KARPOV.COURSES >>>
КОНСПЕКТ

> Конспект > 3 урок > Apache Hive. SQL для Big Data

> Оглавление

> История Hive

- Разработка началась в 2010 году в Facebook
- SQL подобный язык запросов (HiveQL)
- Трансляция в каскад заданий
- Несколько движков исполнения (Map-Reduce, Tez, Spark)
- JDBC и ODBC драйверы для интеграции с существующими системами

Информация хранится в обычных файлах(на HDFS или S3)

- Text file
- Sequence file
- Parquet, ORC
- ...

Hive выглядит как база данных поверх данных на HDFS, но не является ей на самом деле. Это некая псевдо-база, которая не может себя полноценно обслуживать.

Мета-информация хранится в RDBMS

- Apache Derby (по умолчанию)
- MySQL
- Postgres
- Oracle

Стоит отметить, что в Hive можно написать собственную функцию посредством UDF (User Defined Function). Существуют также UDAF (User Defined Aggregate Function) и UDTF (User Defined Tabular Function).

> Иерархия Hive

- Database (папка на HDFS)
  - Table (папка на HDFS)

    - Partition (папка на HDFS) *опционально
    - Bucket (файлы HDFS) *опционально
- View (представление - хранимый запрос)
- Materialized view (query rewrite)

При создании Database (базы данных), Table (таблицы) или Partition (партиции) мы можем явно указывать расположение данных на HDFS.

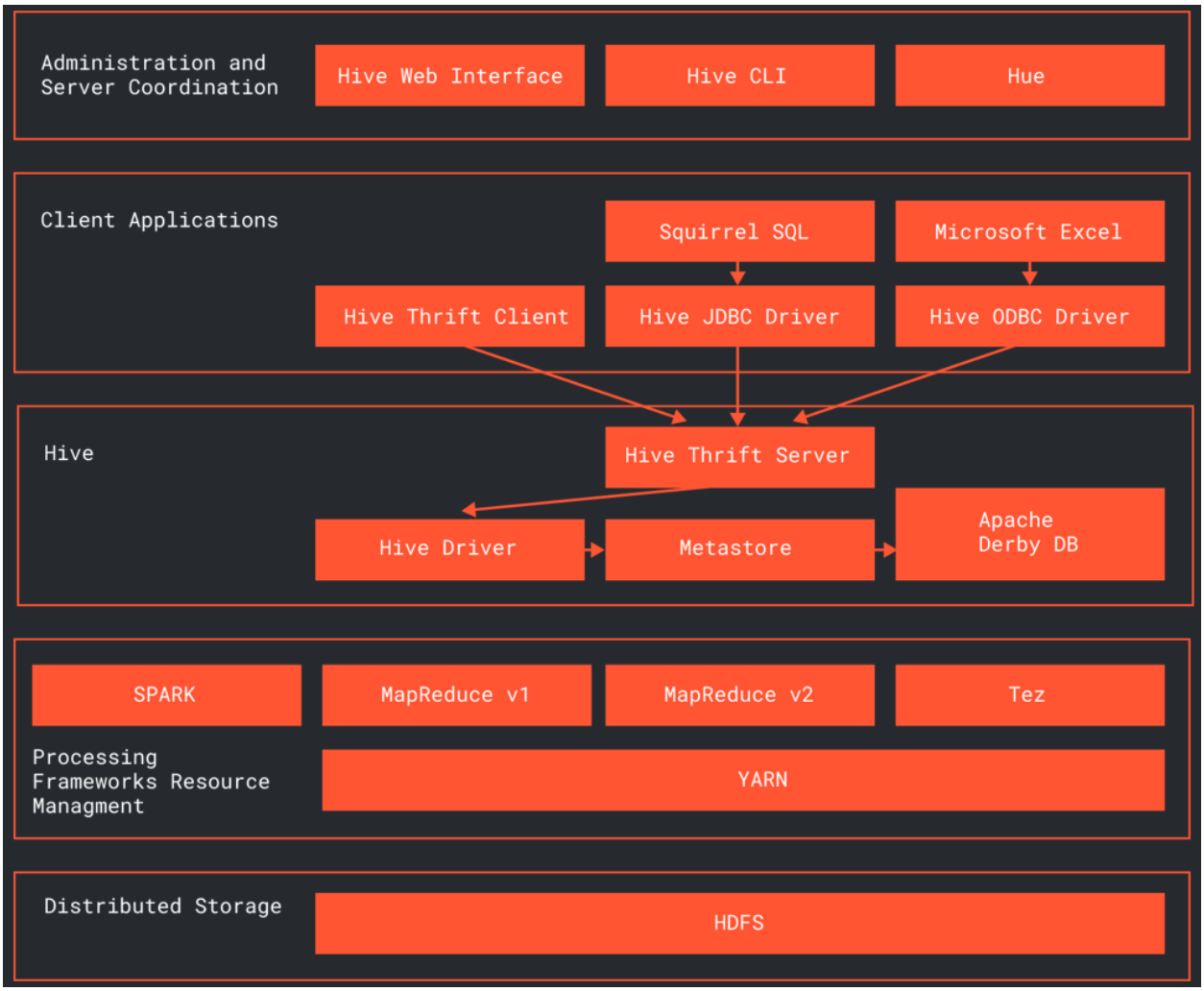
Партиции - разбиение данных в таблице по какому-либо принципу. Например, по дням. Таким образом поиск нужных данных в большой таблице будет занимать меньше времени. Партиции также можно делать вложенными. На примере продаж продуктовой сети: 1 уровень - филиал, 2 уровень - дата продажи.

