Mission 3 – Database Design

This mission aims at improving your understanding of the different levels of database design (conceptual, logical and physical) and their correlated use. You'll learn to apply normalization theory and critical thinking when it comes to designing or reverse-engineering a relational database.

Check your understanding

At the end of the semester, you better understand the differences between designing at the conceptual, logical and physical levels. When it comes to designing a database in practice:

- You are able to use the Entity-Relationship model for building a conceptual schema, identify entities and relationships in the domain at hand, elicit keys, design abstract data types, and state business rules / constraints in natural language.
- You know and can critically apply transformation rules to derive a relational schema from an Entity-Relationship conceptual schema.
- Provided the logical language is expressive enough (e.g. **Tutorial D**), you are able to create views and formally state constraints on the relational schema.
- You are able to use normalization theory (i.e. normal forms, functional & join dependencies)
    - To reason about a relational schema and check/fix it,
    - To start designing from legacy data, and
    - To compare candidate designs with respect to the requirements at hand.
- You are able to translate a logical schema (e.g. in Tutorial D) to an implementation in SQL. You understand that tradeoffs and patterns are needed in practice, since not every domain and constraint has an implementation in SQL.
- You also create necessary indexes in SQL, for basic performance tuning (e.g. of views).

Submission & Deadline

This mission must be completed individually by the **2015/03/25 23h59**. A short report must be submitted on ModdleUCL by then at last.

Open questions are for learning and exploring; your report must not contain answers for them. Closed questions, for which an answer is due, are indicated in green underline. Correctly answering all those questions is a success baseline. If you want to earn more points, do not attempt the "quantity" method (e.g. a very long report with blablabla answers), but the "quality" one (e.g. choose one or two questions for which design exploration is needed and provide a short quality answer that proves that you did your exploration homework and learned something).

References

[1] R. Elmasri, S.B. Navathe, *Data Modeling Using the Entity-Relationship Model*, Chapter 3 in Fundamentals of Database Systems, Fifth Edition, Addison Wesley, 2007.

[2] R. Elmasri, S.B. Navathe, *ER- and EER-to-Relational Mapping*, Chapter 9 in Fundamentals of Database Systems, Fifth Edition, Addison Wesley, 2007.

[3] R. Elmasri, S.B. Navathe, *ER- and EER-to-Relational Mapping*, Chapter 9 in Fundamentals of Database Systems, Fifth Edition, Addison Wesley, 2007.

[4] R. Elmasri, S.B. Navathe, *Functional Dependencies and Normalization for Relational Databases*, Chapter 14 in Fundamentals of Database Systems, Fifth Edition, Addison Wesley, 2007.

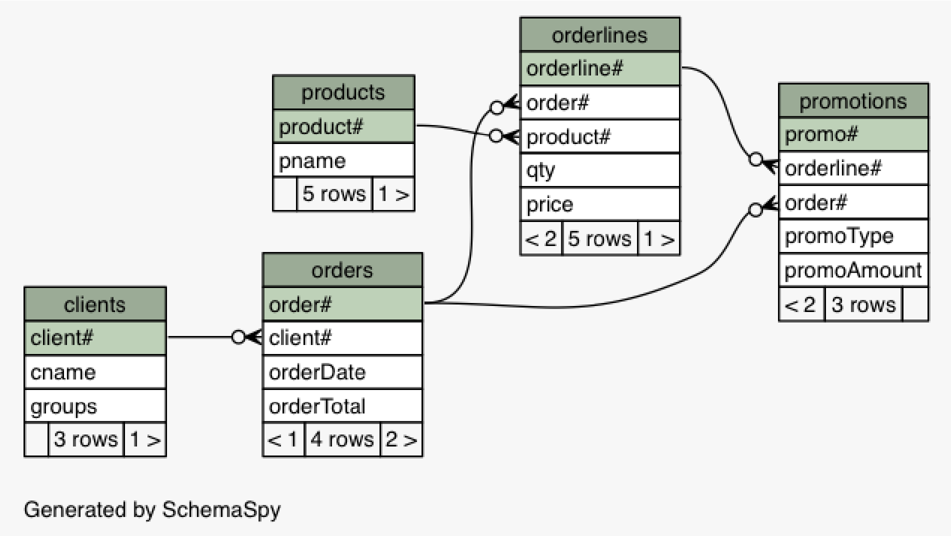
[5] H. Darwen, *Database Design*, Chapters 7 & 8 in An Introduction to Relational Database Theory, Bookboon, 2012

[6] The SQL Language in PostgreSQL , PostgreSQL 9.3.2 documentation, http://www.postgresql.org/docs/9.3/static/sql.html, last retrieved January 2014.

