Apache Hive (Part 3)

Part 3

HiveQL Data Query Language (DQL)

What Does Hive DQL Include and Do?

- The SELECT, SHOW and HELP statements (queries)
- SELECT is the main DQL instruction. It retrieves data you need.
- SHOW retrieves info about the metadata.
- HELP is for people who need help.

The SELECT … FROM Clauses

- The SELECT clause specifies the columns to keep, as well as the outputs of function calls on one or more columns, e.g., the aggregation functions like count(*).
- The FROM clause identifies from which table, view, or nested query we select records.
- For example:

  SELECT e.name, e.salary **FROM** employees e;

  (e: a table alias)

JSON Syntax For a Collection Type

Hive uses JSON (Java-Script Object Notation) syntax for the output:

```
hive> select name, subordinates from employees20;
OK
John Smith      ["Mary Smith","David Wells"]
Mary Smith      ["Bill King"]
Jason Yang      ["Diana Foster","Kelli Doe","Kevin Hoover"]
Ted Wang        ["Diana Johnson","Kevin Li","Dan Jones"]
Mike McPheron   ["Marvin Xu","Elise Lee"]
```

```
hive> select name, deductions from employees20;
OK
John Smith      {"Federal Taxes":0.2,"State Taxes":0.05,"Insurance":0.1}
Mary Smith      {"Federal Taxes":0.2,"State Taxes":0.05,"Insurance":0.1}
Jason Yang      {"Federal Taxes":0.2,"State Taxes":0.05,"Insurance":0.1}
Ted Wang        {"Federal Taxes":0.2,"State Taxes":0.05,"Insurance":0.1}
Mike McPheron   {"Federal Taxes":0.02,"State Taxes":0.05,"Insurance":0.1}
```

```
hive> select name, address from employees20;
OK
John Smith      {"street":"1 Michigan Ave.","city":"Chicago","state":"IL","zip":60600}
Mary Smith      {"street":"100 Ontario St.","city":"Chicago","state":"IL","zip":60601}
Jason Yang      {"street":"101 Uptown Rd.","city":"Ithaca","state":"NY","zip":14850}
Ted Wang        {"street":"21 Farm Rd.","city":"Ithaca","state":"NY","zip":14850}
Mike McPheron   {"street":"12 Bush Ave.","city":"Ithaca","state":"NY","zip":14850}
```

Use Ambari Hive View to Do a Query

```
SELECT name, deductions FROM employees20;
```

	name	deductions
0	John Smith	{"Federal Taxes":0.2,"State Taxes":0.05,"Insurance":0.1}
1	Mary Smith	{"Federal Taxes":0.2,"State Taxes":0.05,"Insurance":0.1}
2	Jason Yang	{"Federal Taxes":0.2,"State Taxes":0.05,"Insurance":0.1}
3	Ted Wang	{"Federal Taxes":0.2,"State Taxes":0.05,"Insurance":0.1}
4	Mike McPheron	{"Federal Taxes":0.02,"State Taxes":0.05,"Insurance":0.1}

```
SELECT name, address FROM employees20;
```

	name	address
0	John Smith	{"street":"1 Michigan Ave.","city":"Chicago","state":"IL","zip":60600}
1	Mary Smith	{"street":"100 Ontario St.","city":"Chicago","state":"IL","zip":60601}
2	Jason Yang	{"street":"101 Uptown Rd.","city":"Ithaca","state":"NY","zip":14850}
3	Ted Wang	{"street":"21 Farm Rd.","city":"Ithaca","state":"NY","zip":14850}
4	Mike McPheron	{"street":"12 Bush Ave.","city":"Ithaca","state":"NY","zip":14850}

Reference Elements of Collections: Array

Use Indexes

```
SELECT name, subordinates[0] FROM employees20;
```

	name	_c1
0	John Smith	Mary Smith
1	Mary Smith	Bill King
2	Jason Yang	Diana Foster
3	Ted Wang	Diana Johnson
4	Mike McPheron	Marvin Xu

```
hive> select name, subordinates[0] from employees20;
OK
John Smith      Mary Smith
Mary Smith      Bill King
Jason Yang      Diana Foster
Ted Wang        Diana Johnson
Mike McPheron   Marvin Xu
```

What will Happen to the Query if

- an employee does not have a subordinate?
- an employee does not have as many subordinates?

