Hive - Part 1

- Abhay Dandekar

Agenda

Agenda continued...

1. SQL v/s HiveQL
2. HiveQL DataTypes - Simple
3. HiveQL DataTypes - Complex
4. Hive Table
5. Managed Table
6. External Table
7. Partitions and Bucketing - Concept
8. Partitions
9. Bucketing

Why was Hive created?

1. Complex java programming
2. Conversion of SQL to MR was complex but repetitive.
3. SQL is lingua franca of Analytics community
4. A hadoop based Datawarehouse

What is a Data-"***Warehouse***"?

Alternatives to Hive

1. Cloudera Impala
2. Presto from Facebook
3. Apache Drill
4. Spark SQL
5. Apache Phoenix

What is Hive?

1. Created by Facebook
2. DWH package built on top of Hadoop
3. Majorly used for data analytics
4. Targeted for users comfortable with SQL
5. It has HiveQL instead on SQL. Similar to SQL.

What is Hive? - continued

1. For managing and querying structured data
2. Abstracts complexity of Hadoop
3. No need to learn Java and hadoop API
4. Known for batch processing
5. Provides a means to attach a structure to your data in HDFS
6. Has Metadata.

Hive v/s RDBMS

1. Schema on read v/s Schema on write
2. Updates, deletes
   a. Managed via delta files
   b. Delta files are periodically merged into the base table by MR Jobs
3. Locking
   a. Managed via zookeeper
   b. Locks can be seen using “SHOW LOCKS” statement on hive prompt

