Занятие 5. SQL-доступ к данным. Apache Hive.

- Развернуть CDH. Инструкция с предыдущего занятия. Или официальная документация.

- В директории /home создать и наполнить файлы:

- vi employee.txt
  Michael|Montreal,Toronto|Male,30|DB:80|Product:Developer^DLead
  Will|Montreal|Male,35|Perl:85|Product:Lead,Test:Lead
  Shelley|New York|Female,27|Python:80|Test:Lead,COE:Architect
  Lucy|Vancouver|Female,57|Sales:89,HR:94|Sales:Lead

- vi employee_id.txt
  Michael|100|Montreal,Toronto|Male,30|DB:80|Product:DeveloperLead
  Will|101|Montreal|Male,35|Perl:85|Product:Lead,Test:Lead
  Steven|102|New York|Female,27|Python:80|Test:Lead,COE:Architect
  Lucy|103|Vancouver|Female,57|Sales:89,HR:94|Sales:Lead
  Mike|104|Montreal|Male,35|Perl:85|Product:Lead,Test:Lead
  Shelley|105|New York|Female,27|Python:80|Test:Lead,COE:Architect
  Luly|106|Vancouver|Female,57|Sales:89,HR:94|Sales:Lead
  Lily|107|Montreal|Male,35|Perl:85|Product:Lead,Test:Lead
  Shell|108|New York|Female,27|Python:80|Test:Lead,COE:Architect
  Mich|109|Vancouver|Female,57|Sales:89,HR:94|Sales:Lead
  Dayong|110|Montreal|Male,35|Perl:85|Product:Lead,Test:Lead
  Sara|111|New York|Female,27|Python:80|Test:Lead,COE:Architect
  Roman|112|Vancouver|Female,57|Sales:89,HR:94|Sales:Lead
  Christine|113|Montreal|Male,35|Perl:85|Product:Lead,Test:Lead
  Eman|114|New York|Female,27|Python:80|Test:Lead,COE:Architect
  Alex|115|Vancouver|Female,57|Sales:89,HR:94|Sales:Lead
  Alan|116|Montreal|Male,35|Perl:85|Product:Lead,Test:Lead
  Andy|117|New York|Female,27|Python:80|Test:Lead,COE:Architect
  Ryan|118|Vancouver|Female,57|Sales:89,HR:94|Sales:Lead
  Rome|119|Montreal|Male,35|Perl:85|Product:Lead,Test:Lead
  Lym|120|New York|Female,27|Python:80|Test:Lead,COE:Architect
  Linm|121|Vancouver|Female,57|Sales:89,HR:94|Sales:Lead
  Dach|122|Montreal|Male,35|Perl:85|Product:Lead,Test:Lead
  Ilon|123|New York|Female,27|Python:80|Test:Lead,COE:Architect
  Elaine|124|Vancouver|Female,57|Sales:89,HR:94|Sales:Lead

- vi employee_hr.txt
  Michael|100|547-968-091|2014-01-29
  Will|101|527-948-090|2013-10-02

Steven|102|647-968-598|2012-11-03
Lucy|103|577-928-094|2010-01-03

- vi employee_contract.txt
  Michael|1000|100|5000|full|2014-01-29
  Will|1000|101|4000|full|2013-10-02
  Will|1000|101|4000|part|2014-10-02
  Steven|1000|102|6400|part|2012-11-03
  Lucy|1000|103|5500|full|2010-01-03
  Lily|1001|104|5000|part|2014-11-29
  Jess|1001|105|6000|part|2014-12-02
  Mike|1001|106|6400|part|2013-11-03
  Wei|1002|107|7000|part|2010-04-03
  Yun|1002|108|5500|full|2014-01-29
  Richard|1002|109|8000|full|2013-09-01

- Запустить beeline
  beeline -u "jdbc:hive2://localhost:10000/default" --silent=true

Data definition and description

1. Data Types

- Создать таблицу

```
CREATE TABLE employee (
  name string,
  work_place ARRAY<string>,
  gender_age STRUCT<gender:string,age:int>,
  skills_score MAP<string,int>,
  depart_title MAP<STRING,ARRAY<STRING>>
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '|'
COLLECTION ITEMS TERMINATED BY ','
MAP KEYS TERMINATED BY ':'
STORED AS TEXTFILE;
```

