Task 1 (2 marks)
Discovering the functionality and processing of unknow HDFS application.

Consider the available source code of Java application in a file `Unknown.java`.

Perform the following steps. Each step listed below is worth 1 mark.

(1) Read an analyse the contents of a file `Unknown.java` and discover what functionality is implemented by the unknown Java application. Insert your explanations as a comment located in the first few lines of the applications. "Few lines" means, that we expect the comprehensive explanations.

Next, insert the comments into a file `Unknown.java`, that explain step by step functionality of each line(s) such that while reading the comments it would it possible to easily understand how the application is implemented. Please note, that a comment like `"An expression on the right-hand side of assignment statement is computed and the results becomes a value of a variable on the left-hand side of assignment statement"` is an absolutely meaningless comment. The comments must explain the semantics of the Java statements in a context of the functionality of the application.

Assume, that solution will be evaluated by reading the comments one-by one and trying to understand how the application is implemented. Assume, that the comments will be read by someone who does not know how to write the computer programs in Java.

(2) At this point you should know what the functionality of the Java application is. Change a name of file `Unknown.java` and a name of application to a name `solution1.java`.

Few text files are zipped in a file `FewFiles.zip`. Compile a Java application `Unknown.java` and create a `jar` file, and use to it to process the files in `FewFiles.zip`.

Report all your Terminal commands and the output to demonstrate, that you successfully run the application. A simple way to create a report is to use Copy from Terminal Window and then Paste it into a text file and later on print it into a file `solution1.pdf`.

Deliverables
A file `solution1.java` with the explanations of the functionality of unknown application and with the comments explaining implementation of the application. A file `solution1.pdf` with the command use to compile and to process the application.

```java
import java.io.IOException;
import java.net.URI;


import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FSDataInputStream;
import org.apache.hadoop.fs.FSDataOutputStream;
import org.apache.hadoop.fs.FileStatus;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;

// The program merges all files located at a folder on a local file system and
// loads the outcomes of merge to HDFS as a single file

public class solution1 {

    public static void main(String[] args) throws IOException {

// The program has  two parameters:
//   a path to folder on a local file system with the files to be merged
//   a path and a name of file that contains the results of merge in HDFS
        String localStr = args[0];
        String hdfsStr = args[1];

// We start from creation of a an object with HDFS configuration
        Configuration conf = new Configuration();

// Next, we create handles for input folder at a local file system and
// and handle for output file in HDFS
        FileSystem hdfs  = FileSystem.get(URI.create(hdfsStr), conf);
        FileSystem local = FileSystem.getLocal(conf);

// Next, we create a string with a path a name of a folder with input files and ..
        Path inputDir = new Path(localStr);
        String folderName = inputDir.getName();
// ... a path to a file in HDFS
        Path hdfsFile = new Path(hdfsStr, folderName);

        try {
// Next, we create a list of names of files localed in a folder on a local file system
and ...
        FileStatus[] inputFiles = local.listStatus(inputDir);
// ... and a handle output file in HDFS
            FSDataOutputStream out = hdfs.create(hdfsFile);

// Next, we iterate over the files in a folder on a local file system and we copy the
files
// to a buffer and buffer is immediately written to an output file in HDFS
            for (int i=0; i<inputFiles.length; i++) {
                System.out.println(inputFiles[i].getPath().getName());
                FSDataInputStream in = local.open(inputFiles[i].getPath());
                byte buffer[] = new byte[256];
                int bytesRead = 0;
                while( (bytesRead = in.read(buffer)) > 0) {
                    out.write(buffer, 0, bytesRead);
                }
                in.close();
            }
            out.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Task 2 (2 marks)
Implementation of MapReduce application without the Reduce phase

The application described in a document `Filter.java` has the functionality equivalent to the functionality of the following SQL statement.

```
SELECT key, value
FROM Sequence-of-key-value-pairs
WHERE value > given-value;
```

The application is a MapReduce application without the Reduce phase.

An objective of this task is to use the Java code included in a file `Filter.java` to implement a MapReduce application, that has the functionality the following `SELECT` statement.

```
SELECT key, value
FROM Sequence-of-key-value-pairs
WHERE value IN (value-1, value-2, value-3);
```

Save your solution in a file `solution2.java`.

When ready, compile, create `jar` file, and process your application. Display the results created by the application. When finished, Copy and Paste the messages from a Terminal screen into a file `solution2.pdf`.

Deliverables
A file `solution2.java` with a source code of the application that implement the functionality of `SELECT` statement given above. A file `solution2.pdf` with a report from compilation, creating `jar` file, processing, and displaying the results of processing `solution2.java`.

```java
import java.io.IOException;
import java.util.StringTokenizer;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.util.GenericOptionsParser;

public class solution2 {

    public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();
        String[] otherArgs = new GenericOptionsParser(conf, args).getRemainingArgs();
        conf.set("value1", otherArgs[0]);
        conf.set("value2", otherArgs[1]);
        conf.set("value3", otherArgs[2]);
    Job job = new Job(conf, "IN Filter");
    job.setJarByClass(solution2.class);
    job.setMapperClass(FilterMapper.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
    job.setNumReduceTasks(0); // Set number of reducers to zero
    FileInputFormat.addInputPath(job, new Path(args[3]));
    FileOutputFormat.setOutputPath(job, new Path(args[4]));
    System.exit(job.waitForCompletion(true) ? 0 : 1);
    }

 public static class FilterMapper
     extends Mapper<Object, Text, Text, IntWritable>{

     private final static IntWritable counter = new IntWritable(0);
     private Text word = new Text();
     private Integer total;
     private Integer value1;
     private Integer value2;
     private Integer value3;
     public void map(Object key, Text value, Context context
             ) throws IOException, InterruptedException {
     StringTokenizer itr = new StringTokenizer(value.toString());

     value1 = Integer.parseInt( context.getConfiguration().get("value1") );
     value2 = Integer.parseInt( context.getConfiguration().get("value2") );
     value3 = Integer.parseInt( context.getConfiguration().get("value3") );

     while (itr.hasMoreTokens()) {
         word.set(itr.nextToken());
             total = Integer.parseInt(itr.nextToken());
         if ( total == value1 || total == value2 || total == value3 )
         { counter.set( total );
           context.write(word, counter); }
     }
     }
 }

}
```

Task 3 (3 marks)
Describing MapReduce implementation

Assume, that a file `customers.txt` has the following contents.

```
00001 James
00002 Harry
00003 Peter
00004 Jane
...   ...
```

The numbers in the first column represent a customer number and the names in the second column represent customer name.

Assume, that a file `orders.txt` has the following contents.

```
0000001 00001 34.5
0000002 00001 23.0
0000003 00002 123.0
0000004 00003 12.3
...     ...   ...
```

The numbers in the first column represent order number, the numbers in the second column represent customer number, and the number in the third column represent a total order value.

An objective of this task is to describe implementation of an application that finds all customers who have not submit any order yet.

Assume that both files have been loaded to HDFS. Explain would you implement Map phase and Reduce phase of MapReduce application, that lists all customers who have not submitted any orders yet.

Save you explanations in a file `solution3.pdf`. This task does not require you to write any code in Java. However, the comprehensive explanations on how to join the rows are expected. You are allowed to support your explanations with the fragments of pseudocode.

Deliverables
A file `solution3.pdf` with the comprehensive explanations on how to implement an application that finds all customers who have not submit any order yet.

Assume, that, a file `customers.txt` has the following contents and the following header.

