



Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ Информатика, искусственный интеллект и системы управления

КАФЕДРА Системы обработки информации и управления

**Методические указания к лабораторным работам
по курсу «Постреляционные базы данных»**

**Лабораторная работа №3
«Создание колоночной базы данных и работа с ней
на примере СУБД Cassandra»**

Виноградова М.В., Королев С.В., Макаров Д.А.

Под редакцией к.т.н. доц. Виноградовой М.В.

Москва, 2022 г.

ОГЛАВЛЕНИЕ

1. ЗАДАНИЕ 3	
1.1. Цель работы.....	3
1.2. Средства выполнения.....	3
1.3. Продолжительность работы.....	3
1.4. Пункты задания для выполнения.....	3
1.5. Содержание отчета.....	4
2. ТЕОРЕТИКО-ПРАКТИЧЕСКИЙ МАТЕРИАЛ	5
2.1. СУБД Apache Cassandra.....	5
2.1.1. Основные сведения.....	5
2.1.2. Основные термины модели данных СУБД Cassandra.....	8
2.1.3. Согласованность данных.....	9
2.1.4. Транзакции.....	13
2.1.5. Доступность.....	13
2.2. Установка для Ubuntu Linux.....	15
2.3. Работа с СУБД CassandraDB через командную строку.....	17
2.4. Создание семейства столбцов.....	19
2.5. Изменение и удаление строк.....	21
2.6. Индексы.....	22
2.7. Выполнение запросов без использования индексов.....	24
2.8. Ограничения WHERE для операторов SELECT.....	25
2.8.1. Ограничения распределительных ключей.....	25
2.8.2. Ограничения кластерных ключей.....	26
2.9. Запросы с использованием агрегатных функций.....	27
2.10. Ключевое слово TTL.....	28
2.11. Группировка и сортировка данных.....	29
2.12. Материализованное представление.....	30
2.13. Команда пакетной обработки.....	32
2.14. Условная конструкция для операторов UPDATE и DELETE.....	34
3. КОНТРОЛЬНЫЕ ВОПРОСЫ	36
4. СПИСОК ИСТОЧНИКОВ	37

1. ЗАДАНИЕ

Лабораторная работа №3 «Создание колоночной базы данных и работа с ней на примере СУБД Cassandra» по курсу «Постреляционные базы данных».

1.1. Цель работы

- Изучить модель представления данных и способы работы с колоночными БД.
- Освоить методы создания колоночной БД и языки запросов к ним.
- Получить навыки работы с колоночной СУБД Apache Cassandra.

1.2. Средства выполнения

- СУБД Apache Cassandra
- Утилита CassandraDb

1.3. Продолжительность работы

Время выполнения лабораторной работы 4 часа.

1.4. Пункты задания для выполнения

Базовое:

1. Создать в среде CassandraDb свое пространство ключей. Определить семейство столбцов по теме, выданной преподавателем.
2. Продемонстрировать добавление, изменение и удаление данных в БД, используя команды API и/или язык Cassandra Query Language.
3. Продемонстрировать (вывести на экран) содержимое БД.
4. Создать второе семейство столбцов по той же теме, определить для них распределительный и кластерный индексы.
5. Определить для семейства столбцов индекс(ы). Выполнить запросы с фильтрацией по ключам и индексам. Продемонстрировать работу allow filtering.

6. Выполнить запросы к базе данных с селекцией и проекцией. Выполнить запрос с использованием скалярных и агрегатных функций.
7. Добавить строку с указанием TTL, продемонстрировать действие TTL.

Расширенное:

8. Выполнить запросы с группировкой и сортировкой данных.
9. Продemonстрировать усечение таблицы и удаление таблицы/индекса.
10. Продemonстрировать создание пакета запросов.

Дополнительное:

11. Создать материализованное представление. Продemonстрировать работу с ним.
12. Продemonстрировать изменение и удаление данных в БД, используя условия IF для Update и Delete.

1.5. Содержание отчета

- Титульный лист;
- Цель работы;
- Задание;
- Тексты запросов и команд в соответствии с пунктами задания.
- Результаты выполнения запросов (скриншоты);
- Вывод;
- Список используемой литературы.

2. ТЕОРЕТИКО-ПРАКТИЧЕСКИЙ МАТЕРИАЛ

2.1. СУБД Apache Cassandra

2.1.1. Основные сведения

Apache Cassandra – это нереляционная отказоустойчивая распределенная СУБД, рассчитанная на создание высокомасштабируемых и надёжных хранилищ огромных массивов данных, представленных в виде хэша. Проект был разработан на языке Java в корпорации Facebook в 2008 году, и передан фонду Apache Software Foundation в 2009. Эта СУБД относится к гибридным [NoSQL](#)-решениям, поскольку она сочетает модель хранения данных на базе семейства столбцов (ColumnFamily) с концепцией key-value (ключ-значение) [1].

Основная идея колоночных СУБД — это хранение данных не по строкам, как это делают традиционные СУБД, а по колонкам. Это означает, что с точки зрения SQL-клиента данные представлены как обычно в виде таблиц, но физически эти таблицы являются совокупностью колонок, каждая из которых по сути представляет собой таблицу из одного поля. При этом физически на диске значения одного поля хранятся последовательно друг за другом — приблизительно так: [A1, A2, A3], [B1, B2, B3], [C1, C2, C3] и т.д.

Такая организация данных приводит к тому, что при выполнении SELECT в котором фигурируют только 3 поля из 50 полей таблицы, с диска **физически** будут прочитаны только 3 колонки. Это означает что нагрузка на канал ввода-вывода будет приблизительно в $50/3=17$ раз меньше, чем при выполнении такого же запроса в традиционной СУБД.

Модель данных Cassandra состоит из следующих элементов (см. Рисунок 1):

- **столбец или колонка (column)** – ячейка с данными, включающая 3 части – имя (column name) в виде массива байтов, метку времени (timestamp) и само значение (value) также в виде байтового массива. С каждым значением связана **метка времени** — задаваемое пользователем 64-битное число, которое используется для разрешения конфликтов во время записи: чем оно больше, тем новее считается столбец. Это учитывается при удалении старых колонок.

- **строка или запись (row)** – именованная коллекция столбцов;
- **семейство столбцов (column family)** – именованная коллекция строк;
- **пространство ключей (keyspace)** – группа из нескольких семейств столбцов, собранных вместе. Оно логически группирует семейства столбцов и обеспечивает изолированные области имен.

