

Hive

Krzysztof Jankiewicz

Plan

- Wprowadzenie
 - Motywacja
 - Organizacja danych
 - Typy danych
- Formaty przechowywania danych
 - Format wierszowy vs kolumnowy
 - ORCFile
- Hive SQL
 - Ładowanie
 - Selekcja i projekcja
 - DML, transakcje w Hive
 - Rozszerzenia grupowania i funkcje analityczne
- Silniki wykonawcze

Wprowadzenie

- Hive jest warstwą przetwarzania danych opartą o platformę Hadoop
- Funkcjonuje głównie jako infrastruktura dla rozwiązań hurtowni danych
- Podstawowe funkcje Hive to:
 - łatwość wykonywania podsumowań
 - zapytania i analizy ad-hoc
 - przetwarzanie dużych wolumenów danych
 - możliwość tworzenia funkcji użytkownika (UDF)
- Historia:
 - Projekt początkowo (2007) był rozwijany przez Facebooka.
 - Pierwsza jego wersja została opublikowana w roku 2008
 - Rozwijany był przez wiele korporacji/instytucji takich jak Netflix, FINRA.
 - Funkcjonują niezależne wersje Hive – np. fork Apache Hive wchodzący w skład Amazon Elastic MapReduce w ramach AWS.
- Daty wydania i wersje:

| 0.10.0 | 1.0.0 | 2.0.0 | 3.0.0 | 2.3.8 – 01.2021 |
|------------------|-------------|--------------|-------------|------------------------|
| 11 Styczeń, 2013 | 4 Luty 2015 | 15 Luty 2016 | 21 Maj 2018 | 3.1.2 – 08.2019 |

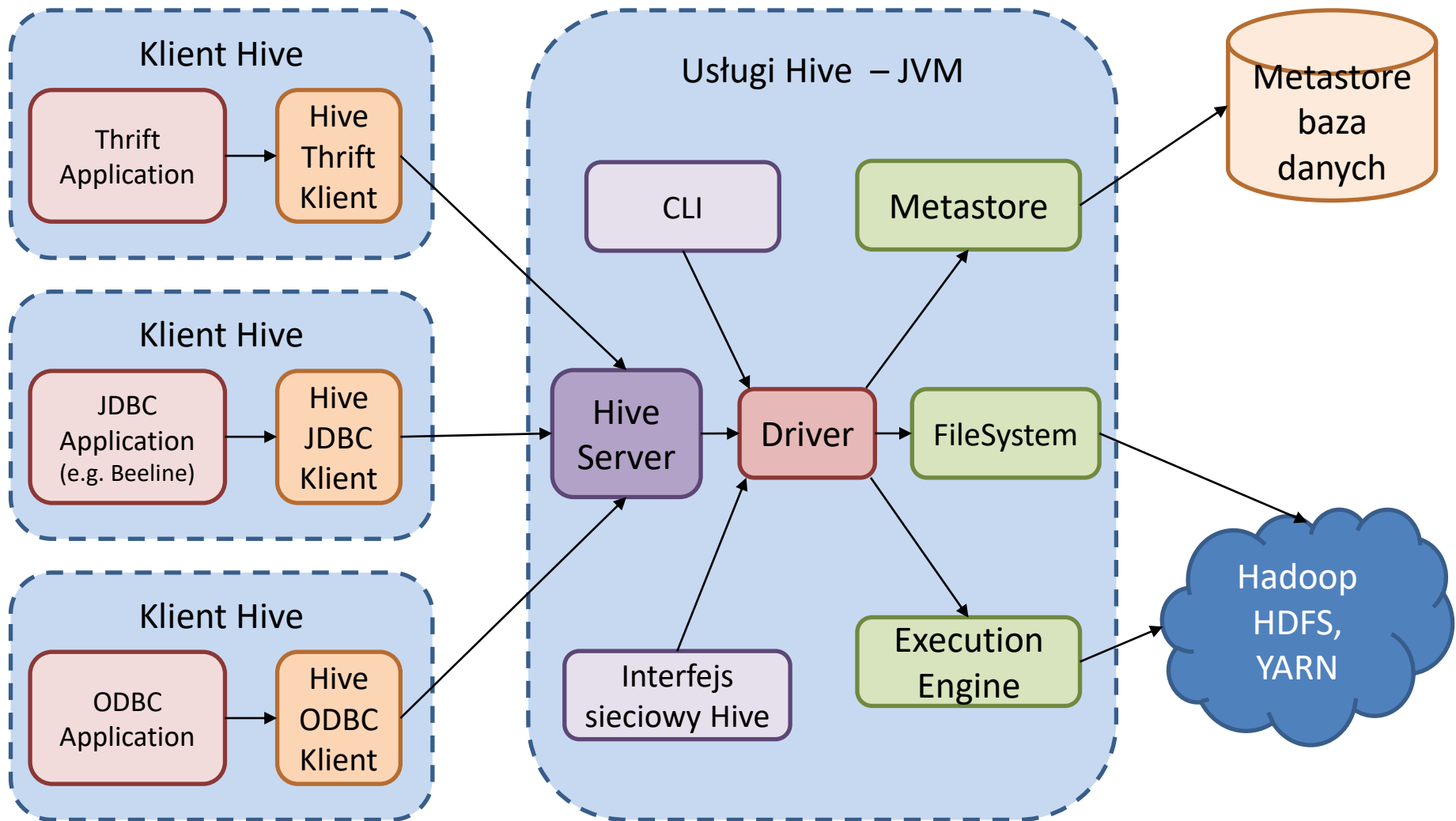
Organizacja danych

Tabela nie musi być partycjonowana albo dzielona na kubeczki.

Dane w Hive są zorganizowane za pomocą:

- **Baz danych** – przestrzeń nazw dla tabel, perspektyw, partycji itp. Wykorzystywana do określania poziomu uprawnień dla użytkowników i grup użytkowników
- **Tabel** – homogeniczne zbiory danych posiadających ten sam schemat
- **Partycji** – każda z tabel może posiadać jeden lub kilka kluczy partycji, które określają sposób składowania danych. Partycje są jednostkami składowania danych.
Każdy unikalny zestaw wartości klucza partycji definiuje partycję tabeli, w której dane posiadające ten zestaw wartości będą przechowywane. Wartości klucza partycji nie są przechowywane – pełnią rolę kolumn wirtualnych.
- **Kubeczków** (ang. *buckets, clusters*) – dane w każdej partycji mogą dodatkowo zostać podzielone na kubeczki. Podział ten następuje na podstawie wartości funkcji haszującej wyznaczonej na podstawie wartości kolumny w tabeli.
Są dwa podstawowe powody wykorzystania kubeczków
 - wydajnościowe (np. łączenie dwóch tabel, których podział na kubeczki oparty był na atrybutach połączeniowych)
 - w celu zwiększenia wydajności przy generowaniu próbek danych

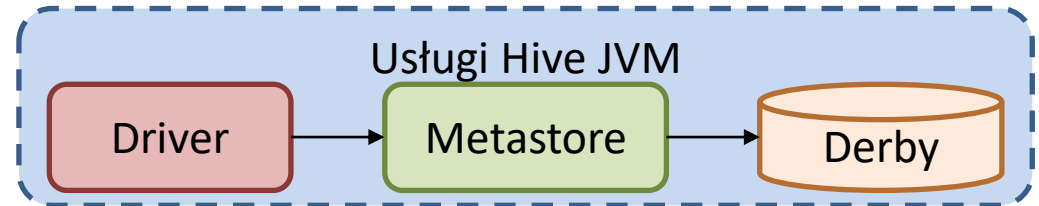
Architektura



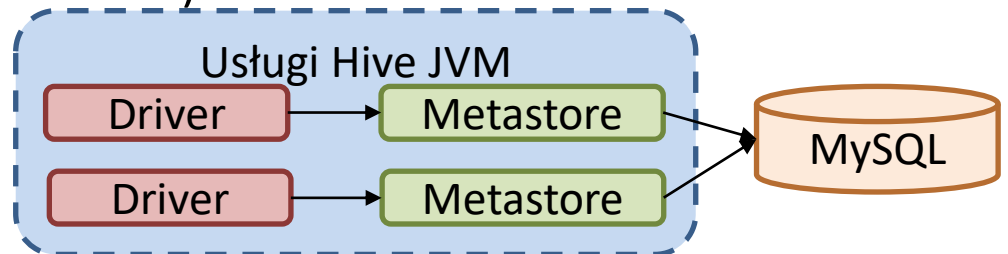
Metastore

- Metastore jest centralnym repozytorium metadanych Hive
- Przechowuje metadane tabel Hive (schemat i lokalizacja) oraz partycji w relacyjnej bazie danych.
- Składa się z dwóch elementów
 - Usługa, która zapewnia dostęp metastore do innych usług Apache Hive
 - Magazyn dyskowy dla metadanych Hive, który jest oddzielony od magazynu HDFS