```
cust# name
00001 James
00002 Harry
00003 Peter
00004 Jane
...   ....
```

A file `orders.txt` has the following contents and the following header

```
order#  cust# value
0000001 00001 34.5
0000002 00001 23.0
0000003 00002 123.0
0000004 00003 12.3
...     ...   ...
```

Assume that both files have been loaded to HDFS.

Implementation of Map phase

A file `customer.txt` is converted into `<key,value>` pairs where `key = cust#` and `value = name`.
A file `orders.txt` is converted into into `<key,value>` pairs where `key = cust#` and `value = order#, value`

Implementation of Reduce phase

Reduce phase operates on two files `customers` and `orders` with `<key,value>` pairs where key is exactly the same for pairs. At this step we find all `<key,value>` pairs from `customer` such that does not exist at least one `<key,value>` pairs from `orders` that has the same value of `key`. Then such `<key,value>` pairs are written to `output`.

Task 4 (3 marks)
Implementation of MapReduce application

Assume, that a bank records in a text file the withdrawals and deposits of certain amounts of money from the bank accounts. A single row in a file with the withdrawal/deposit records consists of an account number, a date when a withdrawal/deposit occurred, and an amount of money involved. Assume, that the withdrawals are represented by the negative numbers and the deposits are represent by the positive numbers and that each withdrawal/deposit modulo 50 = 0. All values in a single record are always separated with a single blank.

An objective of this task is to implement MapReduce application that finds the total amount of money deposited by each customer per year. For example, if a sample file with the withdrawals and deposits contains the following lines

```
1234567 12-DEC-2019 200
1234567 15-DEC-2019 50
9876543 25-JUL-2018 150
9876543 12-FEB-2018 -50
9876543 01-JAN-2019 150
1234567 21-OCT-2020 -250
9876543 22-OCT-2019 300
```

then your application supposed to produce the following outputs.

```
1234567 2019 250
9876543 2018 150
9876543 2019 450
```

The order of the lines listed above is up to you.

Perform the following steps.

Implement the application and save its source code in a file `solution4.java` file. A name of the file with the source code in a local file system is up to you.

Compile the Java source code and create a `jar` file.

Upload to a local file system a small file for the purpose of future testing. The file must contain the withdrawals and deposits and it must have an internal structure the same as it is explained and visualized above. A name of file and location of file in a local file system is up to you.

Use Hadoop to process your application that finds the total amount of money deposited by each customer per year.

Use Hadoop to list an input file with the withdrawals and deposits and the results produced by your application.

Deliverables

A file `solution4.java` with a source code of the application, that implements an application described above. A file `solution4.pdf` with a report from compilation, creating `jar` file, processing, and displaying the results of processing `solution4.java`.

```java
import java.io.IOException;
import java.util.StringTokenizer;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class solution4 {

  public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    Job job = Job.getInstance(conf, "Total deposit");
    job.setJarByClass(solution4.class);
    job.setMapperClass(TokenizerMapper.class);
    job.setCombinerClass(solution4Reducer.class);
    job.setReducerClass(solution4Reducer.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));
    System.exit(job.waitForCompletion(true) ? 0 : 1);
  }

  public static class TokenizerMapper
       extends Mapper<Object, Text, Text, IntWritable>{

    private Text account = new Text();
    private Text date = new Text();
    private final static IntWritable amount = new IntWritable(0);
    private Text year = new Text();
    private Text account_year = new Text();
    private Integer int_amount;

    public void map(Object key, Text value, Context context
                    ) throws IOException, InterruptedException {
      StringTokenizer itr = new StringTokenizer(value.toString());
      while (itr.hasMoreTokens()) {
        account.set(itr.nextToken());
        date.set(itr.nextToken());
        String string_account = account.toString();
        String string_year = date.toString().substring(7);

        account_year.set(string_account+string_year);
        int_amount = Integer.parseInt(itr.nextToken());

        if (int_amount > 0) {
          amount.set(int_amount );
          context.write(account_year, amount);
        }
      }
    }
  }

  public static class solution4Reducer
       extends Reducer<Text,IntWritable,Text,IntWritable> {
    private IntWritable result = new IntWritable();

    public void reduce(Text key, Iterable<IntWritable> values,
                       Context context
                       ) throws IOException, InterruptedException {
      int total = 0;