Количество Buckets (бакеты) указывается на момент создания таблицы. Требуется указать по каким полям мы хотим бакетировать таблицу, с каким количеством бакетов и сортировку (опционально). Бакеты позволяют делать эффективный join таблиц, но требуют внимательного обслуживания.

View (представления) не оптимизируют запрос, а только добавляют удобства в работе с большими запросами.

Materialized view (материализованное представление) - условная витрина, строящаяся по какому-либо запросу. Материализованные представления пришли на смену индексам, так как начиная с версии Hive 3 их нет (они не получили широкого применения).

> Архитектура Hive



На верхнем уровне представлены управляющие средства - Administration and Server Coordination.

В Hive есть собственный веб-интерфейс Hive Web Interface, где можно отслеживать процесс работы запросов в кластере.

Hive CLI представляет собой две утилиты (Hive и Beeline). С точки зрения архитектурной целостности Beeline является более правильным средством.

Hue - веб-интерфейс, подключаемый к кластеру Hadoop и позволяющий использовать HDFS, запускать запросы в Hive, делать минимальную графику.

JDBC (Java Database Connectivity) - стандарт взаимодействия Java-приложений с различными СУБД.

ODBC (Open Database Connectivity) - это программный интерфейс (API) доступа к базам данных, разработанный компанией Microsoft. Помогает, к примеру

подключиться к Hive через Excel.

Hive Driver получает запрос от Hive Thrift Server, который в свою очередь получает его от клиентских приложений, и анализирует его. Затем он обращается к Metastore для семантических и синтаксических проверок, строит план запроса и отправляет на выполнение. В зависимости от выбранного движка (Map-Reduce, Tez, Spark), запрос будет преобразован должным образом. Работа производится внутри YARN.

> Создание таблиц в Hive

Пример создания managed таблицы

```
CREATE TABLE page_view(
    viewTime INT,
    userid BIGINT,
    page_url STRING,
    referrer_url STRING,
    friends ARRAY,
    properties MAP,
    ip STRING COMMENT 'IP Address of the Iser')
COMMENT 'This is the page view table'
PARTITIONED BY(dt STRING, country STRING)
CLUSTERED BY(userid) SORTED BY(viewTime) INTO 32 BUCKETS
ROW FORMAT DELIMETED
    FIELDS TERMINATED BY ';'
    COLLECTION ITEMS TERMINATED BY ','
    MAP KEYS TERMINATED BY '\'
STORED AS SEQUENCEFILE;
```

В общем случае создание таблицы в Hive похоже на создание таблицы в любой реляционной БД. Мы можем задавать комментарии полям, либо таблице в целом.

Выражение PARTITIONED BY в нашем случае партиционирует таблицу по двум уровням. Первый уровень - поле `dt`, второй уровень - `country`. Стоит отметить, что значения этих полей в данных на HDFS не будет. Информация о партициях хранится в Metastore.

CLUSTERED BY позволяет бакетировать таблицу - разбивать данные на бакеты. В данном случае создается 32 бакета по полю `userid`. Также внутри бакета можем задавать сортировку.