Hive Architecture

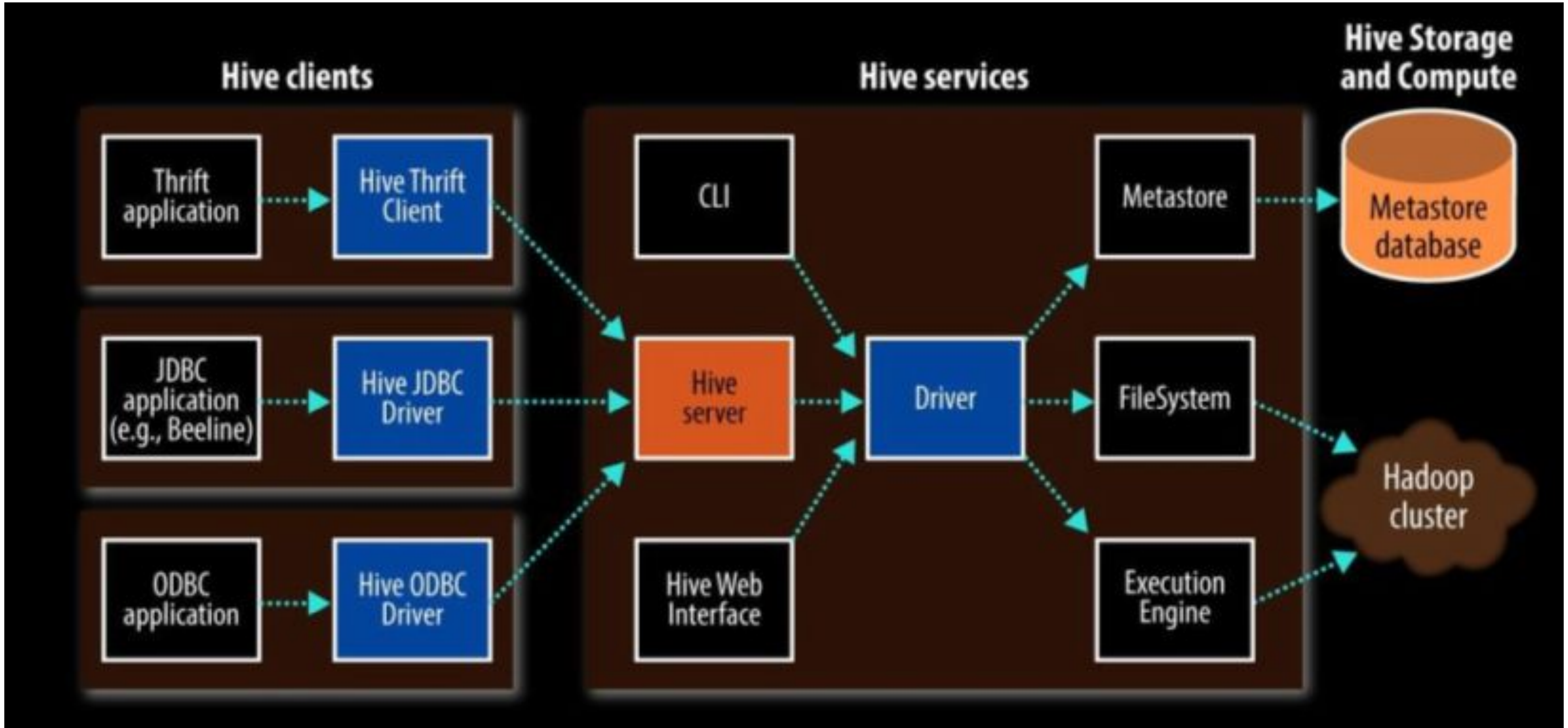


Figure 17-1. Hive architecture

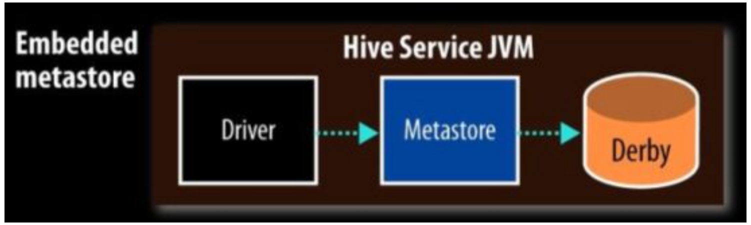
Execution engines

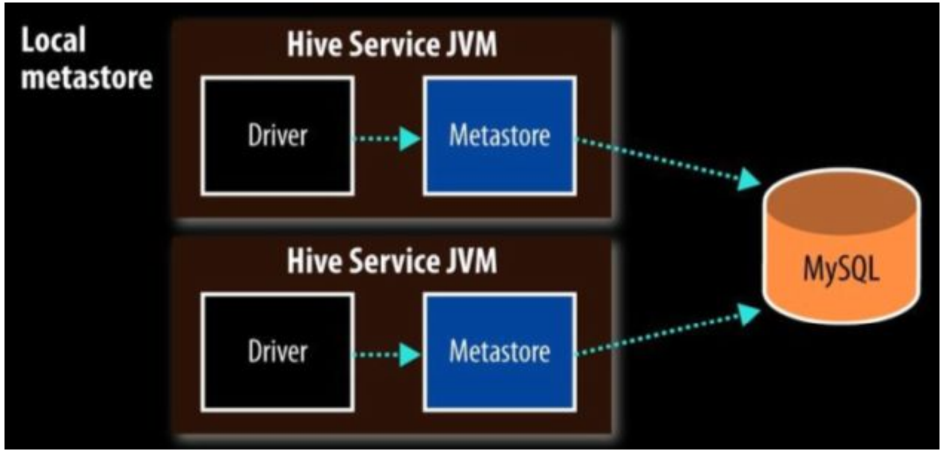
1. Hive on MapReduce
    a. SET hive.execution.engine=mr;
2. Hive on Tez
    a. SET hive.execution.engine=tez;
3. Hive on Spark
    a. SET hive.execution.engine=spark;
    b. Link : https://issues.apache.org/jira/secure/attachment/12652517/Hive-on-Spark.pdf

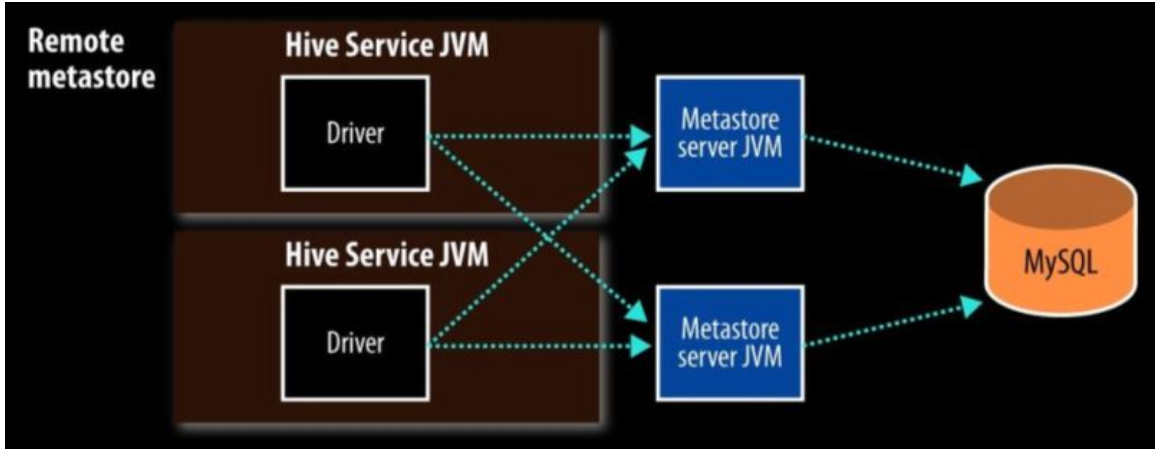
The metastore

1. Central repository for Hive metadata
2. Types of metastore
    a. Embedded metastore - Derby
    b. Local metastore - RDBMS
3. With Embedded metastore, we can only start one session
4. With Local metastore, we can access multiple sessions

