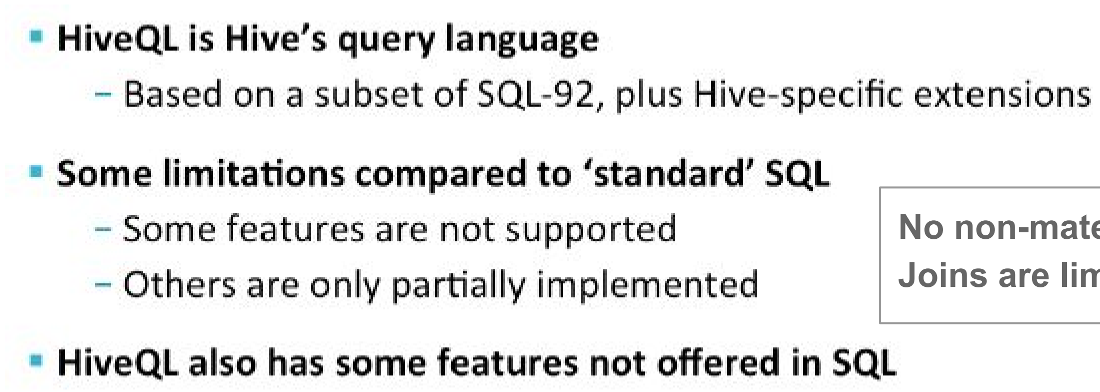
Hive queries

- Select statements
- Joins
- Built-in functions
- Create table as select (CTAS)
- Writing query results

- **HiveQL**
    - **Basic syntax**
    - Joining Datasets
    - Built-in Functions
    - Simplifying queries using views
- Saving queries to tables
- Explaining the query

An Introduction to HiveQL

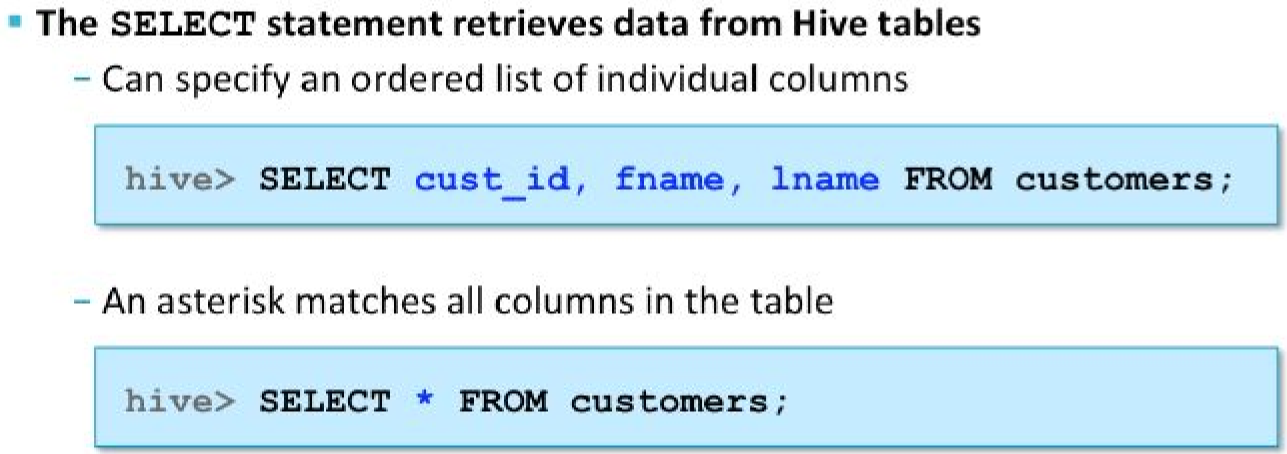


- **HiveQL is Hive's query language**
  - Based on a subset of SQL-92, plus Hive-specific extensions
- **Some limitations compared to 'standard' SQL**
  - Some features are not supported
  - Others are only partially implemented
- **HiveQL also has some features not offered in SQL**

No non-materialized views
Joins are limited (only equality)

- **Maps, array and structs**
- **Built-in table partitioning**
- **Multi-table inserts**

Selecting Data from Hive Tables



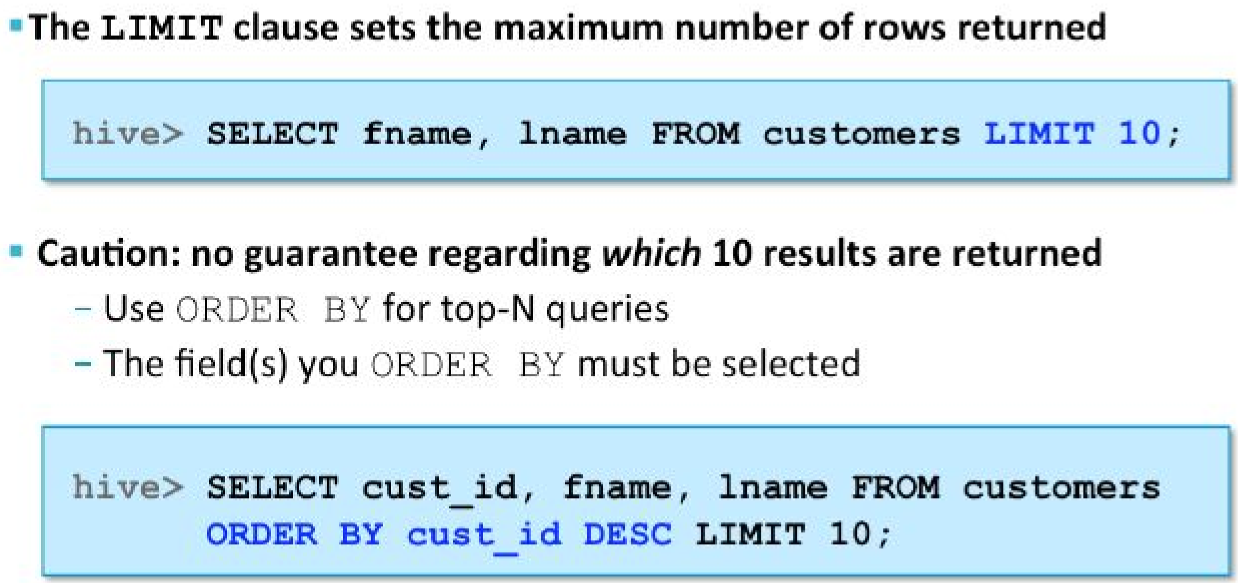
- **The `SELECT` statement retrieves data from Hive tables**
  - Can specify an ordered list of individual columns

```
hive> SELECT cust_id, fname, lname FROM customers;
```

  - An asterisk matches all columns in the table

```
hive> SELECT * FROM customers;
```

Limiting and Sorting Query Results



- **The `LIMIT` clause sets the maximum number of rows returned**

```
hive> SELECT fname, lname FROM customers LIMIT 10;
```

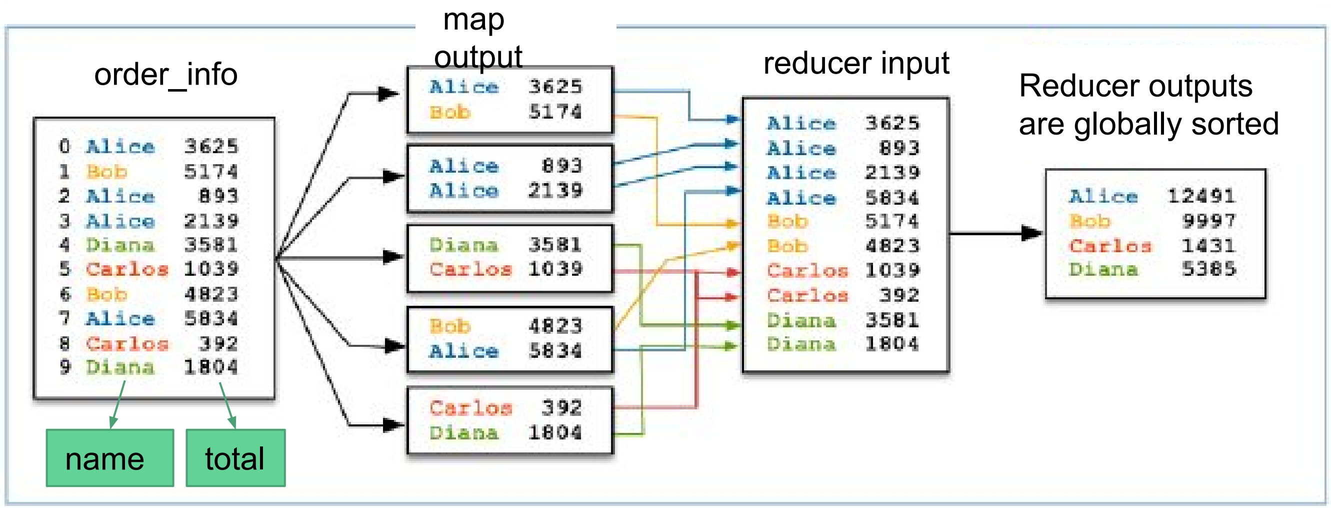
- **Caution: no guarantee regarding *which* 10 results are returned**
  - Use `ORDER BY` for top-N queries
  - The field(s) you `ORDER BY` must be selected

```
hive> SELECT cust_id, fname, lname FROM customers
      ORDER BY cust_id DESC LIMIT 10;
```

Sorting: use ORDER BY for complete sort

- **As in SQL, ORDER BY sorts specified fields in HiveQL**
  - Consider the result from the following query

