Name

CWID

Homework Assignment 4

April 2015

CS520 Results

Please leave this empty!

Sum

Instructions

- Try to answer all the questions using what you have learned in class
- The assignment is not graded
- There is a theoretical and practical part
- **When writing a query, write the query in a way that it would work over all possible database instances and not just for the given example instance!**

Lab Part

- This part of the assignment helps you to practice the techniques we have introduced in class
- In this assignment we will work with Spark to do some basic data integration.
- You will also explore one of the existing ETL tools to do a simple job.

Part 2.1 Spark (Total: 0 Points)

- *Spark* (`https://spark.apache.org/`) is a distributed big data analytics platform written in Scala (`https://scastie.scala-lang.org/`).

Question 2.1.1 Getting comfortable with Scala (0 Points)

- Scala is a functional and object-oriented language that compiles to Java bytecode and is compatible with Java (you can use Java classes from within Scala).
- Scala comes with a repl (an interactive interpreter for the language)

Tasks:

- Install scala (`https://docs.scala-lang.org/getting-started/index.html`)
- Start the scala REPL (`https://docs.scala-lang.org/scala3/book/taste-repl.html`) and play through some simple examples or use the online playground: `https://scastie.scala-lang.org/`
- Learn the concepts of Scala using the online book (`https://docs.scala-lang.org/scala3/book/introduction.html`)
- Learn more Scala using this course: (`https://docs.scala-lang.org/online-courses.html`)

Question 2.1.2 Getting comfortable with Spark (0 Points)

- Install Spark
- Spark can ingest multiple file formats without extensions and query them using a Scala (or Python) API
- Spark provides a REPL that is just an extended Scala repl with the spark libraries made avaiable
- Walk through some of the examples here: `https://spark.apache.org/docs/latest/sql-programming-guide.html`

Part 2.2 ETL tools (Total: 0 Points)

- Have a look at the list of ETL tools here: `http://www.etldatabase.com/etl-tools`
- Select, download, and install one ETL tool of your choice
- Use the ETL tool to implement an ETL workflow for creating a consolidated view of events in parks in Chicago.

The task is to create an ETL workflow that produces a fact table with number of events across several dimensions: time, location, eventtype (e.g., a movie or a concert).

Question 2.2.1 Datawarehouse schema design (0 Points)

Create a datawarehouse cube model and translate it into a star schema. As mentioned above:

- **Measures**: number of events
- **Dimensions**:
  - time (year, month, day)
  - location (park, zip code, city, neighborhood)
  - eventtype (type)

Question 2.2.2 Get the data (0 Points)

Start by downloading the datasets we want to integrate:

- US Zip codes: Download the `geo-data.csv` file from here: `https://github.com/scpike/us-state-county-zip`. We will use this as master data for cities of zip codes.
- Download Chicago park permit data: `https://data.cityofchicago.org/resource/pk66-w54g.csv`
- Download park district park locations: `https://data.cityofchicago.org/api/views/ejsh-fztr/rows.csv?accessType=DOWNLOAD`
- Download Chicago neighborhood boundaries: `https://data.cityofchicago.org/api/views/y6yq-dbs2/rows.csv?accessType=DOWNLOAD`
- Download information about movies in the park:
  - 2018: `https://data.cityofchicago.org/resource/e2v8-k3us.csv`
  - 2017: `https://data.cityofchicago.org/resource/a7gu-2qz6.csv`
  - optionally download for the movie in the park for other years

Question 2.2.3 Implement an ETL workflow (0 Points)

The task is to create an ETL workflow that produces a fact table with number of events across several dimensions: time, location, eventtype (e.g., a movie or a concert). You should implement an ETL workflow that cleans up park location data using the zip code information from the US Zip codes dataset and neightborhood information. Then extract event information from the movies in the park and permit datasets and merge these datasets into a single event table. The event table then needs to be joined with the cleaned up park location dataset to augment each event with more information about the parks. Finally, you should transform this single table into

Theory Part

- This part of the assignment helps you to practice the techniques we have introduced in class.