- Проверить таблицу

```
!table employee
!column employee
```

- Грузим данные

```
LOAD DATA LOCAL INPATH 'home/employee.txt' OVERWRITE INTO TABLE
employee;
```

- Запускаем запросы

```
--Query the whole table
SELECT * FROM employee;

--Query the ARRAY in the table
SELECT work_place FROM employee;

SELECT work_place[0] AS col_1, work_place[1] AS col_2, work_place[2] AS col_3 FROM
employee;

--Query the STRUCT in the table
SELECT gender_age FROM employee;

SELECT gender_age.gender, gender_age.age FROM employee;

--Query the MAP in the table
SELECT skills_score FROM employee;

SELECT name, skills_score['DB'] AS DB,
skills_score['Perl'] AS Perl, skills_score['Python'] AS Python,
skills_score['Sales'] as Sales, skills_score['HR'] as HR FROM employee;

SELECT depart_title FROM employee;

SELECT name, depart_title['Product'] AS Product, depart_title['Test'] AS Test,
depart_title['COE'] AS COE, depart_title['Sales'] AS Sales
FROM employee;

SELECT name,
depart_title['Product'][0] AS product_col0,
depart_title['Test'][0] AS test_col0
FROM employee;
```

2. Databases

```
--Create database without checking if the database already exists.
CREATE DATABASE hivetest;

--Create database and checking if the database already exists.
CREATE DATABASE IF NOT EXISTS hivetest;

--Create database with location, comments, and metadata information
```

```
CREATE DATABASE IF NOT EXISTS hivetest
COMMENT 'hive database demo'
LOCATION '/hdfs/directory'
WITH DBPROPERTIES ('creator'='cloudera','date'='2019-07-17');

--Show and describe database with wildcards
SHOW DATABASES;
SHOW DATABASES LIKE 'hive.*';
DESCRIBE DATABASE hivetest;

--Use the database
USE hivetest;

--Show current database
SELECT current_database();

--Drop the empty database.
DROP DATABASE IF EXISTS hivetest;

--Drop database with CASCADE
DROP DATABASE IF EXISTS hivetest CASCADE;

ALTER DATABASE hivetest SET OWNER user cloudera;
```

3. Table creation

```
--Create internal table and load the data
CREATE TABLE IF NOT EXISTS employee_internal (
  name string,
  work_place ARRAY<string>,
  gender_age STRUCT<gender:string,age:int>,
  skills_score MAP<string,int>,
  depart_title MAP<STRING,ARRAY<STRING>>
)
COMMENT 'This is an internal table'
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '|'
COLLECTION ITEMS TERMINATED BY ','
MAP KEYS TERMINATED BY ':'
STORED AS TEXTFILE;

LOAD DATA LOCAL INPATH 'home/employee.txt' OVERWRITE INTO TABLE
employee_internal;
```

```
--Create external table and load the data
CREATE EXTERNAL TABLE IF NOT EXISTS employee_external (
   name string,
   work_place ARRAY<string>,
   gender_age STRUCT<gender:string,age:int>,
   skills_score MAP<string,int>,
   depart_title MAP<STRING,ARRAY<STRING>>
)
COMMENT 'This is an external table'
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '|'
COLLECTION ITEMS TERMINATED BY ','
MAP KEYS TERMINATED BY ':'
STORED AS TEXTFILE
LOCATION '/user/cloudera/employee';

LOAD DATA LOCAL INPATH 'home/employee.txt' OVERWRITE INTO TABLE
employee_external;

--Temporary tables
CREATE TEMPORARY TABLE IF NOT EXISTS tmp_emp1 (
name string,
work_place ARRAY<string>,
gender_age STRUCT<gender:string,age:int>,
skills_score MAP<string,int>,
depart_title MAP<STRING,ARRAY<STRING>>
);

CREATE TEMPORARY TABLE tmp_emp2 AS SELECT * FROM tmp_emp1;

CREATE TEMPORARY TABLE tmp_emp3 LIKE tmp_emp1;

--Create Table With Data - CREATE TABLE AS SELECT (CTAS)
CREATE TABLE ctas_employee AS SELECT * FROM employee_external;

--Create Table As SELECT (CTAS) with Common Table Expression (CTE)
CREATE TABLE cte_employee AS
WITH r1 AS (SELECT name FROM r2 WHERE name = 'Michael'),
r2 AS (SELECT name FROM employee WHERE gender_age.gender= 'Male'),
r3 AS (SELECT name FROM employee WHERE gender_age.gender= 'Female')
SELECT * FROM r1 UNION ALL select * FROM r3;

SELECT * FROM cte_employee;

--Create Table Without Data - TWO ways
--With CTAS
```

```
CREATE TABLE empty_ctas_employee AS SELECT * FROM employee_internal WHERE
1=2;

--With LIKE
CREATE TABLE empty_like_employee LIKE employee_internal;

--Check row count for both tables
SELECT COUNT(*) AS row_cnt FROM empty_ctas_employee;
SELECT COUNT(*) AS row_cnt FROM empty_like_employee;
```