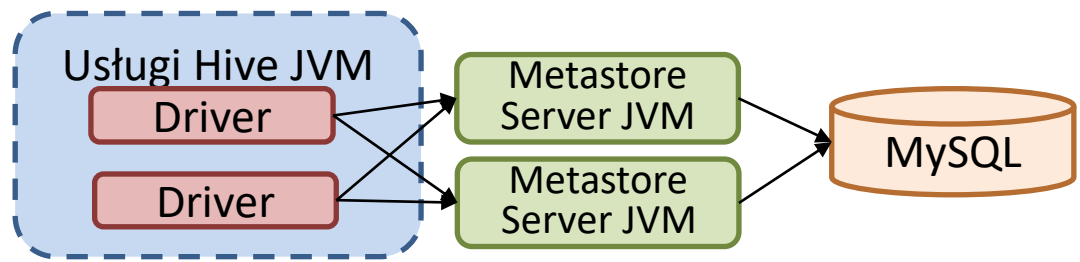
- Istnieją trzy tryby wdrażania Hive Metastore:
 - *Embedded* Metastore (wbudowany, dla testów, Derby)



- *Local* Metastore (wiele sesji, dostęp przez JDBC)



- *Remote* Metastore (działa na osobnej JVM, używa Thrift Network API)



Typy danych (1/2)

- Tabele w Hive posiadają schemat, w którym każda kolumna ma określony typ
- Podstawowe typy danych w Hive to:
 - Liczby całkowite (TINYINT, SMALLINT, INT, BIGINT)
 - Logiczne (BOOLEAN)
 - Liczby zmiennoprzecinkowe (FLOAT, DOUBLE)
 - Liczby stałoprzecinkowe (DECIMAL)
 - Ciągi znaków (STRING, VARCHAR, CHAR)
 - Daty i czas (TIMESTAMP, DATE)
 - Binarne (BINARY)

Typy danych (2/2)

- Oprócz typów podstawowych w Hive występują także typy złożone:
 - Rekordy (STRUCT) – elementy we wnętrzu rekordów osiągalne są za pomocą notacji kropkowej
 - Mapy – pary klucz-wartość, elementy osiągalne są za pomocą wartości klucza
 - Tablice – elementy w tablicy muszą mieć ten sam typ i osiągalne są za pomocą indeksu.
Pierwszy element tablicy posiada indeks 0.

Formaty przechowywania danych

- Format wierszy
(SerDe – Serializator-Deserializator)
- Format plików
 - tekst z ogranicznikami
 - domyślny ogranicznik wartości to Ctrl+A (ASCII 1)
 - domyślny ogranicznik elementów kolekcji to Ctrl+B
 - domyślny ogranicznik kluczy od wartości to Ctrl+C
 - domyślny ogranicznik wierszy to znak nowej wiersza
 - formaty binarne
 - SequenceFile
 - pliki systemu Avro
 - pliki Parqueta
 - pliki RCFile
 - pliki ORCFile

```
CREATE TABLE ...  
ROW FORMAT DELIMITED  
FIELDS TERMINATED BY '\001'  
COLLECTION ITEMS TERMINATED BY '\002'  
MAP KEYS TERMINATED BY '\003'  
LINES TERMINATED BY '\n'  
STORED AS TEXTFILE;
```

```
CREATE TABLE table_json_format(  
  id    int,  
  name  string  
)  
ROW FORMAT SERDE  
'org.apache.hive.hcatalog.data.JsonSerDe'  
STORED AS TEXTFILE;
```

```
STORED AS SEQUENCEFILE;  
STORED AS AVRO;  
STORED AS PARQUET;  
STORED AS RCFile;  
STORED AS ORC;
```

Binarne formaty przechowywania danych

- SequenceFile (układ wierszowy)
 - format danych pozwalający na przechowywanie danych binarnych mających postać par klucz-wartość
 - plik typu SequenceFile zawiera nagłówek, po którym następują rekordy
- Pliki systemu Avro (układ wierszowy)
 - Apache Avro™ to system serializacji danych (AvroSerde)
 - patrz: <http://avro.apache.org/>
- pliki Parqueta (układ kolumnowy)
 - kolumnowy format przechowywania danych
- pliki RCFile (Record Columnar File) (układ kolumnowy)
 - struktura dla hurtowni danych funkcjonujących w oparciu model MapReduce
 - kolumnowy format przechowywania danych
- ORCFile (Optimized Record Columnar File) (układ kolumnowy)
 - wsparcie dla typów złożonych w Hive
 - kompresja na poziomie bloków (run-length encoding dla kolumn liczbowych, słownikowa dla ciągów znaków)

| | K1 | K2 | K3 |
|----|----|----|----|
| W1 | 1 | 2 | 3 |
| W2 | 4 | 5 | 6 |
| W3 | 7 | 8 | 9 |
| W4 | 10 | 11 | 12 |

SequenceFile (układ wierszowy)

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|----|----|----|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|----|----|----|

RCFile (układ kolumnowy)

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|----|---|----|---|----|
| 1 | 4 | 2 | 5 | 3 | 6 | 7 | 10 | 8 | 11 | 9 | 12 |
|---|---|---|---|---|---|---|----|---|----|---|----|

Wycinek wierszy 1

Wycinek wierszy 2

```
select K1
from TAB;
```

```
insert into TAB
values (10,11,12);
```

ORCFile

- Celem wprowadzenia ORCFile było
 - zwiększenie wydajności przetwarzania
 - zmniejszenie wielkości plików
- Podstawowe cechy
 - format kolumnowy
 - świadomość typów – pozwala na optymalizacje składowania
 - dane podzielone są na paski (*stripes*), które wykorzystywane są podczas zrównoległania przetwarzania (standardowo 250MB)
 - wewnętrzne indeksy zawierające statystyki kolumn
 - kompresja zawartości, RLE (run length encoding), słownikowa
- Wiele dużych użytkowników Hadoop wykorzystuje ORC dla przykładu: Facebook – setki petabajtów
- Wiele narzędzi korzysta z plików ORC dla przykładu: Presto, Pig, Spark, Hive, Nifi

Patrz:

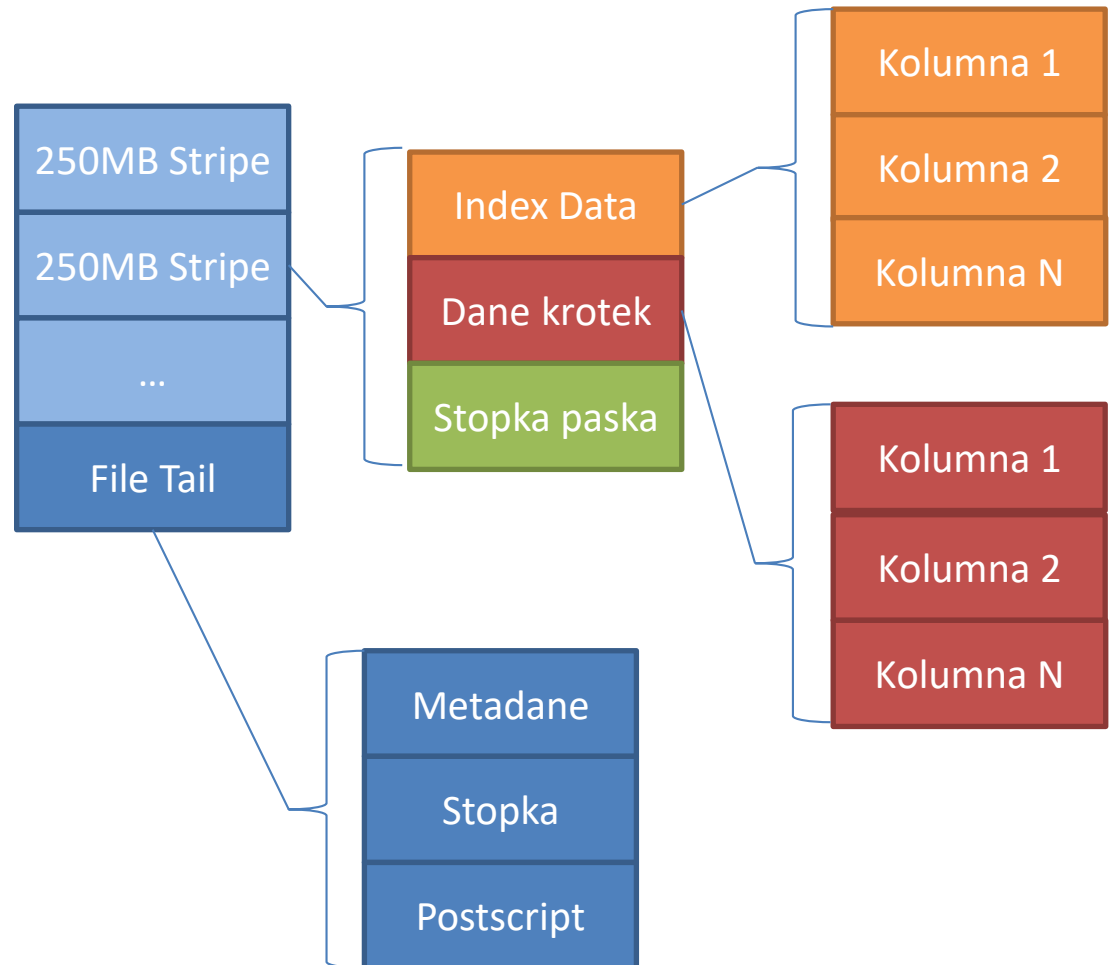
<https://cwiki.apache.org/confluence/display/Hive/LanguageManual+ORC>

