ISIT312/ISIT912 Big Data Management

Spring 2023

Hive Operations and Data Warehouse Designs

In this practice, you will learn how to create partitioned tables and bucket tables, how to export and import tables in Hive, and how to use UMLet to create a conceptual and logical schema of a data warehouse.

Warning: DO NOT attempt to copy the Linux commands in this document to your working Terminal, because it is error-prone. Type those commands by yourself.

Laboratory Instructions.

Part A.

(0) Start Hadoop and Hive

Note. As in the previous lab, you can use ***beeline*** to interact with Hive.

Start five Hadoop services, and Hive Metastore and Hive Server 2 (see the previous laboratory).

(1) How to create a partitioned internal table ?

Create the following internal table to store information about the items.

```
create table item(
 code  char(7),
 name   varchar(30),
 brand  varchar(30),
 price  decimal(8,2) )
    row format delimited fields terminated by ','
    stored as textfile;
```

Next create a text file `item.txt` with sample data given below and save a file in a folder where you plan to keep HQL scripts from this lab. it is assumed that you have already started Hive Server 2 from this folder.

```
B000001,bolt,Golden Bolts,12.34
B000002,bolt,Platinum Parts,20.0
B000003,bolt,Unbreakable Spares,17.25
S000001,screw,Golden Bolts,45.00
S000002,screw,Platinum Parts,12.0
N000001,nut,Platinum Parts,21.99
```

When ready process the following HQL statement to load the contents of a file `item.txt` into `item` table.

```
load data local inpath '.../item.txt' into table item;
```

"`...`" is your path to a file `item.txt`. Verify the contents of a table `item` with a simple:

```
select *
from item;
```

We would like to improve performance of processing a table `item` through partitioning. We expect that a lot of queries will retrieve only the items that have a given name, like for example a query

```
select min(price)
from item
where name ='bolt';
```

Therefore, we create a new table `pitem` as a table partitioned over a column `name`. Then, when processing a query with an equality condition on `name`, like `name ='bolt'` the system will only read a partition where the rows have a string `'bolt'` in a column `name`. The system will not read an entire table. Process the following statement to create a partitioned table `pitem`:

```
create table pitem(
 code  char(7),
 brand  varchar(30),
 price  decimal(8,2) )
    partitioned by (name varchar(30))
    row format delimited fields terminated by ','
    stored as textfile;
```

Note, that a column `name` has been removed from a list of columns in the table and added as a *partition key*.

Next, we shall create the partitions for the values in a column `name` of `item` table. Process the following `alter table` statements to create the partitions for `bolt`, `screw`, and `nut`.

```
alter table pitem add partition (name='bolt');
alter table pitem add partition (name='screw');
alter table pitem add partition (name='nut');
```

Next, process the following statement to check if all partitions have been created.

```
show partitions pitem;
```

Now, we shall copy data from a table `item` into a partitioned table `pitem`. Process the following `INSERT` statements.

```
insert into table pitem partition (name='bolt')
 select code, brand, price
 from item
 where name='bolt';

insert into table pitem partition (name='screw')
 select code, brand, price
 from item
 where name='screw';

insert into table pitem partition (name='nut')
 select code, brand, price
 from item
 where name='nut';
```

Finally, try:

```
select *
from pitem;
```

to check if all data has been copied.

(2*) How to "explain" a query processing plan ?

Are there any differences in the ways how Hive processes an internal table and an internal partitioned table ?

The `explain` statement can be used to find the differences in processing of the following `select` statements.

```
explain extended select * from item where name='bolt';
explain extended select * from pitem where name='bolt';
```