4. Table description

```
--Show tables
SHOW TABLES;
SHOW TABLES '*emp*';
SHOW TABLES '*ext*|*cte*';
SHOW TABLE EXTENDED LIKE 'employee_int*';

--Show columns
SHOW COLUMNS IN employee_internal;
DESC employee_internal;

--Show DDL and property
SHOW CREATE TABLE employee_internal;
SHOW TBLPROPERTIES employee_internal;
```

5. Table clear

```
--Drop table
DROP TABLE IF EXISTS empty_ctas_employee;

DROP TABLE IF EXISTS empty_like_employee;

--Truncate table
SELECT * FROM cte_employee;

TRUNCATE TABLE cte_employee;

SELECT * FROM cte_employee;
```

6. Table ALTER

```
--Alter table name
ALTER TABLE cte_employee RENAME TO cte_employee_backup;

--Alter table properties, such as comments
```

```
ALTER TABLE cte_employee_backup SET TBLPROPERTIES ('comment' = 'New
comments');

--Alter table delimiter through SerDe properties
ALTER TABLE employee_internal SET SERDEPROPERTIES ('field.delim' = '$');

--Alter Table File Format
ALTER TABLE employee_internal SET FILEFORMAT RCFILE;

--Alter Table Location
ALTER TABLE employee_internal SET LOCATION
'hdfs://localhost:9000/user/cloudera/employee';

--Alter Table Location
ALTER TABLE employee_internal ENABLE NO_DROP;
ALTER TABLE employee_internal DISABLE NO_DROP;
ALTER TABLE employee_internal ENABLE OFFLINE;
ALTER TABLE employee_internal DISABLE OFFLINE;

--Alter Table Concatenate to merge small files into larger files
--convert to the file format supported
ALTER TABLE employee_internal SET FILEFORMAT ORC;

--convert to the regular file format
ALTER TABLE employee_internal SET FILEFORMAT TEXTFILE;


--Alter columns
--Change column type - before changes
DESC employee_internal;

--Change column type
ALTER TABLE employee_internal CHANGE name employee_name string AFTER
gender_age;

--Verify the changes
DESC employee_internal;

--Change column type
ALTER TABLE employee_internal CHANGE employee_name name string COMMENT
'updated' FIRST;

--Verify the changes
DESC ctas_employee;

--Add columns to the table
```

```
ALTER TABLE ctas_employee ADD COLUMNS (work string);

--Verify the added columns
DESC ctas_employee;

--Replace all columns
ALTER TABLE ctas_employee REPLACE COLUMNS (name string);

--Verify the replaced all columns
DESC ctas_employee;
```