<https://orc.apache.org/docs>

<https://www.youtube.com/watch?v=5vt2cVyE1GQ>

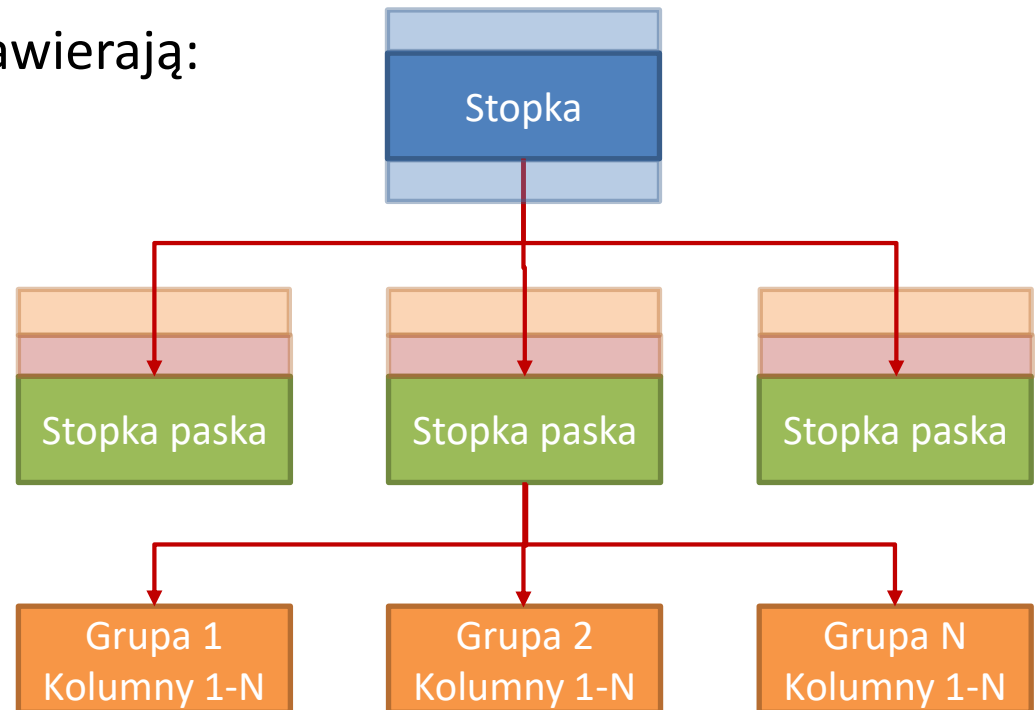
ORCFile – struktura pliku

- Podział pliku na zbiory krotek określone mianem pasków (stripes)
 - domyślny rozmiar 250MB (64MB)
 - większy rozmiar (w porównaniu do RCFile – 4MB) daje większą wydajność odczytu kolumn
- Stopka
 - zawiera listę **lokalizacji pasków**
 - typy i liczbę krotek
 - statystyki (indeksy) dla pliku i każdej z kolumn
- Dopisek (ang. *postscript*)
 - przechowuje parametry kompresji
 - rozmiar skompresowanej stopki
 - wielkość bloku kompresji
 - wersja
 - wielkość metadanych
- Metadane
 - przechowują pary klucz wartość opisujące plik, lub definiowane przez użytkownika



ORCFile – Indeksy

- Trzy typy indeksów przechowują statystyki dotyczące wartości wszystkich kolumn na określonym poziomie
 - poziom pliku – przechowywane w stopce pliku
 - poziom paska – przechowywane w stopce paska
 - poziom grup krotek – dla zbiorów po 10000 krotek wewnątrz paska
- Dane indeksu na poziomie paska zawierają **offsety** dla każdej z grup krotek
- Statystyki dotyczące kolumn zawierają:
 - liczbę wartości
 - informację czy występują wartości puste
 - dla prostych typów
 - wartość minimalną
 - wartość maksymalną
 - dla typów numerycznych
 - sumę wartości
 - dla typów znakowych
 - sumę długości
 - od wersji Hive 1.2 dodatkowo także filtr Bloom



Filtr Blooma

- Zaproponowana w 1970 roku przez Burtona H. Blooma
- W odróżnieniu do indeksów czy tablic mieszających
 - nie posiada wskaźników na dane, "przechowuje" jedynie wartości
 - zajmuje mniej miejsca
 - nie obsługuje usuwania danych
 - możliwość wystąpienia odpowiedzi *false positive*
- Inne zastosowania
 - korekta słownikowa
 - lista zabronionych haseł
 - routery sieciowe (listy niebezpiecznych IP, elementy przechowywane w buforach)

dlaczego wybrane funkcje są niewłaściwe?

Patrz: <https://sites.google.com/site/murmurhash/>
<http://isthe.com/chongo/tech/comp/fnv/>

- Budowa:
 - mapa bitowa o wielkości m
 - funkcje haszujące w liczbie k – mała liczba całkowita, funkcje powinny być niezależne, równomiernie rozmieszczające wyniki w przedziale $0-(m-1)$ – przykłady: *murmur*, *fnv*
- Operacje
 - wstawianie
 - weryfikacja istnienia
- Przykład:
 - $m: 10, k: 2$ ($\text{mod}9, \text{mod}7$)
 - wstawiamy: 11, 23, 49, 33
 - sprawdzamy istnienie: 22, 11, 14

| mod | 9 | 7 |
|-----|---|---|
| 11 | 2 | 4 |
| 23 | 5 | 2 |
| 49 | 4 | 0 |
| 33 | 6 | 5 |

filtr blooma:

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 1 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 0 |

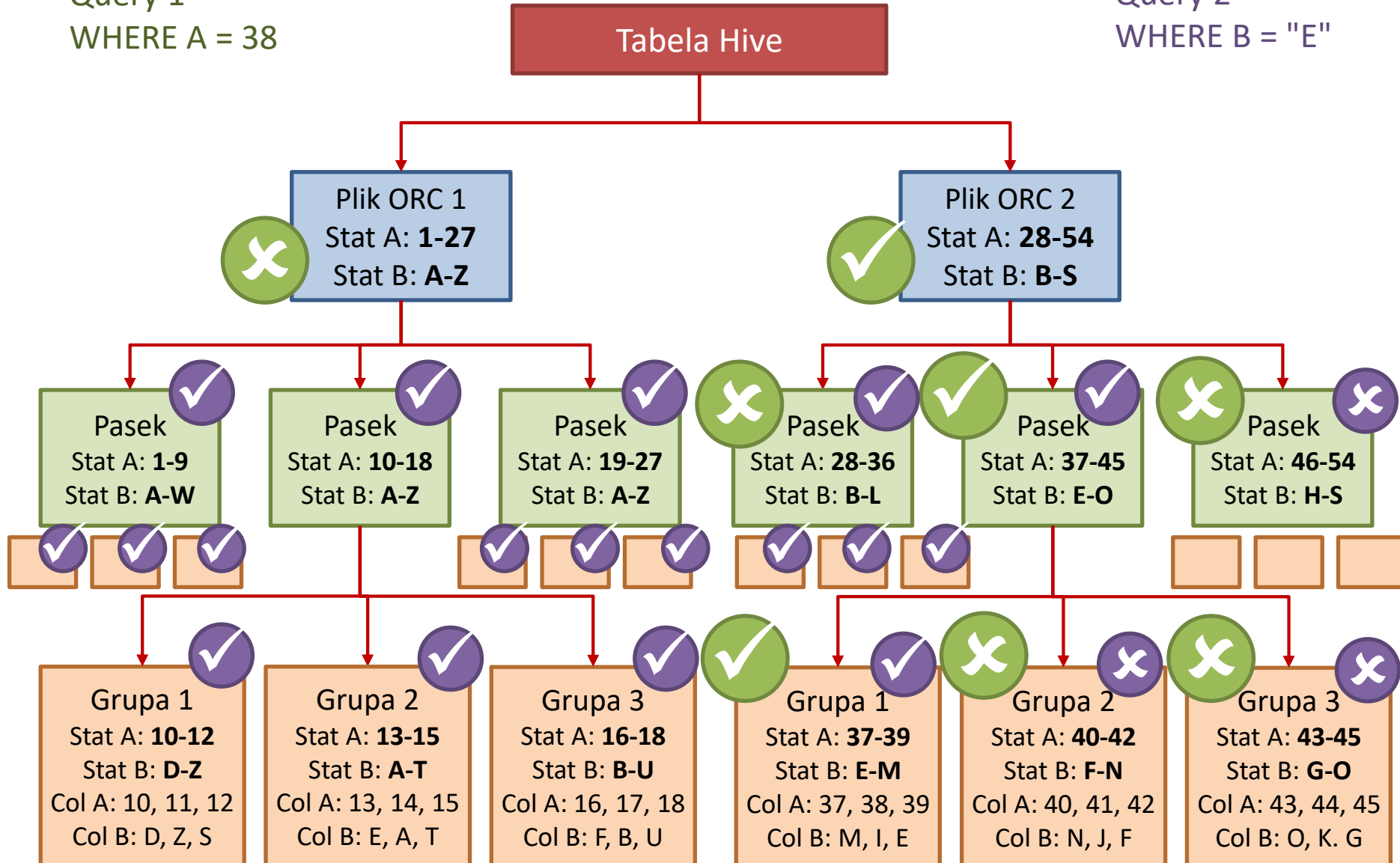
| mod | 9 | 7 |
|-----|---|---|
| 22 | 4 | 1 |
| 14 | 5 | 0 |