      for (IntWritable val : values)
```

```
70          total = total + val.get();
71
72          result.set(total);
73          context.write(key, result);
74        }
75      }
76    }
```

Task 1

Intuitive design of a data cube from a functional specification of operational database

A data warehouse of a train company contains information about train trips. The company would like to implement the following applications.

(i) find the total number of kilometers made by trains in a given year, departing from the stations locating in a given country and arriving at the stations located in a given country.

(ii) find the total duration of international trips in a given year, that is, trips departing from a station located in a country and arriving at a station located in another country,

(iii) find the total number of trips that departed from or arrived at a given city in a given month of a given year,

(iv) find and average duration of train trips in a given country in a given year,

(v) for all trips in a given year, find an average number of passengers on a trip.

(vi) find an average number of passengers all trips between two given city.

(vii) find total number of trips per each driver.

(viii) find the total number of trips that used a given train type in a given year.

(1) Use the specifications of applications listed above to find a data cube, that should be implemented by the train company to create a data warehouse. In your specification of a data cube, list the names of dimensions, hierarchies, measures, and attributes used to describe a data cube.

(2) Pick any three dimensions from a data cube found in the previous step and at least 4 values in each dimension and draw a sample three dimensional data cube in a perspective view similar to a view included in a presentation 03 Data Warehouse Concepts, slide 6.

Deliverables

A file `solution1.pdf` that contains

(1) a specification of data cube as a list of names of dimensions, list of hierarchies, list of measures and a list of attributes as a result of task (1),

(2) a perspective drawing of three dimensional data cube as a result of task (2).

Solution 1

Facts: TRIP (Trip is performed from departure city to arrival city on a day)

Dimensions: DepartureCity, ArrivalCity, Date/Time, Driver, TrainType

Hierarches: Year Consist of Months, Month Conists of Days, Day consists of Hours, Country consists of DepartureCity and ArrivalCity

Measures: Trip length in kms,
Trip duration in hours,
Total number of passengers on a trip

(2)
Obvious

Task 2 (6 marks)
Conceptual modelling of a data warehouse

An objective of this task is to create a conceptual schema of a sample data warehouse domain described below. Read and analyse the following specification of a data warehouse domain.

A large international network of hotels would like to create a data warehouse to store information about their hotels located in the different cities of different countries, hotel guests visiting the rooms in hotels, and employees working at the hotels. The management of the network would like to store the following information in the data warehouse.

Each hotel is described by its location (country, city, building number), email address and link to a Web page. A hotel offers the rooms to its customers. A room has a unique number within a hotel. A room number consists of a floor number and a unique number at a floor. For example, room 25 at 5th floor has a number 0525.

Each hotel has a number of employees. An employee has a unique employee number, first name, last name, and date of birth. Staff members belong to either administration group or maintenance group. Among the other duties, administration staff members are allowed to perform check-in and check-out of hotel guests. Maintenance staff members perform the maintenance works in the rooms occupied by hotel guests.

Hotel guests stay in hotel rooms. On check-in day a start date of a visit is recorded and on check-out day an end date of a visit is recorded. The data warehouse must contain information about the total number days of each visit and amount of money paid by each hotel guest, total number of facilities used by hotel guests, and the total number of maintenances performed in a room during a visit.

A hotel guest is described by a number of identification document, first name, last name, date of birth and nationality. A hotel guest uses a credit card to pay for his/her stay in a hotel. A credit card number and a name of bank that issued a card is recorded.

A data warehouse must be designed such it should be possible to easily implement the following classes of applications.

A management of the hotel network would like to get from a data warehouse information about the total number of visits per hotel and per given period of time like day, month, and year, about total number of visits in hotels per city and per country, about total number of check-ins/outs per employee, and about the total number of visits paid per credit card used, total number of customers per hotel, per room, per month per year, total profits per hotel, per city where the hotels are located, average length of stay per year, per month, per hotel, average discount applied per hotel, per month per year.

To draw a conceptual schema, use a graphical notation explained to you in a presentation 11 Conceptual Data Warehouse Design.

To create a conceptual schema of a sample data warehouse domain, follow the steps listed below.

Step 1 Find a fact entity, find the measures describing a fact entity.
Step 2 Find the dimensions.
Step 3 Find the hierarchies over the dimensions.
Step 4 Find the descriptions (attributes) of all entity types.
Step 5 Draw a conceptual schema.

To draw a conceptual schema, you must use a graphical notation explained to you in a presentation 11 Conceptual Data Warehouse Design.

To draw your diagram, you can use UMLet diagram drawing tool and apply a "Conceptual modelling" notation, Selection of a drawing notation is available in the right upper corner of the main menu of UMLet diagram drawing tool. UMLet 14.3 software is can be downloaded from the subject's Moodle Web site in a section WEB LINKS. A neat hand drawing is still all right.

Deliverables
A file `solution2.pdf` with a drawing of a conceptual schema of a sample data warehouse domain.

Solution

A fact: `VISIT` (in a hotel)

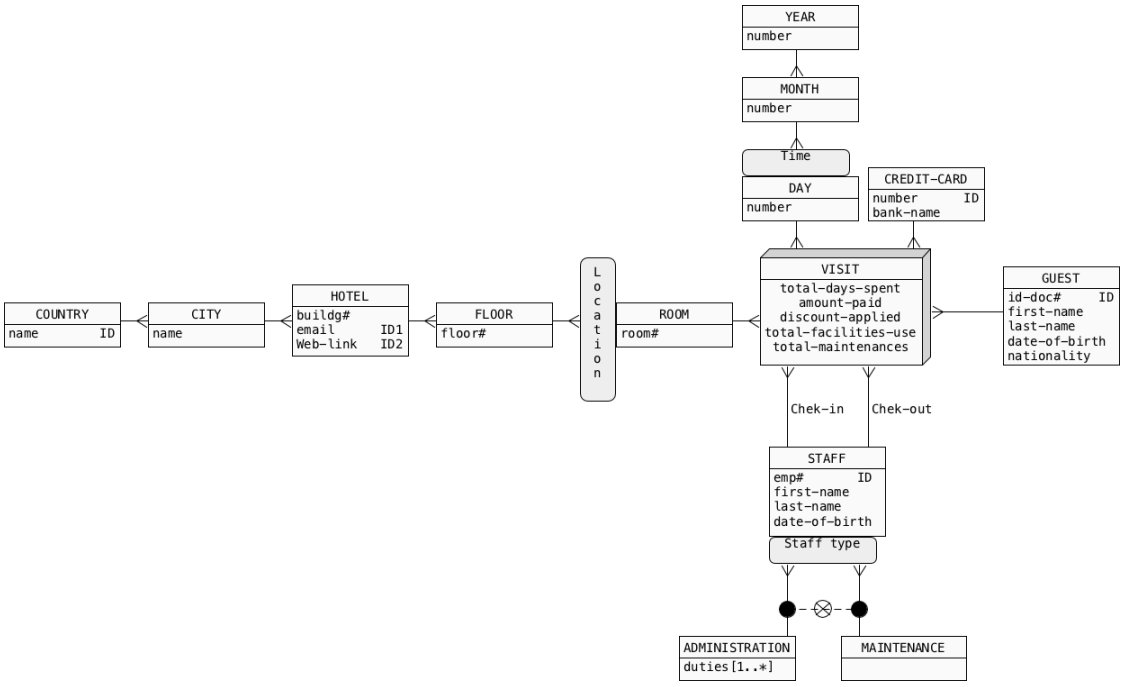
Dimensions:
```
GUEST,
C-CARD,
ROOM,
TIME,
CHECK-IN-ADMIN,
CHECK-OUT-ADMIN,
```

Hierarchies:
```
COUNTRY Consists-of CITY Consists-of HOTEL
Consists-of FLOOR Consists-of ROOM,

YEAR Consists-if MONTH Consists-of DAY

ADMINISTRATION-STAFF ISA STAFF,
MAINTENANCE-STAFF ISA STAFF
```

Measures: `total-days-spent, amount-paid, discount-applied, total-facilities used, total-maintenances`

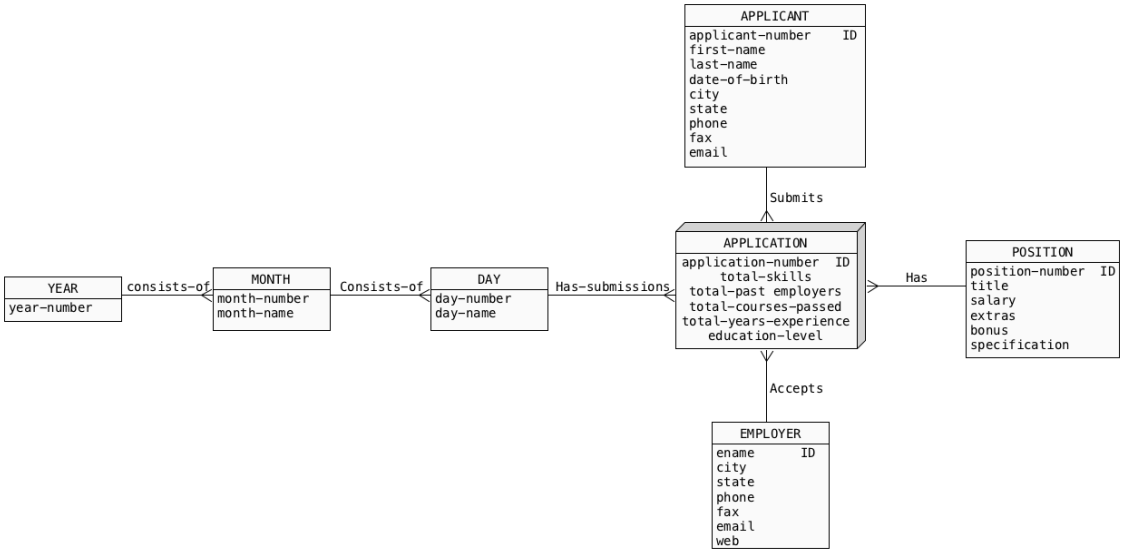
Attributes:

```
HOTEL(country,city,buildg#,email,Web-link)
ROOM(room#)
STAFF(emp#,first-name,last-name,date-of-birth)
ADMINISTRATION-STAFF(duties[1..*])
MAINTENANCE-STAFF()
GUEST(id-doc#, first-name, last-name,
     date-of-borth, nationality)
C-CARD(number, bank-name)
```



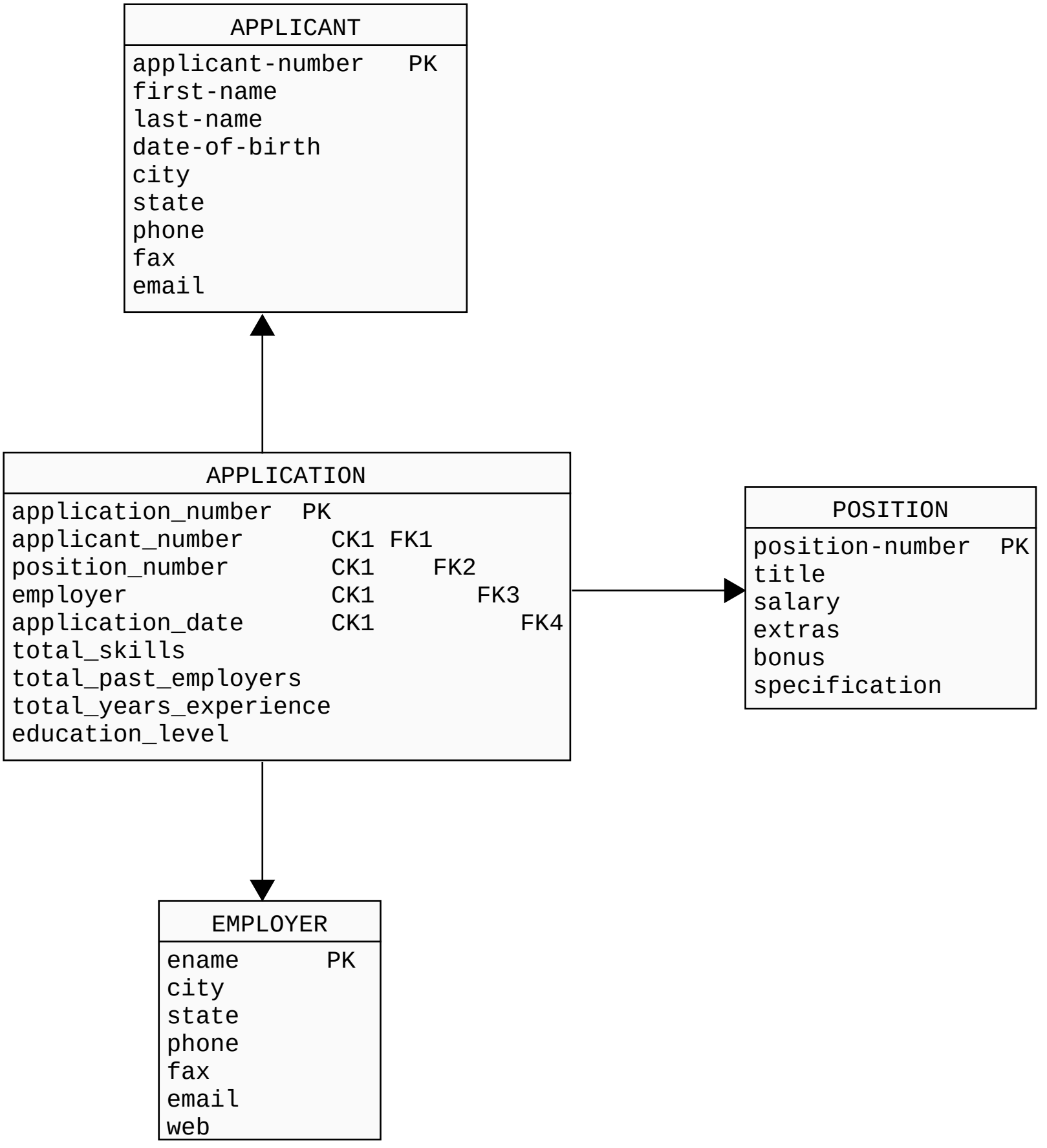
Task 3 (4 marks)
Logical modelling of a data warehouse

Consider the following conceptual schema of a data warehouse.



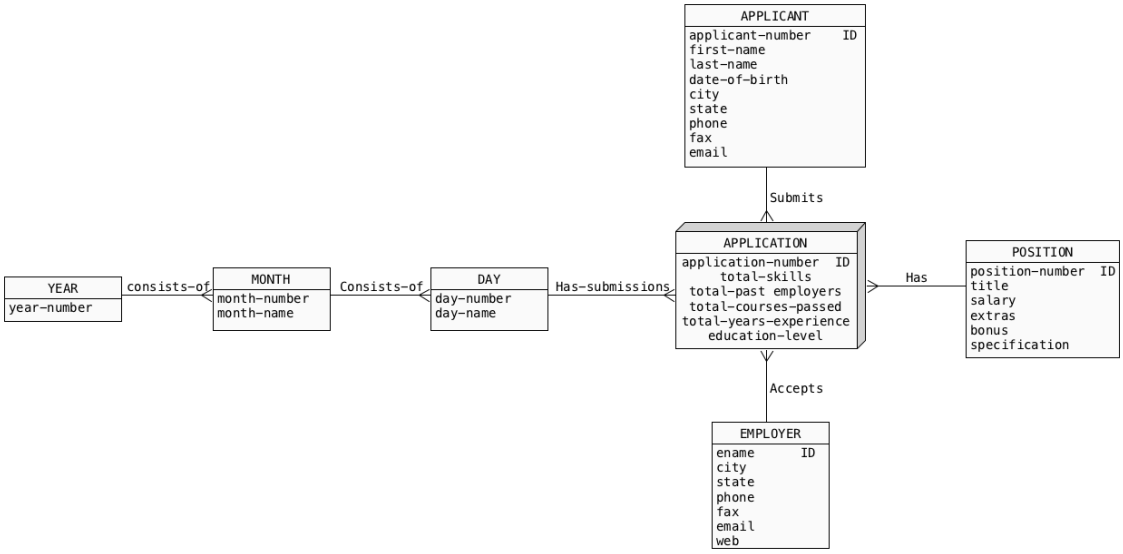
Perform a step of logical design to transform a conceptual schema given above into a logical schema (star schema). Use UMLet diagram drawing tool and apply a "Logical modelling" notation to draw a logical schema. Selection of a drawing notation is available in the right upper corner of the main menu of UMLet. Save a diagram of logical schema in a file `solution3.uxf` and export it to a file `solution3.pdf`.

Deliverables
A file `solution3.pdf` with a drawing of a logical schema.


APPLICANT
applicant-number PK
first-name
last-name
date-of-birth
city
state
phone
fax
email
APPLICATION
application_number PK
applicant_number CK1 FK1
position_number CK1 FK2
employer CK1 FK3
application_date CK1 FK4
total_skills
total_past_employers
total_years_experience
education_level
POSITION
position-number PK
title
salary
extras
bonus
specification
EMPLOYER
ename PK
city
state
phone
fax
email
web

Task 4 (6 marks)
Implementation of a data warehouse as a collection of external tables in Hive

Consider the following conceptual schema of a data warehouse.



Download a file `task4.zip` and unzip it. You should obtain a folder `task4` with the following files: `applicant.tbl`, `position.tbl`, `employer.tbl`, and `application.tbl`.

Use text editor to examine the contents of `*.tbl` files. The order of columns with values is usually consistent with the order of properties in the entity types of a conceptual schema above. In the case of a file `application.tbl` an order of columns with values is a bit different. It is your task to discover the most appropriate order. Note, that you may have to "clean" the files. It means that you may have to remove small mistakes in the files. It is called Extract, Transform, and Load (ETL).

When ready, transfer the files into HDFS.

Implement HQL script `solution4.hql` that creates the external tables obtained from a step of logical design performed earlier. The external tables must overlap on the files transferred to HDFS in the previous step. Note, that you can re-use the outcomes of a logical design performed in Task 3 above.

Include into `solution4.hql` script `SELECT` statements that lists any 3 rows from each one of the external tables implemented in the previous step and the total number of rows included in each table.

When ready, use a command line interface `beeline` to process a script `solution4.hql` and to save a report from processing in a file `solution4.rpt`.