7. Table Partitioning

```
--Create partition table DDL
CREATE TABLE employee_partitioned
(
  name string,
  work_place ARRAY<string>,
  gender_age STRUCT<gender:string,age:int>,
  skills_score MAP<string,int>,
  depart_title MAP<STRING,ARRAY<STRING>>
)
PARTITIONED BY (Year INT, Month INT)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '|'
COLLECTION ITEMS TERMINATED BY ','
MAP KEYS TERMINATED BY ':';

--Check partition table structure
DESC employee_partitioned;

--Show partitions
SHOW PARTITIONS employee_partitioned;

--Add multiple partitions
ALTER TABLE employee_partitioned ADD
PARTITION (year=2018, month=11)
PARTITION (year=2018, month=12);

SHOW PARTITIONS employee_partitioned;

--Drop partitions
ALTER TABLE employee_partitioned DROP PARTITION (year=2018, month=11);
```

```
ALTER TABLE employee_partitioned DROP IF EXISTS PARTITION (year=2017); -- Drop all partitions in 2017

ALTER TABLE employee_partitioned DROP IF EXISTS PARTITION (month=9);

SHOW PARTITIONS employee_partitioned;

--Rename partitions
ALTER TABLE employee_partitioned PARTITION (year=2018, month=12) RENAME TO PARTITION (year=2018,month=10);

SHOW PARTITIONS employee_partitioned;

--Load data to the partition
LOAD DATA LOCAL INPATH 'home/employee.txt'
OVERWRITE INTO TABLE employee_partitioned
PARTITION (year=2018, month=12);

--Verify data loaded
SELECT name, year, month FROM employee_partitioned;

--Partition table add columns
ALTER TABLE employee_partitioned ADD COLUMNS (work string) CASCADE;

--Change data type for partition columns
ALTER TABLE employee_partitioned PARTITION COLUMN(year string);
--Verify the changes
DESC employee_partitioned;

ALTER TABLE employee_partitioned PARTITION (year=2018) SET FILEFORMAT ORC;
ALTER TABLE employee_partitioned PARTITION (year=2018) ENABLE NO_DROP;
ALTER TABLE employee_partitioned PARTITION (year=2018) ENABLE OFFLINE;
ALTER TABLE employee_partitioned PARTITION (year=2018) DISABLE NO_DROP;
ALTER TABLE employee_partitioned PARTITION (year=2018) DISABLE OFFLINE;
ALTER TABLE employee_partitioned PARTITION (year=2018) CONCATENATE;
ALTER TABLE employee_partitioned PARTITION (year=2018) SET FILEFORMAT TEXTFILE;
```

8. Table bucketing

```
--Prepare data for bucket tables
CREATE TABLE employee_id
(
  name string,
```

```
  employee_id int,
  work_place ARRAY<string>,
  gender_age STRUCT<gender:string,age:int>,
  skills_score MAP<string,int>,
  depart_title MAP<STRING,ARRAY<STRING>>
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '|'
COLLECTION ITEMS TERMINATED BY ','
MAP KEYS TERMINATED BY ':';

LOAD DATA LOCAL INPATH '/home/employee_id.txt' OVERWRITE INTO TABLE
employee_id;

--Create the bucket table
CREATE TABLE employee_id_buckets
(
  name string,
  employee_id int,
  work_place ARRAY<string>,
  gender_age STRUCT<gender:string,age:int>,
  skills_score MAP<string,int>,
  depart_title MAP<STRING,ARRAY<STRING>>
)
CLUSTERED BY (employee_id) INTO 2 BUCKETS
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '|'
COLLECTION ITEMS TERMINATED BY ','
MAP KEYS TERMINATED BY ':';

set map.reduce.tasks = 2;

set hive.enforce.bucketing = true;

INSERT OVERWRITE TABLE employee_id_buckets SELECT * FROM employee_id;

hdfs dfs -ls /user/hive/warehouse/employee_id_buckets
```

