Visa

Big Data 101

Student Lab 2

Web Age Solutions Inc.
USA: 1-877-517-6540
Canada: 1-866-206-4644
Web: http://www.webagesolutions.com

For customizations of this book or other sales inquiries, please contact us at:
USA: 1-877-517-6540, email: getinfousa@webagesolutions.com
Canada: 1-866-206-4644 toll free, email: getinfo@webagesolutions.com

Web Age Solutions Inc.
439 University Ave
Suite 820
Toronto
Ontario, M5G 1Y8

Contents

Lab 2 – Working with HiveQL - 1 Hour 4

Step1. Login to the Lab environment 5

Step 2. Filtering 6

Step 3. HiveQL Functions 9

Step 4. Group By 15

Step 5. Order By, Sort By, Cluster By, Distribute By 18

Step 6. Subqueries 21

Review 23

Lab 2 – Working with HiveQL - 1 Hour

Overview

HiveQL Queries

In this lab we will explore HiveQL and Hive CLI settings for more efficient results including aggregation, filtering, grouping, and sorting. We will utilize the accounts, offers, vendors, and transactions tables we briefly explored in Lab 1. Credit Card data that has already been given a schema in the Hive Metastore and data is from the csv files in Hadoop.

Transactions:
- id : auto incrementing ID
- account_id : card holders account
- vendor-id : merchant ID
- times : transaction date & time
- city : where transaction happened
- state : where transaction happened
- amount: in $. (e.g 10.34)

Sample data:
1,5715,4,2015-01-01 00:00:00,El Paso,TX,62.81
2,8236,6,2015-01-01 00:00:00,Pittsburgh,PA,157.33

Vendors
- id : unique id
- name
- city
- state
- category : (General / Grocery / Technology)
-swipe_rate

Sample data:
1,Walmart,Bentonville,AR,General
2,Kroger,Cincinatti,OH,Grocery

Accounts
- id : auto inc id
- accountNo
- bankid
- first_name
- last_name
- email

- gender
- address
- city
- state
- zip

Sample_data:
1,1,1,Ryan,Johnston,rjohnston0@mayoclinic.com,Male,5 Hintze Terrace,Laredo,TX,78044
2,2,9,Todd,Martin,tmartin1@europa.eu,Male,878 Sheridan Point,Columbia,SC,29215

Offers
- start_date
- end_date
- vendor_id
- discount

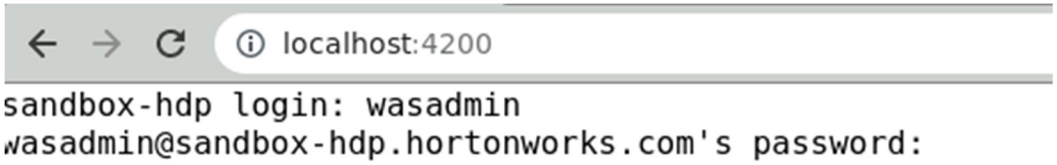
Sample_data:
1/1/2015 0:00, 1/1/2015 0:00, 0.1, 5
1/2/2015 0:00, 1/2/2015 0:00, 0.05, 1

Step1. Login to the Lab environment

Please connect to the assigned Nuvepro Virtual Machine where the labs will be run from. On the Linux desktop you will see the HDP VM has already been started and is showing a couple of ways to login to the system.

Open a web browser on the Nuvepro Virtual Machine. Enter this address into the URL to open the SSH connection to HortonWorks HDP Sandbox.

http://localhost:4200



Login to the console and provide the password when prompted.

Login username: **wasadmin**
Password: **wasadmin**

Start the Hive CLI so we can run the HiveQL commands to investigate the table data.

hive
Now set the database to use with your HiveQL queries.

use wasadmin_db;

Step 2. Filtering

Filtering data is done with conditional statements in HiveQL like SQL. The Where clause in Hive supports many operators and operator precedence is enforced. HiveQL's Where clause is used to filter the column data that satisfies given conditions and supports helpers such as *and, or, not, like, in, between and exists*. There are Relational (see table below), and others from the following link like Arithmetic, Logical, String, and Complex Types Operators available. For more information: https://cwiki.apache.org/confluence/display/Hive/LanguageManual+UDF#LanguageManualUDF-Built-inOperators

Relational Operators

The following operators compare the passed operands and generate a TRUE or FALSE value depending on whether the comparison between the operands holds.