ROW FORMAT DELIMETED указывает формат, в котором мы будем хранить таблицу. В нашем случае это будет SEQUENCEFILE и внутри него мы храним по паттерну, что поля разделяются символом `';'`, сложные типы данных (коллекции) разделяются `','` и тип данных MAP (словарь) разделяется `'\'`.

Пример создания external таблицы

```
CREATE EXTERNAL TABLE page_view_stg (
    viewTime INT,
    userid BIGINT,
    page_url STRING,
    referrer_url STRING
    ip STRING COMMENT 'IP Address of the User',
    country STRING COMMENT 'country of origination'
    )
COMMENT 'This is the staging page view table'
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '44'
LINES TERMINATED BY '10'
STORED AS TEXTFILE
LOCATION '/user/data/staging/page_view';
```

Создание внешней таблицы в Hive тоже похоже на создание такой таблицы в обычных реляционных БД.

Мы можем создать внешнюю таблицу на данных, которые лежат не в папке, в котором расположен Hive, а где-то снаружи.

Главной причиной создания внешних таблиц является их поведение при удалении. При удалении обычной managed таблицы через команду `DROP TABLE`, Hive удаляет всю информацию о ней в Metastore и удаляет данные этой таблицы на HDFS. И проблема в том, что не во всех случаях данные можно восстановить обратно. Но при удалении внешней таблицы, удаляются только метаданные о ней. Сами данные не удаляются и не производится попыток их удаления.

LOCATION - обязательный параметр при создании внешних таблиц.

CTAS (Create Table As Select)

```
CREATE TABLE dataset_42 AS
SELECT
    o.name AS office_name,
    SUM(s.amount) AS total_amount
```

```
FROM fact_sells AS s
JOIN dim_office AS o
    ON s.office_id = o.id
WHERE id BETWEEN 1000 and 2000
GROUP BY o.name
STORED AS ORC;
```

В Hive мы можем также создавать таблицы через CTAS. Это способ создания таблиц с помощью запроса к какой-то другой таблице (таблицам).