[7] J.L. Hainaut, *Bases de Données, Concepts utilisation et développement*, seconde édition, Dunod, 2012.

Exercice 1

We consider the reverse engineering of a (very simplified) e-commerce shop database. The existing PostgreSQL schema (somewhat representative of what Object-Relational Mappers tend to generate in practice) looks like this:



The predicates of those relvars are as in the table below (The files shop-schema.sql and shop-data.sql that are provided with this problem statement provide a more comprehensive documentation and help you playing with that database in real, if needed).

Relvar/table	Predicate
clients	Client \`client#\` is called \`cname\` and belongs to the groups listed in \`groups\`
products	Product \`product#\` is named \`pname\`
orders	Order n° \`order#\` has been sent on \`orderDate\` by client \`client#\` and amounts for a total of \`orderTotal\` euro, all taxes included
orderlines	Order line \`orderline#\` belongs to order n° \`order#\`. It covers the ordering of \`qty\` amount of product n° \`product#\`, one unity of which having cost \`price\`
promotions	Promotion n° \`promo#\` is for order line n° \`orderline#\`, which belongs to order n° \`order#\`. It is a \`promoType\` promotion for a total amount of \`promoAmount\` for

that order line (i.e. it is not a per product promotion).

Question 1.a

Using normalization theory (up to BCNF) find as many design flaws as possible. For each of them:

1. State what design principle (e.g. what normal form) is violated and provide a convincing proof (in natural language)
2. Explain why the design is poor due to that particular violation (e.g. lack of orthogonality, redundancy, missing constraint, etc.)
3. Illustrate your argument at point 2 with an example (e.g. show how the current design accepts wrong data with respect to the domain, find a new requirement that will be hard to tackle)
4. Quickly explain how the design should be corrected to fix the flaw.

You should at least find 3 serious flaws. There are more. The number of flaws you find does not matter much if you cannot explain why these are indeed flaws with convincing arguments.

You can assume usual simplifying assumptions for such a shop (e.g. it does not make much sense to order the same product in multiple order lines since the quantity could be simply adapted). In doubt, state your assumptions first and make your analysis on that basis.

Report 3 flaws (what relvar, what attribute) and what normal form is violated.

Question 1.b (optional)

Extra points: design a schema with all relvars in BCNF for a corrected version of the shop (state constraints not enforced by design). Tip: you can safely remove surrogate keys (SERIAL aka autonumber) that you find unnecessary since they don't really capture domain information (and provided another unique identifier can be found).

No report is due for this question.

Exercice 2

The 6 march of 2015, Robert Sheldon wrote an article called "Hot to get Database Design Horribly Wrong", that you can find at https://www.simple-talk.com/sql/database-administration/how-to-get-database-design-horribly-wrong/ (last retrieved 11 of March 2015).

Go read it. What failures (from those explained in that paper) can you easily spot in Exercice 1?

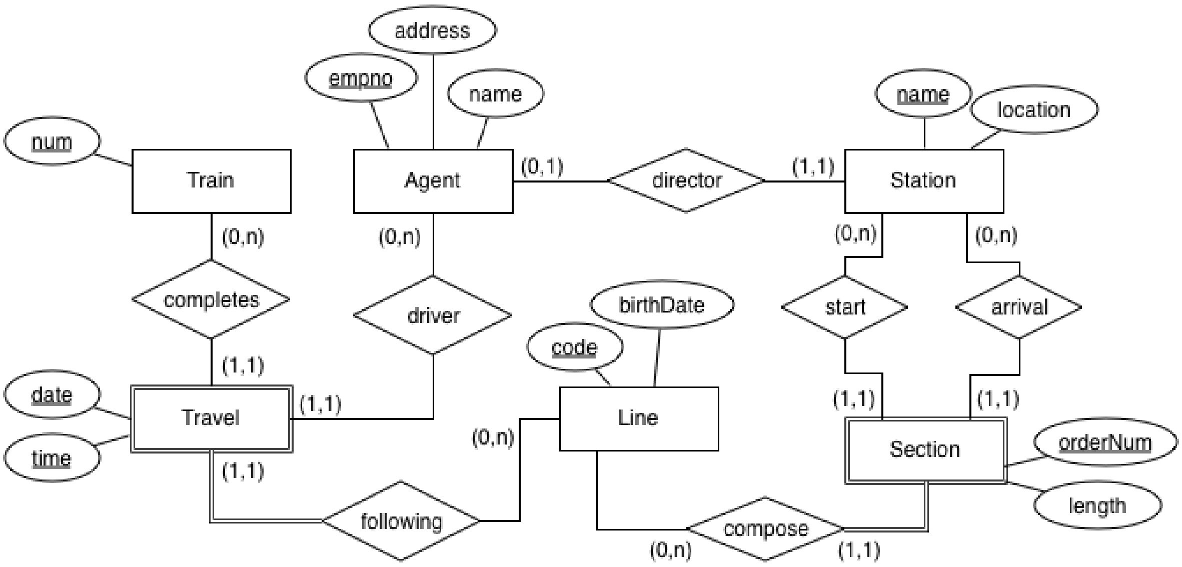
Make a very short summary (3 lines) of the failure that you find the most important to avoid.

