Solutions Data Warehousing Exam January 2013

1. (3p) Carefully read the description stating the requirements for a data warehouse for a bank:

Loan Datawarehouse

A bank wants to build a data warehouse for storing and analyzing data about all loans issued by them.

Every loan has one or more borrowers, a starting date, a type (e.g., fixed rate or one of different types of variable rate), the branch of the bank where the loan was issued, the interest rate at the start of the loan, and the amount. Throughout the duration of the loan the interest rate may change. For every loan the purpose of the loan is recorded; e.g., to buy a car, a house, a personal loan, ... When a borrower applies for the loan, different discounts on the interest rate may be awarded; e.g., fidelity discount, discount because the borrower also bought some additional insurances, VIP discount, etc. For one loan, multiple discounts may apply. The set of discount types is fixed over time, although the magnitude of the discount may change over time due to strategic decisions. The amount of discount is independent of the branch. Every discount that has been awarded needs to be stored. When the loan ends, this is stored as well, together with an indication if the loan was fully repaid or the borrower defaulted.

For the borrowers, their date of birth, family status, monthly income, number of children and address is stored.

The following questions are prototypical for the type of query analysts want to answer based on the data warehouse:

- Give the average interest rate before discount at the start of the loan, per loan type and branch.
- For all branches, give the minimum, maximum and average interest rate per loan type and purpose.
- Give the number of loans per branch and per amount category. The amount category depends on predefined thresholds; amounts are divided into the following classes: `very high`, `high`, `medium`, `low`, and `very low`.
- Give the percentage of defaulted loans per year and per city of the branch where the loan was issued.

(a) Make a dimensional model for the data warehouse. Indicate which cube(s) are needed, what are the dimensions, measures, hierarchies, etc.

(b) Describe the tables for storing the data in a relational database; that is, in a ROLAP solution; make sure that your tables can accommodate changes in the data as much as possible.

Solution: *There was not a single correct answer; different choices may lead to different models. The level of detail given in this explanation was not necessarily expected in your answer.*

A first important observation is what wil be the subjects around which we will be building our data warehouse. In this case the most natural choice is "loan." Another option could have been "loan event"; e.g., change in interest rate. This other option, however, has a couple of disadvantages: given the prototypical queries, loan is a more natural choice. Also, it is hard to define meaningful ways to aggregate measures such as amount over these events.

Hence, every fact will correspond to one particular loan. In principle a data warehouse is read-only; once a fact is entered, it remains the same (unless an error in the database is detected which will be propagated through a type-1 update). Notice that this also holds for type-2 and type-3 updates in the dimensions; these are such that the customer address associated with facts that are already in the database does not change. Following this line of reasoning, when the loan is entered, its data will not change anymore. Therefore, we assume that a new fact will be entered every time a loan ends. Data for ongoing loans will be entered in different tables; a fact here could, e.g., be the downpayements for the loan, or the sale of a loan.

Dimensional model
The dimensions and hierarchies for loan are as follows:

- The set of borrowers; hierarchy: set of cities of borrowers; income class of the borrowers, ...
- Start date and end date of the loan; hierarchy: week, month, quarter, semester, year, ...
- Status of the loan (repaid, defaulted);
- Discounts that have been applied to the loan;
- The loan amount category;
- The loan type;
- The branch; hierarchy: city, region, country, continent where the branch is located.
- The purpose of the loan: maybe the purposes can be grouped into categories. In that case, the category would be a level in the hierarchy.

The measures for the loan are:

- the loan amount;
- a number of aggregations of the loan interest, including the starting rate before and after discount. On top of that other useful aggregations could be stored such as the maximal, minimal and average interest before/after discount;
- Another useful measures could be the total discount.

Adding multiple measures to the fact table could make the table grow very big. In this specific situation, however, it is unlikely that it will lead to a significant footprint; after all the number of loans will not increase with a million per day, in contrast to, e.g., the registration of all customer money transactions.

Relational Model.
We could opt for a star schema or a snowflake schema. As space is unlikely to become an issue for this particular case, we have chosen for a star schema. Notice

however that there could be other reasons to go for a snowflake (e.g., enforcing consistency).