```
! record solution4.rpt

CREATE EXTERNAL TABLE APPLICATION(
application_number     decimal(3),
applicant_number       decimal(6),
position_number        decimal(8),
employer               string,
application_date       string,
total_skills           decimal(1),
total_past_employers   decomal(2),
total_years_experience decimal(2),
education_level        string )
ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
STORED AS TEXTFILE LOCATION '/user/hive/application.tbl';

CREATE EXTERNAL TABLE APPLICANT(
applicant_number       decimal(6),
first_name             string,
last_name              string,
date_of_birth          string,
city                   string,
state                  string,
phone                  string,
fax                    string,
email                  string )
ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
STORED AS TEXTFILE LOCATION '/user/hive/applicant.tbl';

CREATE EXTERNAL TABLE EMPLOYER(
ename                  string,
city                   string,
state                  string,
phone                  string,
fax                    string,
email                  string,
web                    string )
ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
STORED AS TEXTFILE LOCATION '/user/hive/employer.tbl';

CREATE EXTERNAL TABLE POSITION(
position_number        decimal(8),
title                  string,
salary                 decimal(8,2),
extras                 string,
bonus                  decimal(8,2),
specification          string )
ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
STORED AS TEXTFILE LOCATION '/user/hive/position.tbl';
```

```
SELECT * FROM APPLICATION LIMIT 3;
SELECT COUNT(*) FROM APPLICATION;

SELECT * FROM APPLICANT LIMIT 3;
SELECT COUNT(*) FROM APPLICANT;

SELECT * FROM EMPLOYER LIMIT 3;
SELECT COUNT(*) FROM EMPLOYER;

SELECT * FROM POSITION LIMIT 3;
SELECT COUNT(*) FROM POSITION;

! record
```

Task 5 (6 marks)
Querying a data cube

Download a file `task5.zip` and unzip the file. You should obtain a folder `task5` with the following files: `dbcreate.hql`, `dbdrop.hql`, `partsupp.tbl`, `lineitem.tbl`, and `orders.tbl`.

A file `orders.tbl` contains information about the orders submitted by the customers. A file `lineitem.tbl` contains information about the items included in the orders. A file `partsupp.tbl` contains information about the items and suppliers of items included in the orders.

Open Terminal window and use `cd` command to navigate to a folder with the just unzipped files. Start Hive Server 2 in the terminal window (remember to start Hadoop first). When ready process a script file `dbcreate.hql` to create the internal relational tables and to load data into the tables. You can use either `beeline` or SQL Developer. A script `dbdrop.hql` can be used to drop the tables.

The relational tables `PARTSUPP`, `LINEITEM`, `ORDERS` implement a simple two-dimensional data cube. The relational tables `PARTSUPP` and `ORDERS` implement the dimensions of parts supplied by suppliers and orders. A relational table `LINEITEM` implements a fact entity of a data cube.

(1) Implement the following query using `GROUP BY` clause with `CUBE` operator.

For the order clerks (`O_CLERK`) `Clerk#000000522`, `Clerk#000000154`, find the total number of ordered parts per customer (`O_CUSTKEY`), per supplier (`L_SUPPKEY`), per customer and supplier (`O_CUSTKEY`, `L_SUPPKEY`), and the total number of ordered parts.

(2) Implement the following query using `GROUP BY` clause with `ROLLUP` operator.

For the parts with the keys (`L_PARTKEY`) `7`, `8`, `9` find the largest discount applied (`L_DISCOUNT`) per part key (`L_PARTKEY`) and per part key and supplier key (`L_PARTKEY, L_SUPPKEY`) and the largest discount applied at all.

(3) Implement the following query using `GROUP BY` clause with `GROUPING SETS` operator.

Find the smallest price (`L_EXTENDEDPRICE`) per order year (`O_ORDERDATE`), and order clerk (`O_CLERK`).

Implement the following SQL queries as `SELECT` statements using window partitioning technique.

(4) For each part list its key (`PS_PARTKEY`), all its available quantities (`PS_AVAILQTY`), the smallest available quantity, and the average available quantity. Consider only the parts with the keys `5` and `15`.

(5) For each part list its key (`PS_PARTKEY`) and all its available quantities (`PS_AVAILQTY`) sorted in descending order and a rank (position number in an ascending order) of each quantity. Consider only the parts with the keys `10` and `20`. Use an analytic function `ROW_NUMBER()`.

(6) For each part list its key (`PS_PARTKEY`), its available quantity, and an average available quantity (`PS_AVAILQTY`) of the current quantity and all previous quantities in the ascending order of available quantities. Consider only the parts with the keys `15` and `25`. Use `ROWS UNBOUNDED PRECEEDING` sub-clause within `PARTITION BY` clause.

When ready, save your `SELECT` statements in a file `solution5.hql`. Then, process a script file `solution5.hql` and save the results in a report `solution5.rpt`.

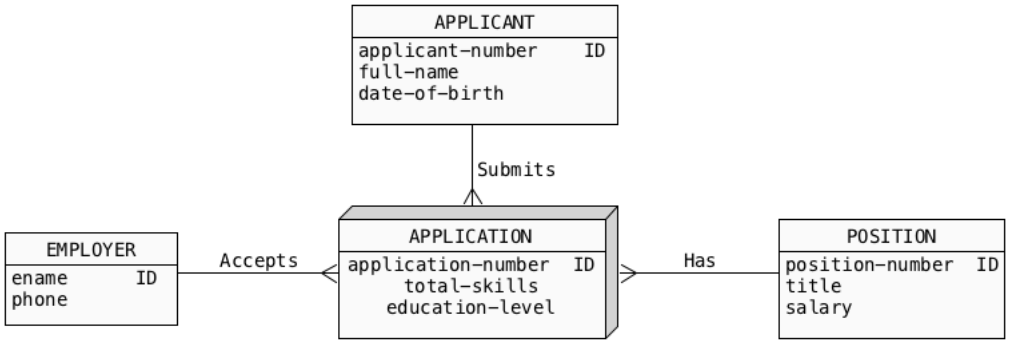
Deliverables
A file `solution5.rpt` that contains a report from processing of `SELECT` statements.

```
! record solution5.rpt

-- For the order clerks (O_CLERK) Clerk#000000522, Clerk#000000154, find the
-- total number of ordered parts per customer (O_CUSTKEY), per supplier
-- (L_SUPPKEY), per customer and supplier (O_CUSTKEY, L_SUPPKEY), and the total
-- number of ordered parts.

   SELECT O_CUSTKEY, L_SUPPKEY, COUNT(*)
   FROM LINEITEM JOIN ORDERS
                 ON LINEITEM.L_ORDERKEY = ORDERS.O_ORDERKEY
   WHERE O_CLERK IN ('Clerk#000000522', 'Clerk#000000154')
   GROUP BY O_CUSTKEY, L_SUPPKEY WITH CUBE;

-- For the parts with the keys (L_PARTKEY) 7, 8, 9 find the largest discount
applied
-- (L_DISCOUNT) per part key (L_PARTKEY) and per part key and supplier key
-- (L_PARTKEY, L_SUPPKEY) and the largest discount applied at all.

