов по стране выпуска (результат отсортировать по количеству просмотров).

2. Для выбранного мультфильма выдать его продолжительность и год выпуска.

3. Для выбранного мультфильма указать список режиссеров.

4. Для выбранного режиссера указать названия его мультфильмов и количество просмотров (результат отсортировать лексикографически по названиям мультфильмов).

Теперь обсудим, какие логические таблицы должны подойти для этих сформулированных запросов.

Мы назвали первую таблицу `CARTOON_BY_COUNTRY`, чтобы отразить тот факт, что мультфильм будет выбираться по стране выпуска и определили колонки `Country` и `Cartoon_id` как ключевые (то есть входящие в состав ключа раздела). Мы включили в состав ключа колонку `Country`, так как по ней бу-

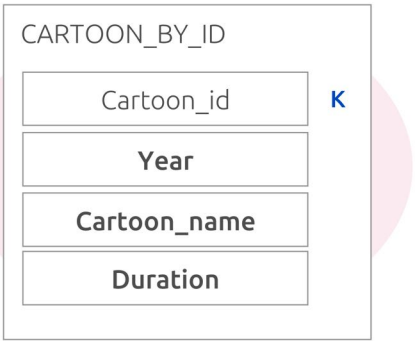
дет происходить запрос и добавили к ней колонку `Cartoon_id` для того, чтобы гарантировать, что такой составной ключ будет с одной стороны полезен для быстрого выполнения запроса, а с другой стороны – он будет однозначным образом идентифицировать запись в таблице. Кроме того, мы указали, что колонка `Views` – кластерная колонка и уточнили, что сортировка по кластерной колонке будет по убыванию. Оставшуюся колонку (`Cartoon_name`) включили для отображения в результате запроса.

Мы называли вторую таблицу `CARTOON_BY_ID`, чтобы отразить тот факт, что мультфильм будет выбираться по его уникальному идентификатору и

включили в состав ключа раздела колонку `Cartoon_id`. Остальные колонки `Cartoon_name`, `Year` и `Duration` включены для отображения в результате запроса.

Логическая таблица для запроса Q2

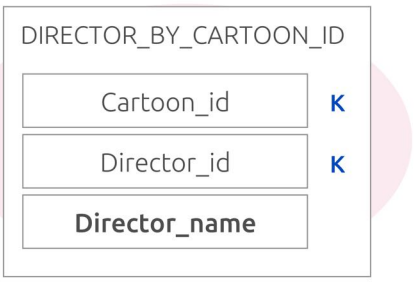
- Q2. Для выбранного мультфильма выдать его продолжительность и год выпуска



Мы называли третью таблицу – `DIRECTOR_BY_CARTOON_ID`, чтобы отразить тот факт, что режиссеры будут выбираться по уникальному идентификатору мультфильма и включили в состав ключа раздела колонки `Cartoon_id` и `Director_id`. Таким образом, мы обеспечили базу для быстрого выполнения запроса и уникальность ключа для идентификации записей в таблице. Оставшуюся колонку `Director_name` включили в состав таблицы для отображения в результате запроса.

Логическая таблица для запроса Q3

- Q3. Для выбранного мультфильма указать список режиссеров

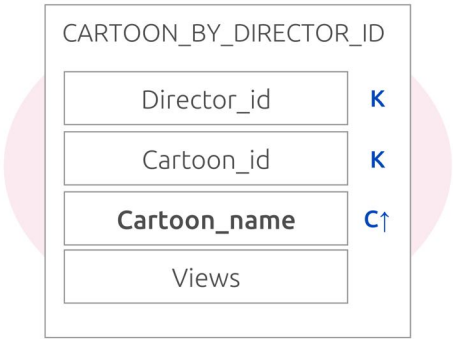


Мы назвали четвертую таблицу – `CARTOON_BY_DIRECTOR_ID`, чтобы отразить тот факт, что мультфильмы будут выбираться по уникальному

идентификатору режиссера и включили в состав ключа раздела колонки `Director_id` и `Cartoon_id`. Кроме того, колонка `Cartoon_name` была определена, как кластерная и для нее указана требуемая сортировка – по возрастанию.

