How to design a database?

Heli Helskyaho, Matias Helskyaho

KCDC 2024

Miracle Finland Oy

KANSAS CITY DEVELOPER CONFERENCE
KDCC
2024

Titanium Sponsors

Progress®

ngrok

DON'T PAN!C LABS™

Platinum Sponsors

Temporal

SUSE

Pulumi

snowflake

Procure SQL
DATA ARCHITECTURE AS A SERVICE

Particular
Software

NAIC
NATIONAL ASSOCIATION OF INSURANCE COMMISSIONERS

NIPR
NATIONAL INSURANCE PRODUCER REGISTRY

LaunchDarkly

H&R BLOCK

ORACLE
Database

LUNAVI

docker®

pangea

CloudBees.

squid™

Gold Sponsors

touchnet
A Global Payments Company

TEKsystems
Own change

SHAMROCK™
TRADING
CORPORATION

QUEST
ANALYTICS

EVENT STORE.

BUILDERTREND

ADAPTIVE™
SOLUTIONS GROUP

honeycomb.io

accumatch
CONSULTING

AMUSEMENT CONNECT

AVITURE

FEDERAL RESERVE BANK of KANSAS CITY

VML

TURNBERRY
SOLUTIONS

sonatype

yugabyteDB

LEAN
TECHNIQUES

Speaker Dinner

Common Room

Friends of KCDC

outsystems

IowaComputer
GURUS

Google Cloud

Gradle

JETBRAINS

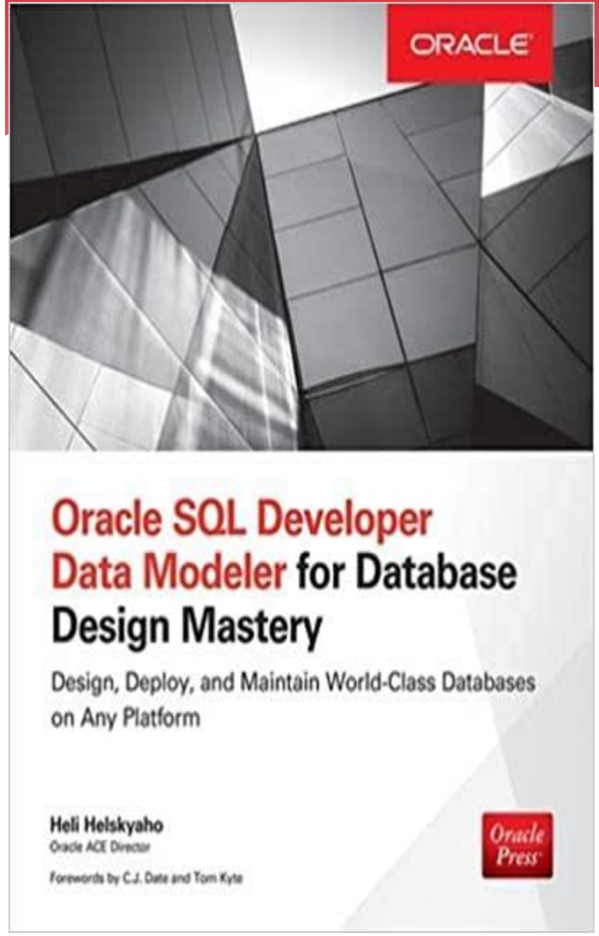
Introduction, Heli

- Graduated from University of Helsinki (Master of Science, computer science), currently a doctoral student, researcher and lecturer (databases, Multi-model/Converged Databases) at University of Helsinki
- Worked with Oracle products since 1993, worked for IT since 1990
- Data and Database!
- CEO for Miracle Finland Oy
- Oracle ACE Director
- Public speaker and an author
- Author of the book Oracle SQL Developer Data Modeler for Database Design Mastery (Oracle Press, 2015), co-author for Real World SQL and PL/SQL: Advice from the Experts (Oracle Press, 2016), Machine Learning for Oracle Database Professionals: Deploying Model-Driven Applications and Automation Pipelines (Apress, 2021), and Extending Oracle Application Express with Oracle Cloud Features: A Guide to Enhancing APEX Web Applications with Cloud-Native and Machine Learning Technologies (Apress, 2022)