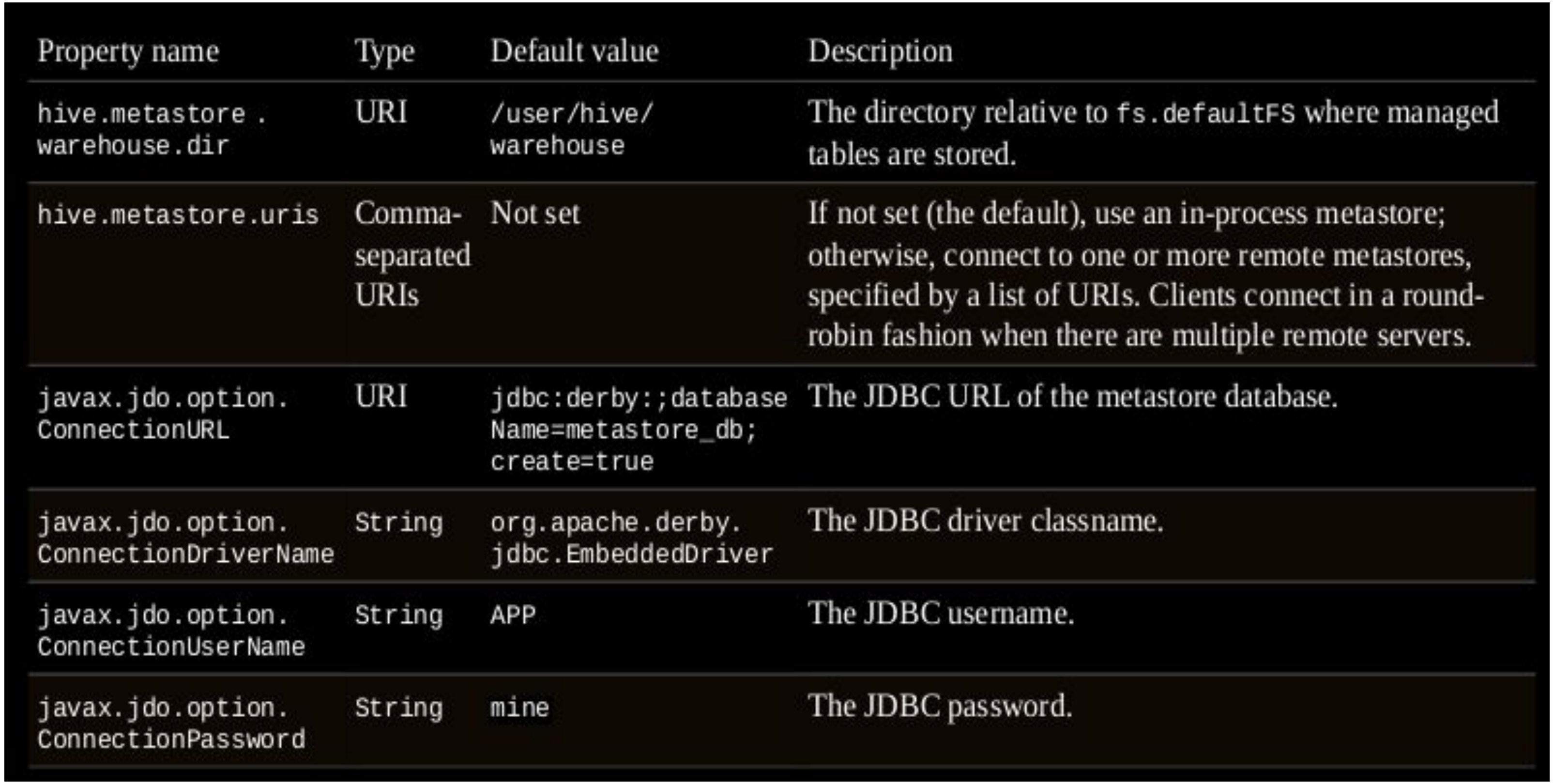
Metastore configurations







Hive Metastore properties



Property name	Type	Default value	Description
`hive.metastore.warehouse.dir`	URI	`/user/hive/warehouse`	The directory relative to `fs.defaultFS` where managed tables are stored.
`hive.metastore.uris`	Comma-separated URIs	Not set	If not set (the default), use an in-process metastore; otherwise, connect to one or more remote metastores, specified by a list of URIs. Clients connect in a round-robin fashion when there are multiple remote servers.
`javax.jdo.option.ConnectionURL`	URI	`jdbc:derby:;databaseName=metastore_db;create=true`	The JDBC URL of the metastore database.
`javax.jdo.option.ConnectionDriverName`	`String`	`org.apache.derby.jdbc.EmbeddedDriver`	The JDBC driver classname.
`javax.jdo.option.ConnectionUserName`	`String`	`APP`	The JDBC username.
`javax.jdo.option.ConnectionPassword`	`String`	`mine`	The JDBC password.

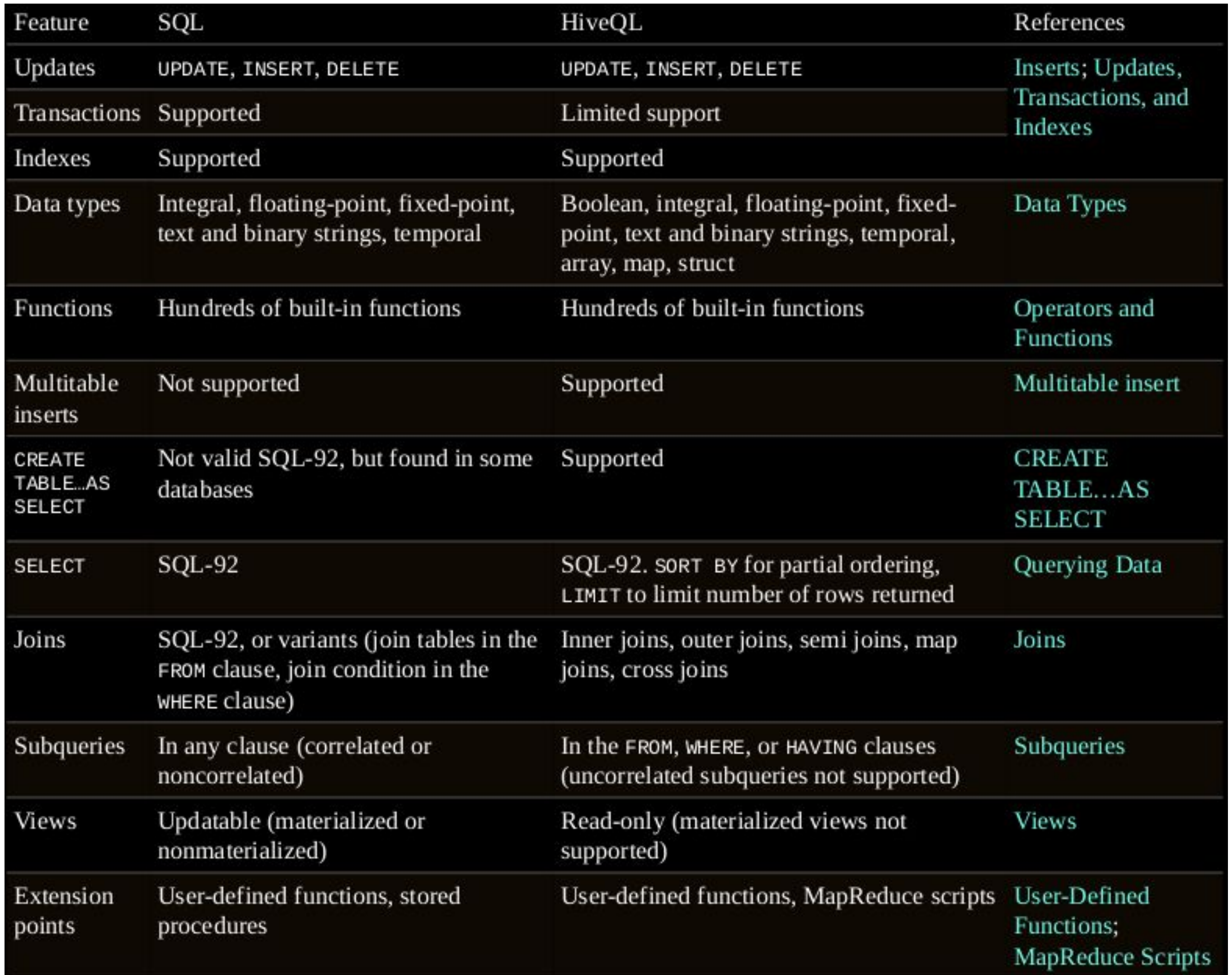
Services provided by Hive

1. CLI a.k.a hive-shell.
2. Hiveserver2
   a. Runs hive as a server exposed via a Thrift service
3. Beeline
   a. A command line interface to Hive
4. Hwi
   a. Hive Web Interface
5. Jar
   a. Hive equivalent to hadoop jar
6. Drivers JDBC and ODBC