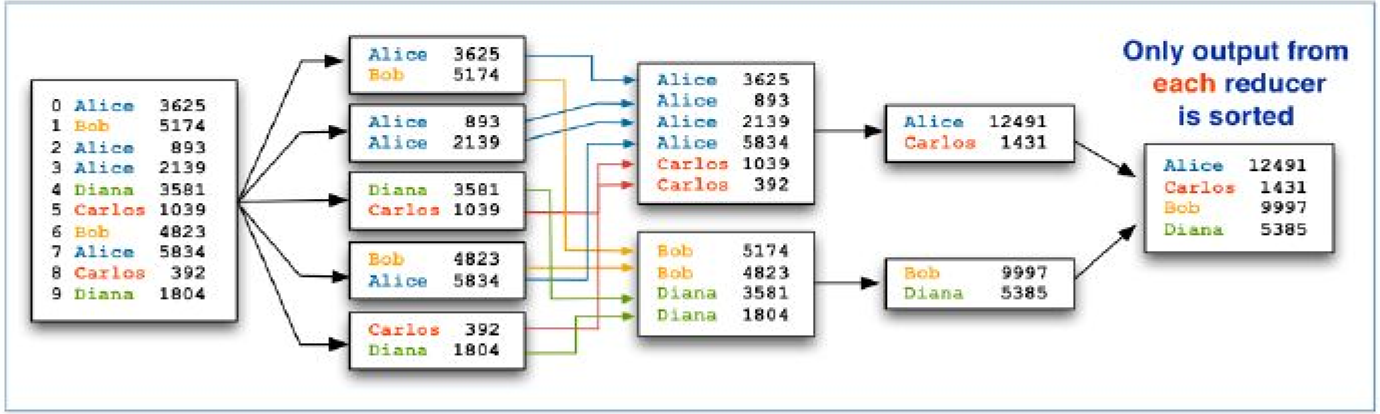
```
hive> SELECT name, SUM(total)
      FROM order_info GROUP BY name
      ORDER BY name;
```



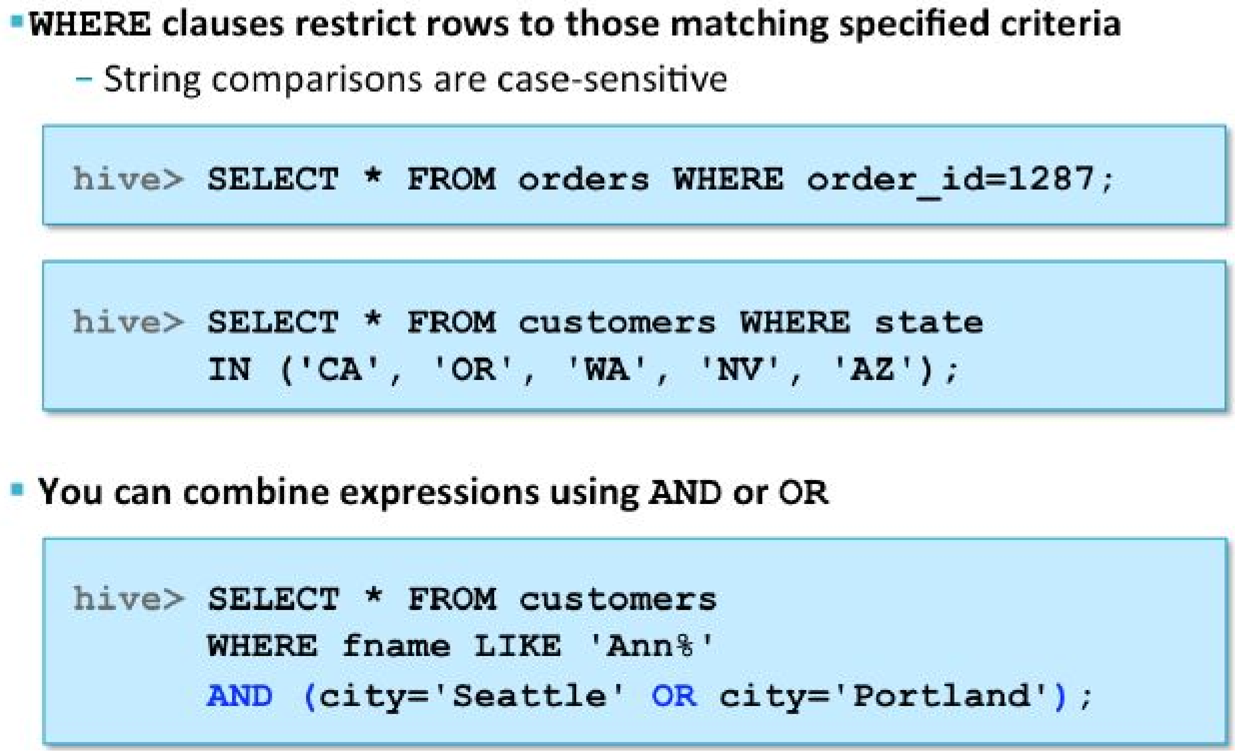
Sorting: use SORT BY for partial sort

- **HiveQL also supports partial ordering via `SORT BY`**
  - Offers much better performance if global order isn't required

```
hive> SELECT name, SUM(total)
      FROM order_info GROUP BY name
      SORT BY name;
```



Using a WHERE Clause to Restrict Results



- **WHERE clauses restrict rows to those matching specified criteria**
  - String comparisons are case-sensitive

```
hive> SELECT * FROM orders WHERE order_id=1287;
```

```
hive> SELECT * FROM customers WHERE state
      IN ('CA', 'OR', 'WA', 'NV', 'AZ');
```

- **You can combine expressions using AND or OR**

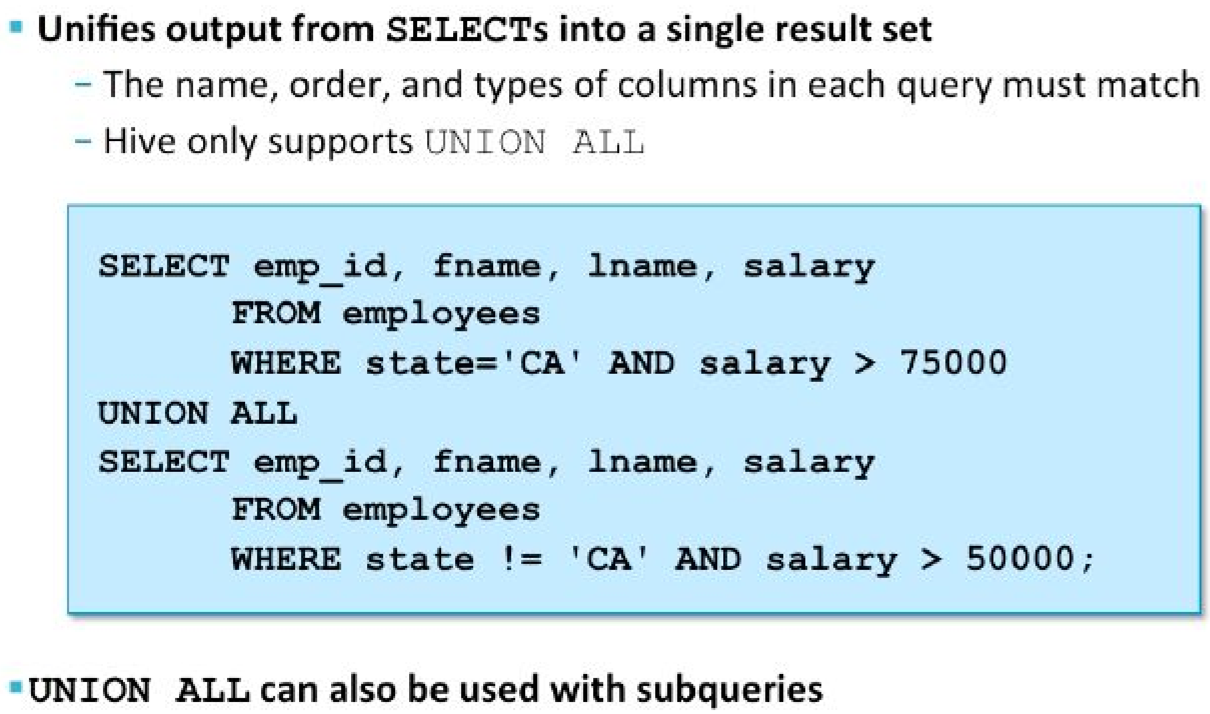
```
hive> SELECT * FROM customers
      WHERE fname LIKE 'Ann%'
      AND (city='Seattle' OR city='Portland');
```

Table Aliases

- **Table aliases can help simplify complex queries**

```
hive> SELECT o.order_date, c.fname, c.lname
      FROM customers c JOIN orders o
      ON c.cust_id = o.cust_id
      WHERE c.zipcode='94306';
```

Combining Query Results with UNION ALL



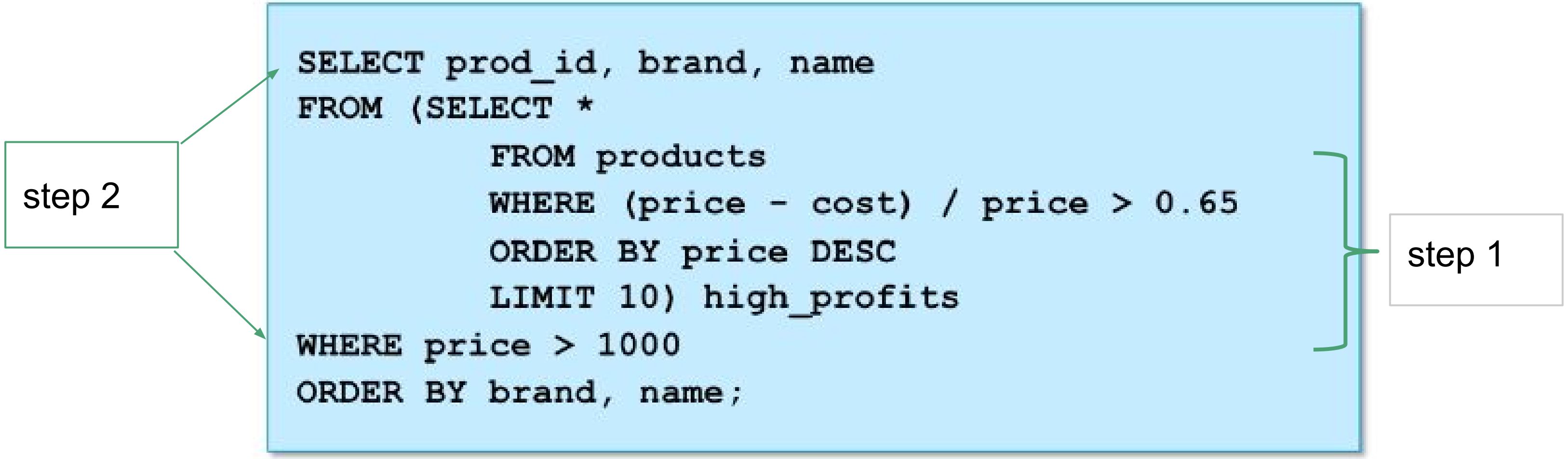
- **Unifies output from `SELECT`s into a single result set**
  - The name, order, and types of columns in each query must match
  - Hive only supports `UNION ALL`

```
SELECT emp_id, fname, lname, salary
      FROM employees
      WHERE state='CA' AND salary > 75000
UNION ALL
SELECT emp_id, fname, lname, salary
      FROM employees
      WHERE state != 'CA' AND salary > 50000;
```