A Oracle ACE Director

MIRACLE
Miracle Finland Oy

Books


ORACLE

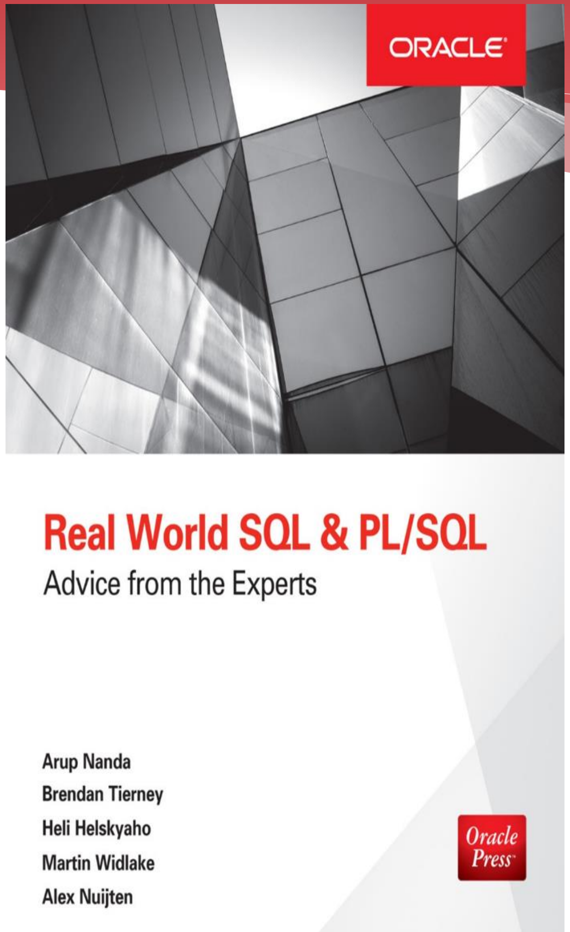
Oracle SQL Developer Data Modeler for Database Design Mastery