A. It causes an error in query and stops processing.
B. It causes an error in query and completes processing.
C. It does not cause an error but produces a NULL for any of such situations.
D. It does not cause an error but produces an empty string for any of such situations.

An Employee does not Have a Subordinate

```
John Smith^100000.0^Mary Smith#David Wells^Federal Taxes@0.2#State Taxes
Mary Smith^80000.0^Bill King^Federal Taxes@0.2#State Taxes@0.05#Insuranc
Jason Yang^150000.0^Diana Foster#Kelli Doe#Kevin Hoover^Federal Taxes@0.
Y#14850
Ted Wang^120000.0^^Federal Taxes@0.2#State Taxes@0.05#Insurance@0.1^21 F
Mike McPheron^150000.0^^Federal Taxes@0.02#State Taxes@0.05#Insurance@0.
```

The last two employees do not have a subordinate. The data file must contain appropriate delimiters, in this case ^.

Reference Elements of Collections: Array

```
SELECT name, subordinates[0] FROM employees10;
```

```
hive> select name, subordinates[0] from employees10;
OK
John Smith      Mary Smith
Mary Smith      Bill King
Jason Yang      Diana Foster
Ted Wang        NULL
Mike McPheron   NULL
```

Reference Elements of Collections: Array

```
SELECT name, subordinates[3] FROM employees10;
```

```
hive> select name, subordinates[3] from employees10;
OK
John Smith      NULL
Mary Smith      NULL
Jason Yang      NULL
Ted Wang        NULL
Mike McPheron   NULL
```

Reference Elements of Collections: Map

Use a Key

```
SELECT name, deductions["Insurance"] FROM employees10;
```

	name	_c1
0	John Smith	0.10000
1	Mary Smith	0.10000
2	Jason Yang	0.10000
3	Ted Wang	0.10000
4	Mike McPheron	0.10000

```
hive> select name, deductions["Insurance"] from employees10;
OK
John Smith      0.1
Mary Smith      0.1
Jason Yang      0.1
Ted Wang        0.1
Mike McPheron   0.1
```

Reference Elements of Collections: Struct

Use Dot Notation

```
SELECT name, address.city, address.state FROM employees10;
```

	name	city	state
0	John Smith	Chicago	IL
1	Mary Smith	Chicago	IL
2	Jason Yang	Ithaca	NY
3	Ted Wang	Ithaca	NY
4	Mike McPheron	Ithaca	IL

```
hive> select name, address.city, address.state from employees10;
OK
John Smith      Chicago IL
Mary Smith      Chicago IL
Jason Yang      Ithaca  NY
Ted Wang        Ithaca  NY
Mike McPheron   Ithaca  IL
```

Computing with Column Values

To do: select the employees' names converted to uppercase, their salaries, actual amount paid after the deductions, rounded to two decimal points.

```
SELECT upper(name), salary,
round(salary * (1 - deductions["Federal Taxes"]
-deductions["State Taxes"]-deductions["Insurance"]),2)
FROM employees23;
```

```
hive> select upper(name), salary, round(salary*(1-deductions["Federal Taxes"]
    > -deductions["State Taxes"]-deductions["Insurance"]),2) from employees23;
OK
JOANN FRASER    100000.0        65000.0
ELISE SMITH     80000.0 52000.0
JASON YANG      150000.0        97500.0
TED WANG        190000.0        123499.99
TOM MCPHERON    210000.0        174300.0
JOHN SMITH      100000.0        65000.0
MARY SMITH      80000.0 52000.0
JASON JONES     150000.0        97500.0
BOB MCPHERON    150000.0        124500.0
SAM JOHNSON     100000.0        65000.0
ELISE SMITH     80000.0 52000.0
JASON YANG      150000.0        97500.0
TED WANG        120000.0        78000.0
TOM MCPHERON    150000.0        124500.0
```