Data manipulation

1. SELECT

```
/* 1. SELECT */
--Query all columns in the table
SELECT * FROM employee;
```

```
--Select only one column
SELECT name FROM employee;

--List columns meet java regular expression
SET hive.support.quoted.identifiers = none;
SELECT `^work.*` FROM employee;

--Select unique rows
SELECT DISTINCT name, work_place FROM employee;

--Select with UDF, IF, and CASE WHEN
SELECT
CASE WHEN gender_age.gender = 'Female' THEN 'Ms.'
ELSE 'Mr.' END as title,
name,
IF(array_contains(work_place, 'New York'), 'US', 'CA') as country
FROM employee;

--Nest SELECT after the FROM
SELECT name, gender_age.gender AS gender
FROM(
SELECT * FROM employee
WHERE gender_age.gender = 'Male'
) t1;

--Nest SELECT using CTE
WITH t1 AS (
SELECT * FROM employee
WHERE gender_age.gender = 'Male')
SELECT name, gender_age.gender AS gender FROM t1;

--Select with expression
SELECT concat('1','+','3','=',cast((1 + 3) as string)) as res;

--Filter data with limit
SELECT name FROM employee LIMIT 2;

--Filter with Where
SELECT name, work_place FROM employee WHERE name = 'Michael';

--Filter with Where and Limit
SELECT name, work_place FROM employee WHERE name = 'Michael' LIMIT 1;

--Filter with in
SELECT name FROM employee WHERE gender_age.age in (27, 30);
```

```
--Subquery in
SELECT name, gender_age.gender AS gender
FROM employee a
WHERE a.name IN (SELECT name FROM employee WHERE gender_age.gender = 'Male');

--Subquery exists
SELECT name, gender_age.gender AS gender
FROM employee a
WHERE EXISTS
(SELECT * FROM employee b WHERE a.gender_age.gender = b.gender_age.gender AND
b.gender_age.gender = 'Male');
```

2. JOIN

```
/* 2. JOIN */

--Prepare another table for join and load data
CREATE TABLE IF NOT EXISTS employee_hr
(
  name string,
  employee_id int,
  sin_number string,
  start_date date
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '|'
STORED AS TEXTFILE;

LOAD DATA LOCAL INPATH '/home/employee_hr.txt' OVERWRITE INTO TABLE
employee_hr;

--Equal JOIN between two tables
SELECT emp.name, emph.sin_number
FROM employee emp
JOIN employee_hr emph ON emp.name = emph.name;

--Join with complex expression - conditional join
SELECT
emp.name, emph.sin_number
FROM employee emp
JOIN employee_hr emph ON
IF(emp.name = 'Will', '1', emp.name) = CASE WHEN emph.name = 'Will' THEN '0' ELSE
emph.name END;
```

```
-- Use Where to limit the output of join
SELECT
emp.name, emph.sin_number
FROM employee emp
JOIN employee_hr emph ON emp.name = emph.name
WHERE
emp.name = 'Will';

--JOIN between more tables
SELECT emp.name, empi.employee_id, emph.sin_number
FROM employee emp
JOIN employee_hr emph ON emp.name = emph.name
JOIN employee_id empi ON emp.name = empi.name;

--Self join is used when the data in the table has nest logic
SELECT emp.name
FROM employee emp
JOIN employee emp_b
ON emp.name = emp_b.name;

--Implicit join, which support since Hive 0.13.0
SELECT emp.name, emph.sin_number
FROM employee emp, employee_hr emph
WHERE emp.name = emph.name;

--Left JOIN
SELECT emp.name, emph.sin_number
FROM employee emp
LEFT JOIN employee_hr emph ON emp.name = emph.name;

--Right JOIN
SELECT emp.name, emph.sin_number
FROM employee emp
RIGHT JOIN employee_hr emph ON emp.name = emph.name;

--Full OUTER JOIN
SELECT emp.name, emph.sin_number
FROM employee emp
FULL JOIN employee_hr emph ON emp.name = emph.name;

--CROSS JOIN in different ways
SELECT emp.name, emph.sin_number
FROM employee emp
CROSS JOIN employee_hr emph;

SELECT emp.name, emph.sin_number
```

```
FROM employee emp
JOIN employee_hr emph;

SELECT emp.name, emph.sin_number
FROM employee emp
JOIN employee_hr emph on 1=1;

--unequal JOIN
SELECT emp.name, emph.sin_number
FROM employee emp
CROSS JOIN employee_hr emph WHERE emp.name <> emph.name;

--LEFT SEMI JOIN
SELECT a.name
FROM employee a
WHERE EXISTS
(SELECT * FROM employee_id b
WHERE a.name = b.name);

SELECT a.name
FROM employee a
LEFT SEMI JOIN employee_id b
ON a.name = b.name;
```