Part 2.3 The Cube Datamodel and Relational Implementations (Snowflake and Star Schemas) (Total: 0 Points)

- In this exercise you will be designing a datawarehouse model and translate it into snowflake and star schemas.
- For an example snowflake and star schema have a look at: `https://github.com/IITDBGroup/cs520/blob/master/SQLexamples/dataware_house_schemas.sql`

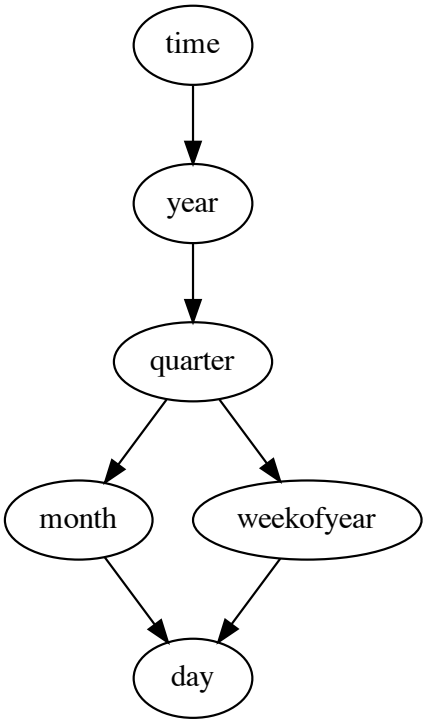
Question 2.3.1 Multidimensional datamodel design (0 Points)

Consider a mobile provider which wants to build a datawarehouse storing information about calls. The following requirements:

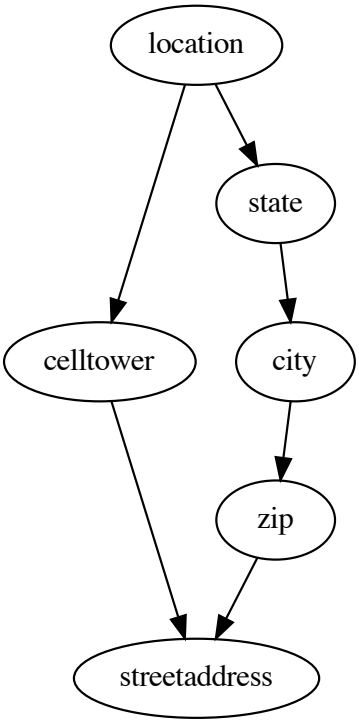
- The datawarehouse should record the number of call, the total length of all calls, and the amount of mobile data usage. This information should be aggregated across multiple dimensions.
- **Dimensions:**
  - It should be possible to analyze calls by time at the granularity of hours, days, months, years, quarters and weeks of the year.
  - Regarding customers placing the calls we would like to record their names as well as at a coarser granularity their homestate
  - Phone calls can also be distinguished based on the calling and receiving phone number. Phone numbers can be grouped by their area code
  - For the calling phone, we would like to record the brand and more specifically also the model
  - Phone calls are placed from and data usage happens in a specific location for which we want to record a streetaddress, cell tower, zip code, city, and state. Cell towers coverage areas may overlap with multiple cities, zip, states. Cities are assumed to belong to a single state and each zip code is contained withing a single city. A streetaddress belongs to exactly one zip code and is covered by one celltower.

