Kantamestarit
Base Master Solutions

MIRACLE

Database design in agile project

DOAG 16.11.2011

Heli Helskyaho

Introduction, Heli

- Graduated from Helsinki University (Master of Science, computer science)
- Worked with Oracle products since 1993, worked for IT since 1990
- Database!
- CEO for Kantamestarit Oy since 2000
- CEO for Miracle Finland Oy since 10/2010
- Board member for OUGF since 2001
- Chairperson for OUGF since 2007 (vice-chair for years before that)
- Ambassador/Spokesperson for EOUC since late 2007
- Oracle ACE since 2011
- Heli.Helskyaho@kantamestarit.fi,Heli@miracleoy.fi.

Introduction, Topic

- Why do we still design the database?
- What is database design?
- How to design a database on agile systems development process? Why is it different than before?

Why to design?

- ”Data is the most valuable property in our company”
- Why do we need to design the database? We already design the application!
- Why is designing a database a different thing:
  - Point of view
  - First increment vs. 20 years from now
  - Same terminology, different meaning -> misunderstandings

  -...

Example

- 1. iteration: ORDER
- Info we have: layout for the user interface, very general use case (analysis) on making an order
- So let’s model....

ORDER

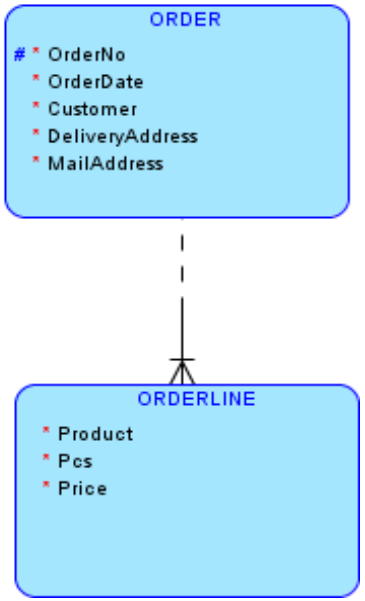
Order#	
Order Date	

Customer (name)	
Delivery address	
Zip code and city	

ORDER LINE

Product	Pcs	Price	Total

Make an Order



What went wrong?

- Where can we find the customer for the order? Who cares. The usecase Create a Customer is on the second iteration.
- And exactly the same with the product information.
- So we have modeled the customer info right to the order -> duplicated information, no references but attributes,... And the same thing with the product information.
- How do we know what a certain customer has ordered?
- How do we know who has ordered certain product?
- We have no way to produce any statistics...but nobody (=application designer, project manager,...) cares because statistics will be in iteration 7. We will think about that later.

What is important?

- It is very important to plan the iterations and increments well and understand where and how much they overlap. The less they overlap the better.
- Always start with the most difficult and the most centric things, do not start with the easy things.
- Always ask questions if you have a feeling that something is not right.

Where to aim?

- A total mistake in modeling is difficult and expensive to fix later.
- When designing the database you must understand **the big picture**: concepts and how they are connected to each others (entities and relationships)

- GOAL 1: **Data integrity** and **quality**.
- GOAL 2: How to **save** and **retrieve data**.

Designing the database

1. **Requirement analysis**: finding and analysing the requirements the future end users have

Result: specification of user requirements

- **data** requirements
- **functional** requirements

Also requirements for security, performance, …

Designing the database

2. **Conceptual design**. ”Interpretation” of all the requirements to a formal presentation (conceptual model).

Result: conceptual schema, also textual documentation is possible (to make sure all the knowledge is documented)

This is a tool for communication with end users.

Designing the database

3. **Logical design**: transforming the conceptual model into a logical data model and a logical schema that the RDMS understands

Result: relational-database schema

(relational schemas and constraints)

Designing the database

4. **Physical design**: instances, tablespaces, indexes, disks …

And all of these phases over and over again…

Designing the database

Might also include (depending who you ask)

- 5. **Transaction design**: trasforming the functional desriptions in conceptual model to operations that will handle the database queries, transactions.

More about …

- Conceptual design
- Logical design
- Physical design

Conceptual design

- Main idea: saving the data and retrieving it -> **DATA**
- Information: everything you can find
- Requirements and analysis usually half way -> need a lot of questions and answers
- Interview the end users! Officially, unofficially
- Model only the **target**, not the whole world
- Use right terminology and clear names, much easier to communicate with the end users (one of the reasons to model!)