- **`UNION ALL` can also be used with subqueries**

Subqueries in Hive

Hive supports subqueries in the FROM clause of the SELECT statement



```
SELECT prod_id, brand, name
FROM (SELECT *
          FROM products
          WHERE (price - cost) / price > 0.65
          ORDER BY price DESC
          LIMIT 10) high_profits
WHERE price > 1000
ORDER BY brand, name;
```

- **Hive allows arbitrary levels of subqueries**
  - Each subquery must be named (like `high_profits` above)

Performance Patterns in HiveQL (1)

- The fastest queries involve only metadata

```
DESCRIBE customers;
```

- The next fastest simply read from HDFS

```
SELECT * FROM customers;
```

- Then a query that requires a map-only job

```
SELECT * FROM customers WHERE zipcode = 94305;
```

Performance Patterns in HiveQL (2)

- The next slowest type of query requires both Map and Reduce phases

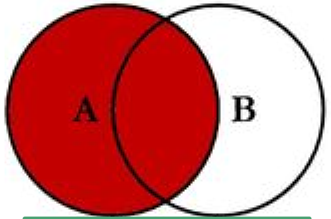
```
SELECT COUNT(cust_id) FROM customers
WHERE zipcode=94305;
```

- The slowest queries require multiple MapReduce jobs

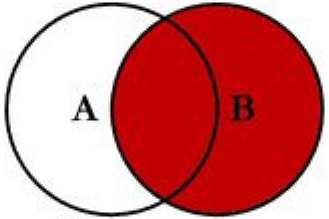
```
SELECT zipcode, COUNT(cust_id) AS num FROM customers
GROUP BY zipcode
ORDER BY num DESC
LIMIT 10;
```

- HiveQL
    - Basic syntax
    - **Joining Datasets**
    - Built-in Functions
    - Simplifying queries using views
- Saving queries to tables
- Explaining the query

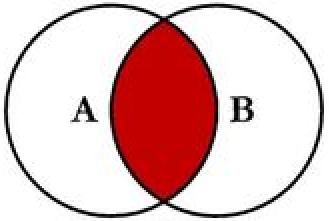
SQL JOINS



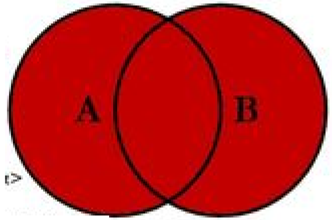
Left Outer Join



Right Outer Join



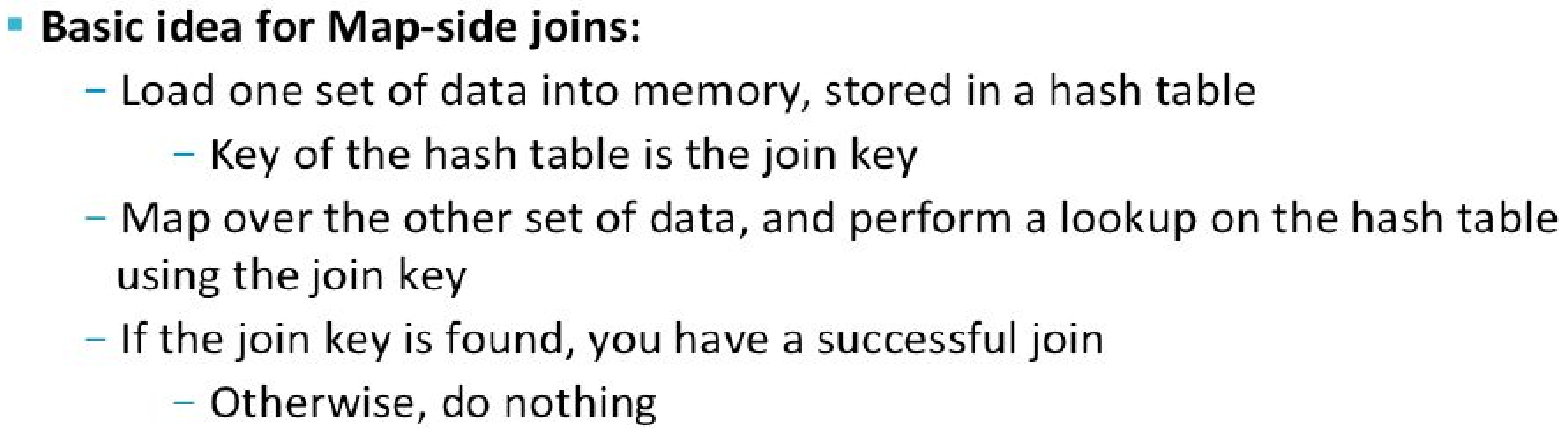
Inner Join



Outer Join

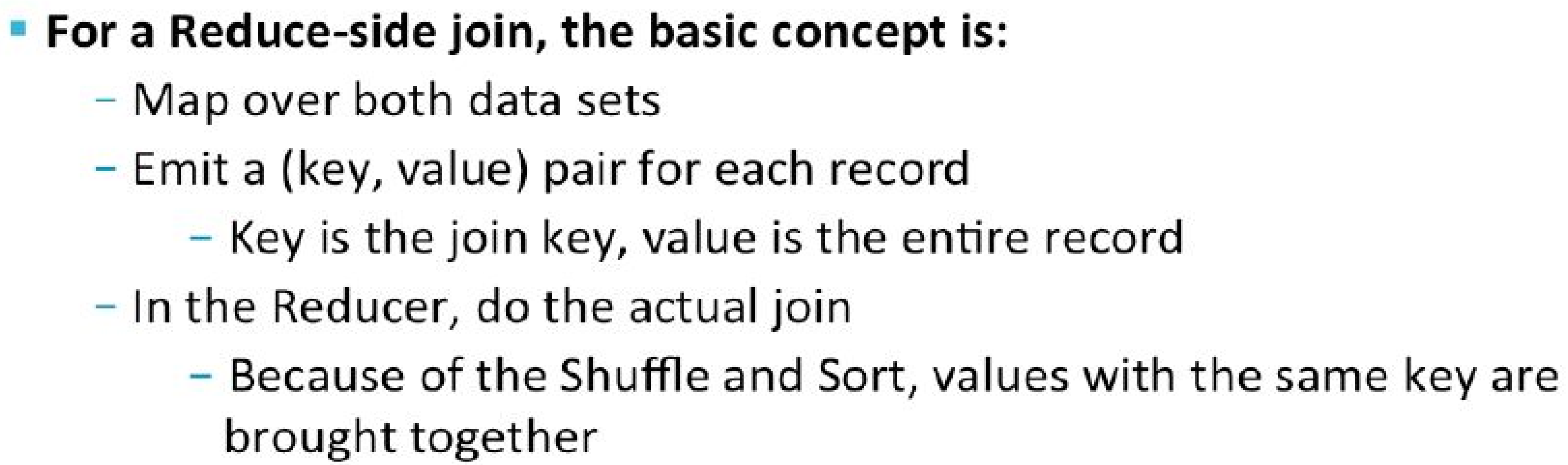
Joins in MapReduce

The rundown on joins (1)



- **Basic idea for Map-side joins:**
  - Load one set of data into memory, stored in a hash table
    - Key of the hash table is the join key
  - Map over the other set of data, and perform a lookup on the hash table using the join key
  - If the join key is found, you have a successful join
    - Otherwise, do nothing

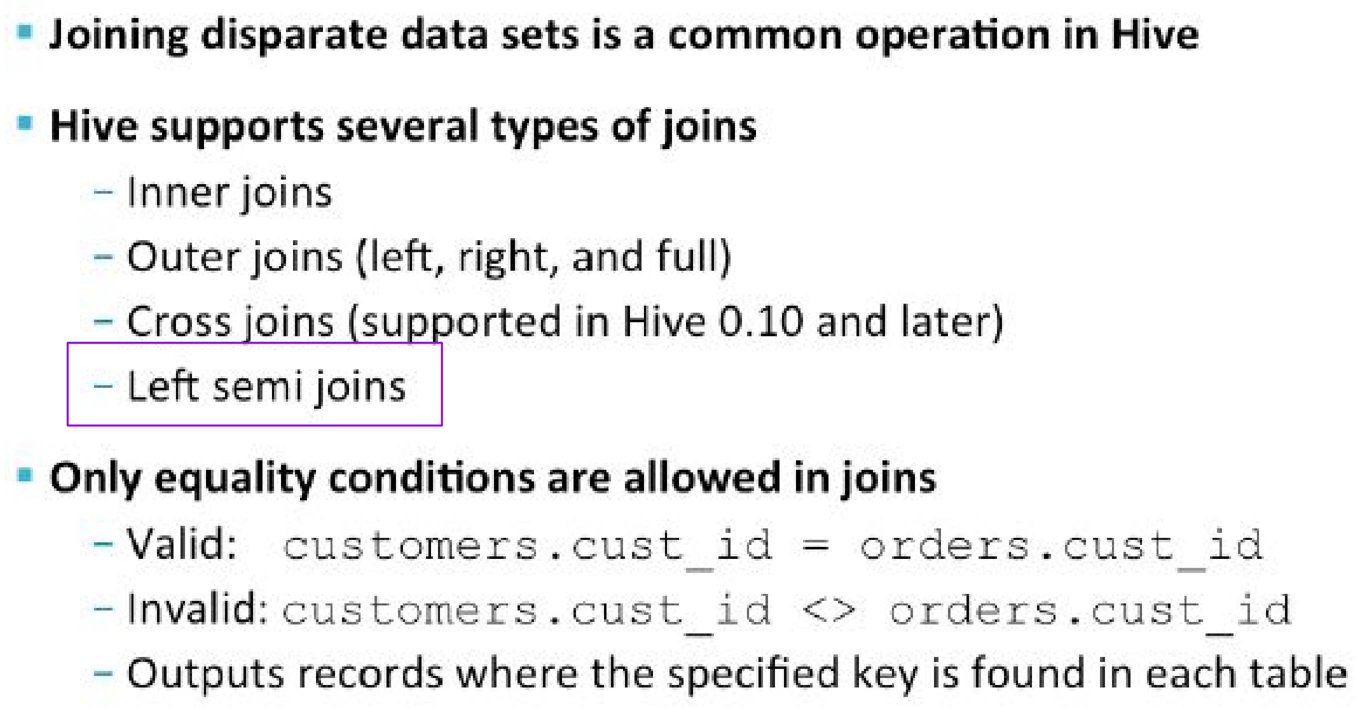
The rundown on joins (2)