   SELECT L_PARTKEY, L_SUPPKEY, MAX(L_DISCOUNT)
   FROM LINEITEM
   WHERE L_PARTKEY IN (7, 8, 9)
   GROUP BY L_PARTKEY, L_SUPPKEY WITH ROLLUP;

-- Find the smallest price (L_EXTENDEDPRICE) per order year (O_ORDERDATE), and
-- order clerk (O_CLERK).

   SELECT O_CLERK, substr(O_ORDERDATE,1,4), MIN(L_EXTENDEDPRICE)
   FROM ORDERS JOIN LINEITEM
               ON LINEITEM.L_ORDERKEY = ORDERS.O_ORDERKEY
   GROUP BY O_CLERK, substr(O_ORDERDATE,1,4)
   GROUPING SETS ( (O_CLERK), (substr(O_ORDERDATE,1,4)) );

  -- For each part list its key (PS_PARTKEY), all its available quantities
(PS_AVAILQTY),
-- the smallest available quantity, and the average available quantity. Consider
only the
-- parts with the keys 5 and 15.

   SELECT PS_PARTKEY, PS_AVAILQTY, MIN(PS_AVAILQTY) OVER (PARTITION BY PS_PARTKEY),
                                   AVG(PS_AVAILQTY) OVER (PARTITION BY PS_PARTKEY)
   FROM PARTSUPP
   WHERE PS_PARTKEY IN (5,15);

-- For each part list its key (PS_PARTKEY) and all its available quantities
-- (PS_AVAILQTY) sorted in descending order and a rank (position number in an
-- ascending order) of each quantity. Consider only the parts with the keys 10 and
20. Use
-- an analytic function ROW_NUMBER().
```

```
   SELECT PS_PARTKEY, PS_AVAILQTY, ROW_NUMBER() OVER (PARTITION BY PS_PARTKEY
                                                      ORDER BY PS_AVAILQTY DESC)
   FROM PARTSUPP
   WHERE PS_PARTKEY IN (10,20);

-- For each part list its key (PS_PARTKEY), its available quantity, and an average
available
-- quantity (PS_AVAILQTY) of the current quantity and all previous quantities in
the
-- ascending order of available quantities. Consider only the parts with the keys
15 and
-- 25. Use ROWS UNBOUNDED PRECEDING sub-clause within PARTITION BY
-- clause.


   SELECT PS_PARTKEY, PS_AVAILQTY, AVG(PS_AVAILQTY) OVER (PARTITION BY PS_PARTKEY
                                                           ORDER BY PS_AVAILQTY
                                                           ROWS UNBOUNDED PRECEDING)
   FROM PARTSUPP
   WHERE PS_PARTKEY IN (15,25);
! record
```

Task 1 (5 marks)
Design and implementation of HBase table

Implement as a single HBase table a database that contains information described by the following conceptual schema.



(1) Create HBase script `solution1.hb` with HBase shell commands that create HBase table and load sample data into the table. Load into the table information about at least two applications such that each involved one applicant and one position.

When ready use HBase shell to process a script file `solution1.hb` and to save a report from processing in a file `solution1.rpt`.

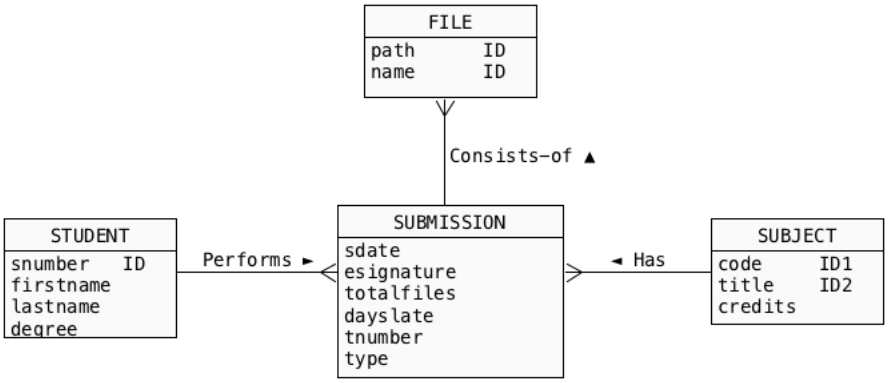
Deliverables
A file `solution1.rpt` that contains a report from processing of `solution1.hb` script with the statements that create HBase table and load sample data.

```
create 'task1', 'APPLICATION','EMPLOYER','APPLICANT','POSITION'

put 'task1','employer|SIM','EMPLOYER:ename','Singapore Institute of Management'
put 'task1','employer|SIM','EMPLOYER:phone','1234567890'

put 'task1','employer|UOW','EMPLOYER:ename','University of Wollongong'
put 'task1','employer|UOW','EMPLOYER:Phone','0987654321'

put 'task1','position|312','POSITION:position-number','312'
put 'task1','position|312','POSITION:title','Big Data Manager'
put 'task1','position|312','POSITION:salary','600000'

put 'task1','position|313','POSITION:position-number','313'
put 'task1','position|313','POSITION:title','Very Big Data Manager'
put 'task1','position|313','POSITION:salary','700000'

put 'task1','applicant|312','APPLICANT:number','007'
put 'task1','applicant|312','APPLICANT:full-name','James Bond'
put 'task1','applicant|312','APPLICANT:date-of-birth','01-01-1960'

put 'task1','applicant|313','APPLICANT:number','666'
put 'task1','applicant|313','APPLICANT:full-name','Very Big Data'
put 'task1','applicant|313','APPLICANT:date-of-birth','01-01-1999'

put 'task1','application|007|312|SIM|01','APPLICATION:application-number','01'
put 'task1','application|007|312|SIM|01','APPLICATION:total-skills','5'
put 'task1','application|007|312|SIM|01','APPLICATION:education-level','high'
put 'task1','application|007|312|SIM|01','POSITION:position-number','312'
put 'task1','application|007|312|SIM|01','APPLICANT:number','007'
put 'task1','application|007|312|SIM|01','EMPLOYER:ename','SIM'

put 'task1','application|666|312|SIM|02','APPLICATION:application-number','02'
put 'task1','application|666|312|SIM|02','APPLICATION:total-skills','5'
put 'task1','application|666|312|SIM|02','APPLICATION:education-level','low'
put 'task1','application|666|312|SIM|02','POSITION:position-number','312'
put 'task1','application|666|312|SIM|02','APPLICANT:number','666'
put 'task1','application|666|312|SIM|02','EMPLOYER:ename','SIM'

put 'task1','application|007|313|UOW|03','APPLICATION:application-number','01'
put 'task1','application|007|313|UOW|03','APPLICATION:total-skills','5'
put 'task1','application|007|313|UOW|03','APPLICATION:education-level','high'
put 'task1','application|007|313|UOW|03','POSITION:position-number','313'
put 'task1','application|007|313|UOW|03','APPLICANT:number','007'
put 'task1','application|007|313|UOW|03','EMPLOYER:ename','UOW'

describe 'task1'

scan 'task1'