The relations should be such that they allow storing the dimensional model, as well as to accomodate changes to the dimensions; e.g., customers may move, discounts can change, etc. It is important to add surrogate keys for the changing dimensions! The OLTP-key will not be unique in the data warehouse because a change in the OLTP database affecting one of the dimensions will result in a new tuple in the dimension table with the same OLTP-key.

The resulting table for the borrower dimension is as follows:

Borrower
Borrower_ID
Borrower_OLTPKey Address Income Gender No_Children ...

We keep the OLTP key if borrower because otherwise it becomes impossible to see which tuples correspond to the same borrower. To record the exact times of the changes, we could timestamp the tuples. In that case attributes to capture the validity period and maybe a flag indication if the tuple represents the most recent version should be added as well.

In case of frequent changes to some of the attributes of borrower (e.g., Income), we may opt to split-off a mini-dimension containing the profile of a borrower.

As a loan can have more than one borrower, we could decide to define a table borrower-group, grouping the borrowers that jointly have a loan. This solution particularly makes sense if the groups themselves have meaning. In this case it is likely that the groups have meaning; e.g., a married couple, or business associates.

This results in the following table:

Borrower_Group
Borrower_Group_ID Borrower_ID

An alternative is to make a bridge-table between the fact table and the borrower table. In that case it could be a good idea to add a dedicated key (e.g., loan number) to the fact table to avoid that the bridge table has to contain the complete composite key of the fact table. Make sure, however, that in that case the cluster index for the fact table should not be on this key.

Both date dimensions are stored into the same table:

Date
Date_ID
Day Week Month Quarter ...

As the discount types are fixed, we opt to combine all of them together into a junk-

dimension:

Discount_Group
Discount_Group_ID
D1_bonification D2_bonification ...

When the rates change, new groups are added.

We consider Status to be a degenerate dimension; i.e., we do not add a dimension table as the fact table already contains the one and only attribute in this dimension. Similarly for Amout_Cat.

Furthermore, for Branch, Purpose and Type also tables are added.

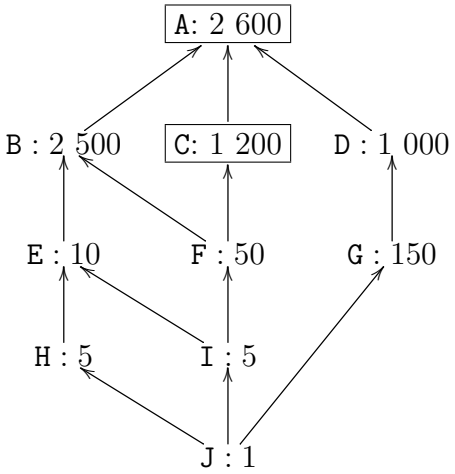
The fact table is as follows:

Loan
Borrower_Group_ID Start_Date End_Date Discount_Group_ID Status Amount_Cat Branch_ID Purpose_ID Type_ID
Amount Start_Rate_BD AVG_intrest_BD MIN_intrest_BD MAX_intrest_BD Start_Rate_Discount ...

The explanation given here is very extensive to more clearly motivate the models. This level of detail and complexity was not expected on the exam. Many other good solutions were given.

Common mistakes on the exam: no surrogate keys—would create problems when updating; sometimes a surrogate key was added to the fact table instead of to the dimension tables
relational schema is invalid; e.g., keys to fact table added into the dimensions instead of the other way around;
No keys were given
Changes cannot be appropriately captured;
Measures that become dimensions and vice versa;
For every example query a cube was defined; this is clearly violating the assumption that the schema should be such that it allows for efficiently answering ad-hoc analysis queries—dealing with ad-hoc queries implies we cannot make a differnet cube for every query;
Inability to deal with multiple discounts or multiple borrowers

2. (3p) Consider the following lattice of views along with a representation of the number of rows in each view where A is the base cuboid:



view	frequency
A	0%
B	5%
C	10%
D	0%
E	5%
F	20%
G	10%
H	10%
I	40%
J	0%

The table on the right expresses how often the different views are requested.