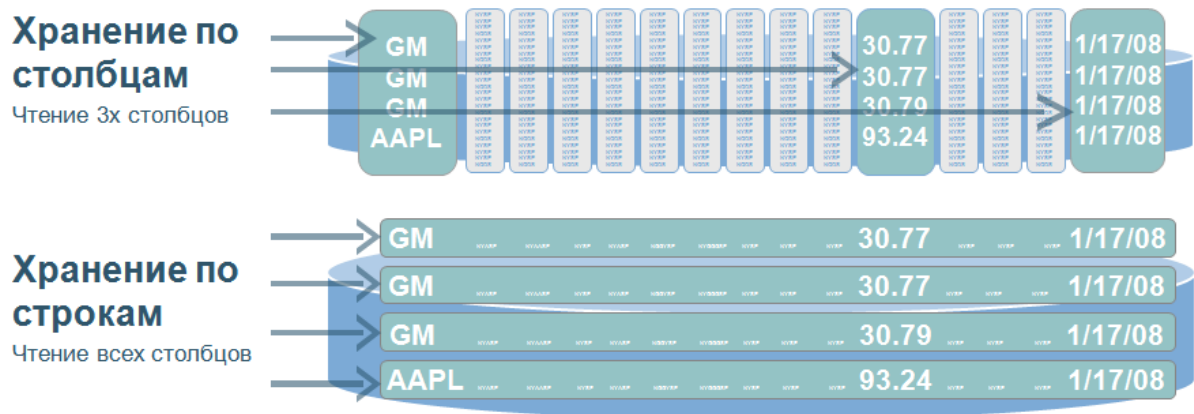


Рисунок 1 – Модель данных Apache Cassandra

В сравнении с хранением данных в строках, как в большинстве реляционных баз данных, преимущества хранения в колонках заключаются в быстром поиске/доступе и агрегации данных. Реляционные базы данных хранят каждую строку как непрерывную запись на диске. Разные строки хранятся в разных местах на диске, в то время как колоночные базы данных хранят все ячейки, относящиеся к колонке, как непрерывную запись, что делает операции поиска/доступа быстрее.

Пример: получение списка заголовков нескольких миллионов статей будет трудоёмкой задачей при использовании реляционных баз данных, так как для извлечения заголовков придётся проходить по каждой записи. А можно получить все заголовки с помощью только одной операции доступа к диску.

В рассматриваемой СУБД доступны следующие **типы данных**:

- **ByteType**: любые байтовые строки (без валидации);
- **AsciiType**: ASCII строка;
- **UTF8Type**: UTF-8 строка;
- **IntegerType**: число с произвольным размером;

- Int32Type: 4-байтовое число;
- LongType: 8-байтовое число;
- UUIDType: UUID 1-ого или 4-ого типа;
- TimeUUIDType: UUID 1-ого типа;
- DateType: 8-байтовое значение метки времени;
- BooleanType: два значения: true = 1 или false = 0;
- FloatType: 4-байтовое число с плавающей запятой;
- DoubleType: 8-байтовое число с плавающей запятой;
- DecimalType: число с произвольным размером и плавающей запятой;
- CounterColumnType: 8-байтовый счётчик.

Пространство ключей соответствует понятию схемы базы данных (database schema) в реляционной модели, а находящиеся в нем семейства столбцов – реляционной таблице (см. Рисунок 2). Столбцы объединяются в записи при помощи ключа (row key) в виде массива байтов, по значению которого столбцы упорядочены в пределах одной записи. В отличие от реляционных СУБД, в NoSQL модели возможна ситуация, когда разные строки содержат разное число колонок или столбцы с такими же именами, как и в других записях.

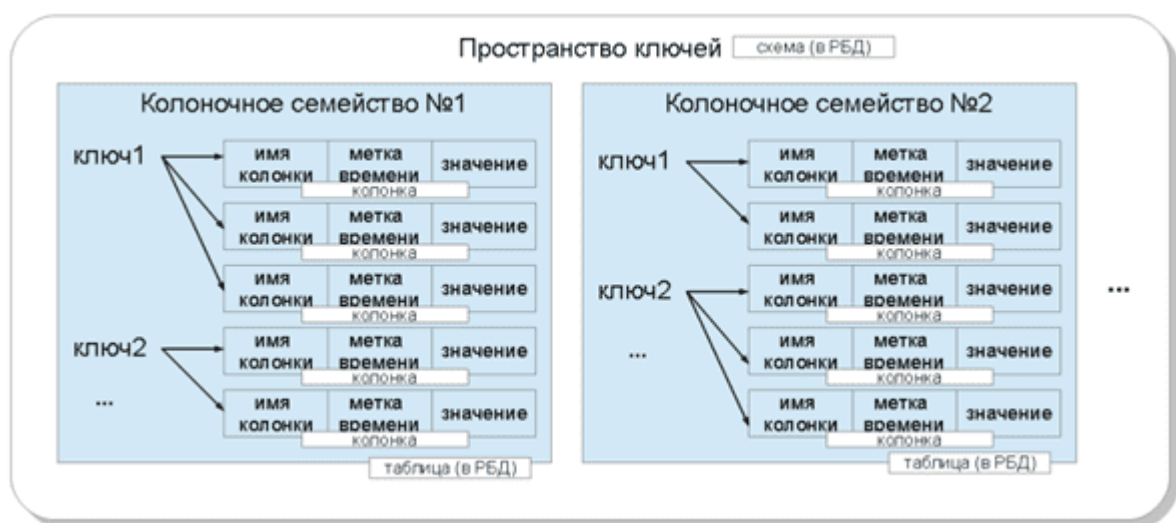


Рисунок 2 – Пространство ключей колоночной БД

2.1.2. Основные термины модели данных СУБД Cassandra

В модели данных СУБД Cassandra используются следующие понятия [2.]:

- **Пространство ключей**, которое помогает привязаться к приложению или предметной области. Это позволяет на одном кластере размещать данные разных приложений;
- **Семейство столбцов**, которое помогает привязаться к запросу;
- **Ключ**, который помогает привязаться к узлу кластера, который определяет, на какие узлы попадут сохранённые колонки;
- **Название колонки**, которое помогает привязаться к атрибуту в записи, что позволяет в одной записи хранить несколько значений.

Структура данных при этом выглядит так (см. Рисунок 3):

- *Keyspace (англ. Пространство ключей)*
 - *ColumnFamily (англ. Семейство колонок)*
 - *Row (англ. Строка)*
 - *Key (англ. Ключ)*
 - *Column (англ. Колонка)*
 - *Name (англ. Название)*
 - *Value (англ. Значение)*
 - *Column (англ. Колонка)*
 - ...
 - ...

Ячейка



Строка



Column Family



Рисунок 3 – Схема хранения данных Apache Cassandra

2.1.3. Согласованность данных

Когда Cassandra получает запрос на запись, данные сначала записываются в *журнал закрепления* (commit log), а затем в структуру, находящуюся в оперативной памяти и называемую таблицей в памяти (memtable). Операция записи считается успешной, если она записана и в журнал закрепления, и в таблицу в памяти. Записи группируются в памяти и периодически записываются в структуры, которые называются SSTable. После очистки структуры SSTable она не перезаписывается; измененные данные записываются в новую структуру SSTable. Неиспользуемые структуры SSTables удаляются в процессе уплотнения.

Посмотрим, как настройки согласованности данных влияют на выполнение операций чтения. Если уровень согласованности для всех операций чтения равен по умолчанию ONE, то после выполнения запроса на чтение Cassandra возвращает данные из первой реплики, даже если эти данные устарели. В последнем случае последующие операции чтения вернут более новые данные, – этот процесс называется *чтением с исправлением* (read repair). Низкий уровень согласованности данных удобно использовать, когда пользователя не волнует, получает ли он устаревшие данные или нет, если обеспечивается высокая производительность чтения.