Hive-shell

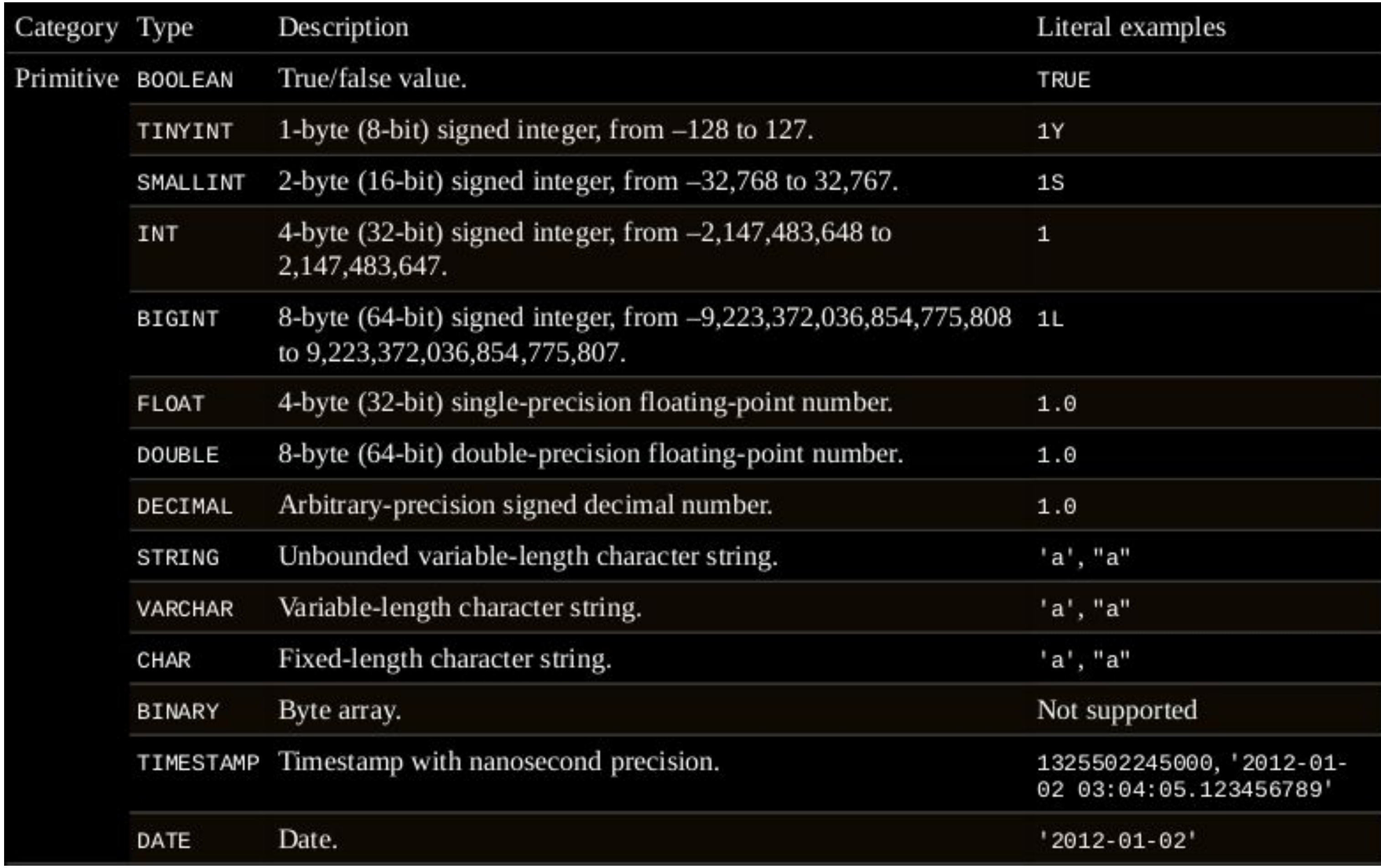
1. Primary way to connect with Hive
2. Uses HiveQL
3. Influenced by MySQL

SQL v/s HiveQL



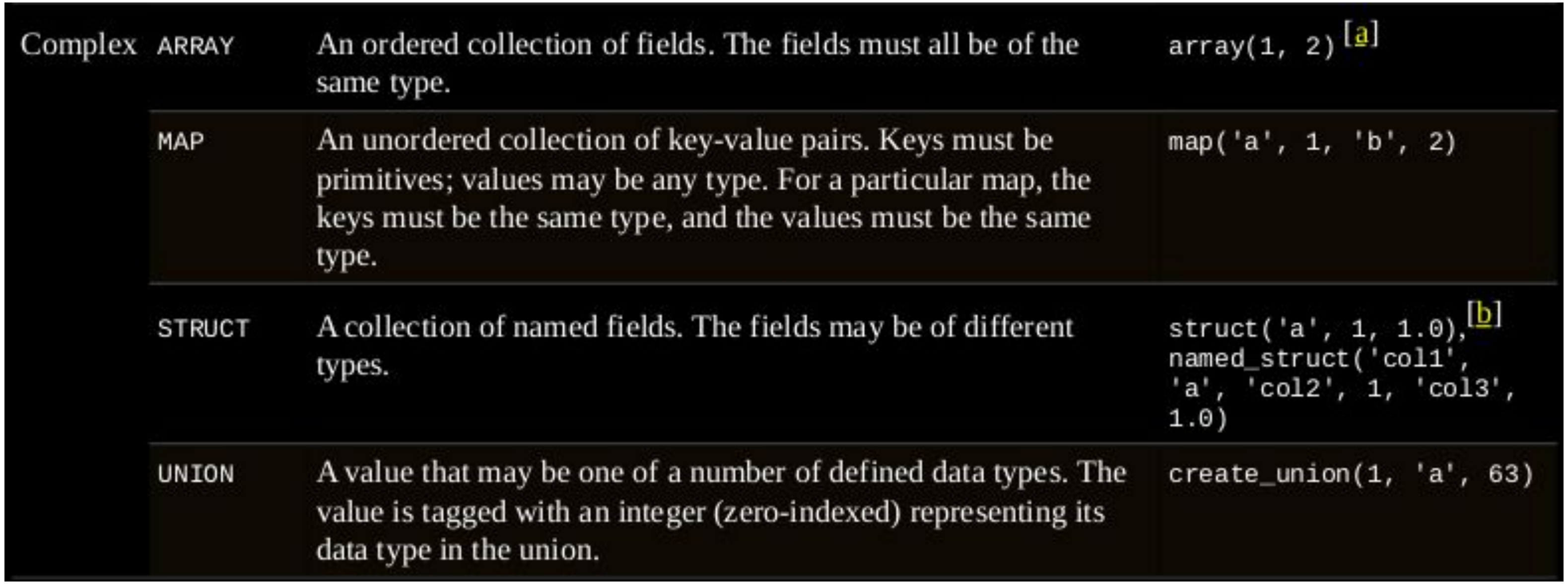
Feature	SQL	HiveQL	References
Updates	`UPDATE`, `INSERT`, `DELETE`	`UPDATE`, `INSERT`, `DELETE`	Inserts; Updates, Transactions, and Indexes
Transactions	Supported	Limited support	
Indexes	Supported	Supported	
Data types	Integral, floating-point, fixed-point, text and binary strings, temporal	Boolean, integral, floating-point, fixed-point, text and binary strings, temporal, array, map, struct	Data Types
Functions	Hundreds of built-in functions	Hundreds of built-in functions	Operators and Functions
Multitable inserts	Not supported	Supported	Multitable insert
`CREATE TABLE...AS SELECT`	Not valid SQL-92, but found in some databases	Supported	CREATE TABLE...AS SELECT
`SELECT`	SQL-92	SQL-92. `SORT BY` for partial ordering, `LIMIT` to limit number of rows returned	Querying Data
Joins	SQL-92, or variants (join tables in the `FROM` clause, join condition in the `WHERE` clause)	Inner joins, outer joins, semi joins, map joins, cross joins	Joins
Subqueries	In any clause (correlated or noncorrelated)	In the `FROM`, `WHERE`, or `HAVING` clauses (uncorrelated subqueries not supported)	Subqueries
Views	Updatable (materialized or nonmaterialized)	Read-only (materialized views not supported)	Views
Extension points	User-defined functions, stored procedures	User-defined functions, MapReduce scripts	User-Defined Functions; MapReduce Scripts