3. UNION

```
/* 3. UNION */

--UNION ALL including duplications
SELECT a.name as nm
FROM employee a
UNION ALL
SELECT b.name as nm
FROM employee_hr b;

--Order with UNION
SELECT a.name as nm FROM employee a
UNION ALL
SELECT b.name as nm FROM employee_hr b
ORDER BY nm;

--Table employee implements INTERCEPT employee_hr
SELECT a.name
FROM employee a
JOIN employee_hr b
ON a.name = b.name;
```

```
--Table employee implements MINUS employee_hr
SELECT a.name
FROM employee a
LEFT JOIN employee_hr b
ON a.name = b.name
WHERE b.name IS NULL;
```

4. INSERT EXPORT IMPORT

```
--Insert specified columns
CREATE TABLE emp_simple( -- Create a test table only has primary types
name string,
work_place string
);

--Insert values
INSERT INTO TABLE emp_simple VALUES ('Michael', 'Toronto'),('Lucy', 'Montreal');
SELECT * FROM emp_simple;

DROP TABLE IF EXISTS ctas_employee ;
CREATE TABLE ctas_employee AS SELECT * FROM employee_external;

--INSERT from CTE
WITH a as (SELECT * FROM ctas_employee )
FROM a
INSERT OVERWRITE TABLE employee
SELECT *;

--Create partition table DDL
DROP TABLE IF EXISTS employee_partitioned ;
CREATE TABLE employee_partitioned
(
  name string,
  work_place ARRAY<string>,
  gender_age STRUCT<gender:string,age:int>,
  skills_score MAP<string,int>,
  depart_title MAP<STRING,ARRAY<STRING>>
)
PARTITIONED BY (Year INT, Month INT)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '|'
COLLECTION ITEMS TERMINATED BY ','
MAP KEYS TERMINATED BY ':';
```

```
--Dynamic partition is not enabled by default. We need to set following to make it work.
SET hive.exec.dynamic.partition=true;
SET hive.exec.dynamic.partition.mode=nostrict;

--Dynamic partition insert
INSERT INTO TABLE employee_partitioned PARTITION(year, month)
SELECT name, array('Toronto') as work_place,
named_struct("gender","Male","age",30) as gender_age,
map("Python",90) as skills_score,
map("R&D",array('Developer')) as depart_title,
year(start_date) as year, month(start_date) as month
FROM employee_hr eh
WHERE eh.employee_id = 102;

SHOW PARTITIONS employee_partitioned;

--Verify the inserted row
SELECT name,depart_title,year,month FROM employee_partitioned
WHERE name = 'Steven';

--Export data and metadata of table
EXPORT TABLE employee TO '/tmp/output';

hdfs dfs -ls -R /tmp/output/

--Import as new table
IMPORT TABLE employee_imported FROM '/tmp/output';

--Import as external table
IMPORT EXTERNAL TABLE empolyee_imported_external
FROM '/tmp/output'
LOCATION '/tmp/outputext' ; --Note, LOCATION property is optional.
```

5. ORDER, SORT, DISTRIBUTE BY, CLUSTER BY

```
--ORDER, SORT
SELECT name FROM employee ORDER BY name DESC;

--Use more than 1 reducer
SET mapred.reduce.tasks = 2;

SELECT name FROM employee SORT BY name DESC;

--Use only 1 reducer
SET mapred.reduce.tasks = 1;
```

```
SELECT name FROM employee SORT BY name DESC;

--Distribute by
SELECT name, employee_id
FROM employee_hr DISTRIBUTE BY employee_id ;

--Used with SORT BY
SELECT name, start_date FROM employee_hr DISTRIBUTE BY start_date SORT BY
name;

--Cluster by
SELECT name, employee_id FROM employee_hr CLUSTER BY name ;
```