22 – nie istnieje (bo bit 1 = 0)
11 – oczywiście tak
14 – *false positive*

Jak to działa?

Query 1
WHERE A = 38

Query 2
WHERE B = "E"



Wpływ na wydajność

- TPC-DS – Decision Support Benchmark
- Zapytanie:
 - `from tpch1000.lineitem where l_orderkey = 1212000001`
- Liczba odczytanych wierszy
 - Bez optymalizacji: 6mld
 - Max/Min (indeksy): 540 000
 - BloomFilter: 10 000
- Czas wykonania zapytania
 - Bez optymalizacji: 74s
 - Max/Min (indeksy): 4,5s
 - BloomFilter: 1,3s

Testy przeprowadzone przez Hortonworks w oparciu o TPC-DS.

Opis TPC-DS.: http://www.tpc.org/information/sessions/tpc_ds.pdf

Silniki przetwarzania

- Hive dostarcza poziom abstrakcji, który tworzy zadania wykonywane przez "niskopoziomowy" silnik przetwarzania
 - MapReduce – klasyczny silnik, wymagający konwersji złożonych zapytań w szereg prostych zadań MR zapisujących swoje wyniki (pośrednie dla zapytania) w ramach HDFS
 - TEZ – silnik, który pozwala na więcej, pojawił się w Hive 0.13 (kwiecień 2014) w wyniku projektu Stinger
 - Spark – wersja Hive 1.1 (wrzesień 2017)

Projekt Stinger

- Cele :
 - Speed (100 x szybciej -> interaktywne zapytania)
 - Skalowalność (z TB do PB)
 - SQL (Szerszy zakres wsparcia obejmujące aplikacje analityczne)
- Kamienie milowe
 - Hive 0.11 (maj 13) – funkcje analityczne, ORCFile
 - Hive 0.12 (październik 13) – typy danych: DATE, VARCHAR, pushdown filter dla ORCFile
 - Hive 0.13 (kwiecień 14) – TEZ, CBO (Optiq), wektoryzacja przetwarzania

Patrz:

<https://issues.apache.org/jira/browse/HIVE-7292>

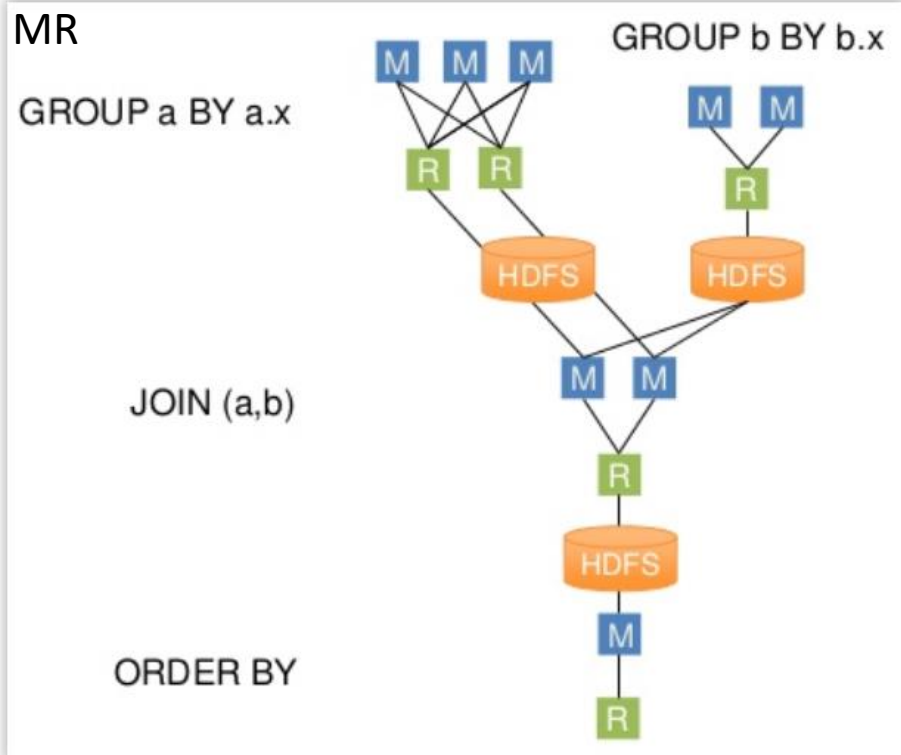
<https://www.slideshare.net/t3rmin4t0r/hivetez-a-performance-deep-dive>

MapReduce vs. TEZ

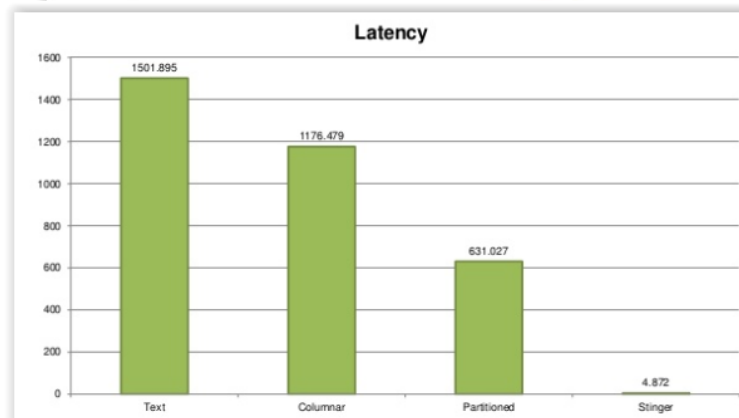
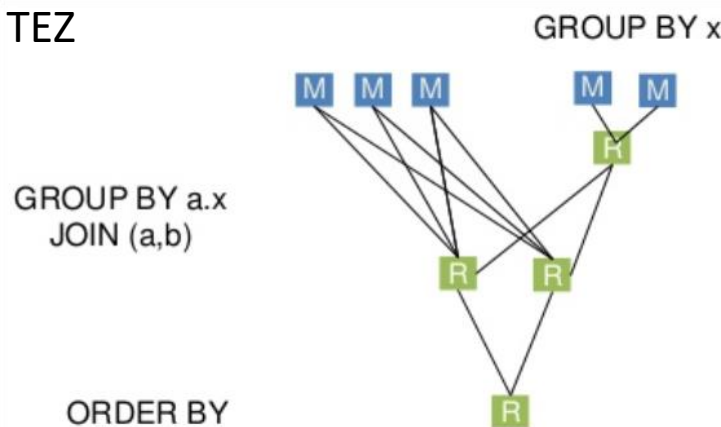
- TEZ rozwijany jest m.in. przez Hortonworks, Facebook, Twitter, Yahoo, Microsoft
- Pozwala na wykonywanie dowolnego DAG w ramach jednego zadania (pośrednie etapy nie są zapisywane w HDFS)

Przykład zapytania:

```
SELECT g1.x, g1.avg, g2.cnt
FROM (SELECT a.x, AVERAGE(a.y) AS avg
      FROM a GROUP BY a.x) g1 JOIN
      (SELECT b.x, COUNT(b.y) AS avg
      FROM b GROUP BY b.x) g2
ON (g1.x = g2.x)
ORDER BY avg;
```



TEZ



Hive SQL

- Hive SQL udostępnia podstawowe operacje SQL. Działają one zarówno na tabelach jak i partycjach tabel
 - Selekcja danych za pomocą klauzuli WHERE.
 - Projekcja danych z tabeli za pomocą klauzuli SELECT.
 - Połączenia równościowe pomiędzy tabelami.
 - Wyznaczanie agregatów dla pogrupowanych data składowanych w tabelach.
 - Składowanie wyników zapytań w tabelach.
 - Pobieranie danych z tabel do „lokalnych” systemów plików (np. NFS).
 - Składowanie wyników zapytań w katalogach HDFS.
 - Zarządzanie (tworzenie, usuwanie i modyfikacja) tabelami i partycjami.
 - Wykorzystanie skryptów do definiowania złożonych zadań map/reduce.