Аналогично при выполнении запроса на запись Cassandra выполняет запись в один из журналов закрепления и возвращает ответ клиенту. Уровень согласованности ONE приемлем в ситуациях, когда обеспечивается высокая производительность записи и пользователь не беспокоится о том, что некоторые записи могут оказаться утерянными, если узел выйдет из строя до того, как запись будет реплицирована на другой узел.

```
quorum = new ConfigurableConsistencyLevel();  
quorum.setDefaultReadConsistencyLevel(HConsistencyLevel.QUORUM);  
quorum.setDefaultWriteConsistencyLevel(HConsistencyLevel.QUORUM);
```

Использование уровня согласованности QUORUM для операций чтения и записи гарантирует, что на запрос чтения ответит большинство узлов и клиенту будет возвращен столбец с новейшей меткой времени, в то время как реплики, не содержащие новейших данных, будут исправлены с помощью операций чтения с исправлением. Уровень согласованности QUORUM для операции записи означает, что операция записи во время выполнения будет размножена по большинству узлов, прежде чем будет признана успешной и клиент будет уведомлен об этом.

Уровень согласованности ALL означает, что на запросы чтения и записи должны отвечать все узлы. В этом случае кластер будет уязвим для отказов – если из строя выйдет хотя бы один узел, операция записи или чтения будет заблокирована и объявлена ошибочной. Таким образом, проектировщик базы должен настраивать уровни согласованности в соответствии с требованиями приложения. В рамках одного и того же приложения требования к согласованности данных могут быть разными; они также изменяются в зависимости от операций, например, демонстрация комментариев к товарам и чтение статуса последнего заказа клиента могут иметь разные уровни согласованности.

Во время создания *пространства ключей* можно указать, сколько реплик данных следует хранить в базе. Это число определяет коэффициент репликации данных. Если коэффициент репликации равен 3, то данные копируются на три узла.

Можно применить команду восстановления узла к пространству ключей и заставить базу Cassandra сравнить каждый ключ с остальными репликами. Эта

операция связана с большими затратами, поэтому можно просто восстановить конкретное семейство столбцов или список семейств столбцов.

```
repair ecommerce
repair ecommerce customerinfo
```

Если узел вышел из строя, предполагается, что его данные были переданы другим узлам. Когда узел вернется в строй, изменения, внесенные в его данные, будут отправлены обратно. Этот метод называется *направленной отправкой* (hinted handoff). Направленная отправка позволяет быстрее восстанавливать узлы, вышедшие из строя [1.].

В СУБД Cassandra представлено несколько различных уровней согласованности чтения (см. Таблица 1).

Таблица 1 – Уровни согласованности чтения

Уровень	Описание
ONE	Координатор шлёт запросы к ближайшему узлу-реплике, читая остальные с целью исправления (read repair) с заранее заданной в конфигурации вероятностью;
TWO	Координатор шлёт запросы к двум ближайшим узлам, выбирая значение с большей меткой времени
THREE	Координатор шлёт запросы к трем ближайшим узлам, выбирая значение с большей меткой времени
QUORUM	Собирается кворум, то есть координатор шлёт запросы к более чем половине узлов-реplik, а именно $\text{round}(N/2)+1$, где N — уровень репликации
ALL	Координатор возвращает данные после прочтения со всех узлов-реplik
LOCAL_QUORUM	Собирается кворум узлов в том датацентре, где происходит координация, и возвращаются данные с последней меткой времени

Уровень	Описание
EACH_QUORUM	Координатор возвращает данные после собрания кворума в каждом из датацентров

В СУБД Cassandra представлено несколько различных уровней согласованности записи (см. Таблица 2).

Таблица 2 – Уровни согласованности записи

Уровень	Описание
ANY	Даёт возможность записать данные, даже если все узлы-реплики не отвечают. Координатор дожидается первого ответа от любого одного узла-реплик или данные сохраняются с помощью механизма направленной отправки (hinted handoff) на координаторе
ONE	Координатор шлёт запросы всем узлам-реплик, но возвращает управление пользователю, дождавшись подтверждения от любого первого узла
TWO	Координатор дожидается подтверждения от двух первых узлов, прежде чем вернуть управление
THREE	Координатор ждет подтверждения от трех первых узлов, прежде чем вернуть управление
QUORUM	Координатор дожидается подтверждения записи от более чем половины узлов-реплик, а именно $\text{round}(N/2)+1$, где N — уровень репликации
ALL	Координатор дожидается подтверждения от всех узлов-реплик
LOCAL_QUORUM	Координатор дожидается подтверждения от более чем половины узлов-реплик в том же центре обработки данных, где расположен координатор. Это позволяет избавиться от задержек, связанных с пересылкой данных в другие датацентры
EACH_QUORUM	Координатор дожидается подтверждения от

Уровень	Описание
	более чем половины узлов-реплик в каждом центре обработки данных

2.1.4. Транзакции

В базе данных нет транзакций в традиционном смысле этого слова – возможности начинать множество операций записи, а затем решать – подтверждать изменения или нет. В базе данных Cassandra операция записи является атомарной на уровне строки. Это значит, что вставка или обновление столбцов по заданному ключу строки трактуется как отдельная запись и, может быть, либо успешной, либо ошибочной. Записи сначала записываются в журнал закрепления и таблицы в памяти и считаются успешными только в тех случаях, когда записи в журнал закрепления и таблицу в памяти оказались успешными. Если узел выходит из строя, для внесения изменений на узел используется журнал закрепления точно так же, как это делает команда `redo log` в базе данных Oracle.

Для синхронизации операций записи и чтения существует возможность использования библиотек внешних транзакций, таких как ZooKeeper [ZooKeeper]. Кроме того, существуют библиотеки, такие как Cages [Cages], позволяющие перекрывать транзакции библиотеки ZooKeeper [1.].

2.1.5. Доступность

База данных Cassandra задумана как очень доступная, потому что в ней нет ведущего узла и все узлы в кластере имеют одинаковые права. Доступность кластера можно увеличить, уменьшив уровень согласованности запросов. Доступность управляется формулой:

$$(R + W) > N \text{ (см. раздел 5.5, "Кворумы" [1.])},$$

где W – минимальное количество узлов, в которых запись должна быть успешно записана;

R – минимальное количество узлов, подтвердивших успешное чтение;

N – количество узлов, участвующих в репликации данных.

Уровень доступности можно настраивать, изменяя значения **R** и **W** при фиксированном значении **N**.

Если в кластере базы данных Cassandra, состоящем из 10 узлов с коэффициентом репликации для пространства ключей равным 3 (**N** = 3), положить **R** = 2 и **W** = 2, то мы получим неравенство $(2 + 2) > 3$. Если в этом сценарии один из узлов выйдет из строя, то уровень доступности снизится незначительно, поскольку данные можно будет получить от других двух узлов. Если **W** = 2 и **R** = 1 и из строя выйдут два узла, то кластер станет недоступным для записи, но по-прежнему будет доступным для чтения. Аналогично, если **R** = 2 и **W** = 1, мы сможем выполнять операции записи, но не сможем читать данные с кластера. При выполнении неравенства $R + W > N$ можно выбирать компромиссы между согласованностью и доступностью.

Настройки пространств ключей и операций чтения/записи следует выбирать в зависимости от ваших потребностей – повысить доступность записи или чтения.

2.2. Установка для Ubuntu Linux

Cassandra — ресурсозатратная СУБД, поэтому рекомендуется иметь не менее 2 Гб оперативной памяти в системе.

Для работы с СУБД Cassandra используется утилита CassandraDB. Скачать и найти основную информацию можно в источнике [3.]. Далее разберем подробнее как установить данную утилиту на Ubuntu Linux.