Design, Deploy, and Maintain World-Class Databases on Any Platform

Heli Helskyaho
Oracle ACE Director

Forewords by C.J. Date and Tom Kyte

Oracle Press

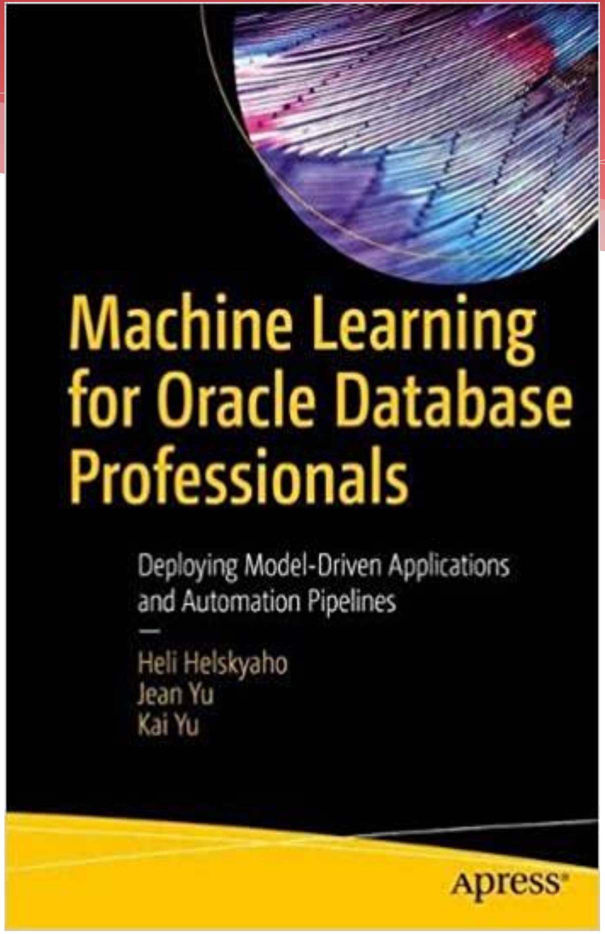

ORACLE

Real World SQL & PL/SQL

Advice from the Experts

Arup Nanda
Brendan Tierney
Heli Helskyaho
Martin Widlake
Alex Nuijten

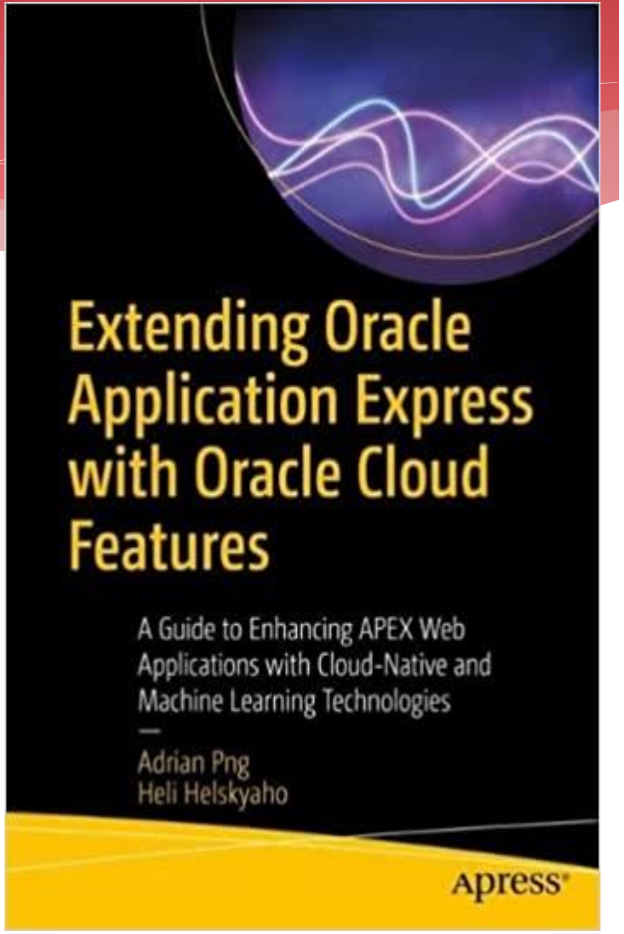
Oracle Press


Machine Learning for Oracle Database Professionals

Deploying Model-Driven Applications and Automation Pipelines

Heli Helskyaho
Jean Yu
Kai Yu

Apress


Extending Oracle Application Express with Oracle Cloud Features

A Guide to Enhancing APEX Web Applications with Cloud-Native and Machine Learning Technologies

Adrian Png
Heli Helskyaho

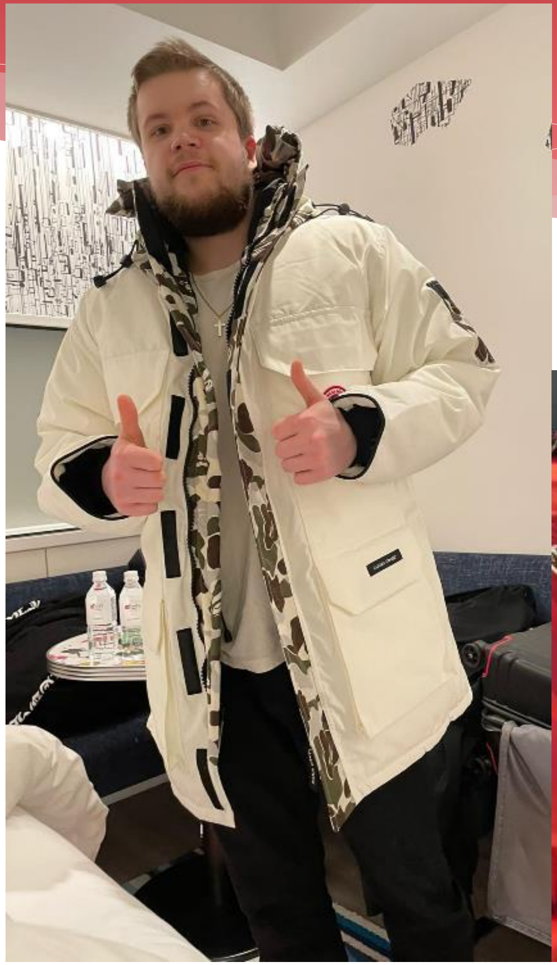
Apress

MIRACLE
Miracle Finland Oy

Matias

Oracle ACE Pro

- Consultant
  - Miracle Finland Oy
- Who am I?
  - On IT since birth
  - Professionally a couple of years
  - OCI, networks, IOT, ML, analytics,..
- Hobbies
  - Love learning cool stuff
  - Playing with tech devices



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 the application not enough?