disable 'task1'
```

```
drop 'task1'
```

Task 2 (5 mark)s
Querying and manipulating data in HBase table

Consider a conceptual schema given below. The schema represents a simple database domain where students submit assignments and each submission consists of several files and it is related to one subject.



Download a file `task2.hb` with HBase shell commands and use HBase shell to process it. Processing of a script `task2.hb` creates HBase table `task2` and loads some data into it.

Use HBase shell to implement the following queries and data manipulations on the HBase table created in the previous step. Save the queries and data manipulations in a file `solution2.hb`.

(1) Find all information about a subject that has code `312`, list two versions per cell.
(2) Find all information about a submission of assignment `1` performed by a student `007` in a subject `312`, list one version per cell.
(3) Delete a column family `FILES`.
(4) Add a column family `ENROLMENT` that contains information about dates when the subjects have been enrolled by the students and allow for 2 versions in each cell of the column family.
(5) Increase the total number of versions in each cell of a column family `ENROLMENT`.

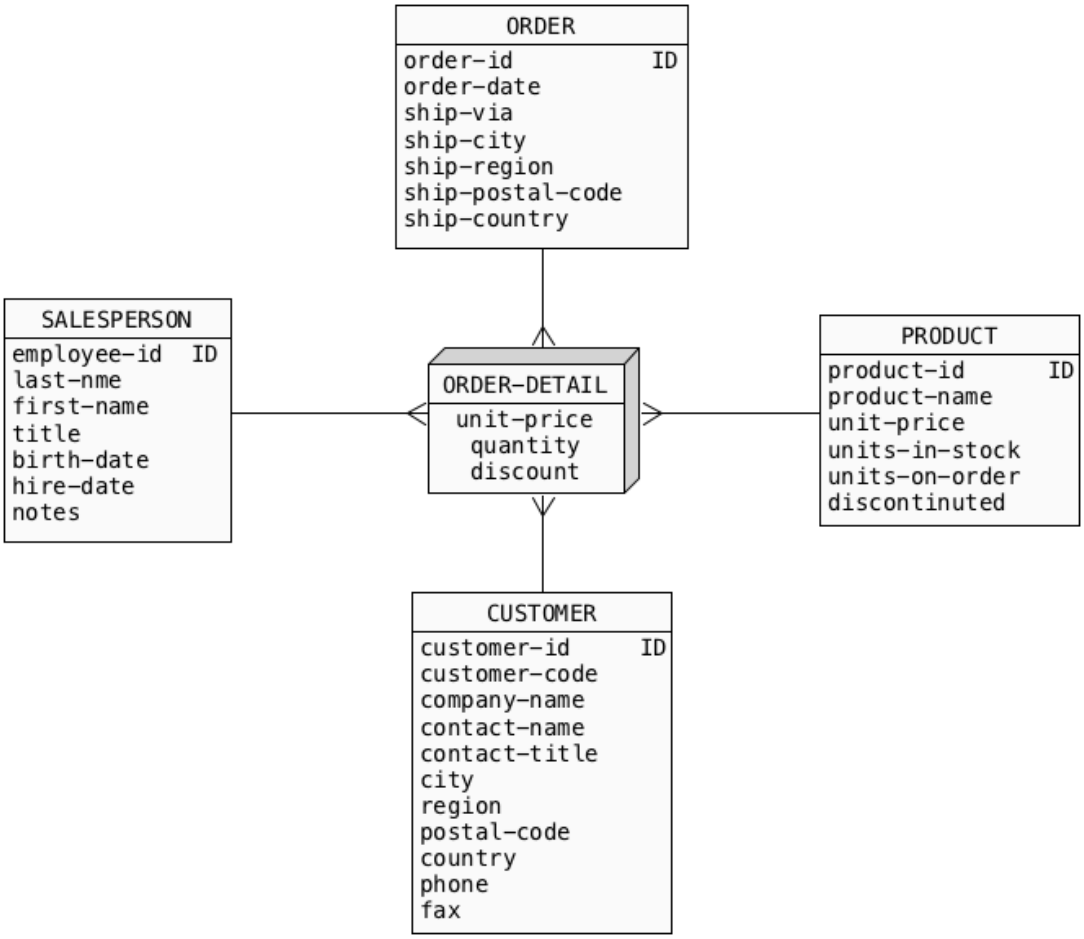
When ready, start HBase shell and process a script file `solution2.hb` with Hbase command shell. When processing is completed copy the contents of Command window with a listing from processing of the script and paste the results into a file `solution2.rpt`. Save the file. When ready submit a file `solution2.rpt`.

Deliverables

```
print '(1) Find all information about a subject that has code 312, list two
versions per cell.'
get 'task2','subject|312',{COLUMN=>'SUBJECT',VERSIONS=>2}

print '(2) Find all information about a submission of assignment 1 performed by a
student 007 in a subject 312, list one version per cell.'
get
'task2','submission|007|312|assignment|1',{COLUMNS=>['FILES','STUDENT','SUBJECT',
'SUBMISSION'],VERSIONS=>1}

print '(3) Delete a column family FILES.'
alter 'task2' , 'delete' => 'FILES'

print '(4) Add a column family ENROLMENT that contains information about dates when
the subjects have been enrolled by the students and allow for 2 versions in each
cell of the column family.'
alter 'task2' , NAME => 'ENROLMENT' , VERSIONS=>2

print '(5) Increase the total number of versions in each cell of a column family
ENROLMENT.'
alter 'task2' , NAME => 'ENROLMENT' , VERSIONS=>5
```

Task 3 (5 marks)
Data processing with Pig Latin

Consider the following conceptual schema of a data warehouse.



Download a file `task3.zip` published on Moodle together with a specification of Assignment 3 and unzip it. You should obtain a folder `TASK3` with the following files: `customer.tbl`, `order_details.tbl`, `order.tbl`, `product.tbl`, `salesperson.tbl`. The files contain data dumped from a data warehouse whose conceptual schema is given above.

Use editor to examine the contents of `*.tbl` files. Note, that each file has a header with information about the meanings of data in each column. A header is not a data component of each file.

(1) Remove the headers and transfer the files into HDFS.

Create Pig Latin script `solution3.pig` that implements the following queries.

(2) Find the first and the last name (`first-name`, `last-name`) of sales people who handled the orders submitted by the customers located in `Mexico`.

(3) Find the total number of sales people who handled the orders submitted in `1996`.

(4) Find the summarizations of prices (`unit-price`) per ordered product (`product-id`).

(5) Find the identifiers of orders (`order-id`) that included both `Ikura` and `Tofu`.

When ready, use `pig` command line interface to process a script `solution3.pig` and to save a report from processing in a file `solution3.rpt`.

Deliverables

A file `solution3.rpt` with a report from processing of Pig Latin script `solution3.pig`.

---

(2)

```
customers = load '/user/hive/customer/customer.tbl' using PigStorage('|')
as
(CUSTOMER_ID:int,CUSTOMER_CODE:chararray,COMPANY_NAME:chararray,CONTACT_NAME:charar
ray,CONTACT_TITLE:chararray,CITY:chararray,REGION:chararray,POSTAL_CODE:chararray,C
OUNTRY:chararray,PHONE:chararray,FAX:chararray);
order_details = load '/user/hive/order_detail/order_detail.tbl' using
PigStorage('|')
as
(ORDER_ID:int,PRODUCT_ID:int,CUSTOMER_ID:int,SALESPERSON_ID:int,UNIT_PRICE:int,QUAN
TITY:int,DISCOUNT:float);
salespeople = load '/user/hive/salesperson/salesperson.tbl' using PigStorage('|')
as
(EMPLOYEE_ID:int,LASTNAME:chararray,FIRSTNAME:chararray,TITLE:chararray,BIRTHDATE:c
hararray,HIREDATE:chararray,NOTES:chararray);
consolidated =  filter customers by COUNTRY == ' Mexico ';
consolidated_details = join consolidated by CUSTOMER_ID, order_details by
CUSTOMER_ID;
consolidated_details_sales = join consolidated_details by SALESPERSON_ID,
salespeople by EMPLOYEE_ID;
results = foreach consolidated_details_sales generate FIRSTNAME, LASTNAME;
dump results;
```

(3)