HiveQL Data-types (Simple)



Category	Type	Description	Literal examples
Primitive	`BOOLEAN`	True/false value.	`TRUE`
	`TINYINT`	1-byte (8-bit) signed integer, from –128 to 127.	`1Y`
	`SMALLINT`	2-byte (16-bit) signed integer, from –32,768 to 32,767.	`1S`
	`INT`	4-byte (32-bit) signed integer, from –2,147,483,648 to 2,147,483,647.	`1`
	`BIGINT`	8-byte (64-bit) signed integer, from –9,223,372,036,854,775,808 to 9,223,372,036,854,775,807.	`1L`
	`FLOAT`	4-byte (32-bit) single-precision floating-point number.	`1.0`
	`DOUBLE`	8-byte (64-bit) double-precision floating-point number.	`1.0`
	`DECIMAL`	Arbitrary-precision signed decimal number.	`1.0`
	`STRING`	Unbounded variable-length character string.	`'a'`, `"a"`
	`VARCHAR`	Variable-length character string.	`'a'`, `"a"`
	`CHAR`	Fixed-length character string.	`'a'`, `"a"`
	`BINARY`	Byte array.	Not supported
	`TIMESTAMP`	Timestamp with nanosecond precision.	`1325502245000`, `'2012-01-02 03:04:05.123456789'`
	`DATE`	Date.	`'2012-01-02'`

HiveQL DataTypes (Complex)



Complex	`ARRAY`	An ordered collection of fields. The fields must all be of the same type.	`array(1, 2)` [a]
	`MAP`	An unordered collection of key-value pairs. Keys must be primitives; values may be any type. For a particular map, the keys must be the same type, and the values must be the same type.	`map('a', 1, 'b', 2)`
	`STRUCT`	A collection of named fields. The fields may be of different types.	`struct('a', 1, 1.0),` [b] `named_struct('col1', 'a', 'col2', 1, 'col3', 1.0)`
	`UNION`	A value that may be one of a number of defined data types. The value is tagged with an integer (zero-indexed) representing its data type in the union.	`create_union(1, 'a', 63)`

Hive Table

1. Hive Table is logically made up of the data being stored and the associated metadata describing the layout.
2. Two types
   a. Managed Tables
   b. External Tables

Managed Tables

1. Hive will manage the data
2. Data will be in warehouse dir
3. Example
    a. CREATE TABLE my_managed_table (column1 STRING);
       LOAD DATA INPATH ‘/user/cloudera/data.txt’ INTO TABLE my_managed_table;
    b. This will move /user/cloudera/data.txt into warehouse dir
4. DROP will delete the meta as well as the data

External Tables

1. User controls the creation and deletion of data
2. Location is specified at table create time
3. Example:
    a. CREATE **EXTERNAL TABLE** my_external_table (column1 STRING) LOCATION ‘/user/cloudera/external_table;
    b. LOAD DATA INPATH ‘/user/cloudera/mydata.txt’ INTO TABLE external_table;
4. Hive does not move it into warehouse dir
5. It does not checks if the data exists or not. Helpful in lazy data creation.
6. DROP TABLE will delete only the metadata. Actual data will remain intact.

Partitions and Buckets

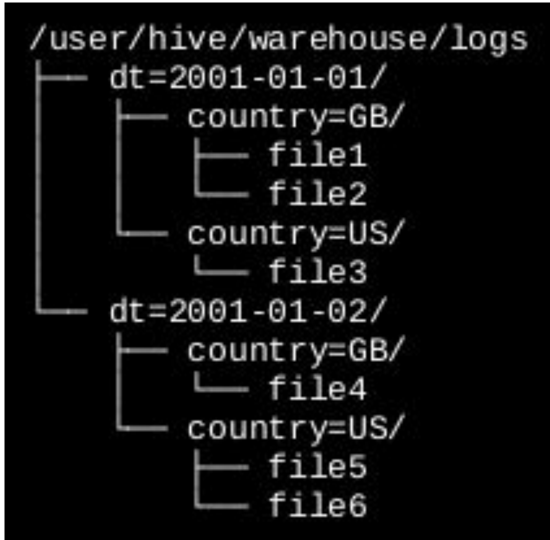
1. Hive organizes data into partitions
2. Data is sliced, so partitions will help fasten the query
3. Date may be a good partition column
4. Tables or Partitions can be further sub-divided into buckets
5. Partitioning and bucketing are ways for data management

Partitions

1. Data is partitioned upon columns
2. Data may be partitioned on multiple dimensions
3. At filesystem, partitions are simply nested dirs
4. The columns over which we partition are called as **partition columns,** and they basically do not exist into the data, but exist onto the data-path location

Partitions - Example

1. CREATE TABLE logs (ts BIGINT, line STRING)
   PARTITIONED BY (**dt** STRING, **country** STRING);
2. LOAD DATA LOCAL INPATH ‘input/hive/partitions/file1’
   INTO TABLE logs
   PARTITION (dt=’2001-01-01’, country =’GB’)
3. To view partitions, SHOW PARTITION logs;
4. SELECT ts, dt, line
   FROM logs
   WHERE country=’US’



```
/user/hive/warehouse/logs
├── dt=2001-01-01/
│   ├── country=GB/
│   │   ├── file1
│   │   └── file2
│   └── country=US/
│       └── file3
└── dt=2001-01-02/
    ├── country=GB/
    │   └── file4
    └── country=US/
        ├── file5
        └── file6
```

Bucketing

1. Bucketing imposes extra structure on table
2. Two reasons for Bucketing
   a. Enable efficient queries esp Join
   b. Make sampling more efficient
3. Use CLUSTERED BY clause
4. Physically, each bucket is just a file in HDFS