Operator	Operand types	Description
A = B	All primitive types	TRUE if expression A is equal to expression B otherwise FALSE.
A == B	All primitive types	Synonym for the = operator.
A <=> B	All primitive types	Returns same result with EQUAL(=) operator for non-null operands, but returns TRUE if both are NULL, FALSE if one of the them is NULL. (As of version 0.9.0.)

Operator	Operand types	Description
A <> B	All primitive types	NULL if A or B is NULL, TRUE if expression A is NOT equal to expression B, otherwise FALSE.
A != B	All primitive types	Synonym for the <> operator.
A < B	All primitive types	NULL if A or B is NULL, TRUE if expression A is less than expression B, otherwise FALSE.
A <= B	All primitive types	NULL if A or B is NULL, TRUE if expression A is less than or equal to expression B, otherwise FALSE.
A > B	All primitive types	NULL if A or B is NULL, TRUE if expression A is greater than expression B, otherwise FALSE.
A >= B	All primitive types	NULL if A or B is NULL, TRUE if expression A is greater than or equal to expression B, otherwise FALSE.
A [NOT] BETWEEN B AND C	All primitive types	NULL if A, B or C is NULL, TRUE if A is greater than or equal to B AND A less than or equal to C, otherwise FALSE. This can be inverted by using the NOT keyword. (As of version 0.9.0.)
A IS NULL	All types	TRUE if expression A evaluates to NULL, otherwise FALSE.
A IS NOT NULL	All types	FALSE if expression A evaluates to NULL, otherwise TRUE.
A IS [NOT] (TRUE|FALSE)	Boolean types	Evaluates to TRUE only if A mets the condition. (since:3.0.0) Note: NULL is UNKNOWN, and because of that (UNKNOWN IS TRUE) and (UNKNOWN IS FALSE) both evaluates to FALSE.
A [NOT] LIKE B	strings	NULL if A or B is NULL, TRUE if string A matches the SQL simple regular expression B, otherwise FALSE. The comparison is done character by character. The _ character in B matches any character in A (similar to . in posix regular expressions) while the % character in B matches an arbitrary number of characters in A (similar to .* in posix regular expressions). For example, 'foobar' like 'foo' evaluates to FALSE whereas 'foobar' like 'foo_ _ _' evaluates to TRUE and so does 'foobar' like 'foo%'.

Operator	Operand types	Description
A RLIKE B	strings	NULL if A or B is NULL, TRUE if any (possibly empty) substring of A matches the Java regular expression B, otherwise FALSE. For example, 'foobar' RLIKE 'foo' evaluates to TRUE and so does 'foobar' RLIKE '^f.*r$'.
A REGEXP B	strings	Same as RLIKE.

The example query notation below uses the equality operator and filters the data if account's id column matches to the numeric value of 3.

```
select * from accounts where id = 3;
```

The opposite of that would be to use a negation for the value looked up;
A <> B or A != B

```
select * from accounts where id <> 3;
```

Q: What would be the best way to find two numeric values low and high in the middle of a number set? Is there more than on way?
A: Hint… find 10, 11, 12 in this sequence 8, 9, 10, 11, 12, 13, 14

```
select * from numbers where numbers between 10 and 12;
```

```
select * from numbers where numbers >= 10 and numbers <=12;
```

Q: What about matching partial known values?
A: Hint… 'bill' from the word 'billings' or '2015' in string values '2012, 2013, 2014, 2015, 2019, 2022, 2023'

```
select * from transactions where account_ID like concat('%', 'bill*', '%');
```

```
select * from transactions where times like concat('%', '2015*', '%');
```

Challenge 1: (Solutions are not shown, confer with others if stuck on a question)

Find the email address in the accounts table which contains 'mayoclinic.com' Hint: use concat('%', value, '%')

Find all the females in the accounts table from Texas

Identify if there is any missing information in the accounts table

Find the count of all transactions from the city of Pittsburgh

Find the count of all transactions from the New York state in 2015

Which vendors sold items between 1/2/2015 and 1/4/2015 in NY

Which vendors sold items between 1/2/2015 and 1/4/2015 in NE

Step 3. HiveQL Functions

HiveQL also supports over 200 Built-in Functions everything from Mathematical, Collection, Type Conversion, and Date, to Conditional, String, Data Masking and Miscellaneous Functions. See this link: https://cwiki.apache.org/confluence/display/Hive/LanguageManual+UDF#LanguageManualUDF-Built-inFunctions

Within the Hive CLI we can also look for and find out more information on the available functions, especially since some are version dependent.