- **For a Reduce-side join, the basic concept is:**
  - Map over both data sets
  - Emit a (key, value) pair for each record
    - Key is the join key, value is the entire record
  - In the Reducer, do the actual join
    - Because of the Shuffle and Sort, values with the same key are brought together

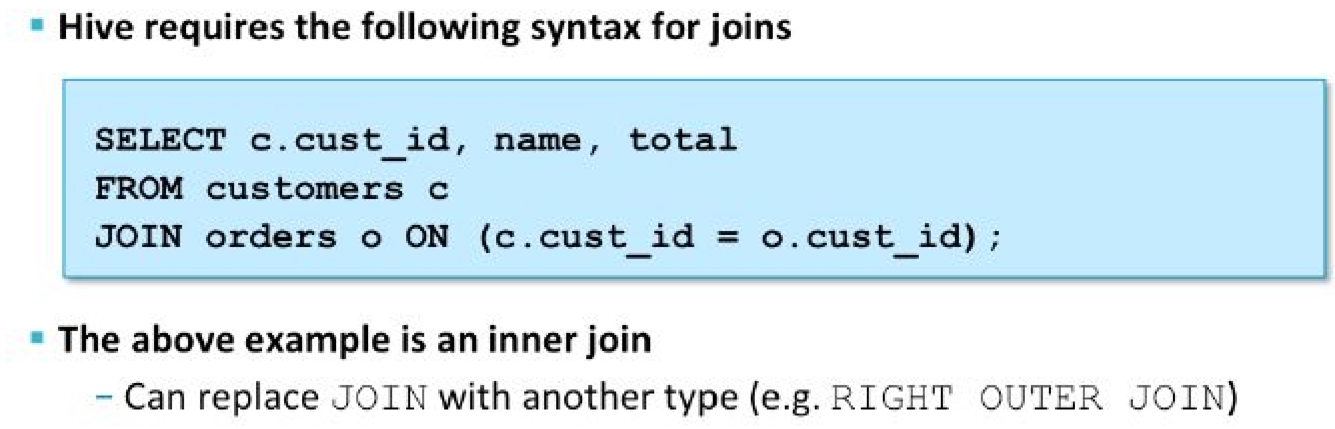
Joins in Hive - use MapReduce joins

Joins in Hive



- **Joining disparate data sets is a common operation in Hive**
- **Hive supports several types of joins**
  - Inner joins
  - Outer joins (left, right, and full)
  - Cross joins (supported in Hive 0.10 and later)
  - Left semi joins
- **Only equality conditions are allowed in joins**
  - Valid: `customers.cust_id = orders.cust_id`
  - Invalid: `customers.cust_id <> orders.cust_id`
  - Outputs records where the specified key is found in each table

Join Syntax



- Hive requires the following syntax for joins

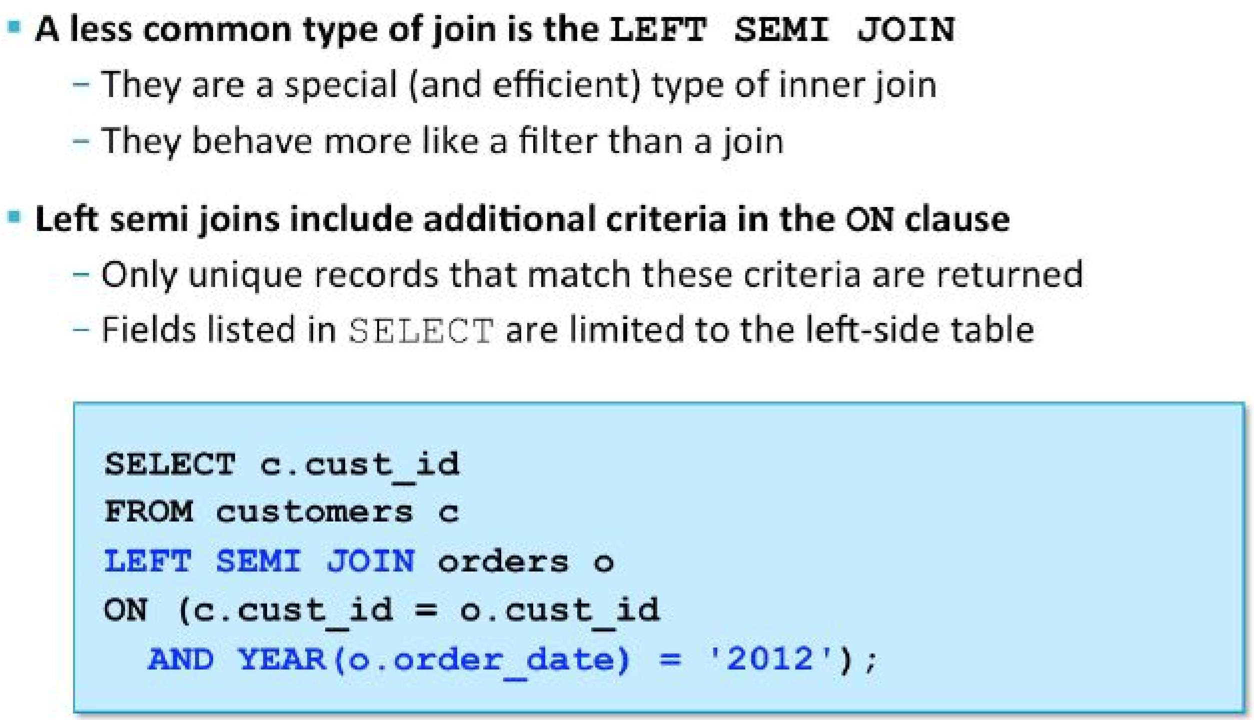
```
SELECT c.cust_id, name, total
FROM customers c
JOIN orders o ON (c.cust_id = o.cust_id);
```

- The above example is an inner join
  - Can replace `JOIN` with another type (e.g. `RIGHT OUTER JOIN`)

Since Hive 0.13, implicit joins are supported

```
SELECT *
FROM table1 t1, table2 t2, table3 t3
WHERE t1.id = t2.id AND t2.id = t3.id AND t1.zipcode = '02535';
```

Left Semi Joins (1)



- **A less common type of join is the `LEFT SEMI JOIN`**
  - They are a special (and efficient) type of inner join
  - They behave more like a filter than a join
- **Left semi joins include additional criteria in the ON clause**
  - Only unique records that match these criteria are returned
  - Fields listed in `SELECT` are limited to the left-side table

```
SELECT c.cust_id
FROM customers c
LEFT SEMI JOIN orders o
ON (c.cust_id = o.cust_id
  AND YEAR(o.order_date) = '2012');
```

Left Semi Joins (2)

Traditional left semi-join

```
SELECT c.customer_id FROM customers c
LEFT SEMI JOIN orders o
ON (c.customer_id = o.order_customer_id)
AND (o.order_date > 2012);
```

Recently added to Hive

```
SELECT c.customer_id FROM customers c
WHERE EXISTS(
     SELECT * FROM orders o
     WHERE (c.customer_id = o.order_customer_id)
     AND (o.order_date > 2012)
);
```

Join performance

- Most tables to be joined are buffered into memory.
- By default, only the last table named (rightmost table) is streamed.
- To change the table to be streamed use a /* hint */

```
SELECT /*+ STREAMTABLE(a)*/ a.val, b.val, c.val FROM a
JOIN b ON (a.key=b.key1)
JOIN c on (c.key=b.key1)
```

Join optimizations

- If an input table fits in memory:
  - table is loaded into memory as a hash table
  - the larger table is input as data to the MapReduce program
  - this is a map-side join

- In “star-schemas”, dimension joins are optimized as Map-side joins
  - Example star:
    - center is the event (sales)
    - Points of the star: dimensions (time, purchaser demographic and store)
  - Joining dimensions is optimized
  - Chaining dimension joins is optimized

https://cwiki.apache.org/confluence/display/Hive/LanguageManual+JoinOptimization

Example of a star-schema chained-join

```
Select count(*) cnt
From sales sl
     join demographics demo on (sl.demo_key = demo.key)
     join time t on (sl.time_key = t.key)
     join store s on (sl.store_key = s.key)
Where
     t.hour = 8
     t.minute >= 30
     demo.num_children = 2
order by cnt;
```

Optimization: Attempts to load all dimensions into hashmaps and use the sales table as Mapper input.