Bucketing - Example

1. CREATE TABLE bucketed_users (id INT, name STRING)
   CLUSTERED BY (id) INTO 4 BUCKETS;
2. Data will be hashed on id and kept into 4 buckets.
3. Data may be SORTED and kept, this will improve efficiency of join queries
4. How can we ensure our data is bucketed? Let hive do it for you.
5. INSERT OVERWRITE bucketed_users
   SELECT * from users;
6. dfs -ls /user/hive/warehouse/bucketed_users;

```
dfs     -ls     /user/hive/warehouse/bucketed_users;
        000000_0
        000001_0
        000002_0
        000003_0
```

To be continued …

Storage Formats

1. Two types of formats
   a. Row format
   b. File format
2. Row format dictates how rows in a particular field are stored
3. File format dictates the container formats for particular fields in a row
   a. Plain-text (default)
   b. Row oriented
   c. Column oriented

Binary Storage formats

1. Storage Formats
    a. Sequence files
    b. Avro datafiles
    c. Parquet files
    d. RCFiles
    e. ORCFiles
2. To change the format, use STORED AS clause
3. Default Storage format is Textfile

Default delimiters

1. Default row delimiter - Ctrl-A (^A)
2. Default collection item delimiter - Ctrl-B (^B)
3. Default map keys delimiter - Ctrl-C (^C)
4. Hence, CREATE TABLE … translates to
    a.

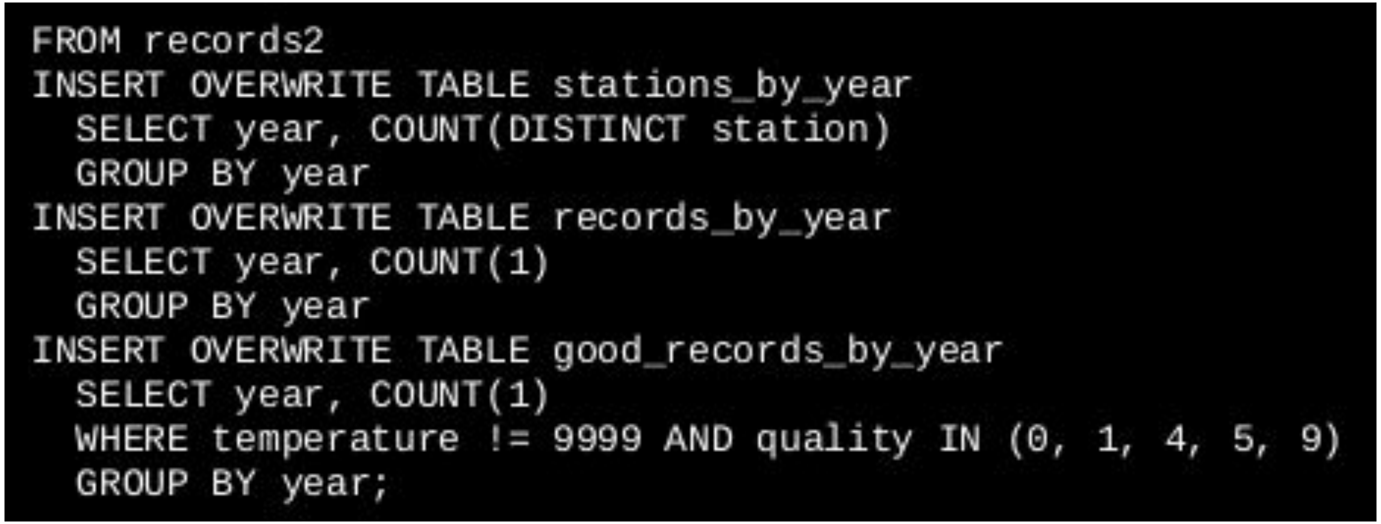
```
CREATE TABLE …
ROW FORMAT DELIMITED
        FIELDS TERMINATED BY ‘\001’
        COLLECTION ITEMS TERMINATED BY ‘\002’
        MAP KEYS TERMINATED BY ‘\003’
        LINES TERMINATED BY ‘\n’
STORED AS TEXTFILE
```

Importing data

1. CTAS
    a. INSERT OVERWRITE TABLE ‘table_name’
       SELECT col1, col2 FROM source_table;
2. Using static partition
    a. INSERT OVERWRITE TABLE ‘table_name’
       PARTITION (dt=’2001-01-01’)
       SELECT col1, col2 FROM source_table;
3. Using dynamic partition insert
    a. INSERT OVERWRITE TABLE ‘table_name’
       PARTITION (**dt**)
       SELECT col1, col2, **dt** FROM source_table;

Multi-table insert

1. We can interchange the INSERT and FROM statement in HiveQL
2. We can insert into multiple table in a single query



```
FROM records2
INSERT OVERWRITE TABLE stations_by_year
  SELECT year, COUNT(DISTINCT station)
  GROUP BY year
INSERT OVERWRITE TABLE records_by_year
  SELECT year, COUNT(1)
  GROUP BY year
INSERT OVERWRITE TABLE good_records_by_year
  SELECT year, COUNT(1)
  WHERE temperature != 9999 AND quality IN (0, 1, 4, 5, 9)
  GROUP BY year;
```