Hive Built-in Functions

- Hive has a large number of built-in functions. The reference can be found at the following link: https://cwiki.apache.org/confluence/display/Hive/LanguageManual+UDF#LanguageManualUDF-Built-inFunctions
- There are many types of functions:
  - Basic mathematical functions
  - Aggregate functions
  - String functions
  - Map functions
  - Array functions
  - Timestamp functions
  - …

Important Built-in Math Functions

Return type	Function	Description
BIGINT	round(d)	Return BIGINT for the rounded value of DOUBLE d
DOUBLE	round(d, N)	Return DOUBLE for value d, rounded to N decimal places
DOUBLE	exp(d)	Return e to the d
DOUBLE	ln(d)	Return the natural logarithm of d
DOUBLE	log10(d)	Return the base-10 logarithm of d
DOUBLE	log2(d)	Return the base-2 logarithm of d
DOUBLE	log(base, d)	Return the base-base logarithm of d
DOUBLE	pow(d,p), power(d, p)	Return d raised to the power p
DOUBLE	sqrt(d)	Return the square root of d
DOUBLE	abs(d)	Return the absolute value of d
DOUBLE	degrees(d)	Return the value of d converted from radians to degrees.
DOUBLE	radians(d)	Return the value of d converted from degrees to radians.
DOUBLE	pi()	Return the value of constant pi, 3.141592653589793

Important Built-in Aggregate Functions

Return type	Function	Description
BIGINT	count(*)	Return the total number of retrieved rows, including rows containing NULL values
BIGINT	count(expr)	Return the number of rows for which the supplied expression is not NULL
BIGINT	count(DISTINCT expr[, expr_.])	Return the number of rows for which the supplied expression(s) are unique and not NULL
DOUBLE	sum(col), sum(DISTINCT col)	Return sum of the elements or sum of distinct values in a group
DOUBLE	avg(col), avg(DISTINCT col)	Return average of the elements or average of distinct values in a group
DOUBLE	min(col), max(col)	Return the minimum or maximum of the values
DOUBLE	variance(col)	Return variance of a set of numbers
DOUBLE	var_samp(col)	Return the sample variance of a set of numbers
DOUBLE	stddev_pop(col)	Return standard deviation of a set of numbers
DOUBLE	stddev_samp(col)	Return sample standard deviation of a set of num
	corr(col1, col2)	Return the correlation of two sets of numbers
ARRAY	collect_set(col)	Return a set with duplicate elements removed

Other Built-in Functions

Return type	Function	Description
BOOLEAN	test in(va1, val2, …)	Return true if test is one of the values in the list
INT	length(s)	Return the length of the string
STRING	reverse(s)	Return a reverse copy of the string
STRING	concat(s1, s2, ...)	Return a concatenation of s1, s2, …
STRING	concat_ws(separator , s1, s2, ...)	Like concat, but using the specified separator.
STRING	substr(s, start_index)	Return the substring of s from start_index position, where 1 is the index of the first character, to the end
STRING	substr(s, int start, int length)	Return the substring of s starting from the start position with the given length
STRING	trim(s)	Return string by removing whitespace at both ends
INT	size(map),size(array)	Return the number of elements in the map or array
<type>	cast(expr as type)	Convert (“cast”) the result of the expr to *type*
ARRAY<STR>	split(s, pattern)	Returns an array of substrings of s, split at pattern
INT	instr(str, substr)	Returns the index of str where substr is found
MAP<S, S>	str_to_map(s, d1,d2)	Creates a map, d1 for k-v pairs, d2 for k-v separators.

The LIMIT Clause

The LIMIT clause puts an upper limit on the number of rows returned by a query.

```
SELECT upper(name), salary,
round(salary * (1 - deductions["Federal Taxes"]
-deductions["State Taxes"]-deductions["Insurance"]),2)
FROM employees23
LIMIT 5;
```

```
JOANN FRASER     100000.0          65000.0
ELISE SMITH      80000.0 52000.0
JASON YANG       150000.0          97500.0
TED WANG         190000.0          123499.99
TOM MCPHERON     210000.0          174300.0
```