Try the following commands and replace <function_name> with your own function name choices from the list returned, for example YEAR:

SHOW FUNCTIONS;

DESCRIBE FUNCTION <function_name>;

DESCRIBE FUNCTION EXTENDED <function_name>;

Q: What functions would be good to determine a low or high columnar value?
A: Did you locate the min() and max() functions? If you did not know the aggregation function for this, look at the following guide.
https://cwiki.apache.org/confluence/display/hive/languagemanual+udf#LanguageManualUDF-Built-inAggregateFunctions(UDAF)

Statistics

Aggregation values are used to return a single row of data. For instance, to find the total of all transactions invoice amounts you could try:

select SUM(amount) from transactions;

```
+-----------------------+
|          _c0          |
+-----------------------+
| 1.37641637560000176E8 |
+-----------------------+
1 row selected (12.63 seconds)
```

That is a very big exponential number see the 'E8' at the values end. Notice the table column name has a weird syntax '_c0', this is because it is a calculated field in the output result. Let's fix that using a specific name for the column with the Alias 'as' and column name just like SQL.

select SUM(amount) as total from transactions;

```
+-----------------------+
|         total         |
+-----------------------+
| 1.37641637560000176E8 |
+-----------------------+
1 row selected (8.585 seconds)
```

A much better result for our reporting.

There are several specialized aggregate queries that are available in HiveQL to help us further understand our datasets.
Built-in Functions such as AVG can help understand the average of purchases across all accounts within the transactions table:

select AVG(amount) from transactions;

Further we can break this down by account_id if we utilize a group by.

The grouping of data, covered in more detail later, is specifically used with aggregates, which helps us further understand the values within columns in the data store. Adding the order by, covered later also, helps sort the record order.

select account_id, AVG(amount) from transactions group by account_id limit 10;

select account_id, SUM(amount) as total from transactions group by account_id order by total desc limit 10;

In statistics it is common to breakdown details in many ways to better understand the data. Quartiles are values that divide your data into quarters. We can use the percentile_approx method to describe the percentages desired along with the explode method.

The explode function explodes an array to multiple rows. Returns a row-set with a single column (col), one row for each element from the array.
https://cwiki.apache.org/confluence/display/hive/languagemanual+udf#LanguageManualUDF-Built-inTable-GeneratingFunctions(UDTF)

select explode(percentile_approx(amount,array(0.01,0.20,0.40,0.60,0.80))) from transactions;

```
+---------------------+
|         col         |
+---------------------+
| 3.9579803143796997  |
| 61.04674384614523   |
| 120.83882739816572  |
| 180.80474779282008  |
| 240.90134786937475  |
+---------------------+
5 rows selected (103.988 seconds)
```

Why percentile_approx and not percentile? Because percentile is not valid for floating point numbers, only for integer data type.

Q: How would we do this by account id?
A: We must do a subquery grouping the account_id, a query within a query.

select explode(percentile_approx(t1.total,array(0.01,0.20,0.40,0.60,0.80))) from (select account_id, sum(amount) as total from transactions group by account_id) t1 limit 10;

```
+---------------------+
|         col         |
+---------------------+
| 10005.34            |
| 12333.210000000001  |
| 13298.63            |
| 14165.630000000001  |
| 15167.220000000001  |
+---------------------+
5 rows selected (11.832 seconds)
```

Q: Why does this work?
A: The subquery creates a data set grouped on transactions' account_id and total the amount then we break it up with the explode User Defined Table Function (UDTF). Grouping functionality is covered more in detail soon.

Challenge 2: (Solutions are not shown, confer with others if stuck on a question)

Find the median amount value. What would the median value be?
Remember, median means 50% is below the value and 50% above.
HINT: The percentile function will help here.

```
+---------------------+
|         col         |
+---------------------+
| 10005.34            |
| 13732.550000000001  |
+---------------------+
2 rows selected (89.411 seconds)
```

Can you find mean and quartiles by vendor?

We've taken median and quartile by account id.
How would we do the same by vendor?

Other interesting commands: Collect

Imagine you have the requirement to group information so that all the results were grouped and returned in a single column.