Conceptual design

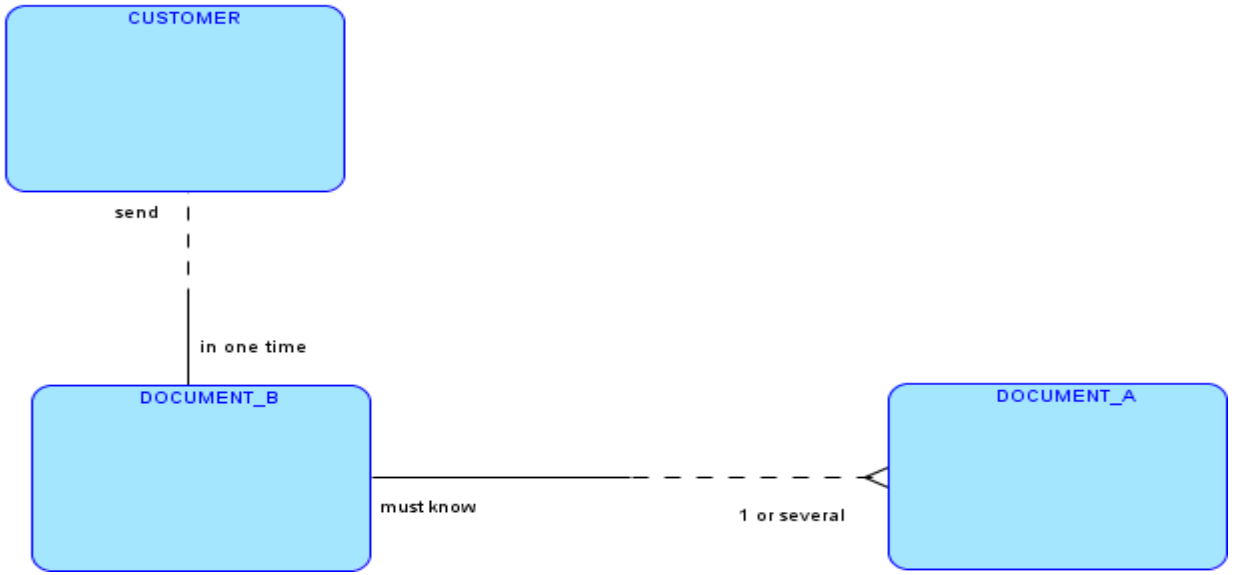
- Try to find and understand the main concepts and their relationships (these are the most difficult to change during the iterations)

Conceptual design

- ”At one time one will make 1 or more Document/-s A and exactly one Document B for a Customer. One must know exactly which Document B was with which Document/-s A.”

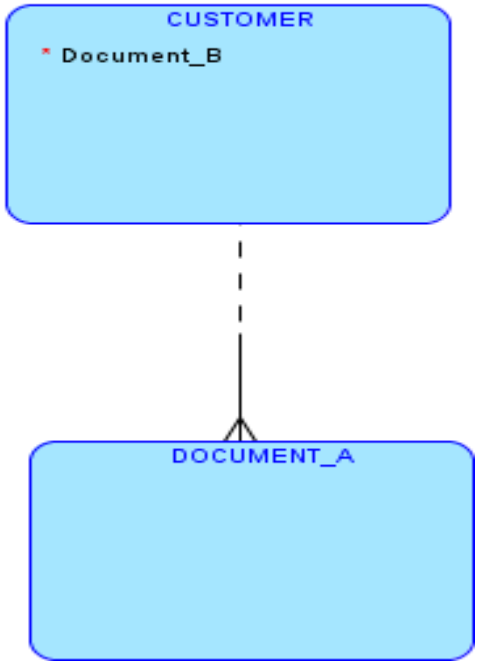
- EASY AND CLEAR!