(a) Suppose that the top-view A and the view C have already been materialized. Select two additional views from the views B, D, E, F, G, H, I, J to materialize. Apply the greedy method described by *Harinarayan, Rajaraman, and Ullman* in their seminal paper "Implementing Data Cubes Efficiently" (SIGMOD 1996)

(b) What benefit gives the additional materialization of these two views?

Solution: In the given situation, the benifits of materializing the different views that have not been materialized yet, are as follows:

$$
\begin{array}{lcll}
B & : & (5\% + 5\% + 10\%) \times (2600 - 2500) & = 20 \\
D & : & 10\% \times (2600 - 1000) & = 160 \\
\boxed{\mathbf{E}} & \boxed{:} & \boxed{(\mathbf{5\% + 10\%}) \times (\mathbf{2600 - 10}) + \mathbf{40\%} \times (\mathbf{1200 - 10})} & \boxed{= \mathbf{864.5}} \\
F & : & (20\% + 40\%) \times (1200 - 50) & = 690 \\
G & : & 10\% \times (2600 - 150) & = 245 \\
H & : & 10\% \times (2600 - 5) & = 259.5 \\
I & : & 40\% \times (1200 - 5) & = 478 \\
J & : & 0\% \times (1200 - 1) & = 0
\end{array}
$$

So, the first extra view to materialize is E. After materializing E, the new benifits of the remaining views become:

$$
\begin{array}{lcll}
B & : & 5\% \times (2600 - 2500) & = 5 \\
D & : & 10\% \times (2600 - 1000) & = 160 \\
F & : & (20\%) \times (1200 - 50) & = 230 \\
\boxed{\mathbf{G}} & \boxed{:} & \boxed{\mathbf{10\%} \times (\mathbf{2600 - 150})} & \boxed{= \mathbf{245}} \\
H & : & 10\% \times (10 - 5) & = 0.5 \\
I & : & 40\% \times (10 - 5) & = 2 \\
J & : & 0\% \times (10 - 1) & = 0
\end{array}
$$

So, the additional benefit of materializing E and G is: 864.5 + 245 = 1109.5.

3. (2p) Consider the following example database and query over the database:

Call

Source_number	Dest_number	Time_of_day	Call_duration
0497456345	0936596743	"1pm-2pm"	30
0497456345	0936596743	"2pm-3pm"	10
0456972890	0936596743	"3am-4am"	120
0456972890	0456972890	"4p-5pm"	2
...	...	...	...

Owner

Number	Owner_ID
0497456345	001
0765382482	001
0456972890	002
0987653197	004
0936596743	005
...	...

Person

ID	Name	Age	Gender	City
001	Pete	32	m	Hasselt
002	Mary	36	f	Antwerp
003	Jean	57	f	Mechelen
004	Jeff	78	m	Brussels
...	...	...	...	...

```
SELECT P1.City,COUNT(*),SUM(C.duration)
FROM Owner S, Call C, Owner D, Person P1, Person P2
WHERE C.Source_number = S.Number
      AND C.Dest_Number = D.Number
      AND (C.Time_of_day = "1pm-2pm" OR C.Time_of_Day = "2pm-3pm")
      AND P1.ID=S.Owner_ID
      AND P2.ID=D.Owner_ID
      AND P2.City="Brussels"
      AND P1.City<>"Antwerp"
      AND P1.Gender="f"
GROUP BY P1.City
```

(a) Give two examples: one of a bitmap and one of a bitmap-join index that would speed-up the execution of this query.

(b) Explain for one of them how it speeds up the computation of the query.

Solution: The following bitmap indices are helpful for the query:

- On Time_of_day in table Call
- On City in table Person
- On Gender in table Person, although it is not selective enough. This bitmap will only be useful if ti can be used in combination with other bitmap indices.

The following indices hardly make sense:

- On Source_Number, Dest_Number in Call or on Number in Owner or on ID, Name in Person because there are too many different attribute values. Other types of indices such as the BTree will be much more useful for these cases.

The following bitmap indices do make sense, but cannot help in optimizing the query:

- On Call_duration in table Call under the condition that Call_duration is discretized or does not contain too many different values; e.g., if duration is recorded in minutes.
- On Age in table Person. For the numeric attriubtes, however, other indexing techniques that allow for range queries may be more appropriate.