Обновляем программные пакеты:

```
$ apt-get update
```

Для работы Cassandra нам предварительно необходимо установить JDK.

```
$ apt-get install default-jdk
```

Проверяем версию java. Результат проверки представлен на Рисунок 4.

```
$ java -version
```

INCLUDEPICTURE
"https://community.vscale.io/hc/user_images/JyE0LtsDDS9darZRuhuj1Q.png" \
* MERGEFORMATINET INCLUDEPICTURE
"https://community.vscale.io/hc/user_images/JyE0LtsDDS9darZRuhuj1Q.png" \
* MERGEFORMATINET INCLUDEPICTURE
"https://community.vscale.io/hc/user_images/JyE0LtsDDS9darZRuhuj1Q.png" \
* MERGEFORMATINET INCLUDEPICTURE
"https://community.vscale.io/hc/user_images/JyE0LtsDDS9darZRuhuj1Q.png" \
* MERGEFORMATINET

```
root@cs57300:~# java -version  
openjdk version "1.8.0_111"  
OpenJDK Runtime Environment (build 1.8.0_111-8u111-b14-2ubuntu0.16.04.2-b14)  
OpenJDK 64-Bit Server VM (build 25.111-b14, mixed mode)
```

Рисунок 4 – Результат проверки версии Java

Переходим непосредственно к самой установке Cassandra. Мы будем устанавливать версию 3.9.

Выполняем в командной строке:

```
$ echo "deb http://debian.datastax.com/datastax-ddc 3.9 main" | sudo  
tee -a /etc/apt/sources.list.d/cassandra.sources.list
```

Устанавливаем утилиту curl

```
$ apt-get install curl
```

Снова обновляем программные пакеты:

```
$ apt-get update
```

И устанавливаем Cassandra

```
$ apt-get install datastax-ddc
```

Проверяем статус

```
$ service cassandra status
```


2.3. Работа с СУБД CassandraDB через командную строку

Чтобы попасть в командную оболочку, необходимо в командной строке ввести **cqlsh** (см. Рисунок 5).

```
osboxes@osboxes:~$ cqlsh
Connected to Test Cluster at 127.0.0.1:9042.
[cqlsh 5.0.1 | Cassandra 3.11.6 | CQL spec 3.4.4 | Native protocol v4]
Use HELP for help.
cqlsh>
```

Рисунок 5 – Применение команды **cqlsh**

Для работы с кассандрой используется язык CQL (Cassandra Query Language (английский) – Язык запросов Cassandra) [4.], который предоставляет возможности управления как структурой базы данных, так и управления записями этой базы данных.

Создаем пространство ключей [5.]:

```
CREATE KEYSPACE [IF NOT EXISTS] keyspace_name
WITH REPLICATION = {
    'class' : 'SimpleStrategy', 'replication_factor' : N }
| 'class' : 'NetworkTopologyStrategy',
    'dc1_name' : N [, ...]
};
```

keyspace_name – название пространства ключей,

dc1_name – имя узла.

```
REPLICATION = { replication_map }
```

Карта репликации определяет, сколько копий данных хранится в данном узле (см. Таблица 3). Этот параметр влияет на согласованность, доступность и скорость запроса.

Таблица 3 – Класс стратегии репликации и настройки факторов

Класс	Фактор	Описание
'SimpleStrategy'	'replication_factor' : N	Для всего кластера будет использоваться один коэффициент репликации. Параметр N означает количество копий данных, должен быть целым

Класс	Фактор	Описание
		числом.
'NetworkTopologyStrategy'	'datacenter_name' : N	Для каждого узла задается свой коэффициент репликации. Параметр N означает количество копий данных, должен быть целым числом.

Примеры задания топологий:

- Простая топология:

```
'class' : 'SimpleStrategy', 'replication_factor' : 1
```

- Сетевая топология:

```
'class' : 'NetworkTopologyStrategy', 'London_dc' : 1, 'Moscow_dc':2
```

Более подробно про параметры Вы можете прочитать в официальной документации [6.].

Создадим пространство ключей test с стратегией репликации 'SimpleStrategy' и коэффициентом репликации 1:

```
cqlsh> CREATE KEYSPACE test WITH REPLICATION = { 'class' :  
'SimpleStrategy', 'replication_factor' : 1};
```

Перейдем в созданное пространство

```
cqlsh> use test
```

2.4. Создание семейства столбцов

Создадим семейство колонок [7., 8.]. Синтаксис создания:

```
CREATE TABLE ИМЯ_ТАБЛИЦЫ (  
ИМЯ_КОЛОНКИ_1 ТИП_ДАНЫХ,  
ИМЯ_КОЛОНКИ_2 ТИП_ДАНЫХ,  
ИМЯ_КОЛОНКИ_3 ТИП_ДАНЫХ,  
PRIMARY KEY (ИМЯ_КОЛОНКИ_1)  
)
```

В **CREATE TABLE** должен быть обязательно объявлен **PRIMARY KEY** для индексации пространства колонок. Пример:

```
CREATE TABLE student (  
id uuid,  
citizenship text,  
first_name text,  
last_name text,  
age int,  
PRIMARY KEY (id)  
);
```

В этом примере будет создано семейство колонок student с полями id типа uuid, citizenship, first_name, last_name типа text, age типа int.

В данном случае первичным ключом будет являться поле **id**.

Создадим таблицу test:

```
> CREATE TABLE IF NOT EXISTS test (  
id timeuuid,  
title text,  
PRIMARY KEY (title, id)  
) WITH default_time_to_live = 120 and CLUSTERING ORDER BY (id DESC);
```

Будет создано семейство колонок с полями **id** и **title**.

Первичный ключ состоит из распределительного и кластерного ключа. Распределительный ключ отвечает за распределение данных по узлам, а кластерный за группировку данных внутри узла. Ключи сами по себе могут быть составными.

В данном случае **title** – распределительный ключ, а **id** - кластерный

Этап проектирования первичного ключа является самым важным, так как от этого напрямую зависит вся эффективность системы.

Конструкция CLUSTERING ORDER BY определяет сортировку данных.

Для изменения структуры таблицы используется команда следующего вида:

```
ALTER TABLE <ИМЯ_ТАБЛИЦЫ> <ДАННЫЕ_ДЛЯ_ИЗМЕНЕНИЯ>
```

Изменим тип id в таблице test:

```
ALTER TABLE test ALTER ID TYPE uuid;
```

Как видно, язык запросов очень похож на традиционный SQL.

Вставим в колонки значения:

```
INSERT INTO student (id, citizenship, first_name, last_name, age)
VALUES (now(), 'Russia', 'Ivan', 'Ivanov', 25);
```

Выполнив select –запрос отобразим все значения.

```
SELECT * FROM tablename;
```

Выполнение запроса для нашей таблицы (см. Рисунок 6).

```
SELECT * FROM student;
```

```
cqlsh:lab7> select * from student;
```

id	age	citizenship	first_name	last_name
3a26e100-9aeb-11ea-b1d1-3148925e06e7	25	Russia	Ivan	Ivanov
6d0317a0-9aeb-11ea-b1d1-3148925e06e7	22	France	test	Petrov
47957270-9aeb-11ea-b1d1-3148925e06e7	35	Russia	Petr	Petrov

Рисунок 6 – Результат выполнения запроса SELECT

2.5. Изменение и удаление строк

Для добавления **INSERT**, для удаления – **DELETE**, для изменения – **UPDATE**.