Alter Tables

1. Hive has schema on read.
2. Hence, its flexible on changes
3. E.G

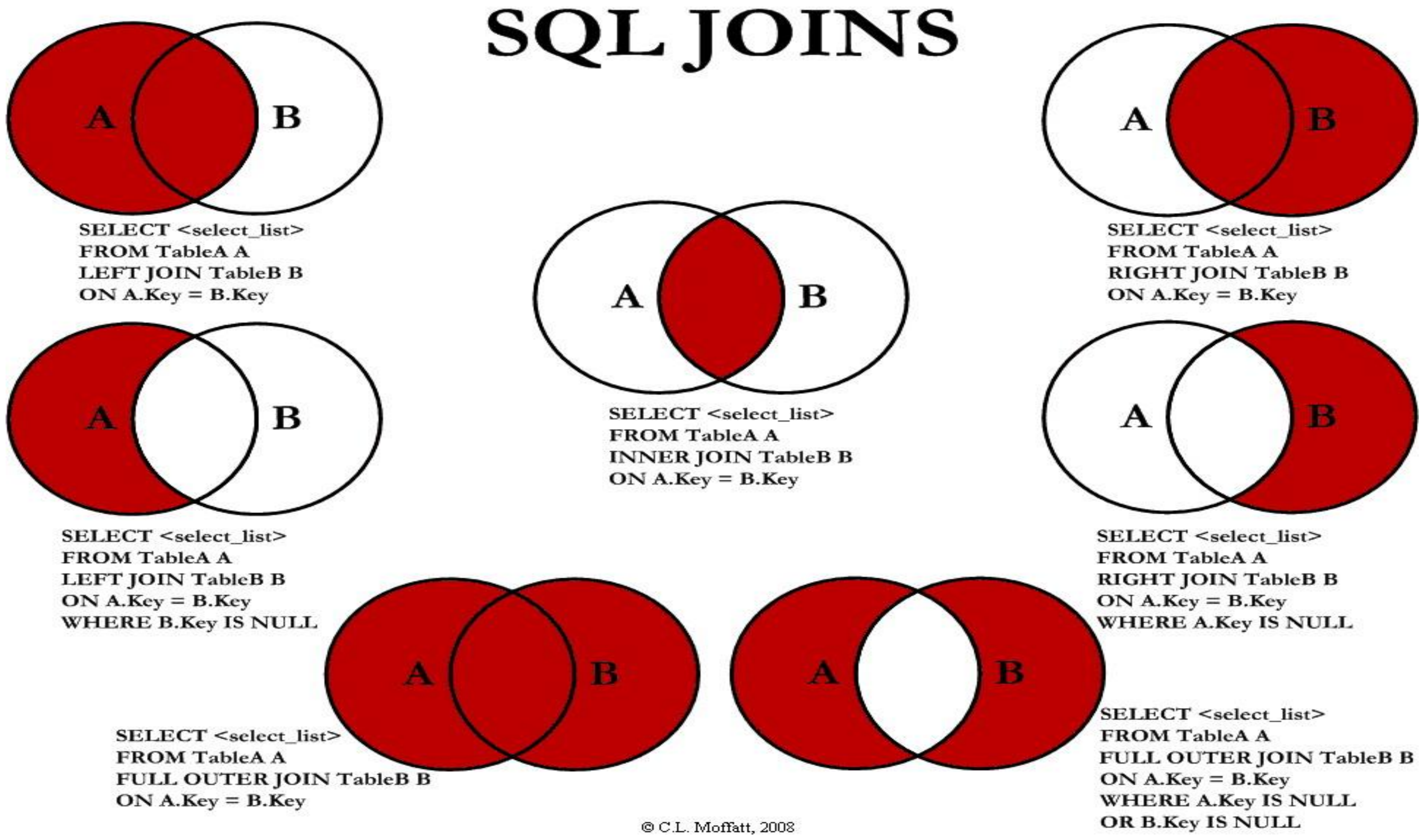
```
ALTER TABLE source RENAME TO target;

ALTER TABLE target ADD COLUMNS (myColumn INT)
```

Drop Table

1. Deletes the data and metadata for a table
2. For external tables, only metadata is removed
3. We can use TRUNCATE to drop only the data and keep metadata intact.

SQL JOINS



SQL JOINS

```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
```

```
SELECT <select_list>
FROM TableA A
INNER JOIN TableB B
ON A.Key = B.Key
```

```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
```

```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
WHERE B.Key IS NULL
```

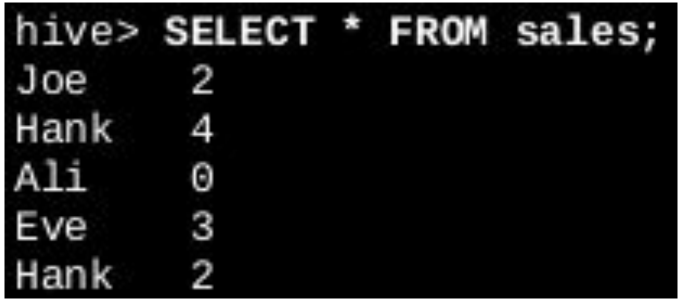
```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
```

```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
```

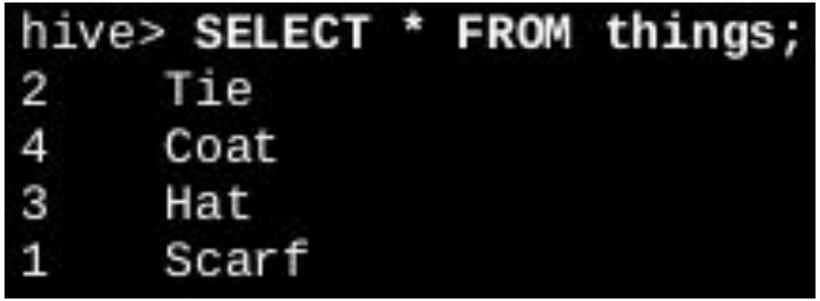
```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
OR B.Key IS NULL
```

Querying data - Joins

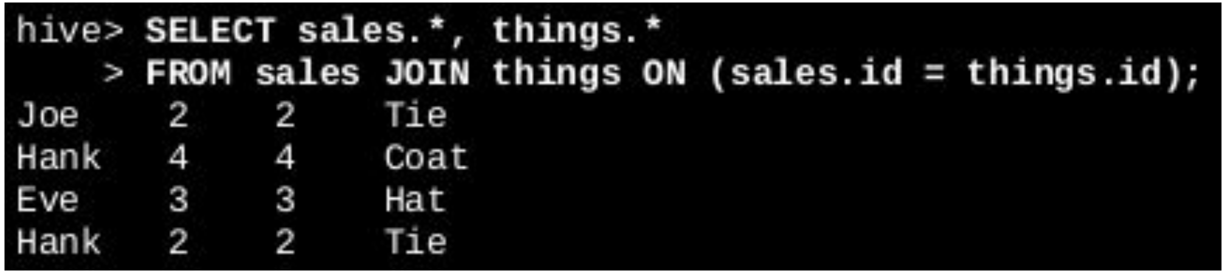
1. Inner Join
2. EXPLAIN helps in understanding the query plan



```
hive> SELECT * FROM sales;
Joe     2
Hank    4
Ali     0
Eve     3
Hank    2
```



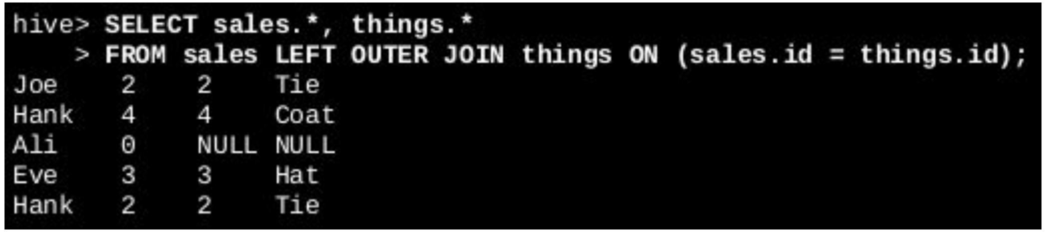
```
hive> SELECT * FROM things;
2     Tie
4     Coat
3     Hat
1     Scarf
```



```
hive> SELECT sales.*, things.*
    > FROM sales JOIN things ON (sales.id = things.id);
Joe     2     2     Tie
Hank    4     4     Coat
Eve     3     3     Hat
Hank    2     2     Tie
```