- HiveQL
  - Basic syntax
  - Joining Datasets
  - **Built-in Functions**
  - Simplifying queries using views
- Saving queries to tables
- Explaining the query

Hive Functions

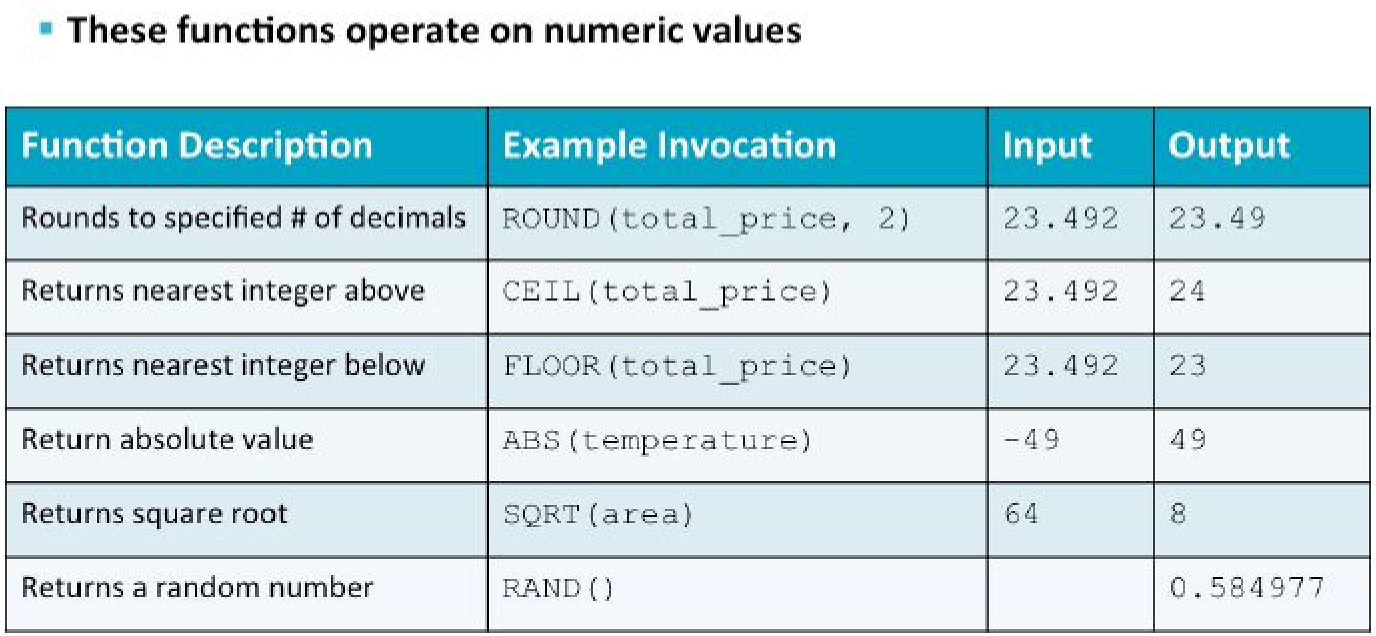
- **Hive offers dozens of built-in functions**
  - Many are identical to those found in SQL
  - Others are Hive-specific
- **Example function invocation**
  - Function names are not case-sensitive

```
hive> SELECT CONCAT(fname, ' ', lname) AS fullname
      FROM customers;
```

- **To see information about a function**

```
hive> DESCRIBE FUNCTION UPPER;
      UPPER(str) - Returns str with all characters
      changed to uppercase
```

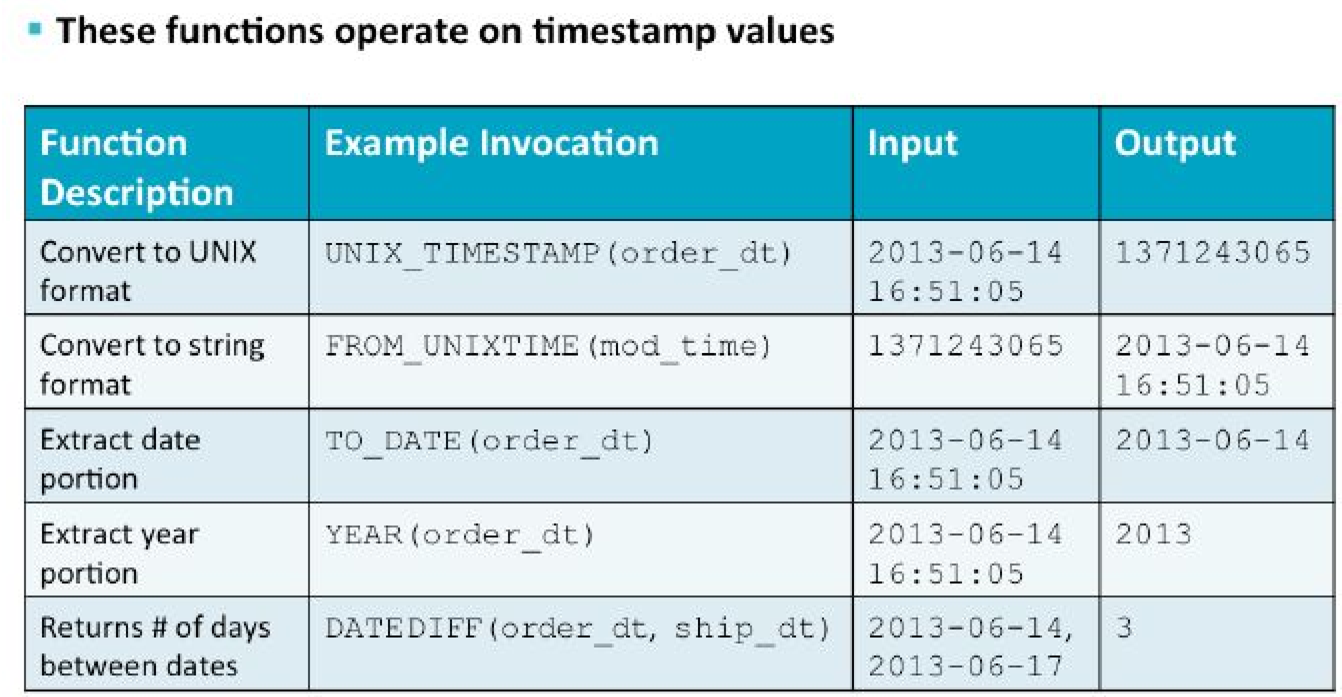
Example Built-in Functions (1)



- **These functions operate on numeric values**

Function Description	Example Invocation	Input	Output
Rounds to specified # of decimals	`ROUND(total_price, 2)`	23.492	23.49
Returns nearest integer above	`CEIL(total_price)`	23.492	24
Returns nearest integer below	`FLOOR(total_price)`	23.492	23
Return absolute value	`ABS(temperature)`	-49	49
Returns square root	`SQRT(area)`	64	8
Returns a random number	`RAND()`		0.584977

Example Built-in Functions (2)



- **These functions operate on timestamp values**

Function Description	Example Invocation	Input	Output
Convert to UNIX format	`UNIX_TIMESTAMP(order_dt)`	2013-06-14 16:51:05	1371243065
Convert to string format	`FROM_UNIXTIME(mod_time)`	1371243065	2013-06-14 16:51:05
Extract date portion	`TO_DATE(order_dt)`	2013-06-14 16:51:05	2013-06-14
Extract year portion	`YEAR(order_dt)`	2013-06-14 16:51:05	2013
Returns # of days between dates	`DATEDIFF(order_dt, ship_dt)`	2013-06-14, 2013-06-17	3

Examine the order_date field in orders

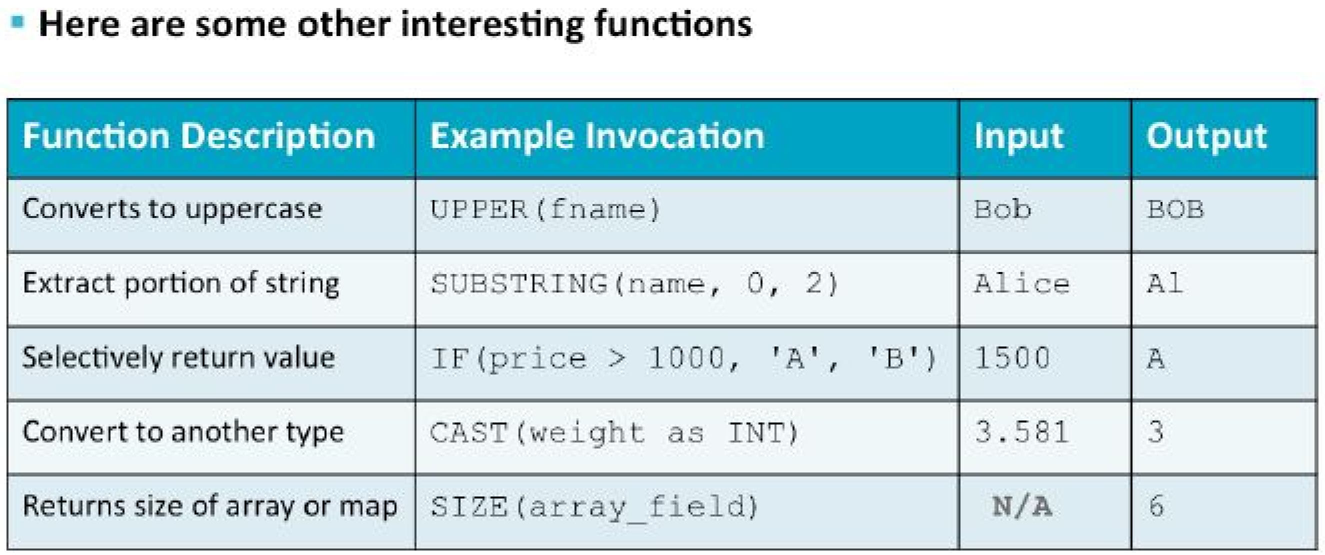
```
hive> select order_date from orders;
```

```
hive> select from_unixtime(order_date) from orders;
```

In your homework you will create a new table, with a “fixed” date-time field so that you can then select as follows:

```
hive> select year(from_unixtime(order_date)) from orders_corrected;
```

Example Built-in Functions (3)