> Join в Hive

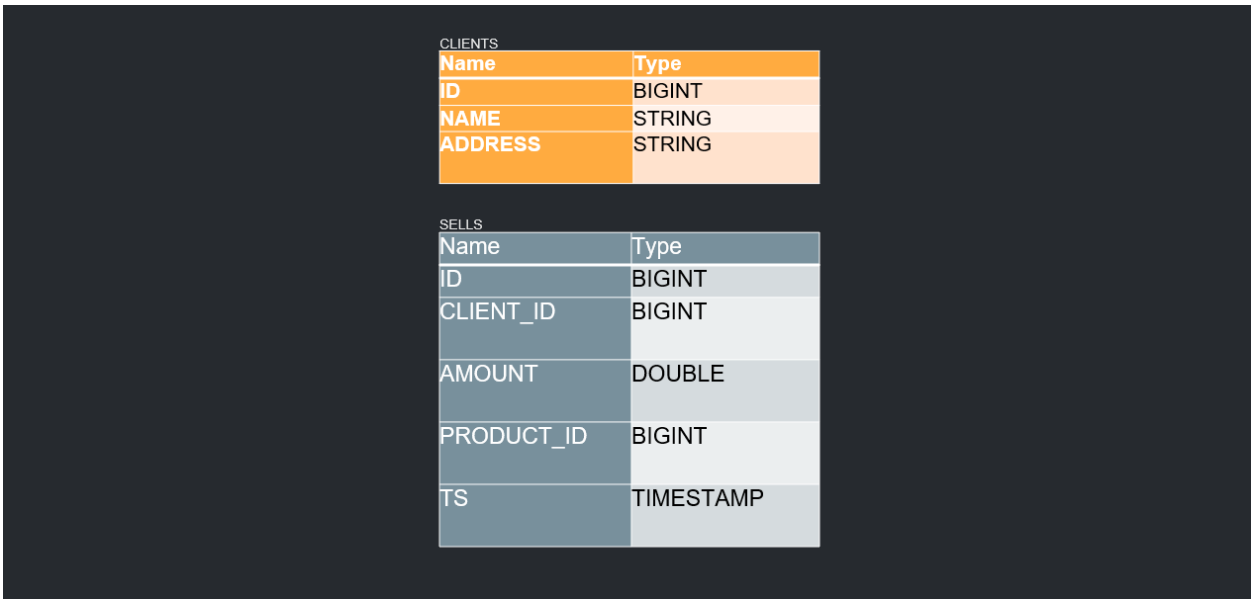
Типы Join

- Классический reduce side join
- Map side join
- Sort Merge Bucket (SMB) join

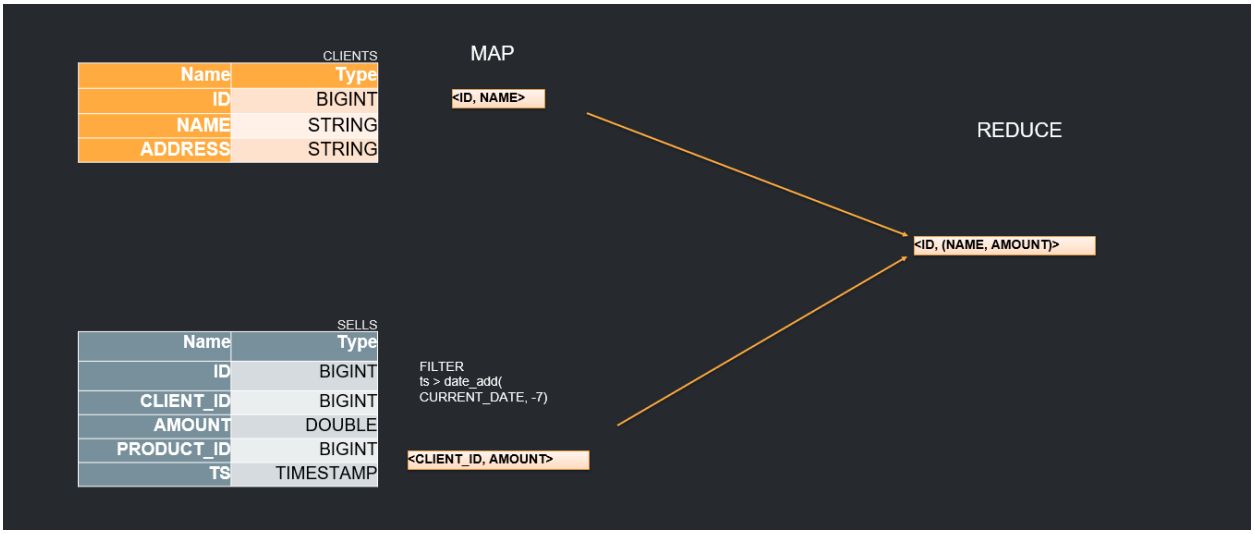
Классический Join очень дорогая операция в Hive – требует сортировки обоих таблиц в MapReduce. При джойне больших таблиц, Hive может падать с ошибкой памяти. Hive позволяет MapReduce писать более лаконично.

Рассмотрим джойн двух таблиц CLIENTS (клиенты) и SELLS (продажи). Допустим, мы хотим по каждому из клиентов рассчитать сумму, которую он потратил за последние 7 дней.

```
SELECT
    SUM(s.amount) AS amount,
    c.name
FROM clients c
JOIN sells s
    ON c.id = s.client_id
WHERE ts > date_add(CURRENT_DATE, -7)
GROUP BY c.name;
```