Klienci CLI, operacje na bazie danych

- **beeline** – klient JDBC oparty na SQLLine CLI

zobacz polecenia w: <http://sqlline.sourceforge.net/>

```
jankiewicz_krzysztof@cluster-0999-m:~$ beeline --silent=true
beeline> !connect jdbc:hive2://localhost:10000/default
Enter username for jdbc:hive2://localhost:10000/default: jankiewicz_krzysztof
Enter password for jdbc:hive2://localhost:10000/default:
Connected to: Apache Hive (version 3.1.2)
Driver: Hive JDBC (version 3.1.2)
```

Transaction isolation: **TRANSACTION_REPEATABLE_READ**

```
0: jdbc:hive2://localhost:10000/default> create table tab (col1 integer);
```

```
0: jdbc:hive2://localhost:10000/default> show tables;
```

```
+-----+
| tab_name |
+-----+
| tab      |
+-----+
```

```
CREATE (DATABASE|SCHEMA) [IF NOT EXISTS] database_name
    [COMMENT database_comment]
    [LOCATION hdfs_path]
    [WITH DBPROPERTIES (property_name=property_value, ...)];
```

```
0: jdbc:hive2://localhost:10000/default> show databases;
```

```
+-----+
| database_name |
+-----+
| default      |
+-----+
```

```
DROP (DATABASE|SCHEMA) [IF EXISTS] database_name [RESTRICT|CASCADE];
```

```
USE database_name;
```

```
0: jdbc:hive2://localhost:10000/default> create database beers;
```

```
0: jdbc:hive2://localhost:10000/default> use beers;
```

Ładowanie danych

```
default> create table beers_sf(line int, abv float, ibu float, id int,  
. . . .> name string, style string, brewery_id int, ounces float)  
. . . .> ROW FORMAT DELIMITED FIELDS TERMINATED BY ',';  
(...)  
INFO : OK  
No rows affected (0.156 seconds)  
default> load data local inpath "/home/jankiewicz_krzysztof/beers.csv" into table beers_sf;  
(...)  
INFO : OK  
No rows affected (0.803 seconds)  
default> create table beers_orc(line int, abv float, ibu float, id int,  
. . . .> name string, style string, brewery_id int, ounces float)  
. . . .> STORED AS ORC;  
INFO : OK  
No rows affected (0.155 seconds)  
default> insert into beers_orc select * from beers_sf where id is not null;  
(...)  
INFO : Total jobs = 1  
INFO : Launching Job 1 out of 1  
INFO : Tez session hasn't been created yet. Opening session  
(...)  
INFO : OK
```

| | VERTICES | MODE | STATUS | TOTAL | COMPLETED | RUNNING | PENDING | FAILED | KILLED |
|-----------------|-----------|-----------|--------|-------|-----------|---------|---------|--------|--------|
| Map 1 | container | SUCCEEDED | 1 | 1 | 0 | 0 | 0 | 0 | |
| Reducer 2 | container | SUCCEEDED | 1 | 1 | 0 | 0 | 0 | 0 | |

VERTICES: 02/02 [=====>>] 100% ELAPSED TIME: 10.65 s

No rows affected (62.89 seconds)


Selekcja i projekcja danych

```
0: jdbc:hive2://localhost:10000/default> select count(*) from beers_orc;
(...)
INFO : OK
+-----+
|  _c0  |
+-----+
| 2410  |
+-----+
1 row selected (0.292 seconds)
0: jdbc:hive2://localhost:10000/default> select count(*) from beers_sf;
(...)
INFO : Tez session was closed. Reopening...
```

```
default> select name, ibu from beers_orc where ibu > 100 order by ibu desc;
(...)
```

```
Map 1 ..... container SUCCEEDED
Reducer 2 ..... container SUCCEEDED

VERTICES: 02/02 [=====>>]
INFO : OK
+-----+
|          name          |      ibu      |
+-----+
| Bitter Bitch Imperial IPA |    138.0    |
| Troopers Alley IPA      |    135.0    |
| Dead-Eye DIPA           |    130.0    |
| ...                     |              |
+-----+
26 rows selected (2.037 seconds)
```

 Home / All DAGs

All DAGs

Hive Queries

Last refresh

DAG Name:

ID:

Submitter:

Status:

Application ID:

Query

Search...

Search...

Search...

All

Search...

\$

| Dag Name | Id | Submitter |
|--|---------------------|----------------------|
| select name, ibu from beers_orc where...desc (Stage-1) | dag_162230350278... | jankiewicz_krzysztof |
| select count(*) from beers_sf (Stage-1) | dag_162230350278... | jankiewicz_krzysztof |
| insert into beers_orc select * from b...null (Stage-1) | dag_162230350278... | jankiewicz_krzysztof |
| insert into beers_orc select * from b...null (Stage-1) | dag_162230350278... | jankiewicz_krzysztof |

Load Counters

Łączenie i składowanie wyników zapytań

```
default> create table breweries_sf(id int, name string, city string, state string)
. . . .> ROW FORMAT DELIMITED FIELDS TERMINATED BY ',';
No rows affected (0.128 seconds)
default> load data local inpath "/home/jankiewicz_krzysztof/breweries.csv" into table breweries_sf;
```

```
default> INSERT OVERWRITE DIRECTORY '/user/jankiewicz_krzysztof/most_biter_beers'
. . . .> ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
. . . .> select breweries_orc.name, breweries_orc.city,
. . . .>         breweries_orc.state, beers_orc.name, beers_orc.ibu
. . . .> from   breweries_orc join beers_orc
. . . .>       on (breweries_orc.id = beers_orc.brewery_id)
. . . .> where  ibu > 100
. . . .> order by ibu desc;
```



| | VERTICES | MODE | STATUS | TOTAL | COMPLETED | RUNNING | PENDING | FAILED | KILLED |
|-----------|----------|-----------|-----------|-------|-----------|---------|---------|--------|--------|
| Map 1 | | container | SUCCEEDED | 1 | 1 | 0 | 0 | 0 | 0 |
| Map 2 | | container | SUCCEEDED | 1 | 1 | 0 | 0 | 0 | 0 |
| Reducer 3 | | container | SUCCEEDED | 1 | 1 | 0 | 0 | 0 | 0 |

VERTICES: 03/03 [=====>>] 100% ELAPSED TIME: 8.35 s

INFO : Moving data to directory /user/jankiewicz_krzysztof/most_biter_beers from hdfs://cluster-0999-m/user/jankiewicz_krzysztof/most_biter_beers/.hive-staging_hive_2021-05-29_16-45-32_044_8129673533934746481-4/-ext-10000

INFO : OK

No rows affected (9.44 seconds)

```
default> !sh hadoop fs -cat /user/jankiewicz_krzysztof/most_biter_beers/000000_0
```

Astoria Brewing Company,Astoria,OR,Bitter Bitch Imperial IPA,138.0

Wolf Hills Brewing Company,Abingdon,VA,Troopers Alley IPA,135.0

Cape Ann Brewing Company,Gloucester,MA,Dead-Eye DIPA,130.0

(. . .)