Операторы **UPDATE** и **DELETE** так же имеют традиционный вид [7., 9.].

Пример **UPDATE**:

```
UPDATE имя_таблицы SET название_колонки=значение WHERE условие
```

Обновим поля `first_name` и `last_name` и строки с заданным `id`:

```
UPDATE student SET first_name = 'Example', last_name='Example' WHERE  
id=54daf810-9aeb-11ea-b1d1-3148925e06e7
```

Пример **DELETE**:

```
DELETE FROM название_таблицы WHERE условие
```

Удалим строки с заданным `id` из таблицы `student`:

```
DELETE FROM student WHERE id=54daf810-9aeb-11ea-b1d1-3148925e06e7;
```

В данном случае происходит поиск по ключу.

Вы можете удалить таблицу с помощью команды **TRUNCATE**. Когда вы усекаете таблицу, все строки таблицы удаляются навсегда. Ниже приведен синтаксис этой команды.

```
TRUNCATE <имя таблицы>
```

Например:

```
TRUNCATE course;
```

2.6. Индексы

В **SELECT** запросах Cassandra Query Language (**CQL**) отсутствуют привычные нам SQL операции **JOIN**, **GROUP BY** [7., 10.]. А операция **WHERE** сильно урезана. В SQL вы можете фильтровать по любой колонке, тогда как в CQL только по **распределительным ключам** (*partition key*), **кластерным ключам** (*clustering columns*) и **вторичным индексам**.

В Cassandra можно создавать вторичные индексы у любой колонки наподобие SQL индексов. Фактически же, вторичные индексы Cassandra — это скрытая дополнительная таблица, поэтому производительность **WHERE** запросов по ним хуже запросов по ключевым колонкам.

СУБД не дает возможности по умолчанию искать по неключевым колонкам, для которых не создан индекс.

Запрос

```
SELECT * FROM student WHERE citizenship='Russia'
```

выдаст ошибку (см. Рисунок 7).

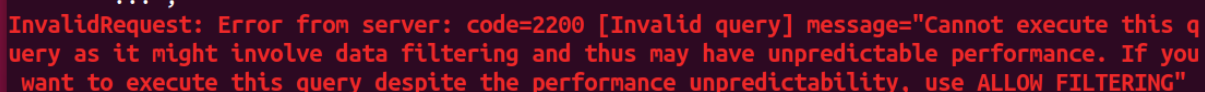


Рисунок 7 – Ошибка при поиске не по ключевым атрибутам

Решением этой проблемы является создание вторичного индекса или использование конструкции **ALLOW FILTERING**.

Рассмотрим создание индекса:

Вы можете создать индекс в Cassandra с помощью команды **CREATE INDEX**. Его синтаксис выглядит следующим образом:

```
CREATE INDEX <identifier> ON <tablename>
```

Создадим индекс по колонке `citizenship`.

```
CREATE INDEX citizenshipIndex ON student (citizenship);
```

После этого можно выполнять запрос с условием на эту колонку. Результат представлен на Рисунок 8.

```
SELECT * FROM student WHERE citizenship='Russia';
```

id	age	citizenship	first_name	last_name
3a26e100-9aeb-11ea-b1d1-3148925e06e7	25	Russia	Ivan	Ivanov
47957270-9aeb-11ea-b1d1-3148925e06e7	35	Russia	Petr	Petrov

Рисунок 8 – Результат поиска по индексу

Удалить индекс можно следующей командой:

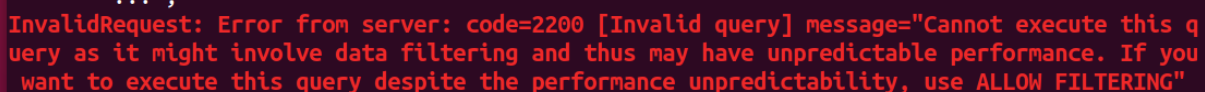
```
DROP INDEX indexname;
```

2.7. Выполнение запросов без использования индексов

Если попытаться выполнить запрос

```
SELECT * FROM student WHERE citizenship='Russia'
```

без использования индекса, возникнет ошибка (см. Рисунок 9).



```
InvalidRequest: Error from server: code=2200 [Invalid query] message="Cannot execute this query as it might involve data filtering and thus may have unpredictable performance. If you want to execute this query despite the performance unpredictability, use ALLOW FILTERING"
```

Рисунок 9 – Ошибка при поиске без использования индекса

Cassandra знает, что она не сможет выполнить запрос эффективным способом. Поэтому он предупреждает вас: «Будьте осторожны. Выполнение этого запроса как такового может быть не очень хорошей идеей, поскольку оно может использовать много ваших вычислительных ресурсов».

Единственный способ, которым Cassandra может выполнить этот запрос, – это извлечь все строки из таблицы, а затем отфильтровать те, у которых есть запрошенное значение для столбца citizenship.

Если ваша таблица содержит, например, 1 миллион строк, и 95% из них имеют запрошенное значение для столбца citizenship, запрос все равно будет относительно эффективным, и вам следует использовать **ALLOW FILTERING**.

С другой стороны, если ваша таблица содержит 1 миллион строк и только 2 строки содержат запрошенное значение для столбца citizenship, ваш запрос крайне неэффективен. Cassandra будет бессмысленно загружать 999.998 строк. Если запрос часто используется, возможно, лучше добавить индекс в столбец citizenship.

К сожалению, у Cassandra нет возможности провести различие между двумя вышеупомянутыми случаями, поскольку они зависят от распределения данных таблицы. Поэтому Cassandra предупреждает вас и полагается на вас, чтобы сделать правильный выбор.

Пример:

```
SELECT * FROM student WHERE last_name='Ivanov' ALLOW FILTERING;
```


В данном запросе будут отображены все студенты с фамилией Иванов (см. Рисунок 10).

```
cqlsh:lab7> SELECT * FROM student WHERE last_name='Ivanov' ALLOW FILTERING;
```

id	age	citizenship	first_name	last_name
3a26e100-9aeb-11ea-b1d1-3148925e06e7	25	Russia	Ivan	Ivanov

Рисунок 10 – Результат выполнения поиска с применением *ALLOW FILTERING*

2.8. Ограничения WHERE для операторов SELECT

2.8.1. Ограничения распределительных ключей

Столбцы распределительного ключа поддерживают только два оператора: = и IN.

К примеру, создадим таблицу с составным распределительным ключом:

```
CREATE TABLE numberOfRequests (  
cluster text,  
date text,  
time text,  
numberOfRequests int,  
PRIMARY KEY ((cluster, date), time)  
)
```

Будет создана таблица с полями cluster, date, time типа text, numberOfRequests типа int и распределительным и кластерным ключами (cluster, date) и time.

В условиях на распределительный ключ вы можете использовать операторы = и IN. Например:

```
SELECT * FROM numberOfRequests WHERE cluster IN ('cluster1',  
'cluster2') AND date = '2015-05-06';
```

Запрос:

```
SELECT * FROM numberOfRequests WHERE cluster = 'cluster1' and time =  
'12: 00 ';
```

будет отклонен, поскольку столбец даты не используется. Причина в том, что Cassandra нужны все столбцы ключа раздела, чтобы иметь возможность вычислять хеш, который позволит находить узлы, содержащие раздел. Если для ключей раздела не указаны ограничения, но некоторые из них указаны для ключей кластеризации, Cassandra потребует, чтобы в запрос был добавлен параметр **ALLOW FILTERING**.