Solution

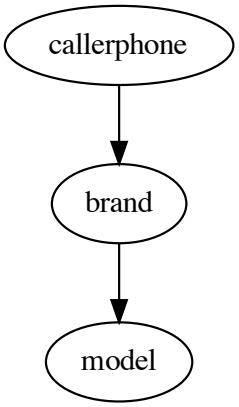
Time



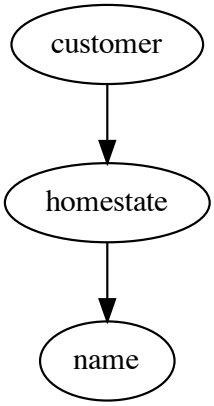
Location



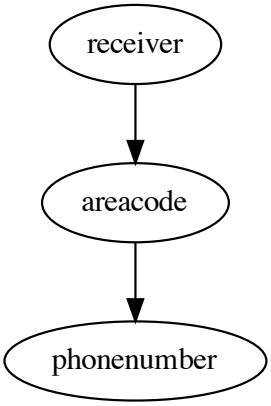
Callerphone



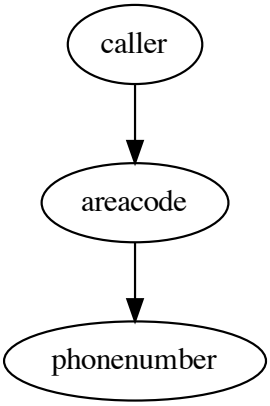
Customer



Receiver



Caller



The fact table will have three measures:

- number of call
- the total length of all calls
- the amount of mobile data usage

Question 2.3.2 Translation into Star and Snowflake Schemas (0 Points)

Translate the cube model you have designed into a star and snowflake schema.

Solution

Star Schema

```
CREATE TABLE timed (
  timeid SERIAL PRIMARY KEY,
  yr INT,
  quarter INT,
  mon INT,
  weekofyear INT,
  day INT
);

CREATE TABLE location (
  locationid SERIAL PRIMARY KEY,
  celltower VARCHAR(100),
  state CHAR(2),
  city VARCHAR(100),
  zip CHAR(5),
  streetaddress VARCHAR(200)
);

CREATE TABLE callerphone (
  callerphoneid SERIAL PRIMARY KEY,
  brand VARCHAR(50),
  model VARCHAR(50)
);

CREATE TABLE customer (
  customerid SERIAL PRIMARY KEY,
  homestate CHAR(2),
  name VARCHAR(100)
);

CREATE TABLE receiver (
  receiverid SERIAL PRIMARY KEY,
  areacode CHAR(3),
  phonenumber CHAR(9)
);

CREATE TABLE caller (
  callerid SERIAL PRIMARY KEY,
  areacode CHAR(3),
  phonenumber CHAR(9)
);
```

Solution

```
CREATE TABLE facts (
  timeid INT,
  locationid INT,
  callerphoneid INT,
  customerid INT,
  receiverid INT,
  callerid INT,
  num_calls INT,
  call_length NUMERIC(12,2),
  mobile_data_usage NUMERIC(15,2),
  PRIMARY KEY (timeid, locationid, callerphoneid, customerid, receiverid, callerid),
  FOREIGN KEY (timeid) REFERENCES timed,
  FOREIGN KEY (locationid) REFERENCES location,
  FOREIGN KEY (callerphoneid) REFERENCES callerphone,
  FOREIGN KEY (customerid) REFERENCES customer,
  FOREIGN KEY (receiverid) REFERENCES receiver,
  FOREIGN KEY (callerid) REFERENCES caller
);
```

Solution

Snowflake Schema

```
-------------------------------------------------------------------------------
-- TIME DIMENSION
CREATE TABLE time_year (
  time_yearid SERIAL PRIMARY KEY,
  yr INT
);

CREATE TABLE time_quarter (
  time_quarterid SERIAL PRIMARY KEY,
  quarter INT,
  time_yearid INT REFERENCES time_year
);

CREATE TABLE time_month (
  time_monthid SERIAL PRIMARY KEY,
  month INT,
  time_quarterid INT REFERENCES time_quarter
);

CREATE TABLE time_weekofyear (
  time_weekofyearid SERIAL PRIMARY KEY,
  weekofyear INT,
  time_quarterid INT REFERENCES time_quarter
);

CREATE TABLE time_day (
  time_dayid SERIAL PRIMARY KEY,
  day INT,
  time_weekofyearid INT REFERENCES time_weekofyear,
  time_monthid INT REFERENCES time_month
);
```