- Point of view (saving and retrieving data vs. UI)
- First increment vs. 20 years from now
- ”the whole picture” vs. increments
- Different goals/targets:
  - Code tables vs. Code files (how about the data integrity?)
  - How about analysis, reports, … everything else but the UI that the data is used for
- Same terminology, different meaning -> misunderstandings
- …

Why to model the data?

- To facilitate communication about the requirements
- To find the questions that should be asked
- To understand the requirements

What is database design?

- 4 (5) phases, over and over again
  - Requirement analysis (DM: logical)
  - Conceptual design (DM: logical)
  - Logical design (DM: relational)
  - Physical design (DM: physical)
  - (Transaction design) (DM: process)

Conceptual design

- Main idea: saving the data and retrieving it -> **DATA**
- Information needed: 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

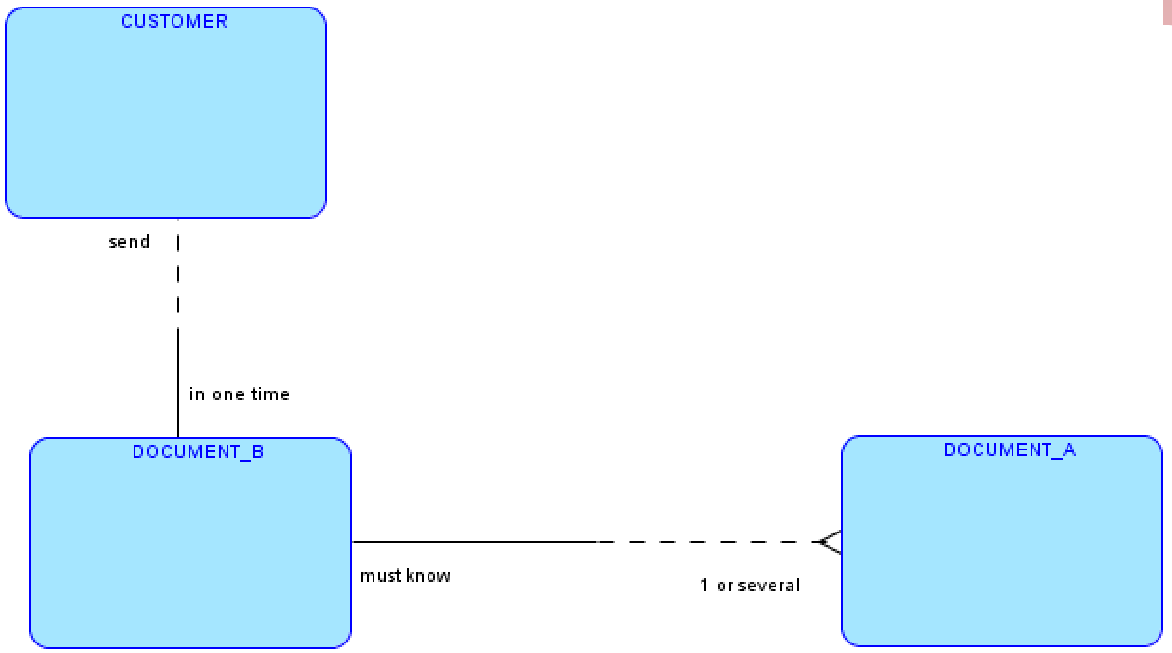
Conceptual design

- Use right **terminology** and clear names, much easier to communicate with the end users (one of the reasons to model!)
- Try to find and understand the **main** concepts and their relationships (these are the most difficult to change during the iterations)

Conceptual design, an example

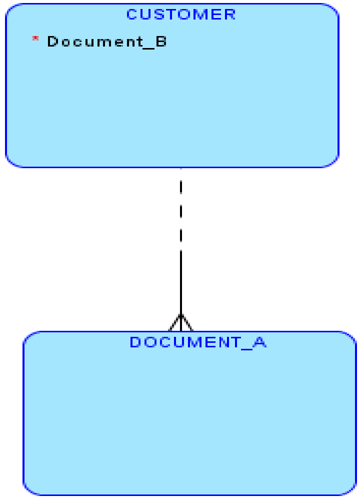
- ”At one time one will make 1 or more Document/-s A and exactly one Document B for a Customer. One must know exactly which Document B was with which Document/-s A.”
- EASY AND CLEAR!