Wsadowe uruchamianie skryptów

- Podobnie jak w przypadku platformy Pig, tak również w przypadku Hive istnieje możliwość wsadowego uruchamiania sekwencji poleceń zawartych w skryptach
- Skrypty mogą być parametryzowane

```
beeline -u jdbc:hive2://localhost:10000/beers \
        -n username -p password \
        --hivevar min_ibu=100 -f file.sql
```

file.sql:

```
. . .
INSERT OVERWRITE DIRECTORY '/user/maria_dev/hive/data/most_biter_beers'
    ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
select * from beers where ibu >= ${min_ibu};
```


Transakcje w Hive

- Początkowo (do wersji 0.13) Hive udostępniał operacje zmieniające zawartość danych (o własności transakcji ACD) na poziomie tabel/partycji (tabela/partycja mogła być dodawana, usuwana)
- W chwili obecnej Hive udostępnia operacje zmiany danych na poziomie wierszy (o własnościach ACD) – operacje DML
- Celem wprowadzenia zmiany było wsparcie dla:
 - aplikacji, które strumieniowo ładują dane do hurtowni danych (liczba tworzonych partycji w czasie jest ograniczona)
 - obsługi SCD (*slowly changing dimensions*)
 - poprawy źle wprowadzonych danych
 - operacji merge
- Aby móc skorzystać z operacji DML, tabela musi być typu ACID
- Istnieje wiele ograniczeń obecnej funkcjonalności transakcji
 - Brak poleceń sterujących transakcjami – każde polecenie jest AUTOCOMMIT
 - Nie wszystkie formaty plików są obsługiwane (tylko ORC)
 - Tabele zewnętrzne nie są obsługiwane
 - Tylko sesje ACID mają dostęp do tabel ACID
 - Jedynym poziomem izolacji jest . . .
 - LOAD DATA nie jest dostępne dla tabel ACID

Transakcje w Hive

Czym fizycznie jest baza danych/tabela Hive?

```
0: jdbc:hive2://localhost:10000/beers> DESCRIBE FORMATTED beers_orc;
```

| col_name | data_type |
|------------------------------|--|
| # col_name | data_type |
| line | int |
| (. . .) | |
| # Detailed Table Information | NULL |
| Database: | beers |
| OwnerType: | USER |
| Owner: | jankiewicz_krzysztof |
| (. . .) | |
| Location: | hdfs://cluster-0999-m/user/hive/warehouse/beers.db/beers |
| Table Type: | MANAGED_TABLE |
| (. . .) | |
| # Storage Information | NULL |
| SerDe Library: | org.apache.hadoop.hive.ql.io.orc.OrcSerde |
| InputFormat: | org.apache.hadoop.hive.ql.io.orc.OrcInputFormat |
| OutputFormat: | org.apache.hadoop.hive.ql.io.orc.OrcOutputFormat |
| Compressed: | No |

```
jankiewicz_krzysztof@cluster-0999-m:~$ hadoop fs -ls /user/hive/warehouse/
```

```
Found 3 items
```

```
drwxr-xr-x - jankiewicz_krzysztof hadoop 0 2021-06-01 12:09 /user/hive/warehouse/beers.db
drwxr-xr-x - jankiewicz_krzysztof hadoop 0 2021-06-01 10:26 /user/hive/warehouse/beers_temp
drwxr-xr-x - jankiewicz_krzysztof hadoop 0 2021-06-01 10:26 /user/hive/warehouse/breweries_temp
```

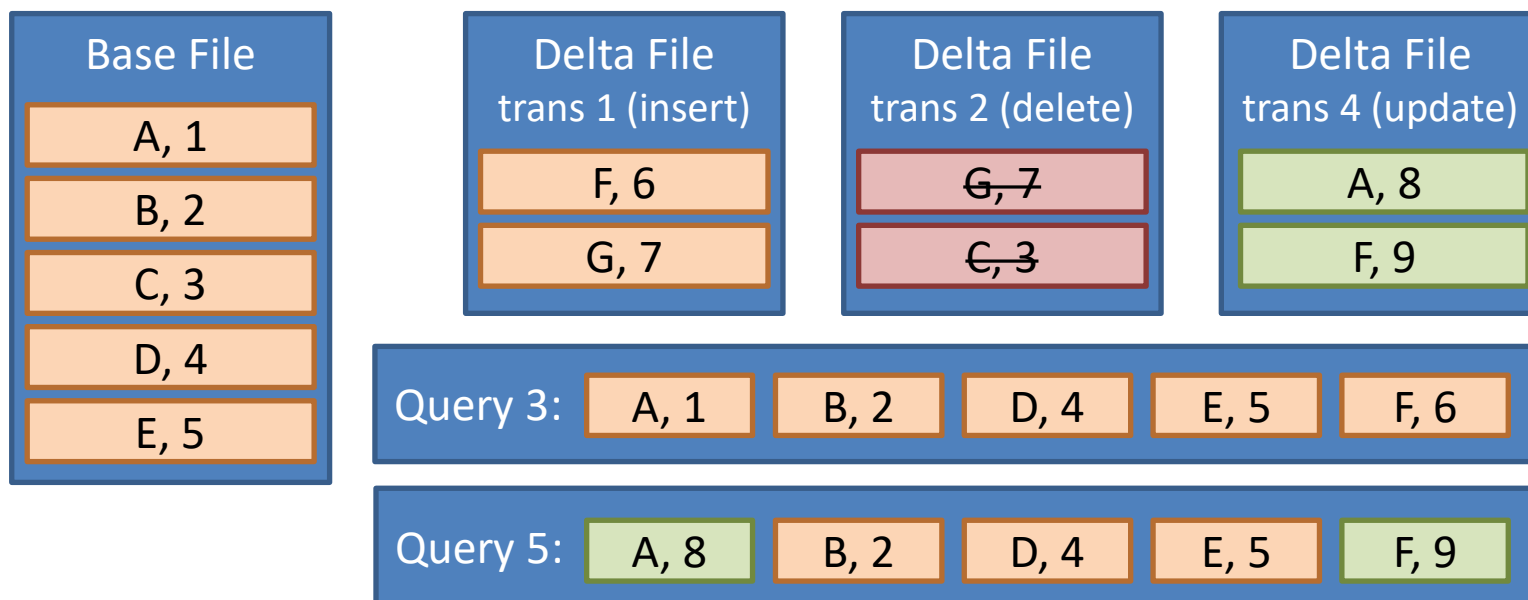
```
jankiewicz_krzysztof@cluster-0999-m:~$ hadoop fs -ls /user/hive/warehouse/beers.db/beers_orc
```

```
Found 1 items
```

```
-rw-r--r-- (. . .) /user/hive/warehouse/beers.db/beers_orc/000000_0
```

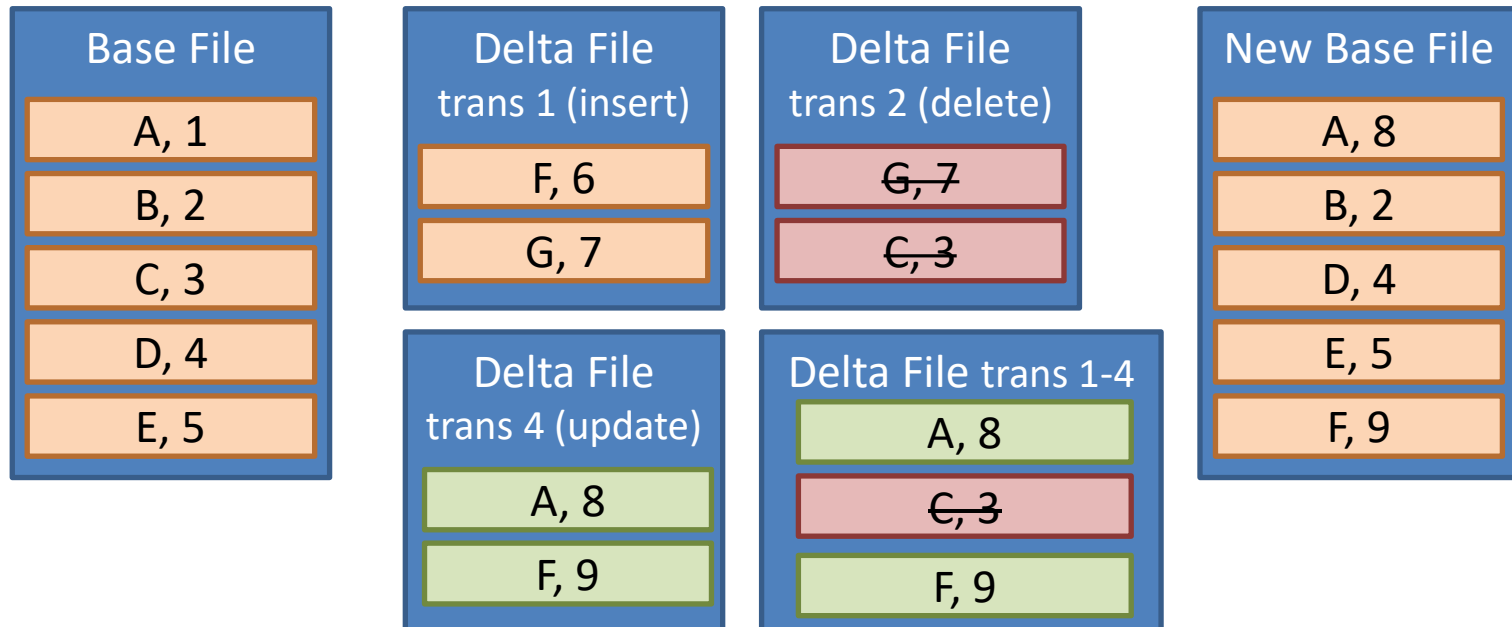
Transakcje w Hive – rozwiązanie

- HDFS nie obsługuje
 - zmiany w środku plików – możemy pisać tylko na końcu pliku
 - spójności, gdy pliki są zmieniane i odczytywane jednocześnie przez dwa różne procesy
- Hive, aby umożliwić powyższe właściwości
 - przechowuje dane tabeli w plikach bazowych
 - nowe, zmodyfikowane i usunięte wiersze są przechowywane w plikach delta
 - dla każdej operacji/transakcji DML tworzone są oddzielne pliki delta dla każdej zmienionej tabeli/partycji/zasobnika.
 - podczas odczytu łączone są dane z obu typów plików