The collect_set() function collapse the records by group and converts the returned result into an array. It returns the set of objects with duplicate elements removed. Here all the results for cities in Texas are returned in the single column without duplicate values.

select gender, collect_set(city) as cities from accounts where state = 'TX' group by gender;

```
+---------+-----------------------------------------------------+
| gender  |                       cities                        |
+---------+-----------------------------------------------------+
| Male    | ["Laredo","Garland","El Paso","Corpus Christi","Dallas","Fort Worth","Odessa
","Houston","Austin","Amarillo","San Antonio","Longview","Galveston","Waco","Denton","Sp
ring"] |
| Female  | ["Lubbock","Conroe","Houston","El Paso","Corpus Christi","Austin","San Anton
io","Dallas","Amarillo","Wichita Falls","College Station","Spring","Laredo","Fort Worth"
,"Bryan","Arlington","Round Rock","Mesquite","Irving"] |
+---------+-----------------------------------------------------+
2 rows selected (8.706 seconds)
```

The collect_list() functions performs similar to above by collapsing the records by group and converting values into an array. In this case though, it returns a list of objects with duplicates. Here duplicate values are returned for the cities.

select gender, collect_list(city) as cities from accounts where state = 'TX' group by gender;

```
+---------+-----------------------------------------------------+
| gender  |                       cities                        |
+---------+-----------------------------------------------------+
| Male    | ["Laredo","Garland","El Paso","Corpus Christi","Dallas","El Paso","Garland",
"Fort Worth","Odessa","Houston","Corpus Christi","Fort Worth","Austin","El Paso","Housto
n","Houston","Amarillo","San Antonio","Houston","Dallas","San Antonio","San Antonio","Au
stin","Longview","Galveston","Fort Worth","Austin","Fort Worth","San Antonio","San Anton
io","Dallas","El Paso","Dallas","Amarillo","Austin","Waco","El Paso","El Paso","El Paso"
,"Denton","Spring","Houston","Odessa","El Paso","El Paso","Dallas"] |
| Female  | ["Lubbock","Conroe","Houston","El Paso","Corpus Christi","Houston","Austin",
"San Antonio","Austin","Dallas","Amarillo","Corpus Christi","Wichita Falls","Corpus Chri
sti","Dallas","College Station","Houston","San Antonio","Dallas","Dallas","Houston","Aus
tin","Houston","Austin","Spring","Laredo","El Paso","Corpus Christi","College Station","
San Antonio","Houston","Fort Worth","Bryan","Austin","Arlington","El Paso","San Antonio"
,"Arlington","Houston","El Paso","El Paso","Amarillo","Houston","Round Rock","El Paso","
El Paso","Fort Worth","Austin","Mesquite","El Paso","Arlington","Irving","Houston","Aust
in","Dallas","El Paso","Austin"] |
+---------+-----------------------------------------------------+
2 rows selected (8.525 seconds)
```

You may investigate further functions as desired here, we will have more challenges later that require some understanding of these.
https://cwiki.apache.org/confluence/display/Hive/LanguageManual+UDF#LanguageManualUDF-Built-inFunctions

Step 4. Group By

As we have seen aggregation values output a single result row. The Group By clause is used for grouping a particular column's values mentioned with the group by command. Selecting aggregation values like sum(), min(), max() and defining a "group by" clause will display results by grouping the particular column values based upon aggregation.

These grouping are typical to count column values pertaining to things like people, places, or things (nouns). For instance, display the total count of employees present in each department or display the total orders from each account.

To calculate total invoice for each account, group by account_id.

select account_id, SUM(amount) from transactions group by **account_id limit 20;**

Notice the number of Map and Reducers running

```
--------------------------------------------------------------------------------
        VERTICES      MODE        STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
--------------------------------------------------------------------------------
Map 1 .......... container     SUCCEEDED      4          4        0        0       0       0
Reducer 2 ...... container     SUCCEEDED      2          2        0        0       0       0
--------------------------------------------------------------------------------
VERTICES: 02/02  [==========================>>] 100%  ELAPSED TIME: 9.28 s
--------------------------------------------------------------------------------
```

Q: Why were additional Map and Reducers needed for this query?
A: The transactions table is quite large so Hive and Hadoop worked together to split the data for parallel operations (faster) hence the 4 Map items. But what about the 2 Reducers? The data result is quite large even though we set a limit it does not get applied until all the results are shuffled. In this case the MapReduce function decided that one Reducer was not sufficient given there was 4 Map indexes produced.