Solution

```
-------------------------------------------------------------------------------
-- LOCATION DIMENSION
CREATE TABLE location_celltower (
  location_celltowerid SERIAL PRIMARY KEY,
  celltower VARCHAR(100)
);

CREATE TABLE location_state (
  location_stateid SERIAL PRIMARY KEY,
  state CHAR(2)
);

CREATE TABLE location_city (
  location_cityid SERIAL PRIMARY KEY,
  city VARCHAR(100),
  location_stateid INT REFERENCES location_state
);

CREATE TABLE location_zip (
  location_zipid SERIAL PRIMARY KEY,
  zip CHAR(5),
  location_cityid INT REFERENCES location_city
);

CREATE TABLE location_streetaddress (
  location_streetaddressid SERIAL PRIMARY KEY,
  streetaddress VARCHAR(200),
  location_zipid INT REFERENCES location_zip,
  location_celltowerid INT REFERENCES location_celltower
);

-------------------------------------------------------------------------------
-- CALLER PHONE DIMENSION
CREATE TABLE callerphone_brand (
  callerphone_brandid SERIAL PRIMARY KEY,
  brand VARCHAR(50)
);

CREATE TABLE callerphone_model (
  callerphone_modelid SERIAL PRIMARY KEY,
  model VARCHAR(50),
  callerphone_brandid INT REFERENCES callerphone_brand
);

-------------------------------------------------------------------------------
-- CUSTOMER DIMENSION
CREATE TABLE customer_homestate (
  customer_homestateid SERIAL PRIMARY KEY,
  homestate CHAR(2)
);

CREATE TABLE customer_name (
  customer_nameid SERIAL PRIMARY KEY,
  name VARCHAR(100),
  customer_homestateid INT REFERENCES customer_homestate
);
```

Solution

```
-------------------------------------------------------------------------------
-- RECEIVER DIMENSION
CREATE TABLE receiver_areacode (
  receiver_areacodeid SERIAL PRIMARY KEY,
  areacode CHAR(3)
);

CREATE TABLE receiver_phonenumber (
  receiver_phonenumberid SERIAL PRIMARY KEY,
  phonenumber CHAR(9),
  receiver_areacodeid INT REFERENCES receiver_areacode
);

-------------------------------------------------------------------------------
-- CALLER DIMENSION
CREATE TABLE caller (
  callerid SERIAL PRIMARY KEY,
  areacode CHAR(3),
  phonenumber CHAR(9)
);

CREATE TABLE caller_areacode (
  caller_areacodeid SERIAL PRIMARY KEY,
  areacode CHAR(3)
);

CREATE TABLE caller_phonenumber (
  caller_phonenumberid SERIAL PRIMARY KEY,
  phonenumber CHAR(9),
  caller_phonenumberid INT REFERENCES caller_phonenumber
);

-------------------------------------------------------------------------------
-- fact table
CREATE TABLE facts (
  time_dayid INT,
  locationid INT,
  callerphoneid INT,
  customerid INT,
  receiverid INT,
  callerid INT,
  num_calls INT,
  call_length NUMERIC(12,2),
  mobile_data_usage NUMERIC(15,2),
  PRIMARY KEY (time_dayid, location_streetaddressid, callerphone_modelid,
               customer_nameid, receiver_phonenumberid, caller_phonenumberid),
  FOREIGN KEY (time_dayid) REFERENCES time_day,
  FOREIGN KEY (location_streetaddressid) REFERENCES location_streetaddress,
  FOREIGN KEY (callerphone_modelid) REFERENCES callerphone_model,
  FOREIGN KEY (customer_nameid) REFERENCES customer_name,
  FOREIGN KEY (receiver_phonenumberid) REFERENCES receiver_phonenumber,
  FOREIGN KEY (caller_phonenumberid) REFERENCES caller_phonenumber
);
```

Part 2.4 SQL Extensions for Data Warehousing (Total: 0 Points)

In this exercise you will be writing queries against the datawarehouse schema (the star schema version) designed in the previous question using the SQL extensions for grouping and window functions discussed in class.