2.8.2. Ограничения кластерных ключей

Кластерные ключи поддерживают операторы =, IN, >, > =, <=, <, **CONTAINS** и **CONTAINS KEY** в ограничениях на один столбец и операторы =, IN, >, > =, <= и < в ограничениях на несколько столбцов.

Использование кластерных ключей в запросе:

Предположим, существует таблица:

```
CREATE TABLE numberOfRequests (  
  cluster text,  
  date text,  
  datacenter text,  
  hour int,  
  minute int,  
  numberOfRequests int,  
  PRIMARY KEY ((cluster, date), datacenter, hour, minute)  
)
```

Будет создана таблица с полями cluster, date, datacenter типа text, hour, minute, numberOfRequests типа int и распределительным и кластерным ключами (cluster, date) и datacenter, hour, minute.

Вы можете видеть, что для эффективного извлечения данных без вторичного индекса вам нужно знать все ключевые кластерные ключи для выбора. Итак, если вы выполните:

```
SELECT * FROM numberOfRequests  
  WHERE cluster = 'cluster1' AND  
         date = '2015-06-05' AND  
         datacenter = 'US_WEST_COAST' AND
```

```
hour = 14 AND  
minute = 00;
```

Cassandra найдет данные эффективно, но если вы выполните:

```
SELECT * FROM numberOfRequests  
WHERE cluster = 'cluster1' AND  
date = '2015-06-05' AND  
hour = 14 AND  
minute = 00;
```

так как в запросе отсутствует условие на кластерный ключ **datacenter**, Cassandra отклонит запрос, поскольку ей придется сканировать весь раздел, чтобы найти запрошенные данные, что неэффективно.

Более подробную информацию Вы можете прочитать в официальной документации [7].

2.9. Запросы с использованием агрегатных функций

В Cassandra стандартные агрегатные функции **min**, **max**, **avg**, **sum** и **count** являются встроенными функциями:

- Функция **min** находит минимальное значение,
- Функция **max** находит максимальное значение,
- Функция **avg** находит среднее значение,
- Функция **sum** суммирует значения в указанной колонке,
- Функция **count** определяет количество значений в колонке.

Воспользуемся функцией **min()** для вычисления минимального возраста.

```
SELECT min(age) AS min_age FROM student WHERE citizenship='Russia';
```

2.10. Ключевое слово TTL

Ключевое слово TTL (Time-to-Live (англ.) – Время жизни) позволяет указать, сколько будет существовать запись (созданная, или измененная).

INSERT и **UPDATE** поддерживают TTL. Время задается в секундах. Например, запрос:

```
INSERT INTO student (id, citizenship, first_name, last_name, age)
VALUES (now(), 'Test', 'Test', 'Test', 34) USING TTL 15;
```

создаст запись, которая будет существовать 15 секунд. Через указанное время она исчезнет.

Поведение аналогично для **UPDATE**. После истечения указанного времени измененное значение заменится на null.

Более подробно про TTL Вы можете прочитать в официальной документации [11.].

Пример:

```
UPDATE student USING TTL 15 SET age=1
WHERE id=54daf810-9aeb-11ea-b1d1-3148925e06e7
```

Видно, что значение изменилось (см. Рисунок 11).

id	age	citizenship	first_name	last_name
3a26e100-9aeb-11ea-b1d1-3148925e06e7	1	Russia	Ivan	Ivanov
6d0317a0-9aeb-11ea-b1d1-3148925e06e7	22	France	test	Petrov
47957270-9aeb-11ea-b1d1-3148925e06e7	35	Russia	Petr	Petrov

Рисунок 11 – Изменение строки с использованием технологии TTL

Через 15 секунд значения возраста изменится на null (см. Рисунок 12).

id	age	citizenship	first_name	last_name
3a26e100-9aeb-11ea-b1d1-3148925e06e7	null	Russia	Ivan	Ivanov
6d0317a0-9aeb-11ea-b1d1-3148925e06e7	22	France	test	Petrov
47957270-9aeb-11ea-b1d1-3148925e06e7	35	Russia	Petr	Petrov

Рисунок 12 – Состояние таблицы по истечении времени TTL указанного в запросе

2.11. Группировка и сортировка данных

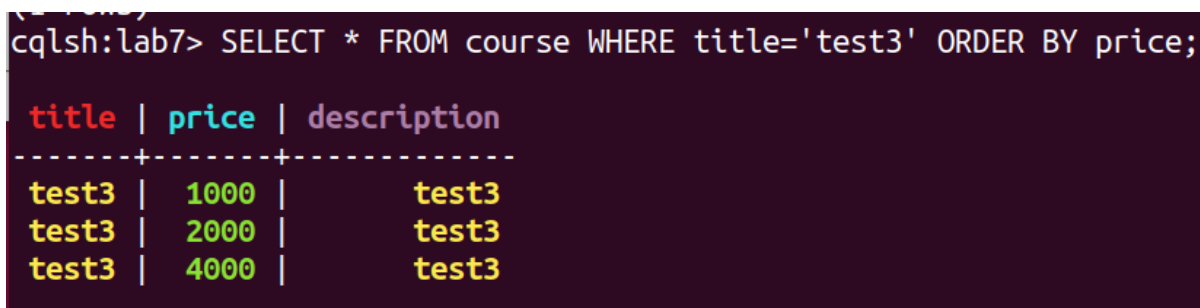
Как и в SQL (Simple Query Language (англ.) – Простая язык запросов), группировка и сортировка осуществляется с помощью ключевых слов **GROUP BY** и **ORDER BY**.

GROUP BY используется для группировки данных, **ORDER BY** для их сортировки.

Отсортируем курсы по цене:

```
SELECT * FROM course WHERE title='test3' ORDER BY price;
```

Результат представлен на Рисунок 13.



```
cqlsh:lab7> SELECT * FROM course WHERE title='test3' ORDER BY price;
```

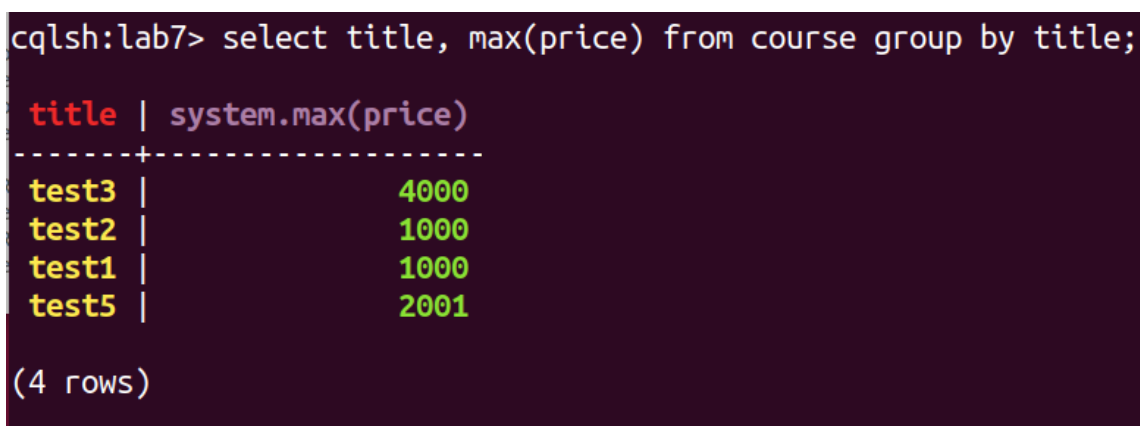
title	price	description
test3	1000	test3
test3	2000	test3
test3	4000	test3