Data aggregation

1. Basic aggregation

```
--Aggregation without GROUP BY columns
SELECT count(*) as rowcnt1, count(1) AS rowcnt2 FROM employee;

--Aggregation with GROUP BY columns
SELECT gender_age.gender, count(*) AS row_cnt FROM employee
GROUP BY gender_age.gender;

--Multiple aggregate functions are called in the same SELECT
SELECT gender_age.gender, AVG(gender_age.age) AS avg_age,
count(*) AS row_cnt FROM employee GROUP BY gender_age.gender;

--Aggregate functions are used with CASE WHEN
SELECT sum(CASE WHEN gender_age.gender = 'Male' THEN gender_age.age
ELSE 0 END)/count(CASE WHEN gender_age.gender = 'Male' THEN 1
ELSE NULL END) AS male_age_avg FROM employee;

--Aggregate functions are used with COALESCE and IF
SELECT
sum(coalesce(gender_age.age,0)) AS age_sum,
sum(if(gender_age.gender = 'Female',gender_age.age,0))
AS female_age_sum FROM employee;

--Nested aggregate functions are not allowed
--FAILED: SemanticException [Error 10128]: Line 1:11 Not yet supported place for UDAF
'count'
--SELECT avg(count(*)) AS row_cnt FROM employee;
```

```
--Aggregate functions cannot apply to null
--SELECT sum(null), avg(null);

--Aggregate functions can be also used with DISTINCT keyword to do aggregation on unique values.
SELECT count(distinct gender_age.gender) AS gender_uni_cnt, count(distinct name) AS name_uni_cnt FROM employee;

--Use max/min struct
SELECT gender_age.gender,
max(struct(gender_age.age, name)).col1 as age,
max(struct(gender_age.age, name)).col2 as name
FROM employee
GROUP BY gender_age.gender;

--Use subquery to select unique value before aggregations for better performance
SELECT count(*) AS gender_uni_cnt FROM (SELECT distinct gender_age.gender FROM employee) a;

--Grouping Set
SELECT
name,
start_date,
count(sin_number) as sin_cnt
FROM employee_hr
GROUP BY name, start_date
GROUPING SETS((name, start_date));
--||-- equals to
SELECT
name,
start_date,
count(sin_number) AS sin_cnt
FROM employee_hr
GROUP BY name, start_date;

SELECT
name, start_date, count(sin_number) as sin_cnt
FROM employee_hr
GROUP BY name, start_date
GROUPING SETS(name, start_date);
--||-- equals to
SELECT
name, null as start_date, count(sin_number) as sin_cnt
FROM employee_hr
GROUP BY name
```

```
UNION ALL
SELECT
null as name, start_date, count(sin_number) as sin_cnt
FROM employee_hr
GROUP BY start_date;

SELECT
name, start_date, count(sin_number) as sin_cnt
FROM employee_hr
GROUP BY name, start_date
GROUPING SETS((name, start_date), name);
--||-- equals to
SELECT
name, start_date, count(sin_number) as sin_cnt
FROM employee_hr
GROUP BY name, start_date
UNION ALL
SELECT
name, null as start_date, count(sin_number) as sin_cnt
FROM employee_hr
GROUP BY name;

--Aggregation condition – HAVING
SELECT gender_age.age FROM employee GROUP BY gender_age.age HAVING
count(*)=1;
SELECT gender_age.age, count(*) as cnt FROM employee GROUP BY gender_age.age
HAVING cnt=1;
```