Transakcje w Hive – aktorzy

- Aktorzy:
 - Compactor – zbiór procesów drugoplanowych (np.: Initiator, Worker, Cleaner, AcidHouseKeeperService) odpowiedzialnych za łączenie danych z plików bazowych i plików zmian (SHOW COMPACTIONS)
 - Minor compaction dla każdego kubetka łączy pliki zmian w pojedynczy plik zmian.
 - Major compaction łączy pliki zmian i plik bazowy w nowy plik bazowy.
 - Transaction/Lock Manager – zarządza blokadami (SHOW LOCKS)
- Domyślnie Hive jest skonfigurowany bez obsługi transakcji



Scalanie (compaction)

- Tabele bez obsługi ACID

| Plik | Zawartość |
|---------|-----------|
| 00000_0 | Kubetek 0 |
| 00001_0 | Kubetek 1 |

- Tabele z obsługą ACID

| Plik | Zawartość |
|---|---|
| 00000_0 | Kubetek 0 – plik bazowy |
| 00001_0 | Kubetek 1 – plik bazowy |
| delta_0000005_0000005/bucket_00000 ... delta_00000010_00000010/bucket_00000 | Transakcje: 5,...,10 kubetek 0 – pliki zmian |

- Minor compaction

| Plik | Zawartość |
|-------------------------------------|--|
| 00000_0 | Kubetek 0 – plik bazowy |
| 00001_0 | Kubetek 1 – plik bazowy |
| delta_0000005_00000010/bucket_00000 | Transakcje od 5 do 10, kubetek 0 – plik zmian |

- Major compaction

| Plik | Zawartość |
|---------------------------|---|
| base_0000010/bucket_00000 | Transakcje do 10, kubetek 0 – plik bazowy |
| 00001_0 | Kubetek 1 – plik bazowy |

DML

```
CREATE TABLE table_with_DML_and_ACID(  
  id          int,  
  name        string  
)  
CLUSTERED BY (id) INTO 2 BUCKETS STORED AS ORC  
TBLPROPERTIES ("transactional"="true",  
  "compactor.mapreduce.map.memory.mb"="2048",  
  "compactorthreshold.hive.compactor.delta.num.threshold"="4",  -- (1)  
  "compactorthreshold.hive.compactor.delta.pct.threshold"="0.5" -- (2)  
);
```

1. uruchamia łączenie typu minor compaction jeśli pojawią się więcej niż 4 katalogi plików zmian
2. uruchamia łączenie typu major compaction jeśli stosunek rozmiaru plików zmian do plików bazowych osiągnie wartość większą niż 50%

```
INSERT INTO table_with_DML_and_ACID(id, name) values (1, 'one');
```

```
UPDATE table_with_DML_and_ACID  
SET    name = 'two'  
WHERE  id = 1;
```

```
MERGE INTO table_with_DML_and_ACID AS T USING new_and_old_data AS S ON T.id = S.id  
WHEN MATCHED AND S.flag = 'U' THEN UPDATE SET T.name = S.name  
WHEN MATCHED AND S.flag = 'D' THEN DELETE  
WHEN NOT MATCHED THEN INSERT VALUES (S.id, S.name);
```

```
DELETE FROM table_with_DML_and_ACID;
```

DDL

- Tabele z partycjami i kubetkami
 - **partycje statyczne** – partycje są określane przez polecenie w czasie kompilacji
 - **partycje dynamiczne** – partycje są określane przez wartość kolumny w czasie wykonywania
- Istnieje wiele poleceń DDL związanych z partycjami, np.:
 - dodawanie / usuwanie partycji
 - wymiana partycji między tabelami
 - łączenie partycji

```
CREATE TABLE flights_orc_part (  
  YEAR int,  
  DAY int ,  
  (. . .)  
  LATE_AIRCRAFT_DELAY int ,  
  WEATHER_DELAY int )  
PARTITIONED BY (MONTH int)  
CLUSTERED BY(ORIGIN_AIRPORT) INTO 32 BUCKETS  
STORED AS ORC;
```

```
INSERT INTO TABLE flights_orc_part PARTITION(MONTH=1)  
select YEAR, DAY, DAY_OF_WEEK, AIRLINE, FLIGHT_NUMBER, TAIL_NUMBER, ORIGIN_AIRPORT,  
  (. . .)  
  SECURITY_DELAY, AIRLINE_DELAY, LATE_AIRCRAFT_DELAY, WEATHER_DELAY  
from flights_ext where MONTH = 1;
```

```
SET hive.exec.dynamic.partition = true;  
SET hive.exec.dynamic.partition.mode = nonstrict;
```

```
INSERT INTO TABLE flights_orc_part PARTITION(MONTH)  
select YEAR, DAY, DAY_OF_WEEK,  
  (. . .)  
  MONTH - THE LAST COLUMN IN QUERY  
from flights_ext where MONTH between 2 and 12;
```

```
jankiewicz_krzysztof@cluster-0999-m:~$ hadoop fs -ls /user/hive/warehouse/beers.db/flights_orc_part  
Found 12 items  
drwxr-xr-x - (...) /user/hive/warehouse/beers.db/flights_orc_part/month=1  
drwxr-xr-x - (...) /user/hive/warehouse/beers.db/flights_orc_part/month=10  
drwxr-xr-x - (...) /user/hive/warehouse/beers.db/flights_orc_part/month=11  
. . .
```

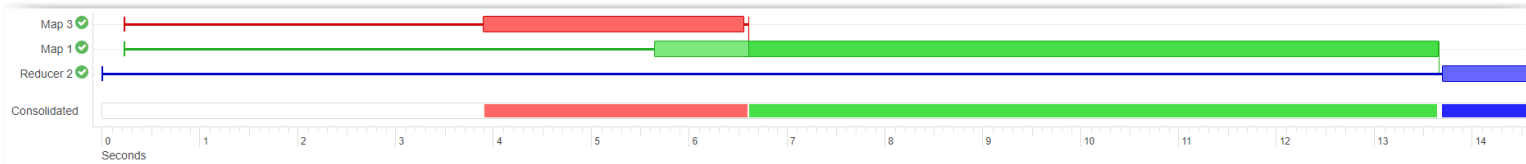
Wykorzystanie partycji i kubełków

```
CREATE TABLE airports_orc_part (  
  IATA_CODE varchar(200),  
  (. . .)  
  STATE varchar(200),  
  LATITUDE double,  
  LONGITUDE double)  
CLUSTERED BY(IATA_CODE) INTO 32 BUCKETS  
STORED AS ORC;
```

```
select a.STATE, sum(f.CANCELLED)/count(*) percent_cancelled  
from   flights_orc f join airports_orc a on (f.ORIGIN_AIRPORT = a.IATA_CODE)  
where  f.MONTH = 6  
group by a.STATE
```

HDFS_BYTES_READ

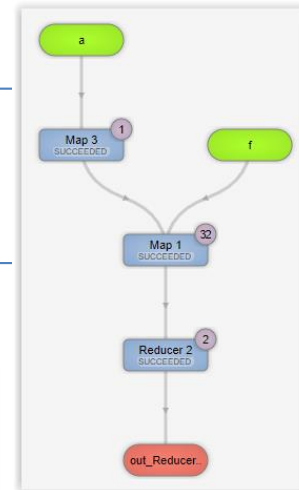
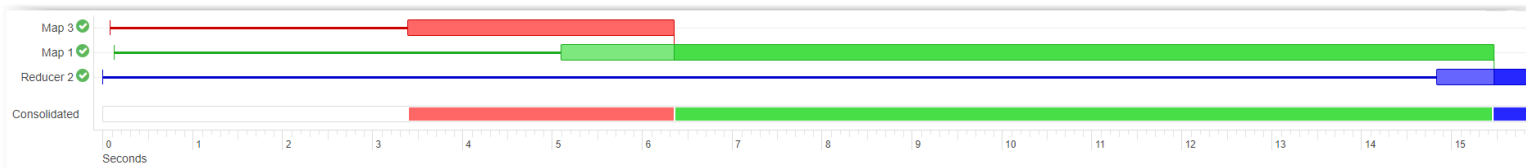
6209187



```
select a.STATE, sum(f.CANCELLED)/count(*) percent_cancelled  
from   flights_orc_part f join airports_orc_part a on (f.ORIGIN_AIRPORT = a.IATA_CODE)  
where  f.MONTH = 6  
group by a.STATE
```