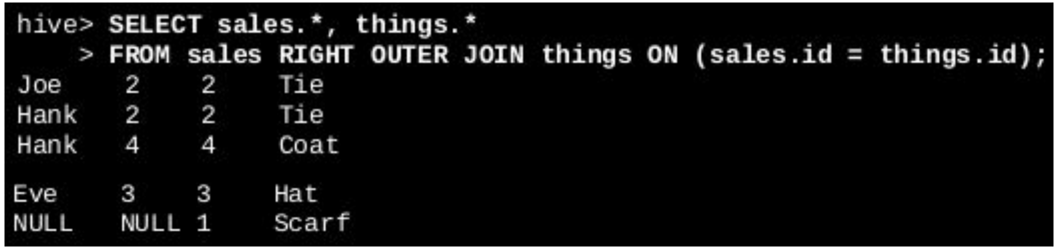
Outer joins

1. Left Outer -



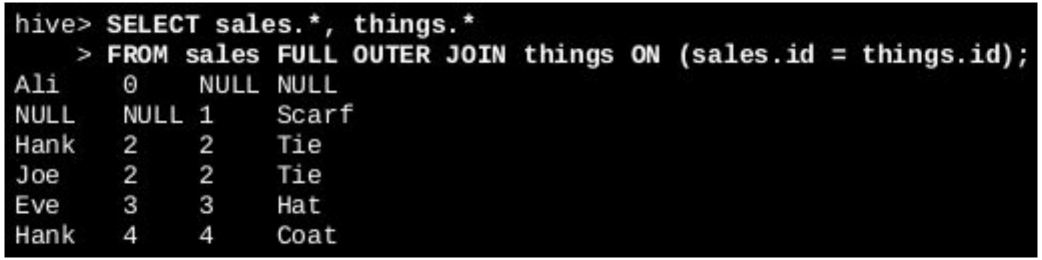
```
hive> SELECT sales.*, things.*
    > FROM sales LEFT OUTER JOIN things ON (sales.id = things.id);
Joe     2    2    Tie
Hank    4    4    Coat
Ali     0    NULL NULL
Eve     3    3    Hat
Hank    2    2    Tie
```

2. Right Outer -



```
hive> SELECT sales.*, things.*
    > FROM sales RIGHT OUTER JOIN things ON (sales.id = things.id);
Joe     2    2    Tie
Hank    2    2    Tie
Hank    4    4    Coat
Eve     3    3    Hat
NULL    NULL 1    Scarf
```

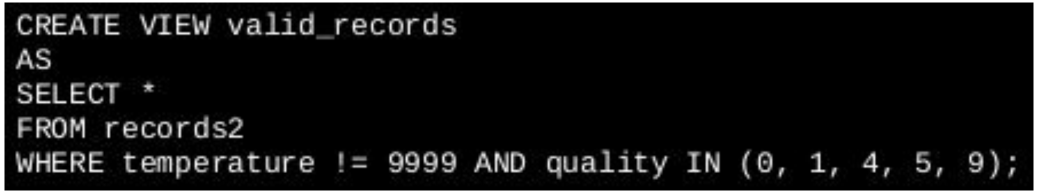
3. Full Outer -



```
hive> SELECT sales.*, things.*
    > FROM sales FULL OUTER JOIN things ON (sales.id = things.id);
Ali     0    NULL NULL
NULL    NULL 1    Scarf
Hank    2    2    Tie
Joe     2    2    Tie
Eve     3    3    Hat
Hank    4    4    Coat
```

Views

1. Views are virtual tables defined by SELECT



```
CREATE VIEW valid_records
AS
SELECT *
FROM records2
WHERE temperature != 9999 AND quality IN (0, 1, 4, 5, 9);
```

2. We cannot update / insert in underlying table via a view.

UDF - What are UDFs

1. Full form of UDF?
2. UDFs can be classified as
   a. UDF : Regular User Defined Functions
      i. Operates on a single row and produces single row as output
   b. UDAF: User Defined Aggregate Functions
      i. Operates on a multiple rows and produces single row as output
   c. UDTF: User Defined Table Functions
      i. Operates on a single row and produces multiple row as output

Why are UDFs needed?

1. Plug in your own logic
2. Use when the built-in hive functions are not able to get the desired result

How are UDFs implemented - 1 (Regular UDF)

1. Extend org.apache.hadoop.hive.ql.exec.UDF
2. Implement evaluate() method

```
import org.apache.hadoop.hive.ql.exec.UDF;
import org.apache.hadoop.io.Text;

public final class MyUpper extends UDF {

        public Text evaluate(final Text s) {
                if (s == null) {
                        return null;
                }
                return new Text(s.toString().toUpperCase());
        }
}
```

How to install and use UDFs

1. Create a jar and load it into HDFS, say /user/hive/UDFs
2. Login to beeline and connect to your cluster
    a. !connect jdbc:hive2://localhost:10000
3. Create your UDF
    a. **create function myupper**
       as 'org.cdac.hive.udf.MyUpper'
       USING JAR
       'hdfs://localhost:9000/user/hive/UDFs/hive.udf-0.0.1-SNAPSHOT.jar';
4. Use it in SQL
    a. **select myupper('data') from ext_loaded;**

How are UDAFs implemented

1. Aggregation is required across nodes
2. Parallel map and reduce operation
3. org.apache.hadoop.hive.ql.exec.UDAF
4. Implement the following methods inside UDAFEvaluator.
   a. init() : resets the status of the aggregation function
   b. iterate() : The iterate() method is called every time there is a new value to be aggregated
   c. merge() : The merge() method is called when Hive decides to combine one partial aggregation with another.
   d. terminatePartial(): a result of partial aggregation.
   e. terminate(): terminate() method is called when the final result of the aggregation is needed.

How are UDTFs implemented

1. Aggregation not required across the nodes
2. Mostly used in cases of data explode
   a. E.g Arrays to row conversion
3. Implement the following:
   a. initalize() : Initialize this UDTF. This will be called only once per instance.
   b. process() : Provide a set of arguments for processing
   c. forward() : Passes an output to the collector
   d. close() : Called to notify the UDTF that there are no more rows to process.

Thank you :)