The first statement processed on a table item returns the following processing plan.

```
+----------------------------------------------------------------------------------------------------------------------------------------------+--+
|                                                           Explain                                                                            |
+----------------------------------------------------------------------------------------------------------------------------------------------+--+
| STAGE DEPENDENCIES:                                                                                                                          |
|   Stage-0 is a root stage                                                                                                                    |
|                                                                                                                                              |
| STAGE PLANS:                                                                                                                                 |
|   Stage: Stage-0                                                                                                                             |
|     Fetch Operator                                                                                                                           |
|       limit: -1                                                                                                                              |
|       Processor Tree:                                                                                                                        |
|         TableScan                                                                                                                            |
|           alias: item                                                                                                                        |
|           Statistics: Num rows: 1 Data size: 203 Basic stats: COMPLETE Column stats: NONE                                                    |
|           GatherStats: false                                                                                                                 |
|           Filter Operator                                                                                                                    |
|             isSamplingPred: false                                                                                                            |
|             predicate: (UDFToString(name) = 'bolt') (type: boolean)                                                                          |
|             Statistics: Num rows: 1 Data size: 203 Basic stats: COMPLETE Column stats: NONE                                                  |
|             Select Operator                                                                                                                  |
|               expressions: code (type: char(7)), 'bolt' (type: varchar(30)), brand (type: varchar(30)), price (type: decimal(8,2))          |
|               outputColumnNames: _col0, _col1, _col2, _col3                                                                                  |
|               Statistics: Num rows: 1 Data size: 203 Basic stats: COMPLETE Column stats: NONE                                               |
|               ListSink                                                                                                                       |
|                                                                                                                                              |
+----------------------------------------------------------------------------------------------------------------------------------------------+--+
22 rows selected (0.294 seconds)
```

The second statement processed on a partitioned table `pitem` returns a plan.

```
+----------------------------------------------------------------------------------------------------------------------------------------------+--+
|                                                           Explain                                                                            |
+----------------------------------------------------------------------------------------------------------------------------------------------+--+
| STAGE DEPENDENCIES:                                                                                                                          |
|   Stage-0 is a root stage                                                                                                                    |
|                                                                                                                                              |
| STAGE PLANS:                                                                                                                                 |
|   Stage: Stage-0                                                                                                                             |
|     Fetch Operator                                                                                                                           |
|       limit: -1                                                                                                                              |
|       Partition Description:                                                                                                                 |
|           Partition                                                                                                                          |
|             input format: org.apache.hadoop.mapred.TextInputFormat                                                                           |
|             output format: org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat                                                        |
|             partition values:                                                                                                                |
|               name bolt                                                                                                                      |
|             properties:                                                                                                                      |
|               COLUMN_STATS_ACCURATE {"BASIC_STATS":"true"}                                                                                   |
|               bucket_count -1                                                                                                                |
|               columns code,brand,price                                                                                                       |
|               columns.comments                                                                                                               |
|               columns.types char(7):varchar(30):decimal(8,2)                                                                                 |
|               field.delim ,                                                                                                                  |
|               file.inputformat org.apache.hadoop.mapred.TextInputFormat                                                                      |
|               file.outputformat org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat                                                   |
|               location hdfs://localhost:8020/user/hive/warehouse/pitem/name=bolt                                                             |
|               name default.pitem                                                                                                             |
|               numFiles 1                                                                                                                     |
|               numRows 3                                                                                                                      |
|               partition_columns name                                                                                                         |
|               partition_columns.types varchar(30)                                                                                            |
|               rawDataSize 86                                                                                                                 |
|               serialization.ddl struct pitem { char(7) code, varchar(30) brand, decimal(8,2) price}                                          |
|               serialization.format ,                                                                                                         |
|               serialization.lib org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe                                                           |
|               totalSize 89                                                                                                                   |
|               transient_lastDdlTime 1657161516                                                                                              |
|             serde: org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe                                                                        |
|                                                                                                                                              |
|               input format: org.apache.hadoop.mapred.TextInputFormat                                                                         |
|               output format: org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat                                                      |
|               properties:                                                                                                                    |
```

```
|                 bucket_count -1                                                                                               |
|                 columns code,brand,price                                                                                      |
|                 columns.comments                                                                                              |
|                 columns.types char(7):varchar(30):decimal(8,2)                                                                |
|                 field.delim ,                                                                                                 |
|                 file.inputformat org.apache.hadoop.mapred.TextInputFormat                                                     |
|                 file.outputformat org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat                                  |
|                 location hdfs://localhost:8020/user/hive/warehouse/pitem                                                      |
|                 name default.pitem                                                                                            |
|                 partition_columns name                                                                                        |
|                 partition_columns.types varchar(30)                                                                           |
|                 serialization.ddl struct pitem { char(7) code, varchar(30) brand, decimal(8,2) price}                         |
|                 serialization.format ,                                                                                        |
|                 serialization.lib org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe                                          |
|                 transient_lastDdlTime 1657161409                                                                              |
|               serde: org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe                                                       |
|               name: default.pitem                                                                                             |
|             name: default.pitem                                                                                               |
|       Processor Tree:                                                                                                         |
|         TableScan                                                                                                             |
|           alias: pitem                                                                                                        |
|           Statistics: Num rows: 3 Data size: 86 Basic stats: COMPLETE Column stats: NONE                                      |
|           GatherStats: false                                                                                                  |
|           Select Operator                                                                                                     |
|             expressions: code (type: char(7)), brand (type: varchar(30)), price (type: decimal(8,2)), 'bolt' (type: varchar(30)) |
|             outputColumnNames: _col0, _col1, _col2, _col3                                                                     |
|             Statistics: Num rows: 3 Data size: 86 Basic stats: COMPLETE Column stats: NONE                                    |
|             ListSink                                                                                                          |
|                                                                                                                               |
+-----------------------------------------------------------------------------------------------------------------------------+--+
68 rows selected (0.955 seconds)
```