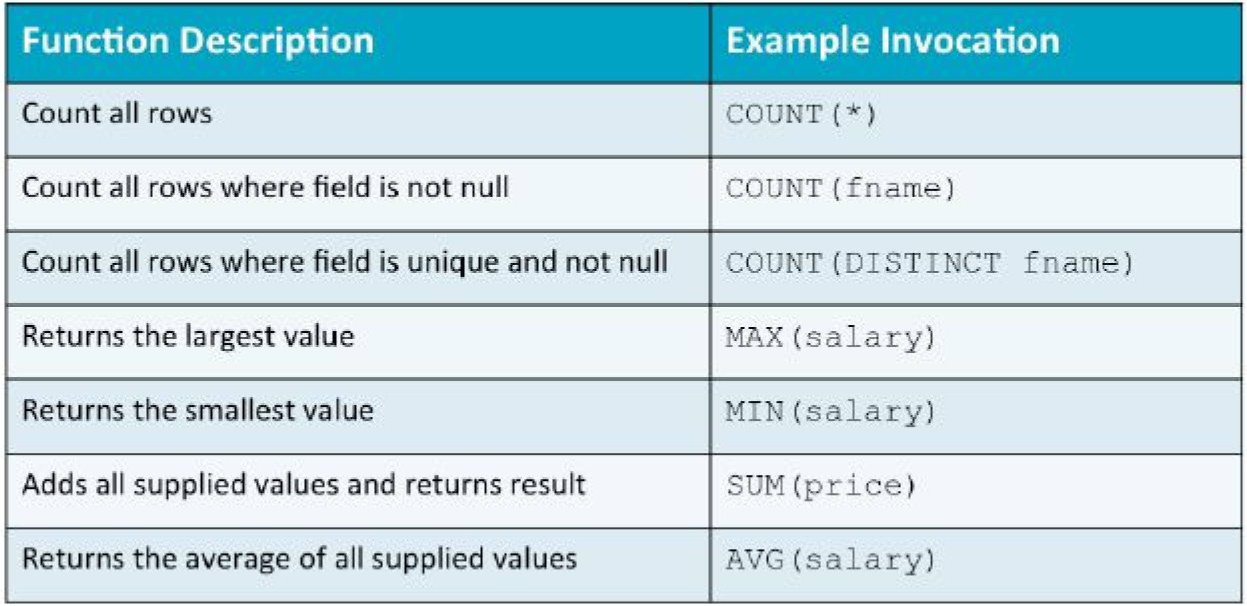


- **Here are some other interesting functions**

Function Description	Example Invocation	Input	Output
Converts to uppercase	`UPPER(fname)`	Bob	BOB
Extract portion of string	`SUBSTRING(name, 0, 2)`	Alice	Al
Selectively return value	`IF(price > 1000, 'A', 'B')`	1500	A
Convert to another type	`CAST(weight as INT)`	3.581	3
Returns size of array or map	`SIZE(array_field)`	**N/A**	6

Built-in Aggregate Functions

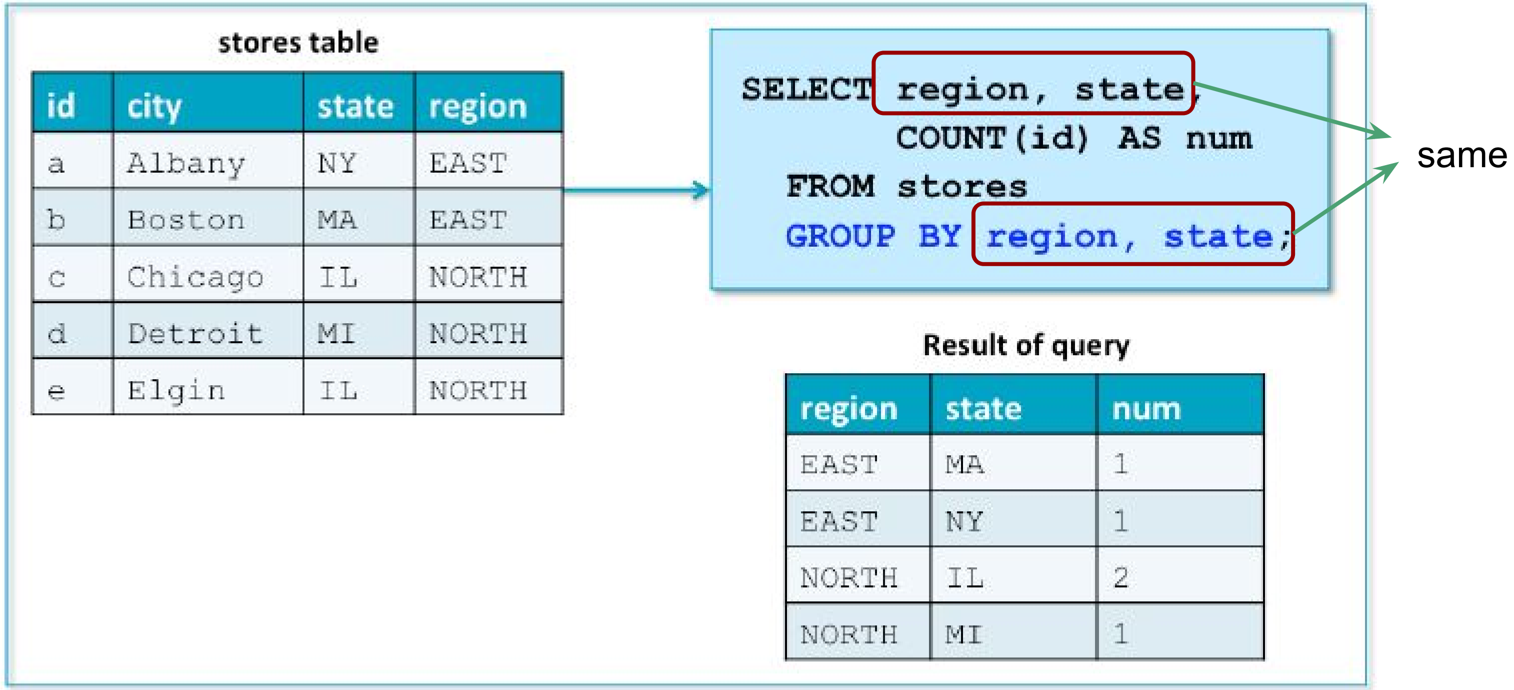
- **Hive offers many aggregate functions, including**



Function Description	Example Invocation
Count all rows	`COUNT(*)`
Count all rows where field is not null	`COUNT(fname)`
Count all rows where field is unique and not null	`COUNT(DISTINCT fname)`
Returns the largest value	`MAX(salary)`
Returns the smallest value	`MIN(salary)`
Adds all supplied values and returns result	`SUM(price)`
Returns the average of all supplied values	`AVG(salary)`

Record Grouping and Aggregate Functions

- **GROUP BY groups selected data by one or more columns**
  - **Caution**: Columns not part of aggregation must be listed in `GROUP BY`



stores table

id	city	state	region
a	Albany	NY	EAST
b	Boston	MA	EAST
c	Chicago	IL	NORTH
d	Detroit	MI	NORTH
e	Elgin	IL	NORTH

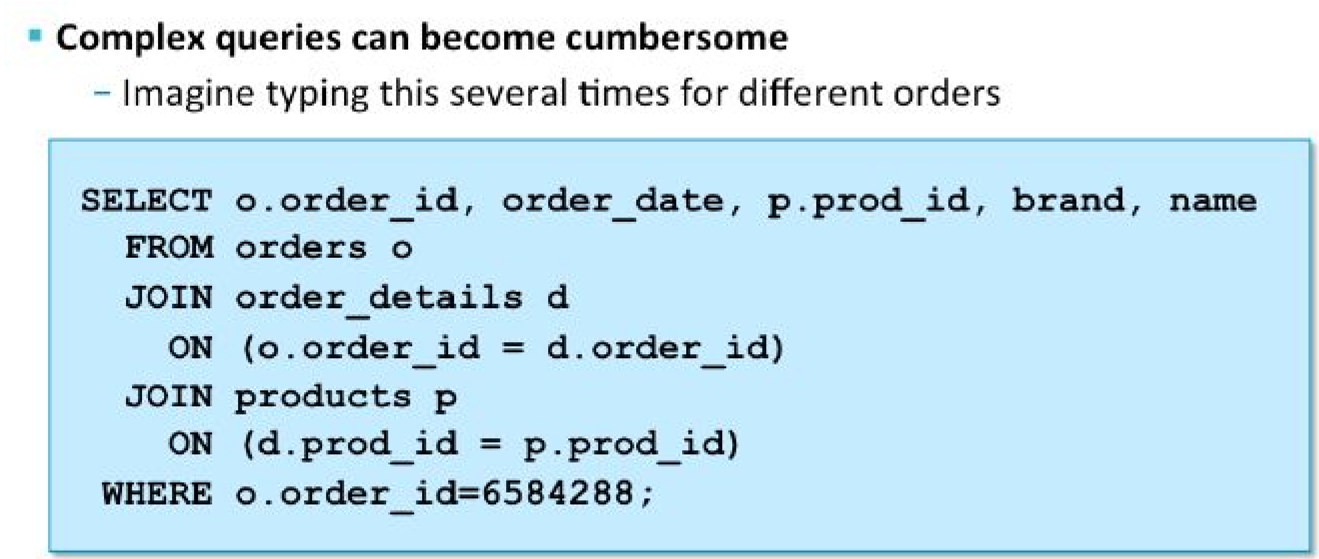
```
SELECT region, state,
       COUNT(id) AS num
  FROM stores
  GROUP BY region, state;
```

Result of query

region	state	num
EAST	MA	1
EAST	NY	1
NORTH	IL	2
NORTH	MI	1

- HiveQL
  - Basic syntax
  - Joining Datasets
  - Built-in Functions
  - **Simplifying queries using views**
- Saving queries to tables
- Explaining the query

Simplifying Complex Queries



- **Complex queries can become cumbersome**
  - Imagine typing this several times for different orders

```
SELECT o.order_id, order_date, p.prod_id, brand, name
  FROM orders o
  JOIN order_details d
    ON (o.order_id = d.order_id)
  JOIN products p
    ON (d.prod_id = p.prod_id)
 WHERE o.order_id=6584288;
```

Creating Views

- **Views in Hive are conceptually like a table, but backed by a query**
  - You cannot directly add data to a view

```
CREATE VIEW order_info AS
SELECT o.order_id, order_date, p.prod_id, brand, name
   FROM orders o
   JOIN order_details d
     ON (o.order_id = d.order_id)
   JOIN products p
     ON (d.prod_id = p.prod_id);
```

- **Our query is now greatly simplified**

```
hive> SELECT * FROM order_info WHERE order_id=6584288;
```

Inspecting and Removing Views

- Use DESCRIBE FORMATTED to see underlying query

```
hive> DESCRIBE FORMATTED order_info;
```

- Use DROP VIEW to remove a view

```
hive> DROP VIEW order_info;
```

Optimization: Multi-Table Insert (1)

- We just saw that you can save output to an HDFS file

```
INSERT OVERWRITE DIRECTORY 'ny_customers'
    SELECT cust_id, fname, lname
        FROM customers WHERE state = 'NY';
```

- This query could also be written as follows

```
FROM customers c
    INSERT OVERWRITE DIRECTORY 'ny_customers'
        SELECT cust_id, fname, lname WHERE state='NY';
```

Optimization: Multi-Table Insert (2)