Reduce side Join



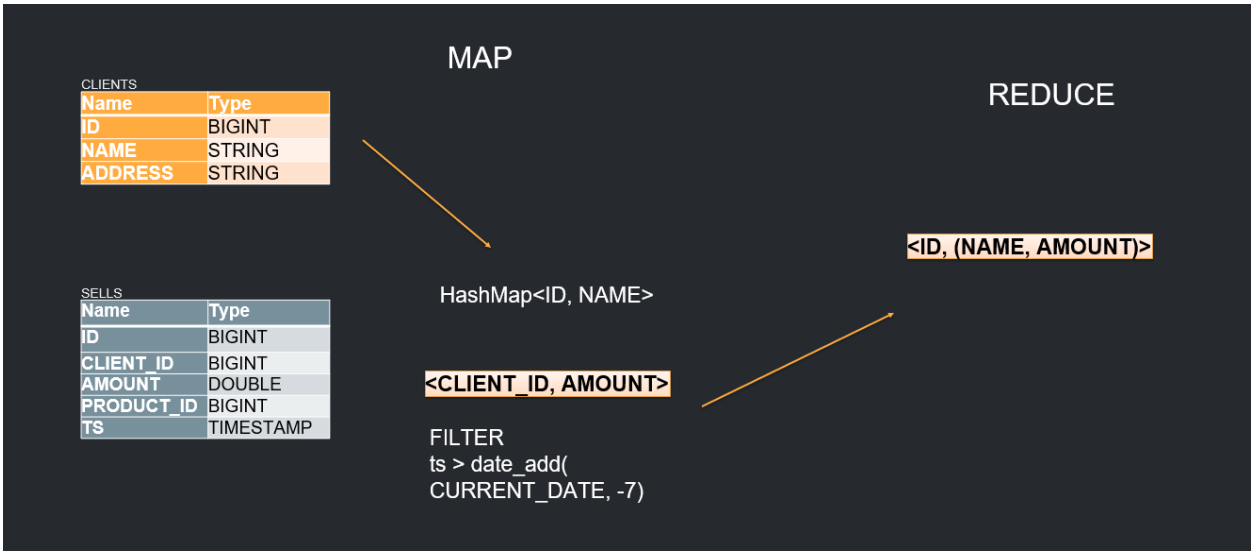
Для обеих таблиц запускаются mappers. Mappers, работающие с таблицей клиентов будет на выходе предоставлять пару ключ-значение. В качестве ключа будет использоваться `id`, в качестве значения - `name`. Mappers, работающие с таблицей продаж будут использовать фильтр и на выходе из них мы получаем ключ `client_id` и значение - `amount`. Эти записи будут приходить на reducer, где мы уже получаем `id`, который для обеих таблиц по значению является одним и тем же, `name` из mappers, которые работали с таблицей клиентов и `amount` от mappers,

работающих с таблицей продаж. Так как все ключи гарантированно на одном reducer, мы можем провести агрегацию по этим ключам и получить результат.

Map side Join

```
SELECT /*+ MAP JOIN(clients) */
    SUM(s.amount) AS amount,
    c.name
FROM clients c
JOIN sells s
    ON c.id = s.client_id
WHERE ts > date_add(CURRENT_DATE, -7)
GROUP BY c.name;
```

Для включения Map side Join, требуется указать хинт `/*+ MAP JOIN(clients) */`.



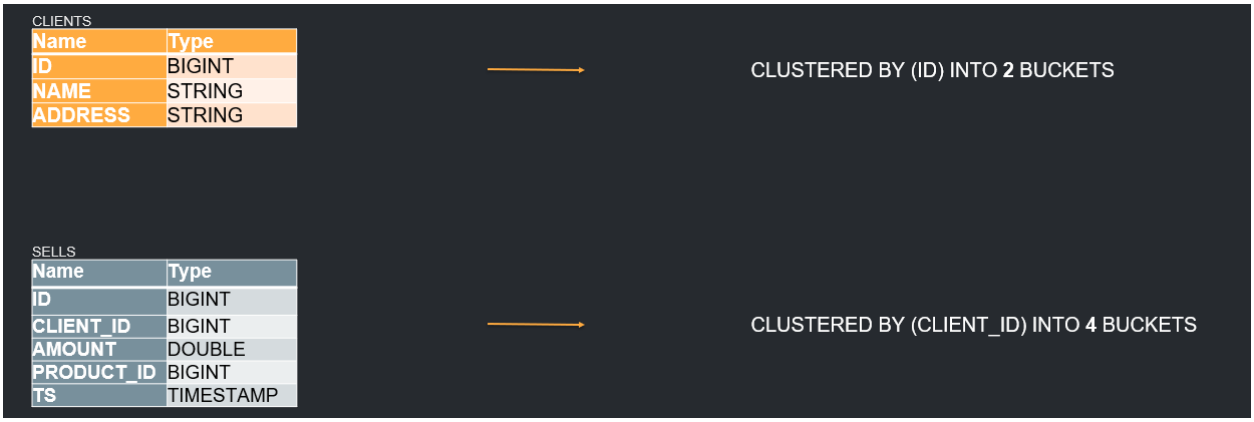
Перед запуском MapReduce задания, таблица `clients`, объявленная в хинте, преобразуется в HashMap (словарь) с ключом и значением. Это выгружается на HDFS и далее каждый mapper, который берет данные таблицы продаж, он берет их из этой таблицы и подтягивает HashMap, который лежит на HDFS. HashMap загружается в память и для джойна таблиц уже не требуется стадия reduce. Джойн двух таблиц будет происходить на mappers, которые берут данные из таблицы с продажами.