Comparison of a value of statistics `Data size:` indicates a smaller amount of data processed in the second case.

```
Statistics: Num rows: 1 Data size: 203 Basic stats: COMPLETE
Column stats: NONE

Statistics: Num rows: 3 Data size: 86 Basic stats: COMPLETE Column
stats: NONE
```

(3) How the partitions are implemented in HDFS ?

To find how the partitions are implemented in HDFS, process the Terminal window the following command:

```
$HADOOP_HOME/bin/hadoop fs -ls /user/hive/warehouse/pitem
```

The partitions of `pitem` tables are implemented as subfolders in `/user/hive/warehouse/pitem`.

(4) How to create a dynamically partitioned table ?

The partitions of a dynamically partitioned table are determined when data is loaded to the table.

Create a new table in the same way as in section (1) above.

```
create table dpitem(
 code  char(7),
 brand  varchar(30),
 price  decimal(8,2) )
    partitioned by (name varchar(30))
    row format delimited fields terminated by ','
    stored as textfile;
```

Next, process the following statements to set the appropriate partition parameters and to copy the rows from a table `item` into a dynamically partitioned table `dpitem`:

```
set hive.exec.dynamic.partition.mode=nonstrict;

set hive.exec.max.dynamic.partitions=5;
```

```
insert overwrite table dpitem partition(name)
 select code, brand, price, name
 from item;
```

[Note. If Zeppelin is used, the above commands need to appear in a single paragraph. If beeline is used, the above commands are processed one by one.]

Process the following statements to set the parameters that allow for dynamic partitioning, and process the statements to display the partitions and the contents of a table `dpitem`.

```
show partitions dpitem;

select *
from dpitem;
```

(4) How to create a bucket table ?

Another method to distribute a table in HDFS is to partition it into *bucket*s. A *bucket* is equivalent to a segment of file in HDFS. A column in a table can be selected to determine a distribution of rows over buckets. A value in such column in row is hashed into a number (*hash value*) and the row will be stored in a bucket with the same number. The rows that have the same hash value in a column selected for hashing (*hash key*) are stored in the same bucket. The same bucket may contain the rows with other values from a selected column when their *hash value* is the same. A concept of *bucket* is equivalent to a concept of *clustered hash index* in the traditional relational database systems.

Process the following statement to create a table `bitem` distributed over two buckets.

```
create table bitem(
 code  char(7),
 name   varchar(30),
 brand  varchar(30),
 price  decimal(8,2) )
    clustered by (name) into 2 buckets
    row format delimited fields terminated by ','
    stored as textfile;
```

Process the following statement to copy the contents of a table `item` into a table `bitem`.

```
insert overwrite table bitem
 select code, name, brand, price
 from item;
```