- **We sometimes need to extract data to multiple tables**
  - Hive `SELECT` queries can take a long time to complete
- **Hive allows us to do this with a single query**
  - Much more efficient than using multiple queries
- **The following example demonstrates multi-table insert**
  - Result is two directories in HDFS

```
FROM customers c
    INSERT OVERWRITE DIRECTORY 'ny_names'
        SELECT fname, lname WHERE state = 'NY';
    INSERT OVERWRITE DIRECTORY 'ny_count'
        SELECT count(DISTINCT cust_id)
            WHERE state = 'NY';
```

Optimization: Multi-Table Insert (3)

- The following query produces the same result

```
FROM (SELECT * FROM customers WHERE state='NY') nycust
    INSERT OVERWRITE DIRECTORY 'ny_names'
        SELECT fname, lname
    INSERT OVERWRITE DIRECTORY 'ny_count'
        SELECT count(DISTINCT cust_id);
```

- HiveQL
    - Basic syntax
    - Joining Datasets
    - Built-in Functions
    - Simplifying queries using views
- **Saving queries to tables**
- Explaining the query

Saving Query Output to a Table

- SELECT statements display their results on screen
- To send results to a Hive table, use INSERT OVERWRITE TABLE
  - Destination table must already exist
  - Existing contents will be deleted

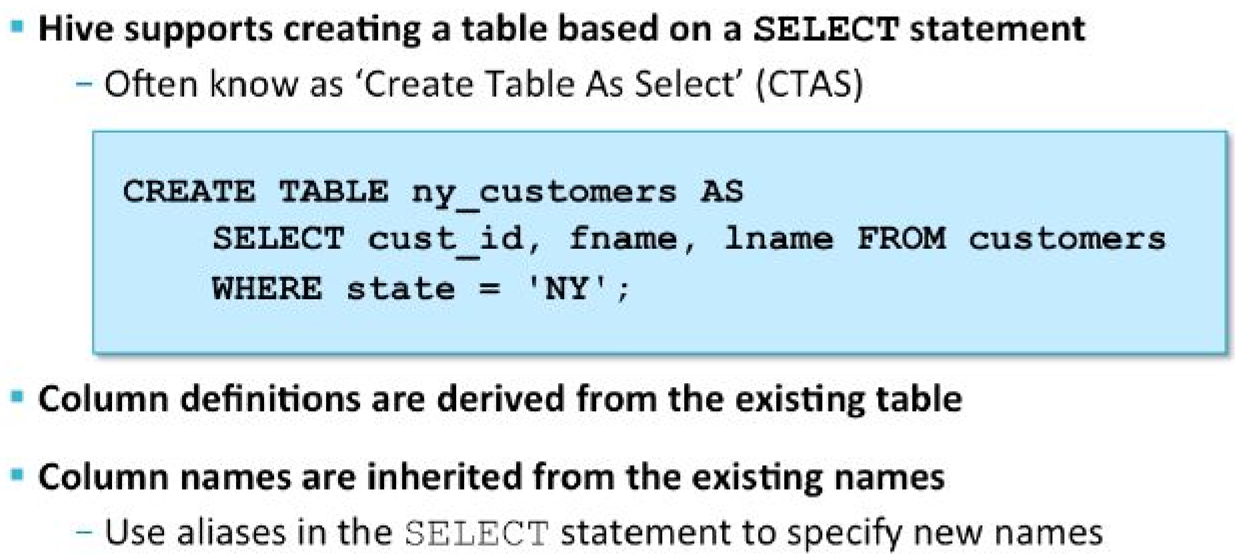
```
hive> INSERT OVERWRITE TABLE ny_customers
        SELECT * FROM customers
        WHERE state = 'NY';
```

- INSERT INTO TABLE adds records without first deleting existing data

```
hive> INSERT INTO TABLE ny_customers
        SELECT * FROM customers
        WHERE state = 'NJ' OR state = 'CT';
```

https://cwiki.apache.org/confluence/display/Hive/LanguageManual+DDL

Creating Tables Based on Existing Data



- **Hive supports creating a table based on a `SELECT` statement**
  - Often know as ‘Create Table As Select’ (CTAS)

```
CREATE TABLE ny_customers AS
    SELECT cust_id, fname, lname FROM customers
    WHERE state = 'NY';
```

- **Column definitions are derived from the existing table**
- **Column names are inherited from the existing names**
  - Use aliases in the `SELECT` statement to specify new names

Writing Output to a Filesystem

- You can save output to a file in HDFS

```
hive> INSERT OVERWRITE DIRECTORY '/dualcore/ny/'
        SELECT * FROM customers
        WHERE state = 'NY';
```

- Add LOCAL to store results to local disk instead

```
hive> INSERT OVERWRITE LOCAL DIRECTORY '/home/bob/ny/'
        SELECT * FROM customers
        WHERE state = 'NY';
```

- Both produce text files delimited by Ctrl-A characters

Writing Output to HDFS, Specifying Format

- **To write the files to HDFS with a user-specified format:**
  - Create an external table in the required format
  - Use `INSERT OVERWRITE TABLE`

```
hive> CREATE EXTERNAL TABLE ny_customers
        (cust_id INT,
         fname STRING,
         lname STRING)
       ROW FORMAT DELIMITED
       FIELDS TERMINATED BY ','
       STORED AS TEXTFILE
       LOCATION '/dualcore/nydata';

hive> INSERT OVERWRITE TABLE ny_customers
       SELECT cust_id, fname, lname
       FROM customers WHERE state = 'NY';
```

Optimization: Multi-Table Insert (1)

- We just saw that you can save output to an HDFS file

```
INSERT OVERWRITE DIRECTORY 'ny_customers'
    SELECT cust_id, fname, lname
        FROM customers WHERE state = 'NY';
```

- This query could also be written as follows

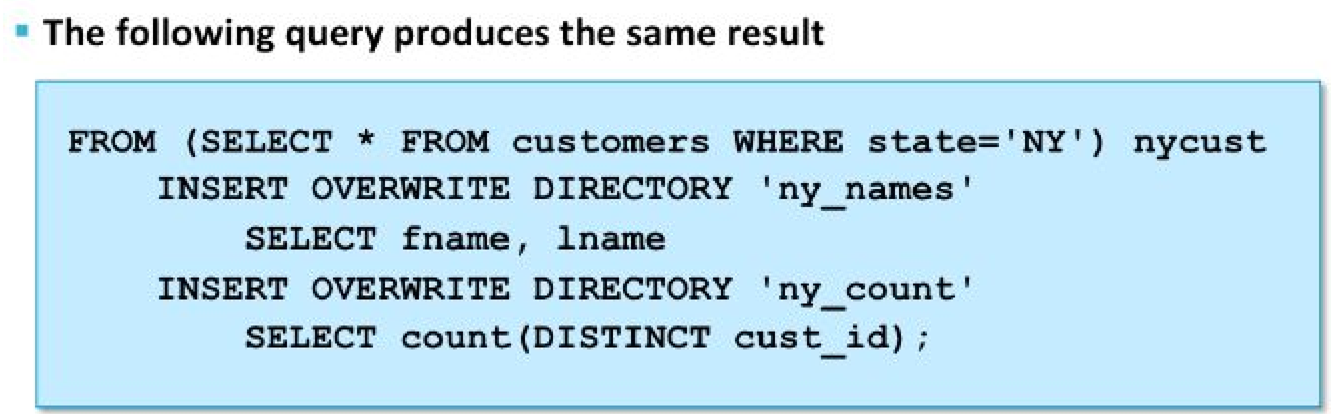
```
FROM customers c
    INSERT OVERWRITE DIRECTORY 'ny_customers'
        SELECT cust_id, fname, lname WHERE state='NY';
```

Optimization: Multi-Table Insert (2)

- **We sometimes need to extract data to multiple tables**
  - Hive SELECT queries can take a long time to complete
- **Hive allows us to do this with a single query**
  - Much more efficient than using multiple queries
- **The following example demonstrates multi-table insert**
  - Result is two directories in HDFS

```
FROM customers c
    INSERT OVERWRITE DIRECTORY 'ny_names'
        SELECT fname, lname WHERE state = 'NY';
    INSERT OVERWRITE DIRECTORY 'ny_count'
        SELECT count(DISTINCT cust_id)
            WHERE state = 'NY';
```

Optimization: Multi-Table Insert (3)

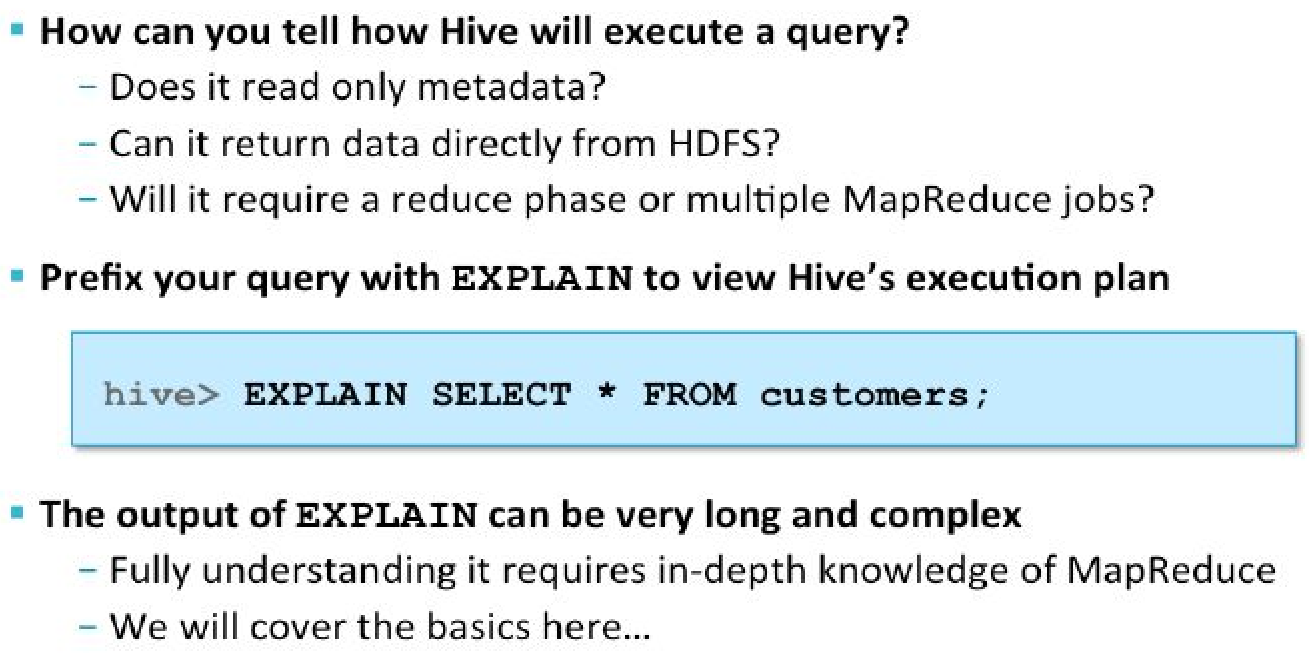


- The following query produces the same result

```
FROM (SELECT * FROM customers WHERE state='NY') nycust
     INSERT OVERWRITE DIRECTORY 'ny_names'
          SELECT fname, lname
     INSERT OVERWRITE DIRECTORY 'ny_count'
          SELECT count(DISTINCT cust_id);
```