Question 2.4.1 (0 Points)

Compute the total number of calls and call length overall and for each combination of the following dimensions and granularities: time (year), location (state), receiver (areacode). For instance, one of these combinations is year and areacode.

Solution

```
SELECT sum(num_calls), sum(call_length),
       yr, state, areacode,
       GROUPING(yr), GROUPING(state), GROUPING(areacode)
FROM facts NATURAL JOIN timed
           NATURAL JOIN location
           NATURAL JOIN receiver
GROUP BY CUBE(yr,state,areacode);
```

Question 2.4.2 (0 Points)

For each state find the three area codes receiving the most calls from locations within this state.

Solution

```
WITH calls_state_ac AS (
  SELECT sum(num_calls) AS num_calls, l.state, r.areacode
  FROM facts NATURAL JOIN location l
             NATURAL JOIN receiver r
  GROUP BY l.state, r.areacode
),

ranked AS (
  SELECT state, areacode, num_calls,
         row_number() OVER (PARTITION BY state
                            ORDER BY num_calls DESC) AS rank
  FROM calls_state_ac)

SELECT state, areacode, num_calls
FROM ranked
WHERE rank <= 3;
```

Question 2.4.3 (0 Points)

Return areacodes that have more outgoing than incoming calls.

Solution

```
WITH aggcalls AS (
  SELECT sum(num_calls) AS num_calls,
         CASE WHEN GROUPING(r.areacode) = 0
              THEN r.areacode
              ELSE c.areacode
         END AS areacode,
         GROUPING(r.areacode) = 0 AS incoming
  FROM facts NATURAL JOIN receiver r NATURAL JOIN caller c
  GROUP BY GROUPING SETS ((r.areacode), (c.areacode)))

SELECT o.areacode
FROM aggcalls o, aggcalls i
WHERE o.areacode = i.areacode
      AND NOT(o.incoming)
      AND i.incoming
      AND o.num_calls > i.num_calls;
```

Question 2.4.4 (0 Points)

Return state and months for which the total call length for calls made from this state during this month is more than 3 times higher than the total call length for each of the three previous months. **Solution**

```
WITH mon_state_len AS (
  SELECT state, yr, mon, sum(call_length) AS total_len
  FROM facts NATURAL JOIN timed
             NATURAL JOIN location
  GROUP BY state, mon, yr
)

SELECT state, total_len
FROM (SELECT state,
             total_len,
             max(total_len) OVER (PARTITION BY state
                                  ORDER BY yr, mon
                                  ROWS BETWEEN 3 PRECEDING AND 1 PRECEDING)
                            AS max3len
      FROM mon_state_len) maxlen
WHERE total_len > 3 * max3len;
```

Question 2.4.5 (0 Points)

Write a query that returns the average of the total length of all calls made per month for all granularities of dimensions location.

Solution

```
SELECT avg(total_len) AS avg_total_len,
       celltower,
       state,
       city,
       zip,
       streetaddress,
       mon,
       GROUPING(celltower) AS gcelltower,
       GROUPING(state) AS gstate,
       GROUPING(city) AS gcity,
       GROUPING(zip) AS gzip,
       GROUPING(streetaddress) AS gstreetaddress
FROM (SELECT sum(call_length) AS total_len,
             mon,
             locationid
        FROM facts NATURAL JOIN timed
       GROUP BY mon, locationid) mcalls
     NATURAL JOIN location l
GROUP BY mon, ROLLUP (celltower, state, city, zip, streetaddress);
```