Column Aliases

We can give anonymous columns a name, called a column alias, in a query.

```
SELECT name, salary, round(salary * (1 - deductions["Federal Taxes"]
-deductions["State Taxes"]-deductions["Insurance"]),1) as Actually_Paid
FROM employees23
LIMIT 10;
```

	name	salary	actually_paid
0	Joann Fraser	100000.0	65000.0
1	Elise Smith	80000.0	52000.0
2	Jason Yang	150000.0	97500.0
3	Ted Wang	190000.0	123500.0
4	Tom McPheron	210000.0	174300.0
5	John Smith	100000.0	65000.0
6	Mary Smith	80000.0	52000.0
7	Jason Jones	150000.0	97500.0
8	Bob McPheron	150000.0	124500.0
9	Sam Johnson	100000.0	65000.0

Nested SELECT Statements

The result of one query is used by another query. The column alias feature is especially useful in nested select statements.

```
FROM (
  SELECT name, salary, round(salary * (1 - deductions["Federal Taxes"]
  -deductions["State Taxes"]-deductions["Insurance"]),1) as Actually_Paid
  FROM employees23
) ap
SELECT ap.name, ap.Actually_Paid
WHERE ap.Actually_Paid > 80000;
```

	ap.name	ap.actually_paid
0	Jason Yang	97500.0
1	Ted Wang	123500.0
2	Tom McPheron	174300.0
3	Jason Jones	97500.0
4	Bob McPheron	124500.0
5	Jason Yang	97500.0
6	Tom McPheron	124500.0

Another Form of SELECT Statement

```
SELECT ap.name, ap.Actually_Paid FROM
(
  SELECT name, salary, round(salary * (1 - deductions["Federal Taxes"]
  -deductions["State Taxes"]-deductions["Insurance"]),1) as Actually_Paid
  FROM employees23
) ap
WHERE ap.Actually_Paid > 80000;
```

	ap.name	ap.actually_paid
0	Jason Yang	97500.0
1	Ted Wang	123500.0
2	Tom McPheron	174300.0
3	Jason Jones	97500.0
4	Bob McPheron	124500.0
5	Jason Yang	97500.0
6	Tom McPheron	124500.0

CASE ... WHEN ... THEN Statements

The **CASE ... WHEN ... THEN clauses** are like **if statements** for individual columns in a query.

```
SELECT name, salary,
  CASE
    WHEN salary < 50000.0 THEN 'low'
    WHEN salary >= 50000.0 AND salary < 70000.0 THEN 'middle'
    WHEN salary >= 70000.0 AND salary < 100000.0 THEN 'high'
    ELSE 'very high'
  END AS bracket FROM employees23 LIMIT 5;
```

	name	salary	bracket
0	Joann Fraser	100000.0	very high
1	Elise Smith	80000.0	high
2	Jason Yang	150000.0	very high
3	Ted Wang	190000.0	very high
4	Tom McPheron	210000.0	very high

LIKE and RLIKE Operators

- The LIKE operator lets us match on strings that begin with or end with a particular substring, or when the substring appears anywhere within the string: **A LIKE B**
  - A LIKE **'x%'**: begins with the prefix 'x' (%: any number of characters)
  - A LIKE **'%x'**: ends with the suffix 'x'
  - A LIKE: **'%x%'**: begins with, ends with, or contains substring 'x'
  - The underscore '_' matches a single character: **'x_'**, **'_x'**, and **'_x_'**
- The RLIKE operator lets us use Java regular expressions, a more powerful minilanguage for specifying matches. A RLIKE B
  - A RLIKE **'.*(x|y).*'**: begins with, ends with, or contains substring 'x' or 'y'

Some Examples Using LIKE and RLIKE

```
hive> select name, address.street from employees23
    > where address.street LIKE '%Ave.';
OK
Joann Fraser    100 Bay Ave.
John Smith      1 Michigan Ave.
Bob McPheron    12 Bush Ave.
Sam Johnson     1 Square Ave.
Tom McPheron    12 Bush Ave.
```

```
hive> select name, address.city from employees23
    > where address.city  LIKE '%Be%';
OK
Joann Fraser    Long Beach
Elise Smith     Long Beach
```

```
hive> select name, address.street from employees23
    > where address.street RLIKE '.*(Michigan|Ontario).*';
OK
John Smith      1 Michigan Ave.
Mary Smith      100 Ontario St.
Elise Smith     100 Ontario St.
```