Conceptual design, an example



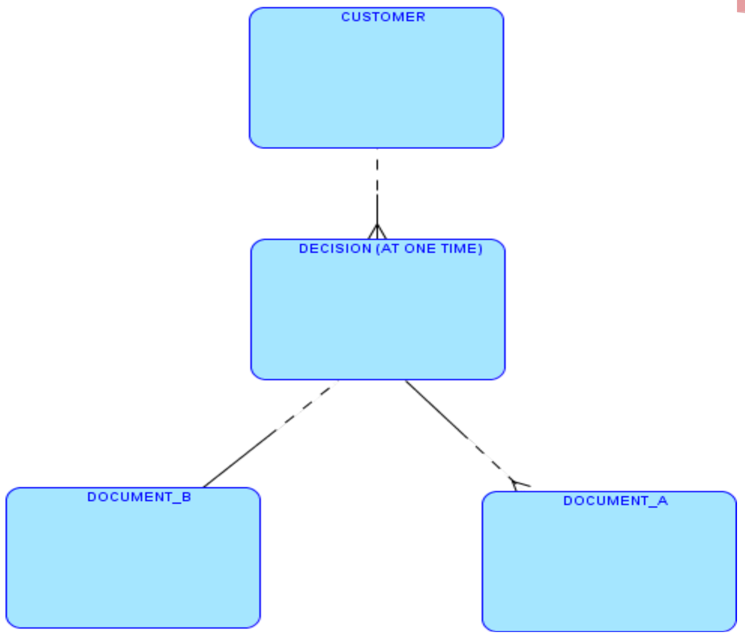
Conceptual design, an example

- OR?



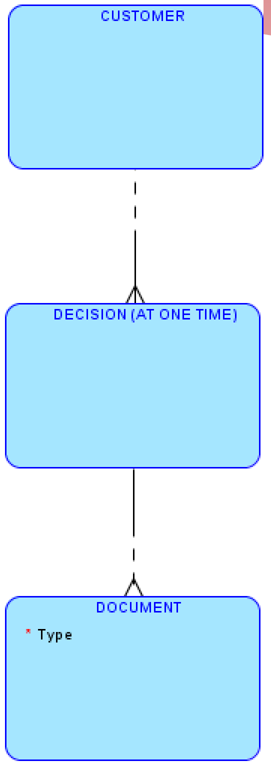
Conceptual design, an example

- OR?



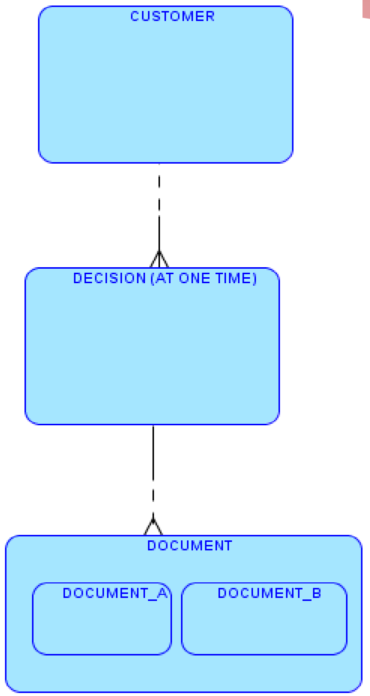
Conceptual design, an example

- OR?



Conceptual design, an example

- OR?



Conceptual design

- Or one of the other dozen different ways to model this requirement?

Conceptual design

- 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 when modeling the database designer 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,…
- …

Design a Data Model for the Application

- Not the ”whole world”
- System definition: the scope and boundaries of the database application (helps to define entities, attributes and relationships, understanding how this system sits to the whole architecture and other IT systems)
- User views (helps to define roles and privileges, understanding how the system will be used)
- database and application design are parallel activities of the database system development lifecycle

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/recommended (to make sure all the knowledge is documented)

This is a tool for communication with end users.