Display a query processing plan for a bucket table.

```
explain extended select * from bitem where name='bolt';
```

```
+----------------------------------------------------------------------------------------------------------------------------+--+
|                                                      Explain                                                               |
+----------------------------------------------------------------------------------------------------------------------------+--+
| STAGE DEPENDENCIES:                                                                                                        |
|   Stage-0 is a root stage                                                                                                  |
|                                                                                                                            |
| STAGE PLANS:                                                                                                               |
|   Stage: Stage-0                                                                                                           |
|     Fetch Operator                                                                                                         |
|       limit: -1                                                                                                            |
|       Processor Tree:                                                                                                      |
```

```
|         TableScan                                                                                                                        |
|           alias: bitem                                                                                                                   |
|           Statistics: Num rows: 6 Data size: 199 Basic stats: COMPLETE Column stats: NONE                                                |
|           GatherStats: false                                                                                                             |
|           Filter Operator                                                                                                                |
|             isSamplingPred: false                                                                                                        |
|             predicate: (UDFToString(name) = 'bolt') (type: boolean)                                                                      |
|             Statistics: Num rows: 3 Data size: 99 Basic stats: COMPLETE Column stats: NONE                                               |
|             Select Operator                                                                                                              |
|               expressions: code (type: char(7)), 'bolt' (type: varchar(30)), brand (type: varchar(30)), price (type: decimal(8,2))       |
|               outputColumnNames: _col0, _col1, _col2, _col3                                                                              |
|               Statistics: Num rows: 3 Data size: 99 Basic stats: COMPLETE Column stats: NONE                                             |
|               ListSink                                                                                                                   |
|                                                                                                                                          |
+------------------------------------------------------------------------------------------------------------------------------------------+--+
22 rows selected (0.221 seconds)
```

(5) How to export/import a table ?

To transfer Hive tables from one Hadoop installation to another one can use `export` and `import` statements. To export an internal table `item` into a folder `expitem` in HDFS process the following statement.

```
export table item to '/user/bigdata/expitem';
```

To verify the storage structures created by `export` statement process the following commands the Terminal:

```
$HADOOP_HOME/bin/hadoop fs -ls /user/bigdata
$HADOOP_HOME/bin/hadoop fs -ls /user/bigdata/expitem/data
```

Just for fun you can also list the contents of metadata.

```
$HADOOP_HOME/bin/hadoop fs -cat /user/bigdata/expitem/_metadata
```

To import a table, assume that we would like to import the contents of exported table `item` into a new table `imported_item`. Process the following statements to import data into a table `imported_item` and list the contents of the table:

```
import table imported_item from '/user/bigdata/expitem';

select * from imported_item;
```

<u>Part B.</u>

You do not need to use Virtual Machine to run UMLet 14.3. You can run UMLet 14.3 on your host operating system like you did it in the subjects CSIT115 and some of you in a subject CSCI235. However, to preserve consistency with the specifications of other laboratory classes in the subject, a specification below assumes that you still use Virtual Machine to run UMLet 14.3.

Connect to Moodle at https://moodle.uowplatform.edu.au/login/index.php and download from Moodle a file `umlet-14.3.zip` located in "Resources" section of ISIT312/912 Moodle Web site.

(1) How to start UMLet ?

Start Terminal program and in the Terminal window navigate to a location where a file `umlet-14.3.zip` has been saved. Unzip the file with a command:

```
unzip umlet-14.3.zip
```

Move to `UMLet-14.3` folder and use Linux command `chmod` to change the access rights on a file umlet.sh:

```
chmod u+x umlet.sh
```

start UMLet 14.3 in the following way:

```
./umlet.sh
```

When UMlet 14.3 started use a menu item `File->Options…` and set an option `DefaultFontfamily` to `Monospaced` and turn on an option `Show grid`. Click at `Ok` button.

Next, change a name of graphical widgets in the right upper corner of UMLet window to `ISIT312Palette`. Note, that in the future we shall also use `Logical modeling` palette.

(2) How to create a <u>conceptual</u> schema of a data warehouse ?

(2.1) Read he following specification of a sample data warehouse domain.

A multinational company consists of departments located in different countries. A department is described by a name and mission statement. The employees work on the projects. A project is described by a name and deadline. An employee is described by an employee number and full name. Employees work on projects. When an employee completes a project then he/she is re-employed by a company to work on another project. The company would like to record in a data warehouse information about the total number of employees, length of each employment, total salary paid on each employment per year, per month, per project, per department, per city and per country. For example, it should be possible to find the projects and the total number of employees working on each project, or the total number of employees employed in each in each month of a given year in each department, or average salary in each month of each year and for each project etc.