Рисунок 13 – Вывод отсортированного списка командой **ORDER BY**

Выберем список курсов и для каждого из них укажем максимальную стоимость:

```
SELECT title, MAX(price) FROM course GROUP BY title;
```

Результат представлен на Рисунок 14.



```
cqlsh:lab7> select title, max(price) from course group by title;
```

title	system.max(price)
test3	4000
test2	1000
test1	1000
test5	2001

(4 rows)

Рисунок 14 – Вывод сгруппированного списка командой **GROUP BY**

2.12. Материализованное представление

В Cassandra все операции записи данных – это всегда операции перезаписи, то есть, если в колоночную семью приходит колонка с таким же ключом и именем, которые уже существуют, и метка времени больше, чем та которая сохранена, то значение перезаписывается. Записанные значения никогда не меняются, просто приходят более новые колонки с новыми значениями.

Запись в Cassandra работает с большей скоростью, чем чтение. Это меняет подход, который применяется при проектировании. Если рассматривать Cassandra с точки зрения проектирования модели данных, то проще представить колоночное семейство не как таблицу, а как *материализованное представление* ([materialized view](#)) — структуру, которая представляет данные некоторого сложного запроса, но хранит их на диске. Вместо того, чтобы пытаться как-либо скомпоновать данные при помощи запросов, лучше постараться сохранить в колоночное семейство все, что может понадобиться для этого запроса. То есть, подходить необходимо не со стороны отношений между сущностями или связями между объектами, а со стороны запросов: какие поля требуются выбрать; в каком порядке должны идти записи; какие данные, связанные с основными, должны запрашиваться совместно — всё это должно уже быть сохранено в колоночное семейство.

Более подробно про параметры материализованного представления Вы можете прочитать в официальной документации [12.].

Синтаксис создания материализованного представления:

```
CREATE MATERIALIZED VIEW [ IF NOT EXISTS ] view\_name AS  
    select\_statement PRIMARY KEY '(' primary\_key ');
```

- [view_name](#) – название материализованного представления
- [select_statement](#) – запрос, на основе которого создается материализованное представление
- [primary_key](#) – первичный ключ для материализованного представления

Ограничения для материализованных представлений:

- Все первичные ключи исходной таблицы включаются в первичный ключ материализованного представления.
- Только один новый столбец может быть добавлен к первичному ключу материализованного представления.

Создадим материализованное представление на основе запроса из семейства колонок course:

```
CREATE MATERIALIZED VIEW course_price AS SELECT * FROM course
WHERE title IS NOT NULL AND
price IS NOT NULL
PRIMARY KEY (title, price)
ORDER BY (price desc);
```

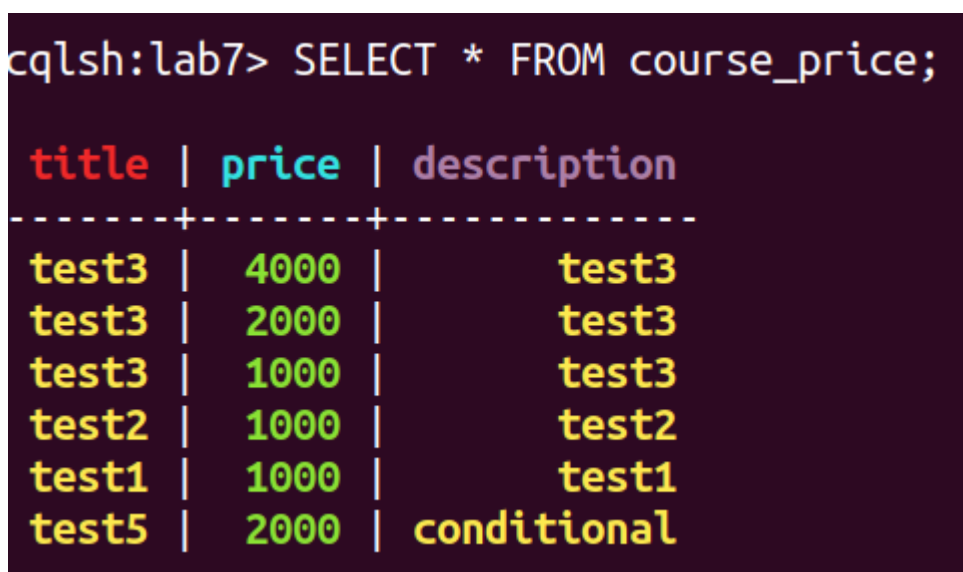
PRIMARY KEY указывает, какие атрибуты из course использовать в качестве ключевых.

Будут выбраны записи, в которых title и price не NULL, они будут отсортированы по полю price.

Теперь оно сохранено на диск, и мы можем к нему обратиться:

```
SELECT * FROM course_price;
```

Результат представлен на Рисунок 15.



```
cqlsh:lab7> SELECT * FROM course_price;
```

title	price	description
test3	4000	test3
test3	2000	test3
test3	1000	test3
test2	1000	test2
test1	1000	test1
test5	2000	conditional

Рисунок 15 – Результат обращения к созданному представлению

2.13. Команда пакетной обработки

Команда **BATCH** объединяет несколько операторов языка изменения данных DML (Data Manipulation Language (англ.) – Язык управления данными) для достижения атомарности и изоляции. Позволяет выполнять запросы «пачками».

Синтаксис использования ключевого слова **BATCH** следующий:

```
BEGIN [ ( UNLOGGED | COUNTER ) ] BATCH
  [ USING TIMESTAMP [ epoch_microseconds ] ]
  dml_statement [ USING TIMESTAMP [ epoch_microseconds ] ] ;
  [ dml_statement [ USING TIMESTAMP [ epoch_microseconds ] ]
[ ; ... ] ]
APPLY BATCH ;
```

dml_statement – DML оператор

BATCH может содержать данные DML операторы:

- INSERT;
- UPDATE;
- DELETE.

Параметры:

UNLOGGED | COUNTER:

Если **UNLOGGED** не указан, будет создан журнал операций. Если задействовано несколько разделов, пакеты регистрируются по умолчанию. Зарегистрированная партия гарантирует, что все или ни одна из пакетных операций будут выполнены успешно (атомарность). Ведение журнала снижает производительность, журнал записывается на два других узла.

Опция **COUNTER** используется для пакетных обновлений типа счетчик.

USING TIMESTAMP:

Устанавливает время записи для транзакций, выполненных в **BATCH**.

Ограничение: **USING TIMESTAMP** не поддерживает LWT (Lightweight transaction (англ.) – облегченные транзакции), такие как операторы DML, которые имеют предложение **IF NOT EXISTS**.

Пример:

```
BEGIN BATCH
  INSERT INTO course (title, price, description)
    VALUES ('test5', 2000, 'test5');
  UPDATE course SET description='update1' WHERE title='test5' AND
    price=2000;
APPLY BATCH;
```

Данный запрос вставит значения в поля title, price, description, а после этого обновит поле description.