```
orders = load '/user/hive/order/order.tbl' using PigStorage('|')
as
(ORDER_ID:int,ORDER_DATE:chararray,SHIP_VIA:int,SHIP_CITY:chararray,SHIP_REGION:cha
rarray,SHIP_POSTAL_CODE:chararray,SHIP_COUNTRY:chararray);
order_details = load '/user/hive/order_detail/order_detail.tbl' using
PigStorage('|')
as
(ORDER_ID:int,PRODUCT_ID:int,CUSTOMER_ID:int,SALESPERSON_ID:int,UNIT_PRICE:int,QUAN
TITY:int,DISCOUNT:float);
products = load '/user/hive/product/product.tbl' using PigStorage('|')
as
(PRODUCT_ID:int,PRODUCT_NAME:chararray,UNIT_PRICE:int,UNITS_IN_STOCK:int,UNITS_ON_O
RDER:int,DISCONTINUED:chararray);
orders_1996 = filter orders by ENDSWITH(ORDER_DATE,'1996 ');
order_details_1996 = join orders_1996 by ORDER_ID, order_details by ORDER_ID;
products_1996 = foreach order_details_1996 generate PRODUCT_ID;
products_all = foreach products generate PRODUCT_ID;
leftouter = join products_all by PRODUCT_ID left outer, products_1996 by
PRODUCT_ID;
notordered = filter leftouter by products_1996::order_details::PRODUCT_ID is null;
notordered_grouped = group notordered all;
notordered_counted = foreach notordered_grouped generate COUNT(notordered);
dump notordered_counted;
```

(4)

```
order_details = load '/user/hive/order_detail/order_detail.tbl' using
PigStorage('|')
as
(ORDER_ID:int,PRODUCT_ID:int,CUSTOMER_ID:int,SALESPERSON_ID:int,UNIT_PRICE:int,QUAN
TITY:int,DISCOUNT:float);
grouped_products = group order_details by PRODUCT_ID;
summarized_products = foreach grouped_products generate group,
SUM(order_details.UNIT_PRICE);
dump summarized_products;

(5)
products = load '/user/hive/product/product.tbl' using PigStorage('|')
as
(PRODUCT_ID:int,PRODUCT_NAME:chararray,UNIT_PRICE:int,UNITS_IN_STOCK:int,UNITS_ON_O
RDER:int,DISCONTINUED:chararray);
order_details = load '/user/hive/order_detail/order_detail.tbl' using
PigStorage('|')
as
(ORDER_ID:int,PRODUCT_ID:int,CUSTOMER_ID:int,SALESPERSON_ID:int,UNIT_PRICE:int,QUAN
TITY:int,DISCOUNT:float);
ikura = filter products by PRODUCT_NAME == ' Ikura ';
tofu = filter products by PRODUCT_NAME == ' Tofu ';
ikura_orders = join ikura by PRODUCT_ID, order_details by PRODUCT_ID;
ikura_orders_id = foreach ikura_orders generate order_details::ORDER_ID;
tofu_orders = join tofu by PRODUCT_ID, order_details by PRODUCT_ID;
tofu_orders_id = foreach tofu_orders generate order_details::ORDER_ID;
ikura_and_tofu_orders_id = join ikura_orders_id by order_details::ORDER_ID,
tofu_orders_id by order_details::ORDER_ID;
dump ikura_and_tofu_orders_id;
```

Task 4 (5 marks)
Data processing with Spark

In this task we use the files uploaded to HDFS in the Task 3 of this Assignment. If you have not uploaded the files then download a file `task3.zip` published on Moodle together with a specification of Assignment 3 and unzip it. You should obtain a folder `TASK3` with the following files: `customer.tbl`, `order_details.tbl`, `order.tbl`, `product.tbl`, `salesperson.tbl`.

When ready create a script `solution4.sc` that implements the following Spark-shell operations:

(1) Create a DataFrame named `orderDetailsDF` that contains information about the details of orders included in a file `order-details.tbl`.
(2) Lists all order details where `quantity` is greater than `50`.
(3) Find the total number of orders submitted in `Germany`.
(4) Find the total number of orders per each country.
(5) Find 5 most expensive (use attribute `unit-price`) products.

When ready, start Spark-shell and process a script `solution4.sc` in Spark-shell using `:paste` command.

Save a report in a file `solution4.rpt`.

Deliverables
A file `solution4.rpt` with a report from processing of a file `solution4.sc`.

```
case class CUSTOMER(
customer_id: Int,
code: String,
company_name: String,
contact_name: String,
contact_title: String,
city: String,
region: String,
postal_code: String,
country: String,
phone: String,
fax: String)

val DF_CUSTOMER =
spark.sparkContext.textFile("./customer.tbl").map(_.split(",")).map(attributes=>CUSTOMER(
attributes(0).trim.toInt, attributes(1).trim(), attributes(2).trim(),
attributes(3).trim(), attributes(4).trim(), attributes(5).trim(), attributes(6).trim(),
attributes(7).trim(), attributes(8).trim(), attributes(9).trim(),
attributes(10).trim())).toDF()

CustomerDF.show()


case class ORDERDETAIL(
order_id: Int,
product_id: Int,
customer_id: Int,
supplier_id: Int,
unit_price: Float,
quantity: Int,
discount: Float)

val DF_ORDERDETAIL =
spark.sparkContext.textFile("./order_detail.tbl").map(_.split(",")).map(attributes=>ORDER
DETAIL(attributes(0).trim.toInt, attributes(1).trim.toInt, attributes(2).trim.toInt,
attributes(3).trim.toInt, attributes(4).trim.toFloat, attributes(5).trim.toInt,
attributes(6).trim.toFloat)).toDF()

OrderDetailDF.show()


case class ORDER(
order_id: Int,
order_date: String,
ship_via: String,
city: String,
region: String,
postal_code:
String, country: String)

val DF_ORDER =
spark.sparkContext.textFile("./order.tbl").map(_.split(",")).map(attributes=>ORDER(attrib
utes(0).trim.toInt, attributes(1).trim(), attributes(2).trim(), attributes(3).trim(),
attributes(4).trim(), attributes(5).trim(), attributes(6).trim())).toDF()

DF_ORDER.show()

case class PRODUCT(product_id: String, name: String, unit_price: String, unit_in_stock:
String, unit_on_order: String, discontinued: String)

val DF_PRODUCT =
spark.sparkContext.textFile("./product.tbl").map(_.split(",")).map(attributes=>PRODUCT(at
tributes(0), attributes(1), attributes(2), attributes(3), attributes(4),
attributes(5))).toDF()

DF_PRODUCT.show()

case class SALESPERSON(
employee_id:Int,
```

```
54  last_name:String,
55  first_name:String,
56  title:String,
57  dob:String,
58  hire_date:String,
59  notes:String)
60
61  val DF_SALESPERSON =
    spark.sparkContext.textFile("./salesperson.tbl").map(_.split(",")).map(attributes=>SALESP
    ERSON(attributes(0).trim.toInt,attributes(1).trim(),attributes(2).trim(),
    attributes(3).trim(), attributes(4).trim(), attributes(5).trim(),
    attributes(6).trim())).toDF()
62
63  DF_SALESPERSON.show()
64
65  DF_ORDERDETAIL.filter(col("quantity") > 50).show()
66
67  DF_ORDER.filter(col("country").like("Germany")).count()
68
69  DF_ORDER.groupBy(col("country")).count().show()
70
71  DF_PRODUCT.sort(col("unit_price").desc).show(20)
```