And the results of the query are shown here:

account_id	_c1
1	13285.299999999997
2	13739.18
3	11646.52
4	10724.279999999999
5	16656.16
8	12592.369999999999
10	14929.560000000001
11	12287.920000000002
13	13748.949999999999
15	13917.619999999999
18	14030.810000000001
23	113.41
27	47.68
29	279.76
6	13142.0
7	13603.51
9	15849.209999999997
12	15800.530000000002
14	14451.27
16	15851.52

20 rows selected (10.271 seconds)

The sorted ordered is off in the above query and the column has the computed name instead of something more correct. How can we fix this?
Notice we now add the additional alias and order by to the query.
We will cover Order By in more detail shortly, skip ahead if needed.

select account_id, SUM(amount) as total from transactions group by account_id order by account_id limit 20;

VERTICES	MODE	STATUS	TOTAL	COMPLETED	RUNNING	PENDING	FAILED	KILLED
Map 1	container	SUCCEEDED	4	4	0	0	0	0
Reducer 2	container	SUCCEEDED	2	2	0	0	0	0
Reducer 3	container	SUCCEEDED	1	1	0	0	0	0

VERTICES: 03/03 [==========>>] 100% ELAPSED TIME: 8.29 s

Q: Why were additional Reducers used in this resultset compared to the one without the order by clause? Notice Reducer 3 above.
A: This adds another Reducer phase and forces a single Reducer for the data set to pass through. This is necessary because we want to ensure that global ordering is maintained.

And below ordered and column Alias applied.

account_id	total
1	13285.299999999997
2	13739.18
3	11646.52
4	10724.279999999999
5	16656.16
6	13142.0
7	13603.51
8	12592.369999999999
9	15849.209999999997
10	14929.560000000001
11	12287.920000000002
12	15800.530000000002
13	13748.949999999999
14	14451.27
15	13917.619999999999
16	15851.52
17	15132.27
18	14030.810000000001
19	14911.82
20	15924.390000000003

20 rows selected (9.346 seconds)

Challenge 3: (Solutions are not shown, confer with others if stuck on a question)

Using two columns from the transactions table, show the greatest and smallest items sold by value and the account_id

When and what was the highest sale item for each account between 2015-01-02 00:00:00 and 2015-01-02 00:10:00? Answer is 696 rows.

Find top 10 spending customers from the billings table?

Calculate total of all accounts amounts per year for transactions table

Step 5. Order By, Sort By, Cluster By, Distribute By

Order By:

The syntax used in SQL is the same for HiveQL ORDER BY command.

Order By in Hive queries helps sort selected column data. Sorting is generally performed on a specific column or columns when the presentation of the order is important for the query result in either ascending or descending order in relation to the column values. DESC for sorting the order in descending order and ASC for Ascending order of the sort (Default so can be left off written query).

In the case of values that are of string data type the order would be done lexicographically versus numeric values sorted by ordinal number. The reference for this and other types of ordering data is found at:
https://cwiki.apache.org/confluence/display/Hive/LanguageManual+SortBy

Try the following and observe the differences in the resultsets.

select * from accounts order by state limit 20;

select * from accounts order by state, city limit 20;

select * from accounts order by state desc, city limit 20;

Sort By:

Sort By performs it's sorting on the column names of Hive tables producing sorted output just like the Order By clause. The sort by clause will sort the rows in *each Map result* before feeding to the Reducer. The sort by depends on column types just like order by.

select account_id, amount as total from transactions sort by account_id limit 20;

Compare this to an order by clause, output of both commands below.

select account_id, amount as total from transactions order by account_id limit 20;

Q: Which command was faster? Why do you think that is?
Q: What is the major difference here?
Q: What is the extra Reducer doing?

```
--------------------------------------------------------------------------------
        VERTICES      MODE        STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
--------------------------------------------------------------------------------
Map 1 .......... container     SUCCEEDED      4          4        0        0       0       0
Reducer 2 ...... container     SUCCEEDED      2          2        0        0       0       0
Reducer 3 ...... container     SUCCEEDED      1          1        0        0       0       0
--------------------------------------------------------------------------------
VERTICES: 03/03  [==========================>>] 100%  ELAPSED TIME: 2.92 s
--------------------------------------------------------------------------------
```

account_id	total
1	282.88
1	183.7
1	134.96
1	105.06
1	214.43
1	131.82
1	259.06
1	280.13
1	119.65
1	234.43
1	132.12
1	107.79
1	7.65
1	298.66
1	258.32
1	99.18
1	13.7
1	62.29
1	4.12
1	188.83

20 rows selected (10.816 seconds)