More HiveQL Clauses/Statements

- **GROUP BY** is used with aggregate functions to group the result by one or more columns and then does an aggregation over each group.
- **HAVING** constrains the groups produced by GROUP BY.
- **INNER JOIN** is only for equi-joins.
- **LEFT OUTER JOIN** returns all records from left table plus those matching the WHERE clause.
- **RIGHT OUTER JOIN** returns all records in right table plus those matching the WHERE clause.
- **FULL OUTER JOIN** returns all records from all tables plus those matching the WHERE clause.
- **LEFT SEMI-JOIN** returns records from left table if records are found in right table that satisfy the ON predicates.
- **Cartesian Product JOINs** is not executed in parallel, not optimized using M/R.
- **MAP-SIDE JOINs** do all joining on map-side and eliminate the reduce step.
- **ORDER BY** is for a total ordering of the query result set in a single reducer (slow).
- **SORT BY** is for a local ordering within each reducer (faster).
- **DISTRIBUTE BY** controls how map output is divided among reducers.
- **CLUSTER BY** is a short-hand way of DISTRIBUTED BY… SORT BY combination.
- **UNION ALL** combines two or more tables.

Join Optimizations

- Hive assumes that the last table in the query is the largest.
- It attempts to buffer the other tables and then stream the last table through, while performing joins on individual records.
- You should structure your join queries so the largest table is last.
- Fortunately, you don't have to put the largest table last in the query. Hive also provides a "hint" mechanism to tell the query optimizer which table should be streamed. **SELECT** */*+ STREAMTABLE(s) */ …;*

LEFT SEMI-JOIN

Returns records from the left table if records are found in the right table that satisfy the ON predicates. It's a special, optimized case of the more general inner join. For example, return stock records only on the days of dividend payments.

```
SELECT s.ymd, s.symbol, s.price_close
FROM stocks s LEFT SEMI JOIN dividends d
ON s.ymd = d.ymd AND s.symbol = d.symbol;
```

```
1985-04-25      ZAP     13.67
1985-01-09      ZAP     14.63
1984-10-09      ZAP     16.92
1984-07-09      ZAP     14.72
1984-04-09      ZAP     19.72
1984-01-09      ZAP     14.51
1983-10-07      ZAP     15.83
1983-07-11      ZAP     17.57
1983-04-11      ZAP     13.39
Time taken: 36.537 seconds, Fetched: 132722 row(s)
```

Map-side Joins

This is an optimization when all but one table is small. The largest table is streamed through the mappers while the small tables are cached in memory so that Hive can do all the joining on the map-side by looking up every possible match against the small tables in memory, thereby eliminating the reduce step required in common join scenarios. Processing is faster.

To do Map-side joins, first set a the following property to true:

```
set hive.auto.convert.join=true;
```

The join:

```
SELECT s.ymd, s.symbol, s.price_close, d.dividend
FROM stocks s JOIN dividends d ON s.ymd = d.ymd
AND s.symbol = d.symbol
WHERE s.symbol = 'IBM';
```

Example: SORT BY

```
hive> select s.ymd, s.symbol, s.price_close
    > from stocks s
    > sort by s.ymd asc, s.symbol desc
    > limit 18;
```

```
1962-01-02      KO      101.0
1962-01-02      IBM     572.0
1962-01-02      HPQ     35.0
1962-01-02      GE      74.75
1962-01-02      EK      110.75
1962-01-02      DIS     37.25
1962-01-02      DD      241.5
1962-01-02      CAT     38.5
1962-01-02      BA      50.0
1962-01-02      AA      65.37
1962-01-03      KO      98.75
1962-01-03      IBM     577.0
1962-01-03      HPQ     34.63
1962-01-03      GE      74.0
1962-01-03      EK      109.62
1962-01-03      DIS     37.75
1962-01-03      DD      241.75
1962-01-03      CAT     38.88
Time taken: 67.763 seconds, Fetched: 18 row(s)
```