Логическая таблица для запроса Q4

- Q4. Для выбранного режиссера указать названия его мультфильмов и количество просмотров (результат отсортировать лексикографически по названиям мультфильмов)



Следующим шагом после определения логической структуры таблиц будет построение физической модели данных. В несколько упрощенном варианте построение можно разбить на три этапа. На первом – определяются первичные ключи для таблиц, на втором – создаются пространства ключей, а на третьем – определяются возможные типы данных для колонок и таблицы сопоставляются подходящим пространствам ключей.

Построение физической модели данных

- Разнесение таблиц по пространствам ключей
- Определение типов полей
- Определение PRIMARY KEY для каждой таблицы (поля, отмеченные символом K + поля, отмеченные символом C)

Очередной этап проектирования – создание пространства ключей. Однако до того, надо определить какие пространства и в каком количестве понадобятся в принципе. В нашем случае представляется уместным создать два пространства ключей: `Cartoon` и `Director`.

В первом будут располагаться две логически связанные между собой таблицы: `CARTOON_BY_COUNTRY`, `CARTOON_BY_ID`, а во втором – `DIRECTOR_BY_CARTOON_ID` и `CARTOON_BY_DIRECTOR_ID`.

Разнесение таблиц по пространствам ключей

(CARTOON_BY_COUNTRY, CARTOON_BY_ID) → • KEYSPACE CARTOON

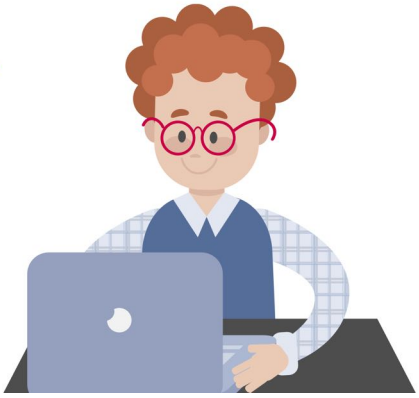
(DIRECTOR_BY_CARTOON_ID, CARTOON_BY_DIRECTOR_ID) → • KEYSPACE DIRECTOR

На следующем слайде демонстрируется, как можно определить пространство ключей. Не будем детализировать задаваемые характеристики про-

Определение пространства ключей на языке CQL

```
CREATE KEYSPACE CARTOON
WITH replication = { 'class' : 'SimpleStrategy' ,
'replication_factor' : 3};

CREATE KEYSPACE DIRECTOR
WITH replication = {'class' : 'SimpleStrategy' ,
'replication_factor' : 3};
```

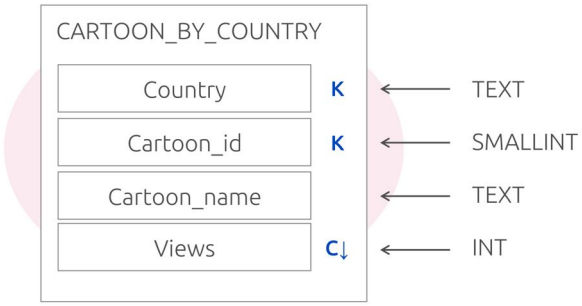


странства ключей, а лишь поясним, что они задают особенности механизма репликаций и количество создаваемых реплик.

Далее нужно определить типы колонок и привязать таблицы к подходящим пространствам ключей. Мы не будем подробно рассматривать все возможные типы колонок в Cassandra. В рамках одной лекции это просто невозможно. Любознательные слушатели легко смогут найти соответствующую информацию в открытой документации Cassandra. Для нашего примера вполне достаточно представлять, что есть три простейших типа: текстовый, целочисленный и короткий целочисленный. Именно их мы будем использовать при построении базы.

Итак, добавим к логической таблице `CARTOON_BY_COUNTRY` типы колонок, чтобы в затем определить таблицу на языке `CQL`. Определение типов –

Определение типов для таблицы CARTOON_BY_COUNTRY



достаточно несложная операция. Страна – текстовое поле, идентификатор мультфильма – короткое целое, название фильма – текстовое, количество просмотров – целое.