HDFS_BYTES_READ

735575



Perspektywy, indeksy i perspektywy materializowane

```
CREATE OR REPLACE VIEW cancelation_by_state as
select a.STATE, sum(f.CANCELLED)/count(*) percent_cancelled
from   flights_orc_part f join airports_orc_part a on (f.ORIGIN_AIRPORT = a.IATA_CODE)
where  f.MONTH = 6
group by a.STATE
```

- Wysoką wydajność przetwarzania zapytań w Hive można uzyskać
 - wykorzystując składowanie kolumnowe
 - partycjonowanie danych
 - kubetki
- Indeksy Hive zostały usunięte w wersji 3.0
 - używaj materializowanych perspektyw z przepisywaniem zapytań
 - używaj formatów plików kolumnowych (Parquet, ORC) – mogą wykonywać skanowanie selektywne (pliki lub bloki można pominąć)
- Perspektywy materializowane zostały wprowadzone w Apache Hive 3.0.0
 - Własności
 - mogą być używane w przepisywaniu zapytań
 - muszą być ręcznie odświeżane po zmianach w tabelach źródłowych (ALTER ... REBUILD)
 - odświeżanie przyrostowe jest dostępne po operacjach INSERT
 - UPDATE i DELETE wymuszają pełną przebudowę MV
 - domyślnie nieaktualne dane w MV nie są używane do przepisywania zapytań, możemy zmienić to zachowanie, aby wykorzystać dane, które są nieaktualne

```
set hive.materializedview.rewriting.time.window=10min;
```

Rozszerzenia grupowania

- Jak na platformę hurtowni danych przysłało, Hive udostępnia rozszerzenia grupowania
 - CUBE np.:
GROUP BY a, b, c WITH CUBE
 - ROLLUP np.:
GROUP BY a, b, c WITH ROLLUP
 - GROUPING SETS np.:
GROUP BY a, b GROUPING SETS ((a, b), a, b, ())
- Funkcja GROUPING__ID zwraca wartość bitvector, określa ona to które z elementów grupowania występują w grupie, a które nie

```
select beers_orc.style, breweries_orc.state, round(avg(beers_orc.ibu)) avg_ibu, GROUPING__ID
from   breweries_orc join beers_orc on (breweries_orc.id = beers_orc.brewery_id)
where  beers_orc.ibu is not null
and    (breweries_orc.state in ('OR','AL') or breweries_orc.state is null)
and    (beers_orc.style in ('American IPA','American Pale Ale (APA)') or beers_orc.style is null)
group by beers_orc.style, breweries_orc.state WITH ROLLUP
order by breweries_orc.state
```

| beers_orc.style | breweries_orc.state | avg_ibu | grouping__id |
|-------------------------|---------------------|---------|--------------|
| null | null | 68.0 | 3 |
| American IPA | null | 80.0 | 1 |
| American Pale Ale (APA) | null | 48.0 | 1 |
| American IPA | AL | 67.0 | 0 |
| American Pale Ale (APA) | AL | 40.0 | 0 |
| American IPA | OR | 81.0 | 0 |
| American Pale Ale (APA) | OR | 50.0 | 0 |

Funkcje analityczne

Obsługiwane przez Hive funkcje analityczne obejmują ich szeroki zakres:

- Funkcje rankingu
 - RANK
 - ROW_NUMBER
 - DENSE_RANK
 - CUME_DIST
 - PERCENT_RANK
 - NTILE
- Funkcje okna
 - LEAD
 - LAG
 - FIRST_VALUE
 - LAST_VALUE

- Funkcje raportujące

- COUNT
- SUM
- MIN
- MAX
- AVG

| | | | | |
|-----------------------|-----------|------|---|------|
| East Asia & Pacific | Indonesia | 75.0 | 1 | 30.0 |
| East Asia & Pacific | China | 63.0 | 2 | 25.0 |
| East Asia & Pacific | Vietnam | 31.0 | 3 | 12.0 |
| Europe & Central Asia | Romania | 57.0 | 1 | 26.0 |
| Europe & Central Asia | Ukraine | 56.0 | 2 | 25.0 |
| Europe & Central Asia | Turkey | 48.0 | 3 | 22.0 |

```
select * from (  
  select c.region, i.countryname,  
         round(sum(i.value)/1000000000) val_mld,  
         RANK () OVER (PARTITION BY region  
                       ORDER BY sum(i.value) desc) rank_in_region,  
         round(sum(i.value)/SUM(sum(i.value))  
               OVER (PARTITION BY region)*100) ratio  
  from indicators_part i join country_buck c  
    on (i.countrycode = c.countrycode)  
 where i.year = 2015  
  group by c.region, i.countryname  
 ) t  
where t.rank_in_region <= 3
```

UDF i HPL/SQL

- Do wbudowanych funkcji UDF wlicza się:
 - wbudowane operatory
 - wbudowane funkcje
 - wbudowane funkcje agregujące (UDAF)
 - wbudowane funkcje generujące tabele (UDTF)
 - własne funkcje użytkownika (CREATE FUNCTION)
 - implementowane w Javie
 - rejestrowane w Hive za pomocą plików jar

Patrz:

- SHOW FUNCTIONS;
- <https://cwiki.apache.org/confluence/display/Hive/LanguageManual+UDF>

- Hive od wersji 2.0 implementuje Hybrid Procedural SQL On Hadoop (HPL/SQL) jako proceduralny SQL - <http://www.hp1sql.org/doc>
 - główny cel to implementacja procesów ETL

Dodatkowa funkcjonalność

- Tabele tymczasowe
- Tabele zewnętrzne
- Wektoryzacja przetwarzania

Tabele tymczasowe

- Często złożone przetwarzanie przy wykorzystaniu poleceń SQL wymaga składowania pośrednich etapów
- W Hive mogą do tego celu służyć tabele tymczasowe

```
CREATE TEMPORARY TABLE tmp1 (c1 string);
CREATE TEMPORARY TABLE tmp2 AS ...
CREATE TEMPORARY TABLE tmp3 LIKE ...
```
- Istnieją tylko na poziomie sesji – nie trzeba ich kasować
- Składowane są w przestrzeni roboczej użytkownika (/tmp/hive-username)
- Użytkownicy mogą tworzyć tabele tymczasowe o tych samych nazwach (poziom sesji)
- Nazwy tabel tymczasowych „przykrywają” nazwy tabel trwałych

Tabele zewnętrzne

- Tabele zewnętrzne mogą służyć jako źródła danych do dalszego przetwarzania

```
[maria_dev@sandbox-hdp hive]$ hadoop fs -ls labs/hadoop/hive/beereries
Found 1 items
-rw-r--r--  1 maria_dev hdfs          22504 2018-07-15 15:00 labs/hadoop/hive/beereries/beereries.csv
```

```
CREATE EXTERNAL TABLE IF NOT EXISTS beereries_ext(
    id INT,
    name STRING,
    city STRING,
    state STRING)
COMMENT 'craft beereries'
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS TEXTFILE
location '/user/maria_dev/labs/hadoop/hive/beereries';
```

```
CREATE TABLE IF NOT EXISTS beereries_orc(
    id INT,
    name STRING,
    city STRING,
    state STRING)
COMMENT 'craft beereries'
STORED AS ORC;
```

```
INSERT OVERWRITE TABLE beereries_orc
SELECT * FROM beereries_ext
WHERE id IS NOT NULL;
```

Wektoryzacja przetwarzania zapytań

- Standardowo wykonywanie zapytań przetwarza krotki pojedynczo
- Wektoryzacja pozwala przetwarzać jednocześnie całe tablice krotek – wektory kolumn
- W wersji 1.2 w ramach dystrybucji Hortonworks wektoryzacja wspiera
 - operacje takie jak: projekcji, selekcji oraz grupowania
 - również tabele partycjonowane
 - cały zestaw typów prostych (tinyint, smallint, int, bigint, date, boolean, float, double, timestamp, string, char, varchar, binary)
 - szereg wyrażeń i agregatów
 - porównań: >, >=, <, <=, =, !=
 - arytmetycznych: plus, minus, multiply, divide, modulo
 - logicznych: AND, OR
 - agregatów: sum, avg, count, min, max
 - tylko **zapytania**,
 - tylko tabele w formacie **ORC**

Podsumowanie

- Organizacja danych
- Typy danych
- Formaty przechowywania danych
 - Format wierszowy vs kolumnowy
 - ORCFile
- Hive SQL
 - Ładowanie
 - Selekcja i projekcja
 - DML, transakcje w Hive, łączenie (konkatenacja)
 - Rozszerzenia grupowania i funkcje analityczne
- Silniki wykonawcze