Q: Why are the MapReduce efforts different? Why more Reducers?

```
--------------------------------------------------------------------------------
        VERTICES      MODE        STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
--------------------------------------------------------------------------------
Map 1 .......... container     SUCCEEDED      4          4        0        0       0       0
Reducer 2 ...... container     SUCCEEDED      1          1        0        0       0       0
--------------------------------------------------------------------------------
VERTICES: 02/02  [==========================>>] 100%  ELAPSED TIME: 7.71 s
--------------------------------------------------------------------------------
```

account_id	total
1	173.62
1	282.88
1	107.79
1	234.43
1	7.65
1	148.6
1	14.47
1	298.66
1	280.13
1	197.86
1	132.12
1	191.11
1	162.34
1	21.36
1	259.06
1	144.17
1	6.31
1	119.65
1	169.34
1	183.7

20 rows selected (8.8 seconds)

To see additional explanations for these HiveQL examples refer to: https://cwiki.apache.org/confluence/display/Hive/LanguageManual+SortBy

Cluster By:
Cluster By used as an alternative for both Distribute By and Sort By clauses in HiveQL.

Hive uses the columns in Cluster By to distribute the rows among Reducers. Cluster By columns are sent to the multiple Reducers.

- It ensures sorting orders of values present in multiple reducers

Let's use a Cluster By clause on the account_id column name of the transactions table. The output when executing this query will give results to multiple reducers at the back end. It is an alternative clause for both Sort By and Distribute By. If you want to store results into multiple reducers, use the Cluster By clause. This will take some time though, you might want to limit the values used, try 200.

select account_id, amount from transactions cluster by account_id limit 200;

Distribute By:
Hive uses the columns in Distribute By to distribute the rows among reducers. All Distribute By columns will go to the same reducer.

- It ensures each of N reducers gets non-overlapping ranges of column values
- It doesn't sort the output of each reducer

select account_id, amount from transactions distribute by account_id limit 10;

Q: Which ordering query was the fastest? Why?

Step 6. Subqueries

A subquery is a query that references another query's result to perform additional query tasks. Simple subqueries have better performance than a more complex correlated Where clause subquery.

Simple Subquery

There are two forms of simply subqueries that are available in HiveQL

SELECT ... FROM (subquery) name ...

SELECT ... FROM (subquery) as name

select first_name || ' ' || last_name as fullname from (select first_name, last_name from accounts) clients limit 10;

In the above example we use concatenation to join two columns into a single result with spacing and an alias for adding the column header of fullname. Notice the subquery is assigned a table name of clients.

```
+------------------+
|     fullname     |
+------------------+
| Ryan Johnston    |
| Todd Martin      |
| Sharon Martin    |
| Theresa Hughes   |
| Mary Wright      |
| Margaret Fuller  |
| Carolyn Martin   |
| Aaron Castillo   |
| Juan George      |
| Jimmy Russell    |
+------------------+
10 rows selected (0.251 seconds)
```

With or without the Alias usage the query will work, it's just preference.

select first_name, last_name from (select first_name, last_name, zip from accounts where zip = 19125) as clients;

```
+-------------+------------+
| first_name  | last_name  |
+-------------+------------+
| Jimmy       | Russell    |
| Nicholas    | Rivera     |
+-------------+------------+
```

Here this example uses the alias ‘as’ with the subquery client table

Columnar Subqueries

Subquery in where clause is typically used to compare a child result to a parent value. A where clause on an email address is made to match and then used to return particular account name.

select first_name, last_name from accounts where email = (select email from accounts where email = 'rhowardqf@facebook.com');

```
+-------------+------------+
| first_name  | last_name  |
+-------------+------------+
| Russell     | Howard     |
+-------------+------------+
1 row selected (7.136 seconds)
```

Correlated Subqueries

In a correlated subquery the parent value is compared within the child query by matching the values in a where clause in the child query.

SELECT A FROM T1 WHERE EXISTS (SELECT B FROM T2 WHERE T1.X = T2.Y)

HiveQL does have limitations on the usage for instance IN/NOT IN subqueries select a single column and EXISTS/NOT EXISTS must have one or more correlated predicates.

For additional information on Subqueries see the following:
https://cwiki.apache.org/confluence/display/hive/languagemanual+subqueries

Review

We have seen that HiveQL command syntax and SQL are very similar but there are some subtle differences like extra commands available with Operators and Functions. Next, we looked at clustered commands that utilized parallel programming which has advantages with very large datasets. Lastly, we employed various subquery methods to improve the performance of the resultsets.