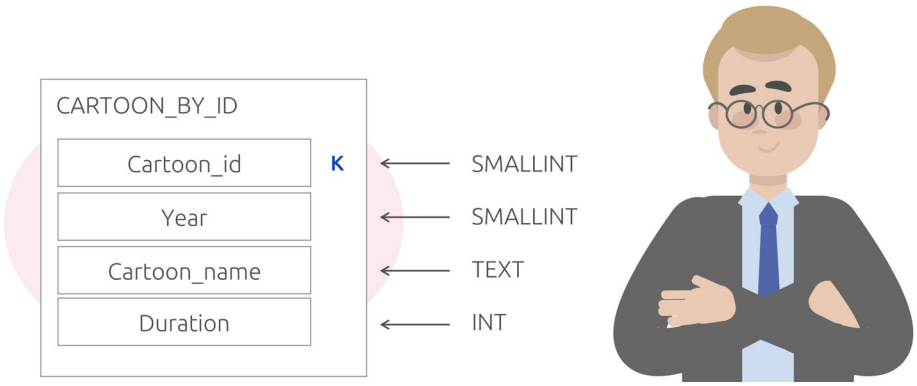
Вот как на языке CQL выглядит оператор, создающий новую таблицу. Обратите внимание, что `PRIMARY KEY` образуется из всех полей, объявленных как поля из ключа раздела плюс кластерные поля. В нашем случае – это три поля: `Country`, `Cartoon_id`, `Views`. Кроме того, в предложении `CREATE TABLE` встречается фрагмент `WITH CLUSTERING ORDER BY`, который явным образом указывает возможную сортировку в таблице по полю `Views`. И наконец, перед именем таблицы `CARTOON_BY_COUNTRY` указано имя пространства ключей `Cartoon`, в котором будет размещена таблица.

Определение таблицы CARTOON_BY_COUNTRY в CQL

```
CREATE TABLE CARTOON.CARTOON_BY_COUNTRY (
COUNTRY TEXT,
CARTOON_ID SMALLINT,
CARTOON_NAME TEXT,
VIEWS INT,
PRIMARY KEY((COUNTRY, CARTOON_ID),VIEWS)
WITH CLUSTERING ORDER BY (VIEWS DESC);
```

Далее определим типы для таблицы `CARTOON_BY_ID`: идентификатор мультфильма – короткое целое, год выпуска фильма – короткое целое, название фильма – текстовое, продолжительность фильма – целое.

Определение типов для таблицы CARTOON_BY_ID



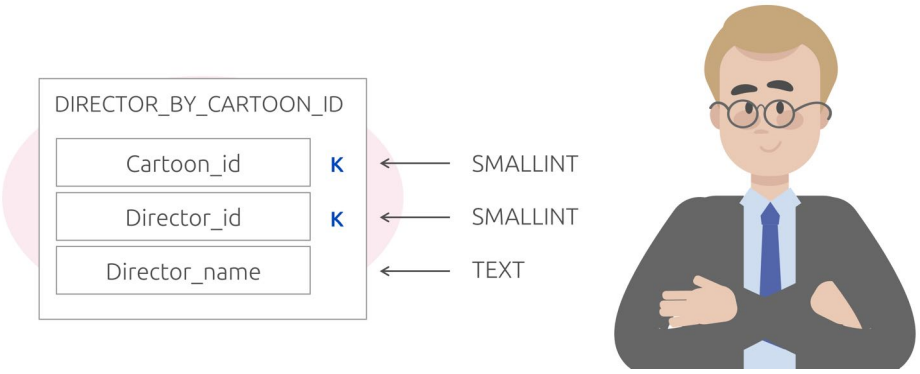
На экране можно видеть, как будет выглядеть оператор `CREATE TABLE`, создающий таблицу `CARTOON_BY_ID`. Обратите внимание, что первичный ключ имеет простейший вид, так как образуется из единственной колонки – идентификатор мультфильма, а перед именем таблицы `CARTOON_BY_ID` указано имя пространства ключей `CARTOON`, в котором будет размещена таблица.

Определение таблицы CARTOON_BY_ID в CQL

```
CREATE TABLE CARTOON.CARTOON_BY_ID (
CARTOON_ID SMALLINT PRIMARY KEY,
YEAR SMALLINT,
CARTOON_NAME TEXT,
DURATION INT);
```

На следующем слайде определение типов для таблицы `DIRECTOR_BY_CARTOON_ID`: идентификатор мультфильма – короткое целое, идентификатор режиссера – короткое целое, имя режиссера – текстовое.