Conceptual design



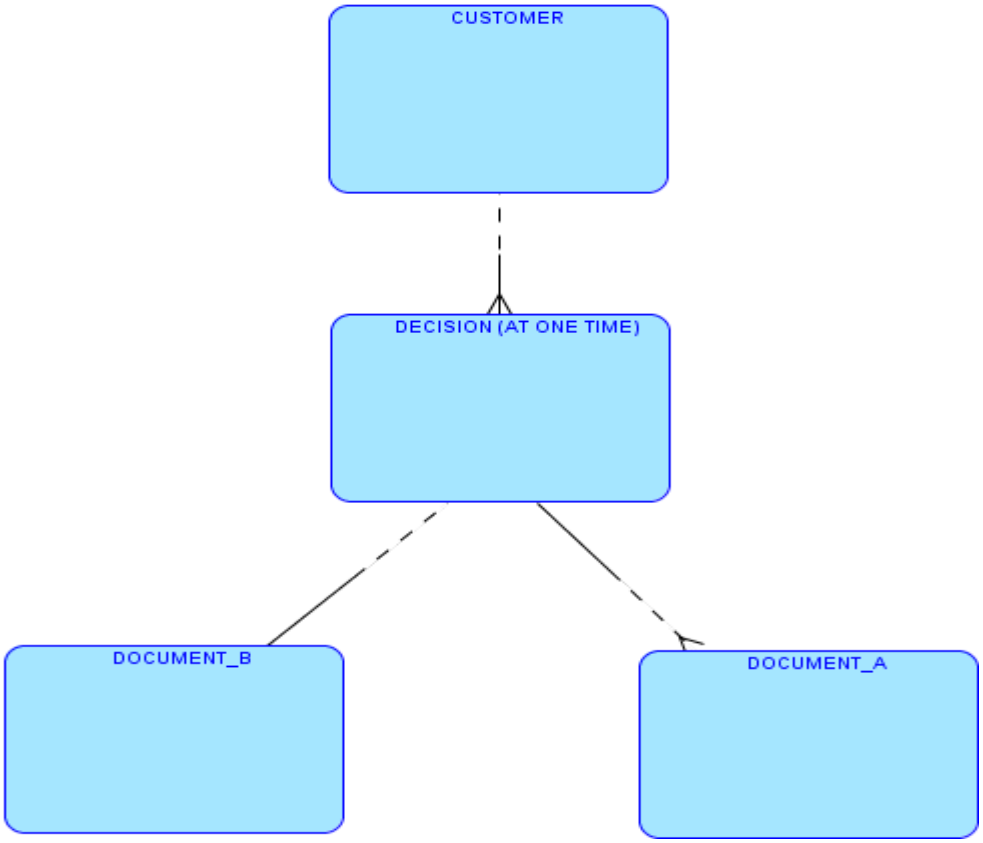
Conceptual design

- OR?



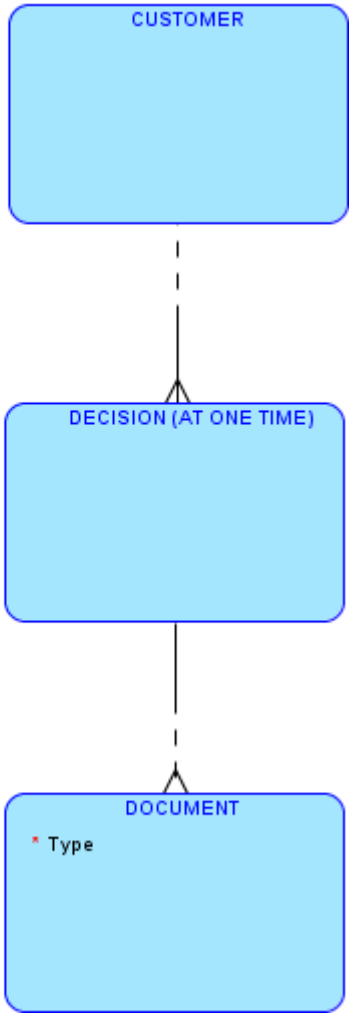
Conceptual design

- OR?



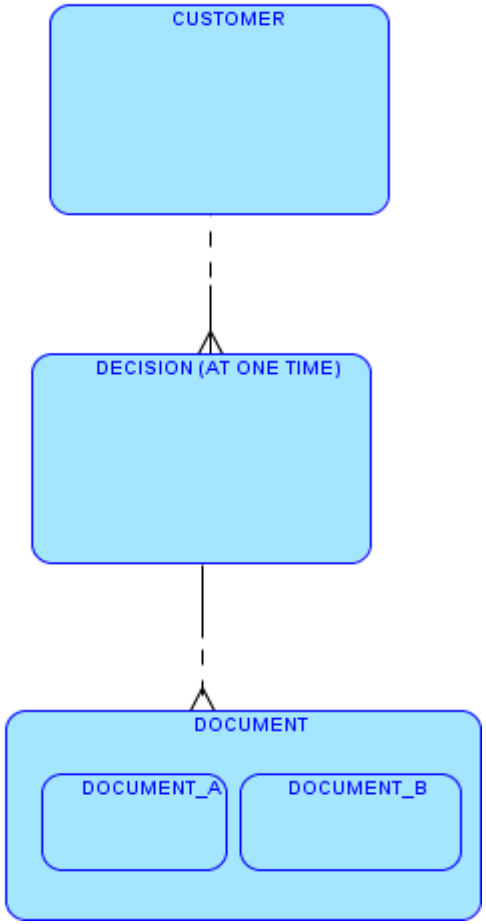
Conceptual design

- OR?