2. Window functions

```
--Prepare table and data for demonstration
CREATE TABLE IF NOT EXISTS employee_contract
(
name string,
dept_num int,
employee_id int,
salary int,
type string,
start_date date
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '|'
STORED AS TEXTFILE;
```

```
LOAD DATA LOCAL INPATH
'/home/employee_contract.txt'
OVERWRITE INTO TABLE employee_contract;

--window aggregate functions
SELECT
name,
dept_num as deptno,
salary,
count(*) OVER (PARTITION BY dept_num) as cnt,
sum(salary) OVER(PARTITION BY dept_num ORDER BY dept_num) as sum1,
sum(salary) OVER(ORDER BY dept_num) as sum2
FROM employee_contract
ORDER BY deptno, name;

--window sorting functions
SELECT
name,
dept_num as deptno,
salary,
row_number() OVER () as rnum,
rank() OVER (PARTITION BY dept_num ORDER BY salary) as rk,
dense_rank() OVER (PARTITION BY dept_num ORDER BY salary) as drk,
percent_rank() OVER(PARTITION BY dept_num ORDER BY salary) as prk,
ntile(4) OVER(PARTITION BY dept_num ORDER BY salary) as ntile
FROM employee_contract
ORDER BY deptno, name;

--window analytics function
SELECT
name,
dept_num as deptno,
salary,
round(cume_dist() OVER (PARTITION BY dept_num ORDER BY salary), 2) as cume,
lead(salary, 2) OVER (PARTITION BY dept_num ORDER BY salary) as lead,
lag(salary, 2, 0) OVER (PARTITION BY dept_num ORDER BY salary) as lag,
first_value(salary) OVER (PARTITION BY dept_num ORDER BY salary) as fval,
last_value(salary) OVER (PARTITION BY dept_num ORDER BY salary) as lvalue,
last_value(salary) OVER (PARTITION BY dept_num ORDER BY salary RANGE BETWEEN
UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) AS lvalue2
FROM employee_contract
ORDER BY deptno, salary;

--window expression preceding and following
SELECT
name, dept_num as dno, salary AS sal,
```

```
max(salary) OVER (PARTITION BY dept_num ORDER BY name ROWS BETWEEN 2
PRECEDING AND CURRENT ROW) win1,
max(salary) OVER (PARTITION BY dept_num ORDER BY name ROWS BETWEEN 2
PRECEDING AND UNBOUNDED FOLLOWING) win2,
max(salary) OVER (PARTITION BY dept_num ORDER BY name ROWS BETWEEN 1
PRECEDING AND 2 FOLLOWING) win3,
max(salary) OVER (PARTITION BY dept_num ORDER BY name ROWS BETWEEN 2
PRECEDING AND 1 PRECEDING) win4,
max(salary) OVER (PARTITION BY dept_num ORDER BY name ROWS BETWEEN 1
FOLLOWING AND 2 FOLLOWING) win5,
max(salary) OVER (PARTITION BY dept_num ORDER BY name ROWS 2 PRECEDING)
win6,
max(salary) OVER (PARTITION BY dept_num ORDER BY name ROWS UNBOUNDED
PRECEDING) win7
FROM employee_contract
ORDER BY dno, name;

--window expression current_row
SELECT
name, dept_num as dno, salary AS sal,
max(salary) OVER (PARTITION BY dept_num ORDER BY name ROWS BETWEEN
CURRENT ROW AND CURRENT ROW) win8,
max(salary) OVER (PARTITION BY dept_num ORDER BY name ROWS BETWEEN
CURRENT ROW AND 1 FOLLOWING) win9,
max(salary) OVER (PARTITION BY dept_num ORDER BY name ROWS BETWEEN
CURRENT ROW AND UNBOUNDED FOLLOWING) win10,
max(salary) OVER (PARTITION BY dept_num ORDER BY name ROWS BETWEEN
UNBOUNDED PRECEDING AND 1 PRECEDING) win11,
max(salary) OVER (PARTITION BY dept_num ORDER BY name ROWS BETWEEN
UNBOUNDED PRECEDING AND CURRENT ROW) win12,
max(salary) OVER (PARTITION BY dept_num ORDER BY name ROWS BETWEEN
UNBOUNDED PRECEDING AND 1 FOLLOWING) win13,
max(salary) OVER (PARTITION BY dept_num ORDER BY name ROWS BETWEEN
UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) win14
FROM employee_contract
ORDER BY dno, name;

--window reference
SELECT name, dept_num, salary,
MAX(salary) OVER w1 AS win1,
MAX(salary) OVER w2 AS win2,
MAX(salary) OVER w3 AS win3
FROM employee_contract
WINDOW
w1 as (PARTITION BY dept_num ORDER BY name ROWS BETWEEN 2 PRECEDING
AND CURRENT ROW),
```

```
w2 as w3,
w3 as (PARTITION BY dept_num ORDER BY name ROWS BETWEEN 1 PRECEDING
AND 2 FOLLOWING)
;
```