- HiveQL
    - Basic syntax
    - Joining Datasets
    - Built-in Functions
    - Simplifying queries using views
- Saving queries to tables
- **Explaining the query**

Viewing the Execution Plan



- **How can you tell how Hive will execute a query?**
  - Does it read only metadata?
  - Can it return data directly from HDFS?
  - Will it require a reduce phase or multiple MapReduce jobs?
- **Prefix your query with `EXPLAIN` to view Hive's execution plan**

```
hive> EXPLAIN SELECT * FROM customers;
```

- **The output of `EXPLAIN` can be very long and complex**
  - Fully understanding it requires in-depth knowledge of MapReduce
  - We will cover the basics here…

EXPLAIN: understanding the Query Plan (1)

- **The query plan contains three main sections**
  - Abstract syntax tree details how Hive parsed query (excerpt below)
  - The stage dependencies and plans are more useful to most users

```
hive> EXPLAIN CREATE TABLE cust_by_zip AS
      SELECT zipcode, COUNT(cust_id) AS num
      FROM customers GROUP BY zipcode;

ABSTRACT SYNTAX TREE:
    (TOK_CREATETABLE (TOK_TABNAME cust_by_zip) ...

STAGE DEPENDENCIES:
    ... (excerpt shown on next slide)

STAGE PLANS:
    ... (excerpt shown on upcoming slide)
```

EXPLAIN: understanding the Query Plan (2)

- **Our query has three stages**
- **Dependencies define order**
  - Stage-1 runs first
  - Stage-0 runs next
  - Stage-2 runs last

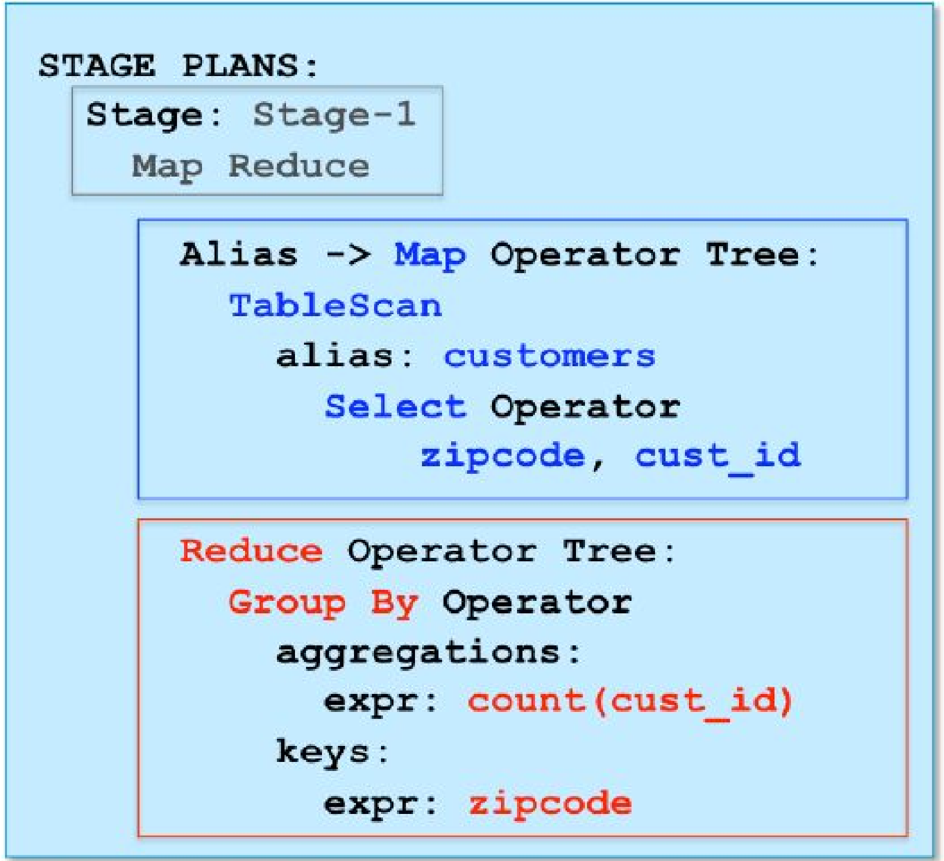
```
ABSTRACT SYNTAX TREE:
 ... (shown on previous slide)

STAGE DEPENDENCIES:
  Stage-1 is a root stage
  Stage-0 depends on stages: Stage-1
  Stage-2 depends on stages: Stage-0

STAGE PLANS:
 ... (shown on next slide)
```

EXPLAIN: understanding the Query Plan (3)

- **Stage-1: MapReduce job**
- **Map phase**
  - Read customers table
  - Selects `zipcode` and `cust_id` columns
- **Reduce phase**
  - Group by `zipcode`
  - Count `cust_id`



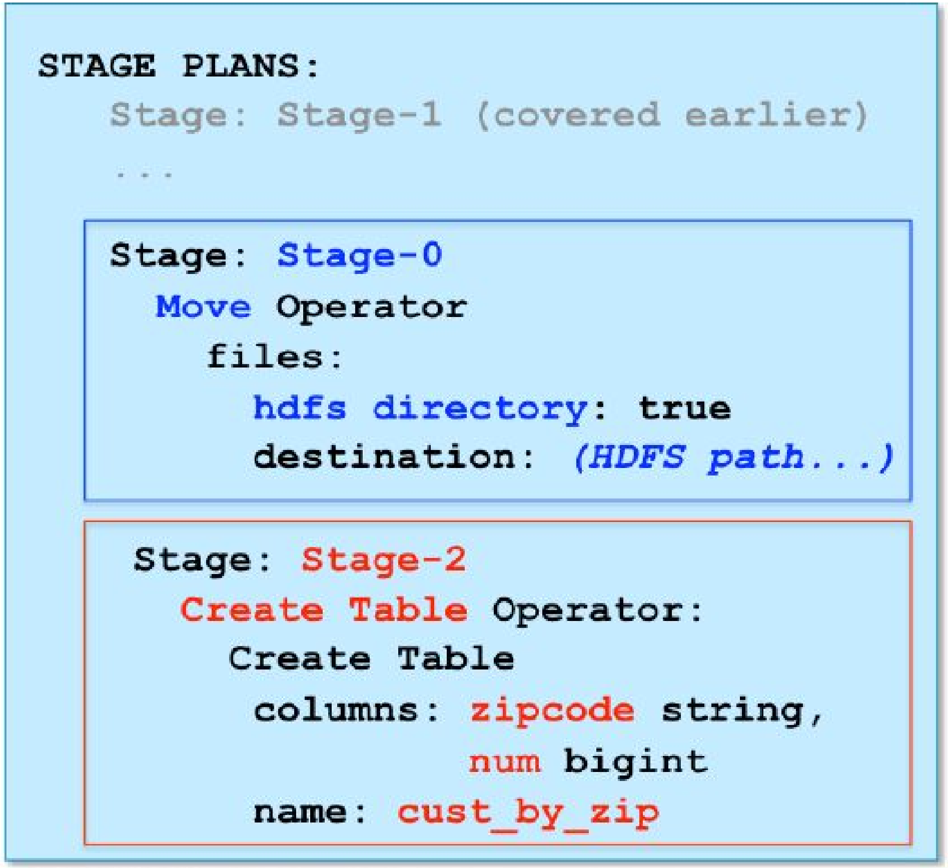
```
STAGE PLANS:
  Stage: Stage-1
    Map Reduce

      Alias -> Map Operator Tree:
        TableScan
          alias: customers
            Select Operator
              zipcode, cust_id

      Reduce Operator Tree:
        Group By Operator
          aggregations:
            expr: count(cust_id)
          keys:
            expr: zipcode
```

EXPLAIN: understanding the Query Plan (4)

- **Stage-0: HDFS action**
  - Move previous stage's output to Hive's warehouse directory
- **Stage-2: Metastore action**
  - Create new table
  - Has two columns



```
STAGE PLANS:
   Stage: Stage-1 (covered earlier)
   ...

   Stage: Stage-0
     Move Operator
       files:
         hdfs directory: true
         destination: (HDFS path...)

    Stage: Stage-2
      Create Table Operator:
        Create Table
         columns: zipcode string,
                  num bigint
         name: cust_by_zip
```

Bibliography

The following offer more information on topics discussed in this section

- HiveQL Language Manual
  - https://cwiki.apache.org/confluence/display/Hive/LanguageManual
- Hive Built-in Functions
  - https://cwiki.apache.org/confluence/display/Hive/LanguageManual+UDF

Hive transactions

As of Hive version **0.14.0**: INSERT...VALUES, UPDATE, and DELETE are now available with full ACID support.

INSERT ... VALUES Syntax:

```
INSERT INTO TABLE tablename [PARTITION (partcol1[=val1], partcol2[=val2] ...)] VALUES values_row [, values_row ...]
```

Where values_row is: (value [, value ...]) where a value is either null or any valid SQL literal

UPDATE Syntax:

```
UPDATE tablename SET column = value [, column = value ...] [WHERE expression]
```

DELETE Syntax:

```
DELETE FROM tablename [WHERE expression]
```

Hive transactions - references

Additionally, from the Hive Transactions doc:

If a table is to be used in ACID writes (insert, update, delete) then the table property "transactional" must be set on that table, starting with Hive 0.14.0. Without this value, inserts will be done in the old style; updates and deletes will be prohibited.

Hive DML reference:

https://cwiki.apache.org/confluence/display/Hive/LanguageManual+DML

Hive Transactions reference:

https://cwiki.apache.org/confluence/display/Hive/Hive+Transactions