Conceptual design

- OR?



Conceptual design

- Or one of the other dozen different ways to model this requirement?
- Modeling is difficult because
  - Spoken/written language is not exact
  - Usually all the ”important” things are those that ”everybody knows” so they are not told.
  - At this stage we do not know one important thing: how the data will be **retrieved**? That will be on iteration 9…
- Modeling is **mandatory** because then the person modeling the database realizes **what must be asked**!

Conceptual design, principles

- Everybody has their own and preferred ways of modeling (”handwriting”) and that is ok, but it is important to be **consistent** at least inside one model or system or application.
- Naming standards
- Documentation
- Comments from collegues
- Reviews, audits,…
- …

Logical design

- Start with the conceptual model
- **Specify and sharpen**, make **decisions** on more difficult modeling issues
- Now you should already have the information how the database is supposed to be used, functional descriptions (usually you do not have it, not all at least ☹)

Logical design, Transformation

Two different kind of transformations

(in both the idea behind is conceptual model -> logical model):

1. Transformation from notation to another
   - UML -> ER
   - ER -> UML
   - ER and UML
2. Transformation from conceptual ER to logical ER

ER to UML

- Entities, attribute - > easy
- Relationships -> easy
- Relationship cardinalities -> easy
- Generatization, specialization, inheritance -> more to think about

ER to UML

- Transformation is quite simple from ER to UML (can even be automated) but...

Difference between ER and UML

- UML: both data requirements and functional requirements (not all the data requirements though)
- ER: data requirements
- Notation is not the issue but **what** are you trying to model. The **point of view** and the **need**.
- **Is it possible in just one ”picture” to model both data requirements and functional requirements? All of them?**

Conceptual to logical, transformation

- Usually very simple (depending of course on the quality of the conceptual model in terms of the relational theory)
- Usually the cardinality and participation constraint of the relationships and the fact if the relationship has its own attributes affect the most

Conceptual to logical, transformation

- Entity -> entity
  - CUSTOMER -> CUSTOMER
- Attribute -> attribute
  - FIRSTNAME -> FIRSTNAME
- Relationship without attributes -> relationship
- Relationship with attributes -> entity+ all needed relationships

Conceptual to logical, transformation

- M:n-relationship-> new entity with relationship to all needed entities, primary key=combination of the relationships, attributes

Example M:n

- A customer can order several products and a product can be ordered by several customers.

CUSTOMER ——— makes ——— PRODUCT

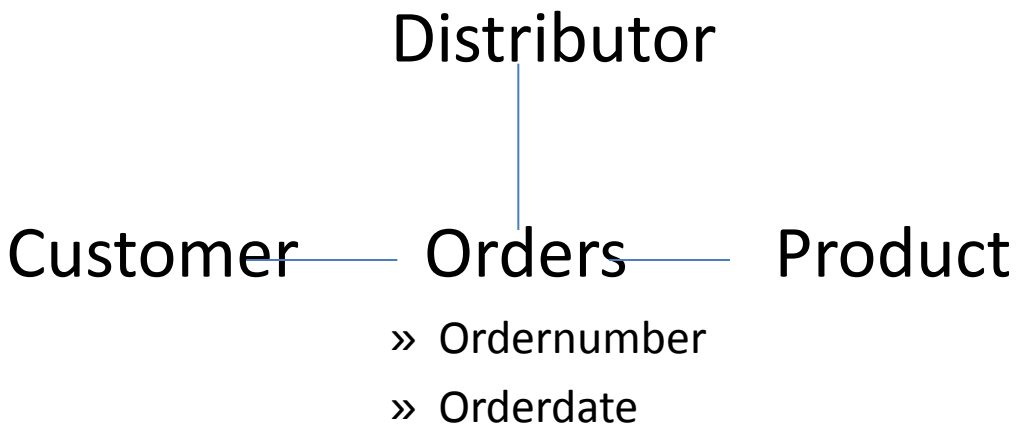
an order

CUSTOMER-ord- ORDER - comb - PRODUCT

ER conceptual to ER logical, transformation

- Complex relationship -> entity + all needed relationships
- Having a complex relationship means **THINK**
- ER is only two dimentional...
- Usually questions are needed about the relationships.

Complex relationship



Relationship has attributes? -> new entity+all the relationships

Conceptual to logical, transformation

- Recursion (product hierarchy)
- Multivalued attributes -> new entity unless other arguments to do something else
- 1:1 relationship. Why? There might be an explanation like lifecycle.
- Subtypes, supertypes, hierarchy: think
- ...