Определение типов для таблицы DIRECTOR_BY_CARTOON_ID



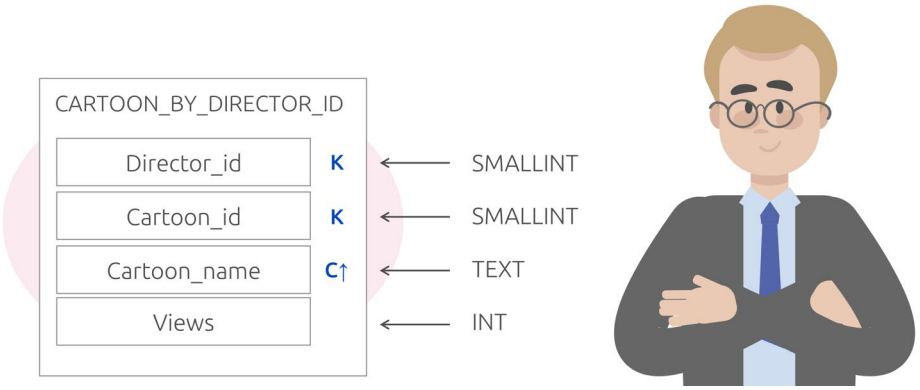
А вот и оператор `CREATE TABLE` для таблицы `DIRECTOR_BY_CARTOON_ID`. Как видите – первичный ключ образуется из идентификаторов мультфильма и режиссера, а перед именем таблицы `DIRECTOR_BY_CARTOON_ID` указано имя пространства ключей `DIRECTOR`, в котором будет размещена таблица.

Определение таблицы DIRECTOR_BY_CARTOON_ID в CQL

```
CREATE TABLE DIRECTOR.DIRECTOR_BY_CARTOON_ID (
CARTOON_ID SMALLINT,
DIRECTOR_ID SMALLINT,
DIRECTOR_NAME TEXT,
PRIMARY KEY(CARTOON_ID, DIRECTOR_ID));
```

И, наконец, определение типов для таблицы `CARTOON_BY_DIRECTOR_ID`: идентификатор режиссера – короткое целое, идентификатор мультфильма – короткое целое, название мультфильма – текстовое, количество просмотров – целое.

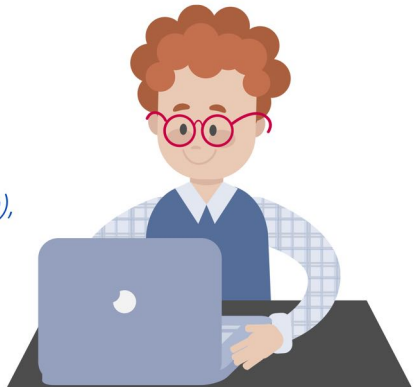
Определение типов для таблицы CARTOON_BY_DIRECTOR_ID



И последний оператор `CREATE TABLE` для таблицы `CARTOON_BY_DIRECTOR_ID`. Первичный ключ здесь образуется из идентификаторов мультфильма и режиссера, но, кроме этого, в него включается кластерное поле `CARTOON_NAME`. Кроме того, в предложении `CREATE TABLE` встречается фрагмент `WITH CLUSTERING ORDER BY`, который явным образом указывает возможную сортировку в таблице по полю `CARTOON_NAME`, а перед именем таблицы `CARTOON_BY_DIRECTOR_ID` указано имя пространства ключей `DIRECTOR`, в котором будет размещена таблица.

Определение таблицы CARTOON_BY_DIRECTOR_ID в CQL

```
CREATE TABLE DIRECTOR.CARTOON_BY_DIRECTOR_ID (
DIRECTOR_ID SMALLINT,
CARTOON_ID SMALLINT,
CARTOON_NAME TEXT,
VIEWS INT,
PRIMARY KEY ((DIRECTOR_ID, CARTOON _ID),
CARTOON_NAME)
WITH CLUSTERING ORDER BY
(CARTOON_NAME ASC);
```



Итак, мы продемонстрировали, как создаются базы в Cassandra, а в следующем фрагменте лекции познакомимся с операторами для работы с данными.

3 Работа с данными

В СУБД Cassandra существую таблицы, а, следовательно, должны существовать операторы, которые позволяют добавлять новые записи в таблицы, редактировать существующие и удалять морально устаревшие.