(2.2) First, we identify a fact entity and the measures.

Consider the following fragment of a specification given above.

The company would like to record in a data warehouse information about the total number of employees, length of each employment, total salary paid on each employment per year, per project, per department, per city and per country.

The fragment contributes to a fact entity `EMPLOYMENT` described by the measures `length` and `salary`.

To create a fact entity `EMPLOYMENT` drag a graphical component `fact entity` that looks like a cube (`Fact name` and `Measure1, Measure 2,…`) from a panel with the graphical widgets to the drawing area of UMLet. Next, change a name of fact from `Fact name` to `EMPLOYMENT` and measures `Measure1, Measure 2, …` to `length` and `salary` (the measures are yellow highlighted in a fragment of the specification above). To do so you have to leftclick at the fact entity to make it blue and then modify its properties in a panel "Properties" in the right lower corner of UMLet window.

(2.3) Next, we add the dimensions and attributes describing the entity types in dimensions.

The following fragment of specification indicates the existence of Time dimension that consists of entity types MONTH and YEAR (see a yellow highlighted text in a fragment below).

The company would like to record in a data warehouse information about the total number of employees, length of each employment, total salary paid on each employment per year, per month, per project, per department, per city and per country.

First, we create Time dimension that consists of the entity types MONTH and YEAR. To do so drag two entity type graphical components (Level name, attribute, …) from a panel with graphical widgets to the drawing area of UMLet. Then, change the names of entity types to MONTH and YEAR and connect the entities with one-to-many relationship graphical component, Finally, use one-to-many relationship graphical component to connect MONTH entity to a fact entity EMPLOYMENT.

Next, add an attribute name to an entity MONTH and attribute number to an entity YEAR. To do so you have to leftclick at an entity to make it blue and then modify its properties in a panel "Properties" in the right lower corner of UMLet window.

In the same way add three more dimensions: PROJECT, EMPLOYEE, and DEPARTMENT and describe each entity type with the attributes listed in the specification (see below).

A project is described by a name and deadline.

An employee is described by an employee number and full name.
A department is described by a name and mission statement.

(2.4) Finally, we create the hierarchies over the dimensions.

In this step we create Location and Time hierarchies over the dimensions DEPARTMENT and TIME(MONTH-YEAR). First add a hierarchy blob Criterion between entity type MONTH and many-to-one relationship leading to an entity YEAR. Replace a text Criterion with a text Time.

Next, create two more entity types CITY and COUNTRY and describe both of them with an attribute name. Add one-to-many relationship between COUNTRY and CITY and one-to-many relationship between CITY and DEPARTMENT.

Finally, to create Location hierarchy add a blob Criterion between entity DEPARTMENT and many-to-one relationship leading to an entity CITY. Replace a text Criterion with a text Location.

To save your design in a file employment.uxf use File->Save option from the main menu. To create pdf file use File->Export as … option from the main menu. When creating pdf file make sure that none of the graphical component in the main window of UMLet is blue highlighted !

(3) How to create a logical schema of a data warehouse ?

To create a logical schema (a collection of relational schemas) of a data warehouse we start from a diagram of conceptual schema created in the previous step. First, we change in a panel located in the

right upper corner of UMLet window a collection of graphical widgets from ISIT312Palette to Logical modeling.

Next, we add to each entity type, except fact entity EMPLOYMENT, a surrogate key. For example, we add a surrogate key year_ID to entity YEAR, month_ID to entity MONTH, project_ID to entity PROJECT, etc. We nominate all _ID attributes to be primary keys in the respective relational tables. The names of relational tables remain the same as the names of the respective entity types.

Next, we migrate the primary keys from one side of one-to-many relationships to the relational tables to many side of the relationships and we nominate the migrated _ID attribute as foreign keys.

Next, we nominate a collection of all foreign keys in a table obtained from fact entity type EMPLOYEMENT as a composite primary key in a relational table EMPLOYMENT.

Finally, we replace one-to-many relationships with arrows directed from the locations of foreign keys to the locations of the respective primary keys. A logical schema can be saved and exported in the same way as a conceptual schema.

[A sample solution will be attached separately.]