Normalization

Always the main principle and with a reason!

- All the attributes are **atomic**. And the data has been saved only **once**.
- **Why?** To make sure the data is good quality and correct.
- Optimising by not normalising? OK, but you are now getting data which is not necessary good quality...

ER, UML?

- I would recommend, ER for data requirements and UML for functional requirements
- Then check the UML model agains the ER model and make sure all data requirements that should have been implemented at the UML model are there (there is a meaning for them)
- I would recommend to do it manually, so you can also review the ER model at the same time.
- Try to test the models really work.

ER, UML?

- Remember that it is possible that the class model has information that the data model does not have
  - Age etc.
- Also data model can have information that class model does not
  - Creator, created (triggers, UI knows nothing about it)
- So the models are not necessary identical!

Other issues

- Database privileges
- History data
- Cleaning the database
- Documentation
- …

Physical design

- In short:
  - Start with the logical model
  - Design the disc usage, count the need for the space etc.
  - Indexes
  - Schemas
  - Implementations to all objecs (tablespace, storage, user rights,...)
  - DDLs

Physical design

- How, who and where to document?
- How to document any changes (patch, changes in database objects, ...)?
- Backup and recovery (test that is works every now and then!)

Performance

- Plan, count, ...
- Test! With a real material (size, quality,...) and the right versions of program.

Tuning

- Hardware and operating system
- DBMS
- Model level (logical, physical, transaction)
- All three levels affect each others

Tuning the logical model

- Vertical partitioning
  - Creating entities with just some attributes and using additional entities for remaining attributes (vehicle, car, lorry, motorcycle or ”the most commonly used data” and ”less used”), no joins for the ”common ones”, joins only for non-common ones
- Horizontal partitioning
  - Several entities for different values or meanings
- All possible in one table, no joins

Tuning the logical model

- Always problematic because affects the code, tests, test material,…
- Denormalization: even more problematic because the data will be multiplied and must be taken care by coding…

Tuning the physical model

- Among others:
  - clustering
  - Physical vertical partitioning, ”partitioning”
  - Materialized views

DBMS will handle (no need for coding). Usually no changes for the code needed (sometimes it is needed, partitioning)

Indexing

- S **might** be faster
- IUD slower
- Not a silver bullet

Tuning the queries

- Transform the query to a more efficent form (execution plan) with same outcome (result) than the original query (equivalent).

Tuning

- Ask direct questions. (”Nothing is working” is not an answer)
- Be systematic and find the real problem
- Solve the problem
- Document the problem and the solution (new problems because of the solution?)

Refactoring the database

- Analyze the change from the **database** perspective
  - The phase of the database, development vs production
  - The phase of usage of the database, development vs production
  - Completely new database vs ”old” database
- Analyze the change from the **project** perspective (cheep coding now is not an excuse for a bad database design)

Refactoring the database

- Analyze the change from the maintenance point of view (would it be more clear to call discount DISCOUNT than AMOUNT?)
- Versioning
- Always remember somebody will maintain this system and this database
- Always remember clear and understandable solutions, do not try to be too clever

Refactoring the database

- The problems related to refactoring the database usually have nothing to do with the database but everything to do ”on top” of the database
- Refactoring the database is nothing special, just normal database design process

Agile database design

- Agile makes database design harder than ever. Long experience in different projects is most valuable so you might be able to predict some problems beforehand.
- **Agile is no excuse for not designing!**
- Agility costs. It is very important to analyze everythig well so you know what is the price tag of your desicion. (short term, long term)
- Maintenance!
- The more agile the more you must design and analyze.

A tool

- To be able to be agile or at least flexible one needs a tool for database design.
- From Oracle the solution is Data Modeler (part of the SQL Developer)
- To have less complains about the changes in database people testing needs also tools to be able to change their test material and testcases...

Conclusions

- If you think you have/will have valuable data in your database, you MUST design the database.
- The database must be designed by a person who understands how the database works and knows how the DATA should be modeled.
- The database can not be designed alone in your room, you need all the different experts to give you the right info for making desicions.

Conclusions

- The database must be designed and for the right purpose .
- Data integrity and data quality must be the guiding rules.

Conclusions

- If the timetable is tight you need more and more thinking and designing to be sure you are making the right desicions to be sure you will not need to change everything everytime.
- Prioritization! For everybody (even the end users). Nothing is more stressful than too much work in no order and too little time.
- Team spirit will help you through anything! Positive attitude!

THANK YOU!

QUESTIONS?

heli@miracleoy.fi