DISTRIBUTE BY with SORT BY

These two clauses are often used together. In the following example, clause DISTRIBUTE BY is to ensure that the records for each stock symbol go to the same reducer, then use SORT BY to order the data.

```
SELECT s.ymd, s.symbol, s.price_close
FROM stocks s
DISTRIBUTE BY s.symbol
SORT BY s.symbol ASC, s.ymd ASC
limit 10;
```

```
1962-01-02      AA      65.37
1962-01-03      AA      66.37
1962-01-04      AA      66.37
1962-01-05      AA      66.25
1962-01-08      AA      64.0
1962-01-09      AA      63.75
1962-01-10      AA      63.88
1962-01-11      AA      63.5
1962-01-12      AA      62.0
1962-01-15      AA      60.75
Time taken: 62.771 seconds, Fetched: 10 row(s)
```

Note: DISTRIBUTE BY clause must come before the SORT BY clause.

CLUSTER BY = DISTRIBUTE BY with SORT BY

CLUSTER BY clause is a shorthand way of expressing the same query on last slide.

```
SELECT s.ymd, s.symbol, s.price_close
FROM stocks s
CLUSTER BY s.symbol, s.ymd
limit 10;
```

```
1962-01-02      AA      65.37
1962-01-03      AA      66.37
1962-01-04      AA      66.37
1962-01-05      AA      66.25
1962-01-08      AA      64.0
1962-01-09      AA      63.75
1962-01-10      AA      63.88
1962-01-11      AA      63.5
1962-01-12      AA      62.0
1962-01-15      AA      60.75
Time taken: 64.017 seconds, Fetched: 10 row(s)
```

List the Smaller Table Last

```
hive> select s.ymd, s.symbol, s.price_close, d.dividend
    > from stocks s JOIN dividends d ON s.ymd=d.ymd and s.symbol=d.symbol
    > where s.symbol='APL';
```

```
2002-12-27      APL     25.5    0.54
2002-09-26      APL     25.0    0.54
2002-06-26      APL     23.7    0.535
2002-03-26      APL     29.0    0.52
2001-12-27      APL     24.5    0.58
2001-09-26      APL     25.8    0.6
2001-06-27      APL     29.5    0.67
2001-03-28      APL     24.0    0.65
2000-12-27      APL     17.94   0.56
2000-09-27      APL     17.5    0.535
2000-06-28      APL     13.88   0.45
2000-03-29      APL     11.38   0.295
Time taken: 28.492 seconds, Fetched: 37 row(s)
```

List the Larger Table Last

```
hive> select s.ymd, s.symbol, s.price_close, d.dividend
    > from dividends d JOIN stocks s ON s.ymd=d.ymd and s.symbol=d.symbol
    > where s.symbol='APL';
```

```
2002-12-27      APL     25.5    0.54
2002-09-26      APL     25.0    0.54
2002-06-26      APL     23.7    0.535
2002-03-26      APL     29.0    0.52
2001-12-27      APL     24.5    0.58
2001-09-26      APL     25.8    0.6
2001-06-27      APL     29.5    0.67
2001-03-28      APL     24.0    0.65
2000-12-27      APL     17.94   0.56
2000-09-27      APL     17.5    0.535
2000-06-28      APL     13.88   0.45
2000-03-29      APL     11.38   0.295
Time taken: 23.732 seconds, Fetched: 37 row(s)
```

Use Hive "Hint" Mechanism to Optimize

```
hive> select /*+ STREAMTABLE(S) */ s.ymd, s.symbol, s.price_close, d.dividend
    > from stocks s JOIN dividends d ON s.ymd=d.ymd and s.symbol=d.symbol
    > where s.symbol='APL';
```

```
2002-12-27      APL     25.5    0.54
2002-09-26      APL     25.0    0.54
2002-06-26      APL     23.7    0.535
2002-03-26      APL     29.0    0.52
2001-12-27      APL     24.5    0.58
2001-09-26      APL     25.8    0.6
2001-06-27      APL     29.5    0.67
2001-03-28      APL     24.0    0.65
2000-12-27      APL     17.94   0.56
2000-09-27      APL     17.5    0.535
2000-06-28      APL     13.88   0.45
2000-03-29      APL     11.38   0.295
Time taken: 23.773 seconds, Fetched: 37 row(s)
```