The following bitmap-join indices are helpful for the query:

- On Person.City into table Owner; join on Owner.Owner_ID=Person.ID
- On Person.Gender into table Owner; join on Owner.Owner_ID=Person.ID
- On Person.City into table Call; join on Call.Source_Number=Owner.Number and Owner.Owner_ID=Person.ID;
- On Person.Gender into table Call; join on Call.Dest_Number=Owner.Number and Owner.Owner_ID=Person.ID.

Bitmap-join indices on ID and Name are not useful since the domains are too large; Bitmap-join index on Age cannot be used in the query; on Time_of_day is not preferable since the relation from call to owner and (indirectly) to person is many-one—most bitmaps will only contain 1's. Bitmap-join index on Gender is not selective enough; only useful in combination with other bitmap-join indices.

The query is helped by, e.g., the bitmap-join index on Person.City into table Call: directly find the Calls for numbers owned by persons from Brussels, skipping two joins. Using the index to find the tuples corresponding to callers not in Antwerp is likely to be far less useful because this selection criterium is insufficiently selective.

Common errors: bitmap index is given instead of bitmap-join index;
bitmap join index maps to rows of the joined table (must be: index maps values v in an attribute A of one table T to tuples in another table S that join with a tuple t of the first table having $t.A = v$)
bitmap index is used to combine the condition $City = Brussels$ and $City \neq Antwerp$, which does not makes sense since it concerns two different tuples representing different persons.

4. (1p) Compute the edit distance between the following two strings: "Belgium" and "Bulgeria".

   Solution: Compute the edit distance using dynamic programming:

		B	e	l	g	i	u	m
	0	1	2	3	4	5	6	7
B	1	0	1	2	3	4	5	6
u	2	1	1	2	3	4	4	5
l	3	2	2	1	2	3	4	5
g	4	3	3	2	1	2	3	4
e	5	4	3	3	2	2	3	4
r	6	5	4	4	3	3	3	4
i	7	6	5	5	4	3	4	4
a	8	7	6	6	5	4	4	5

   Distance equals 5.

5. (1p) Explain the concept of "Data temperature" and "warm data" that was introduced in the Teradata presentation (slide 33 and following), and how it is exploited by Teradata to improve performance.

   Solution: data temerature refers to how often data is accessed. Hot data is accessed most, warm data less, cold data the least. The warmer the data, the faster the disk it will be stored on. So, the most accessed data will be stored on the fastest (part of the) disk. See also `http://d289lrf5tw1zls.cloudfront.net/media/whitepapers/The_Data_Temperature_Spectrum.pdf` for detailed information.

BONUS (+1) Assume a large graph is stored on disk as a binary relation $R(A, B)$. Describe an efficient method to approximate how often edges in the graph are symmetric; an edge (u, v) is called symmetric if the edge (v, u) exists as well. In other words, we want to compute: $|\{(u, v) \in R \mid (v, u) \in R\}|$.

Solution: The solution relies on the approximation of the Jaccard-index (and hence indirectly for the size of the intersection of two sets) based upon hashing that we have seen in class as one of the main building blocks of the semi-streaming algorithm for counting the number of triangles in a disk-resident graph.

In one scan over the database, construct two "signatures" per node v; one for its forward neighbors $F(v)$, and one for its backward neighbors $B(v)$. This can be stored in memory. [If the signatures are too big to fit into the memory we can process the hash functions one by one, similarly as for counting the triangles. We omit this complexity here.]

Then go over all nodes v, and compute $\sum_{v \in V} |F(v) \cap B(v)|$. The size of the intersection can be estimated directly from the signatures.

Common errors: solutions based on first counting triangles; since there is no relation to the number of symmetric edges and the number of triangles this approach is automatically doomed.
Using sketching methods to count the number of edges (u, v) and then check for every edge (u, v) if (v, u) has a non-zero frequency. This won't work as the streaming methods for counting frequencies are approximate and very inaccurate for low frequencies. The frequencies that need to be estimated in this approach are 0 or 1.