Exercice 3

The following excerpt is extracted from [7], Chapter 11, page 326.

> "We consider train travels. A travel is done by a train (for which a unique train number is known) at a given date and given time within that day, following a known train line (identified by a code, and characterized by a "birth" date, i.e. when the line has been put in service). The train driver is a so-called Agent. The latter has a unique employee number, a name, and an address. A train line is composed of consecutive sections, each of a given length. A line section starts at a station and arrives to another. A station is identified by a name and has a location. Each station is directed by an agent, yet not all agents are responsible for one."

The following entity-relationship diagram is proposed in [7] as conceptual model for the domain at hand (except that we use the syntactic notation from the Elmasri's textbook [1], not the one used in [7]).



Question 3.a

Make sure to you fully understand the notation by reading [3] first, then answer the following questions:

1. Why are doubly stroked lines sometimes used for entities (e.g. Travel) and roles lines (e.g. towards the "following" relationship)? What does that mean?
2. What attributes compose the identifier of Section?
3. What does Section/orderNum capture?

Question 3.b

Using the ER-to-Relational rules explained in [2] (and possibly [3]), convert this conceptual model to a relational model.

You can use a tool such a DB-Main (www.db-main.be) or any diagrammatic software, even those having conceptual-to-logic automated support; alternatively, you can create the logical design manually (we strongly encourage you to choose this strategy, as your understanding of design will be definitely better).

Provide the relational schema, either as an image or as a short Tutorial D script. In either case, the attribute types, candidate keys and foreign keys must be included.

Question 3.c

Did the conversion rules generate NULLable attributes? If yes, which ones and how can you avoid them?

Question 3.d

Suppose the last domain requirement becomes "Stations are directed by at most one agent, yet not all stations are. Not all agents are responsible for a station either".

What modification of the conceptual schema does that imply? What consequences does it have on the logical schema? How can you avoid a NULLable attribute here? Why should you?

Question 3.e

The multiplicities of the "compose" relationship between Line and Section allow a Line to be composed of no section at all, which does not make much sense.

What modification of the conceptual schema is needed to enforce every line to have at least one section? What consequences does it have on the logical schema? Provide a constraint definition in Tutorial D for enforcing a line to have at least one section? What about SQL?

Question 3.f

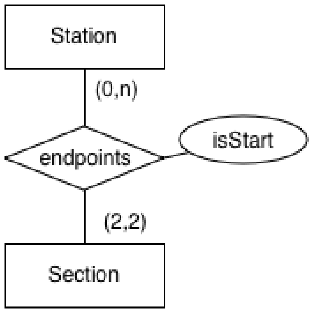
Our conceptual modeling might have a weakness.

Currently, the length of a Section is dependent of a triple {start, end, orderNum}. Under the domain assumption that only one physical rail track directly connects two stations, it seems obvious that the length of a Section is not dependent of where it is used in a Line (orderNum).

How can you state this weakness rigorously using normalization theory? How can you fix it (enforcing the domain assumption just stated above)? What consequences does that have on the conceptual model?

Question 3.g

The "start" and "arrival" relationships have generated two different attributes in the relvar for sections. An alternative modeling could be as illustrated below:



What consequence does that have on the identifier for Section? What logical design do you end up with if you follow this conceptual modeling?

Among the two, what relational design do you prefer? Answer by explaining at least one advantage it has. Any drawback?

No report is required for this question.

Question 3.h

Suppose the following requirements hold:

- Both train drivers and station directors are employees.
- No driver is also a station director and vice-versa.

Modify both the conceptual and the logical design to capture those requirements more adequately. Explain those changes in max 3 lines or a small, yet clear, picture.