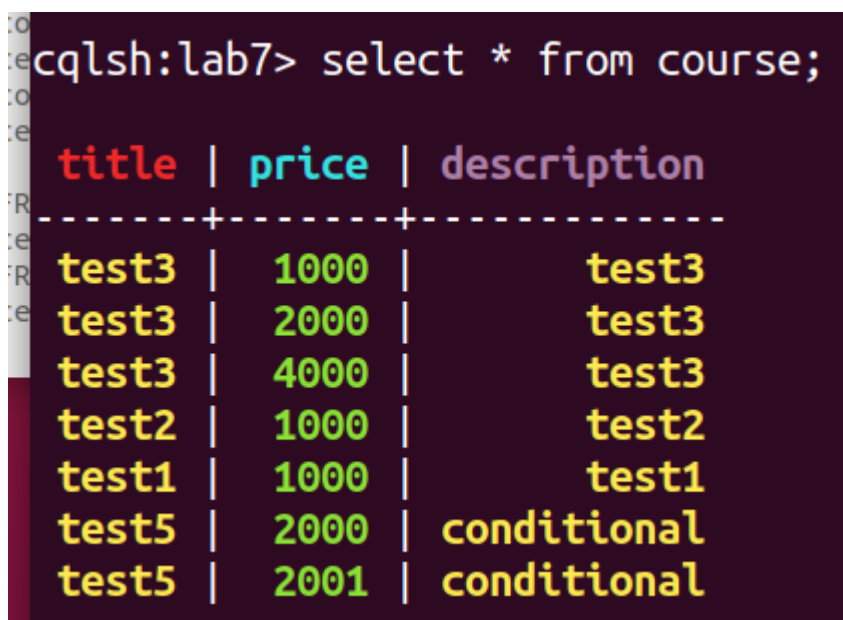
2.14. Условная конструкция для операторов UPDATE и DELETE

Ключевое слово **UPDATE** работает следующим образом: если значения, удовлетворяющего условию, не существует, оно будет создано.

Так, следующий запрос создаст новую запись:

```
UPDATE course SET description='conditional' WHERE title='test5' AND  
price=2001;
```

Результат представлен на Рисунок 16.

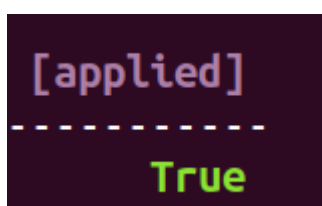


title	price	description
test3	1000	test3
test3	2000	test3
test3	4000	test3
test2	1000	test2
test1	1000	test1
test5	2000	conditional
test5	2001	conditional

Рисунок 16 – Результат выполнения UPDATE без IF EXISTS

Чтобы избежать такого поведения, можно использовать **IF EXISTS**.

```
UPDATE course SET description='conditional' WHERE title='test6'  
AND price=2000 IF EXISTS;
```



```
[applied]  
-----  
True
```

Рисунок 17 – Результат выполнения UPDATE с конструкцией IF EXISTS

Значения будут обновлены. В случае отсутствия данной записи запрос вернет false и новая запись не будет создана.

Так же данная конструкция позволяет выиграть в производительности, если записей не существует.

DELETE так же можно использовать с **IF**:

```
DELETE FROM course WHERE title='test5' AND price=2000 IF EXISTS;
```

В **IF** может содержаться любое условие, например:

```
DELETE FROM employees WHERE department_id = 2 AND employee_id = 1 IF  
name = 'Joe';
```

3. КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Особенности, модель данных и функциональные возможности колоночных СУБД?
2. Что такое пространство ключей, семейство столбцов, строка, столбец и суперстолбец?
3. Что такое ключи и индексы? Как их объявлять и использовать? Отличия распределительного и кластерного индексов?
4. Функциональные возможности и языки запросов для СУБД Cassandra?
5. Типы данных и TTL в СУБД Cassandra?
6. Возможности и конструкции для определения данных, работы с данными и запросами в СУБД Cassandra?
7. Технология репликации и фрагментации в СУБД Cassandra? Как организован и работает кластер?
8. Уровни согласованности данных в СУБД Cassandra?
9. Восстановление данных в СУБД Cassandra?
10. Запись на диск и уплотнение данных в СУБД Cassandra?
11. Поддержка транзакций в СУБД Cassandra?
12. Что такое материализованное представление и как с ним работать?

4. СПИСОК ИСТОЧНИКОВ

1. Фаулер, Мартин, Садаладж, Прамодкумар Дж. NoSQL: новая методология разработки нереляционных баз данных. : Пер. с англ. - М.: ООО "И.Д. Вильямс", 2013г.
2. Habr. Как устроена apache Cassandra. – Текст. Изображение. : электронные // Habr : [сайт]. – URL: <https://habrahabr.ru/post/155115/> (дата обращения 01.05.2022)
3. Apache Cassandra – Текст. Изображение. : электронные // Apache Cassandra. : [сайт]. – URL: https://cassandra.apache.org/_/index.html (дата обращения 01.05.2022)
4. Apache Cassandra: Cassandra documentation: The Cassandra Query Language (CQL) – Текст. Изображение. : электронные // Apache Cassandra. : [сайт]. – URL: <http://cassandra.apache.org/doc/latest/cql/index.html> (дата обращения 01.05.2022)
5. Datastax. Documentation. CQL for Apache Cassandra 3.0: Create Keyspace. – Текст. : электронные // Datastax: Documentation : [сайт]. – URL: https://docs.datastax.com/en/cql-oss/3.3/cql/cql_reference/cqlCreateKeyspace.html (дата обращения 01.06.2020)
6. Datastax. Documentation. CQL for Apache Cassandra 3.0: A deep look at the CQL where clause. – Текст. : электронные // Datastax: Documentation : [сайт]. – URL: <https://www.datastax.com/blog/2015/06/deep-look-cql-where-clause> (дата обращения 01.06.2020)
7. Apache Cassandra: Cassandra documentation. – Текст. Изображение. : электронные // Apache Cassandra. : [сайт]. – URL: <http://cassandra.apache.org/doc/latest/> (дата обращения 01.05.2022)
8. Habr. Моделирование данных в Cassandra 2.0 на CQL3. – Текст. Изображение. : электронные // Habr : [сайт]. – URL: <https://habr.com/ru/post/203200/> (дата обращения 01.05.2022)
9. Habr. SELECT...WHERE запросы в Cassandra 2.0 на CQL3. – Текст. Изображение. : электронные // Habr : [сайт]. – URL: <https://habr.com/ru/post/205176/> (дата обращения 01.05.2022)
10. How to Program Blog: Using Group By in Apache Cassandra. – Текст. Изображение. : электронные // How to Program Blog: [сайт]. – URL:

<https://howtoprogram.xyz/2017/02/18/using-group-apache-cassandra/> (дата обращения 01.05.2022)

11. Datastax. Documentation. CQL for Apache Cassandra 3.0: Time-to-live. – Текст. : электронные // Datastax: Documentation : [сайт]. – URL: https://docs.datastax.com/en/cql-oss/3.3/cql/cql_using/useTTL.html (дата обращения 01.06.2020)
12. Datastax. Documentation. CQL for Apache Cassandra 3.0: Materialized View. – Текст. : электронные // Datastax: Documentation : [сайт]. – URL: https://docs.datastax.com/en/cql-oss/3.3/cql/cql_reference/cqlCreateMaterializedView.html (дата обращения 01.06.2020)