Ничего особенного в этих операторах в Cassandra нет. Это обычные SQL – подобные операторы `INSERT`, `UPDATE` и `DELETE` (то есть такие, как в реляционных СУБД). Синтаксис использования – почти такой же, как и в SQL. Тем не менее, посмотрим конкретные примеры, как добавлялись записи в созданные таблицы.

Вот как выглядит добавление записи в таблицу `CARTOON_BY_COUNTRY`. Это обычный оператор `INSERT` в котором указано в какую таблицу и какие именно значения следует добавить. Имена колонок, в которые добавляются

```
INSERT INTO
CARTOON.CARTOON_BY_COUNTRY
(COUNTRY, CARTOON_ID, CARTOON_NAME, VIEWS)
VALUES ( 'USA', 1, 'The Lion King', 5495742);
```

значения, перечислены через запятую. Сами добавляемые значения упоминаются после ключевого слова `VALUES` и также перечислены через запятую. Строковые значения `('USA', 'The Lion King')` выделяются одиночными кавычками. Единственной особенностью этого оператора можно считать использование имени пространства ключей `CARTOON` перед именем таблицы.

Еще один пример демонстрирует использование оператора `INSERT` при добавлении новой записи в таблицу `CARTOON_BY_ID`, расположенную в пространстве ключей `CARTOON`. Добавляется идентификатор мультфильма

```
INSERT INTO
CARTOON.CARTOON_BY_ID
(CARTOON_ID, YEAR, CARTOON_NAME, DURATION)
VALUES (1, 1994, 'The Lion King', 88);
```

`CARTOON_ID`, год создания `YEAR`, название `CARTOON_NAME` и продолжительность `DURATION`.

Следующий пример демонстрирует использование оператора `INSERT` при добавлении пары записей в таблицу `DIRECTOR_BY_CARTOON_ID`, расположенную в пространстве ключей `DIRECTOR`. В таблицу добавляются колонки: идентификатор мультфильма `CARTOON_ID`, идентификатор режиссера

```
INSERT INTO
DIRECTOR.DIRECTOR_BY_CARTOON_ID
(CARTOON_ID,DIRECTOR_ID,DIRECTOR_NAME)
VALUES (1,1, 'Rodger Allers');

INSERT INTO
DIRECTOR.DIRECTOR_BY_CARTOON_ID
(CARTOON_ID,DIRECTOR_ID,DIRECTOR_NAME)
VALUES (1,2, 'Rob Minkoff');
```

`DIRECTOR_ID`, имя режиссера `DIRECTOR_NAME` и соответствующие им значения для режиссеров `Rodger Allers` и `Rob Minkoff`.

И, наконец, последний пример на добавление записей демонстрирует добавление пары записей в последнюю из упомянутых ранее таблиц – `CARTOON_BY_DIRECTOR_ID` при этом, как обычно, перед именем таблицы указывается имя соответствующего пространства ключей. В последнем случае

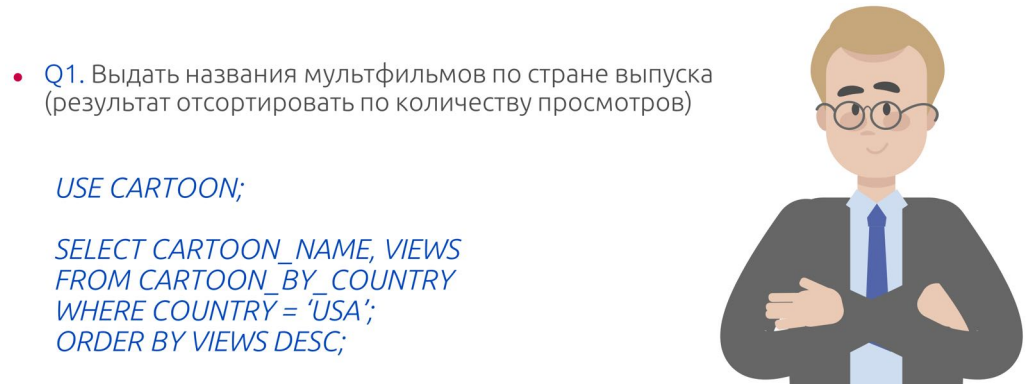
```
INSERT INTO
DIRECTOR.CARTOON_BY_DIRECTOR_ID
(DIRECTOR_ID,CARTOON_ID,CARTOON_NAME,VIEWS)
VALUES(1,1, 'The Lion King', 5495742);

INSERT INTO
DIRECTOR.CARTOON_BY_DIRECTOR_ID
(DIRECTOR_ID,CARTOON_ID,CARTOON_NAME,VIEWS)
VALUES(2,1, 'The Lion King', 5495742);
```

это пространство `DIRECTOR`.