SMB (Sort Merge Bucket) Join

Для того, чтобы сработал SMB Join должно быть выполнено два условия:

- Таблицы должны быть бакетированы по полю, которому будет происходить join
- Количество бакетов в обоих таблицах должно быть кратно друг другу (2-4, 2-8, 5-10).
- Убедиться, что указанные параметры выставлены в значение True

```
SET hive.auto.convert.sortmerge.join=true;
SET hive.optimize.bucketmapjoin=true;
SET hive.optimize.bucketmapjoin.sortedmerge=true;
SET hive.auto.convert.sortmerge.join.noconditionaltask=true;
```



Принцип работы SMB Join

Есть левая и правая таблица. В левой (`clients`) имеются два бакета, а в правой (`sales`) - четыре. Для работы SMB Join нужно соответствие в количестве бакетов между двумя таблицами. Берется хэш от значения поля, делится на количество бакетов и остаток от них будет соответствовать бакету, в которым лежат данные. Бакет 0 в таблице `clients` - означает, что остаток от деления хэша `id` из таблицы `clients`, деленный на 2 (количество бакетов) будет равняться 0. Этому бакету будут соответствовать бакет 0 и бакет 2 из таблицы `sales`. Таким образом все эти данные могут попасть на один mapper и в рамках него заджойниться. Стадии reduce и стадии shuffle в данном случае не будет.

Следует иметь в виду, что при insert в таблицу, которая ранее была поделена на бакеты, может произойти такое, что часть данных, которая была загружена этой

операцией, не будет поделена бакеты и таким образом SMB Join отработает некорректно.

> Транзакции и виртуальные поля

Транзакции

При создании Hive не подразумевалось использование транзакций (INSERT, DELETE, UDPATE).

С появлением формата ORC, ограничения обойти удалось. Все изменения хранятся в delta-файлах и при запросе данных из Hive они применяются к оригинальным данным и в итоге мы получаем тот результат, который ожидаем увидеть.

Механизм compaction производит процесс, при котором сравниваются оригинальная версия и delta-файлы, производится "накатка" вторых на первую и сохраняется как новая версия данных с удалением старой.

BEGIN, COMMIT, ROLLBACK в Hive не поддерживается.

Виртуальные поля

```
SELECT
    INPUT_FILE_NAME,
    BLOCK_OFFSET_INSIDE_FILE,
    m.*
FROM movies m;
```

INPUT_FILE_NAME - возвращает название файла, где хранятся данные

BLOCK_OFFSET_INSIDE_FILE - физическое смещение данных внутри файла

Поля полезны при разборе инцидентов или ошибок.

> Глоссарий

Hive - система управления базами данных на основе платформы Hadoop. Позволяет выполнять запросы, агрегировать и анализировать данные,

хранящиеся в Hadoop.

JDBC (Java Database Connectivity) - стандарт взаимодействия Java-приложений с различными СУБД.

ODBC (Open Database Connectivity) - это программный интерфейс (API) доступа к базам данных, разработанный компанией Microsoft. Помогает, к примеру подключиться к Hive через Excel.

Hue - веб-интерфейс, подключаемый к кластеру Hadoop и позволяющий использовать HDFS, запускать запросы в Hive, делать минимальную графику.

Sequence file - это двоичный формат для хранения данных в виде сериализованных пар ключ/значение в экосистеме Apache Hadoop, позволяющий разбивать файл на участки (порции) при сжатии.

Parquet - то бинарный, колоночно-ориентированный формат хранения больших данных, изначально созданный для экосистемы Hadoop, позволяющий использовать преимущества сжатого и эффективного колоночно-ориентированного представления информации.

ORC (Optimized Row Columnar) - это колоночно-ориентированный (столбцовый) формат хранения данных в экосистеме Apache Hadoop.

Apache Derby - реляционная СУБД, написанная на Java, предназначенная для встраивания в Java-приложения или обработки транзакций в реальном времени.