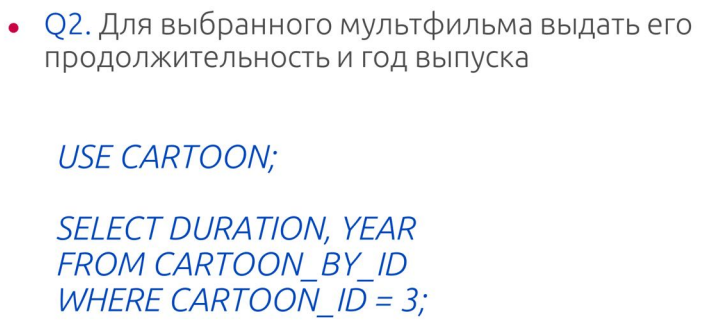
Итак, база данных создана и заполнена. Возникает следующая проблема – выборка данных. Для выборки данных в CQL используется оператор `SELECT`. Он чрезвычайно похож на `SELECT`, который используется в реляционных базах, однако имеет большое количество ограничений в использовании. Так, например, в операторе `SELECT` нельзя указывать условие для соединения двух таблиц, так как операции соединения в Cassandra в принципе нет. Можно сортировать записи, используя оператор `SELECT`, но только в том случае, если соответствующее поле было объявлено как поле для сортировки. Рассмотрим несколько примеров его использования.

Первый пример демонстрирует запрос типа `Q1`, который был предусмотрен при проектировании базы. Устанавливается подходящее пространство ключей (`USE CARTOON`), а затем выбираются названия мультфильмов и количество просмотров из таблицы `CARTOON_BY_COUNTRY` в соответствии с заданным фильтром `COUNTRY = 'USA'`. Используется конструкция `ORDER BY`



`VIEWS DESC` так как именно она была предусмотрена при создании модели базы.

Второй пример демонстрирует запрос типа `Q2`, который также был предусмотрен при проектировании базы. Выбираются продолжительность и



год выпуска мультфильма из таблицы `CARTOON_BY_ID` в соответствии с заданным фильтром `CARTOON_ID = 3`.

Третий пример демонстрирует запрос типа `Q3`, который был предусмотрен при проектировании базы. Устанавливается соответствующее пространство ключей (`USE DIRECTOR`), а затем выбирается список режиссеров мультфильма из таблицы `DIRECTOR_BY_CARTOON_ID`. Мультфильм определяется в соответствии с заданным фильтром `CARTOON_ID = 3`.

- Q3. Для выбранного мультфильма указать список режиссеров

```
USE DIRECTOR;

SELECT DIRECTOR_NAME
FROM DIRECTOR_BY_CARTOON_ID
WHERE CARTOON_ID = 3;
```

Последний пример демонстрирует запрос типа `Q4`, который был предусмотрен при проектировании базы. Выбираются названия мультфильмов и количество просмотров из таблицы `CARTOON_BY_DIRECTOR_ID` в соответствии с заданным фильтром `DIRECTOR_ID = 5`. Используется конструкция ORDER

- Q4. Для выбранного режиссера указать названия его мультфильмов и количество просмотров (результат отсортировать лексикографически по названиям мультфильмов)

```
USE DIRECTOR;

SELECT CARTOON_NAME, VIEWS
FROM CARTOON_BY_DIRECTOR_ID
WHERE DIRECTOR_ID = 10
ORDER BY CARTOON_NAME ASC;
```

BY `CARTOON_NAME ASC` так как именно она была предусмотрена при создании модели базы.

В рамках одной лекции мы рассмотрели, разумеется, не все аспекты использования Cassandra. Поэтому слушателям, которым этих сведений оказалось недостаточно, как обычно, советуем читать официальную документацию Cassandra: `https://cassandra.apache.org/doc/latest/`