Requirement analysis and Conceptual design

- Collecting requirements and analyzing them
- Fact-finding: interviews, questionnaires, existing documentation,… (recordings)
- Requirements specifications
  - data requirements
  - functional requirements (performance, security, backup/recovery,..)
- Completely neutral to any technology

Requirement analysis and Conceptual design

- Why entity-relationship model (ER)
  - Defining the tables directly based on requirements can be too difficult and lead to a wrong db schema.
  - Based on a good ER it is easy to generate the relational model (which is at least on 3NF)
- Data Flow Diagrams (DFD)

Methods

- Collecting requirements
  - Centralized: all requirements merged into a single set of requirements
  - View: requirements for each user view as separate lists. Separate data models (local data model), that in later stage are merged as global data model.
  - A combination of these two
- Strategy
  - bottom-up
  - top-down
  - Inside-out
  - mixed

ER model, An Entity Type

- A group of objects with the same properties
- In spoken language: an Entity, a real life object (”A Customer”, ”An Invoice”)
- ”A noun”

Strong and weak entity

- strong entity: is not existence-dependent on some other entity (“Customer”, “Invoice”)
- weak entity: is existence-dependent on another entity (“Invoiceline”)
  - owner entity is identifying
  - identifying relationship

ER model, An Attibute

- Attribute is a property of an entity or a relationship (”The Firstname”, ”The OrderDate”)

ER model, An Attibute

- Atomic (”Firstname”)
- Composite attribute (”Name” = ”Firstname”+”Lastname”)
- Multivalued/set-valued (”Phonenumber”)
- Derived attribute (”Age”, ”Duration”)
- A Key (”StudentID”)
- Mandatory/non-mandatory (NULLs)

ER model, A Relationship Type

- In spoken language: a relationship
- A Relationship between 2 or more entities
- Entities *participate* the relationship
- An Entity can participate a relationship in different Roles (Recursive Relationship)
- *Degree* of a relationship is the number of participating entities in a relationship.
- A relationship can have its own Attributes
- ”A verb”

ER model, A Relationship

- Constraints
  - **Cardinality** constraint
    - 1:1
    - 1:m
    - m:1
    - m:n
  - **Participation** constraint
    - Total/partial (mandatory/optional)

ER model, A Key

- A **Candidate Key:** The minimal set of attributes that uniquely identifies each occurrence of an entity.
- Composite key is a candidate key that consists of two or more attributes.
- The **Primary Key:** one selected Candidate Key to identify each occurrence of an entity. Other possible candidate keys are called **alternate keys.**
- Which one to choose? (will it stay unique, not changing, data type, lenght,...)

ER model, A Key

- The Primary Key
  - Natural Key
  - Surrogate Key
  - A combination of these two…
- Not a required feature for an ER model, but relational model needs a Primary Key.

ER model, A Key

- Natural Key usually more efficient because joins are sometimes not needed at all
- A surrogate key is not often used in retrieving data (because it has no meaning to the business), but PK always has an index -> wasted space
- Surrogate does not stop inserting logical duplicates (UK needed)

Designing

Miracle Finland Oy

Avoid Redundance

- A wrong place for an attribute will cause redundancy
  - For example Country in the address entity
- Too many relationships may cause redundancy

Normalization

- Normalization is a method that **removes data redundancy** from a relation (minimizing the insertion, deletion and update anomalies that degrade the performance of databases)
- Normalization uses a series of tests (normal forms) to help identify the **optimal grouping for these attributes** to identify a set of suitable relations that supports the data requirements.
- examining the **functional dependencies** between attribute

Functional Dependency

- a relationship between two attributes in a relation (determinant, dependent)
  - ISBN -> BookTitle (ISBN determines the book title)
  - or
  - Employee_id → name job salary
  - or
  - street city→ postcode and
  - postcode →city

Armstrong’s axioms

- **Axiom of reflexivity**
  - If $Y \subseteq X$, then $X \longrightarrow Y$
  - (If Y is a subset of X, then X determines Y.)
- **Axiom of augmentation**
  - If $X \longrightarrow Y$, then $XZ \longrightarrow YZ$
  - (If X determines Y, then XZ determines YZ.)
- **Axiom of transitivity**
  - If $X \longrightarrow Y$ ja $Y \longrightarrow Z$, then $X \longrightarrow Z$
  - (If X determines Y and Y determines Z, then X must also determine Z.)

Why Functional Dependency?

- To define keys
  - *X = attributes never on the right hand side*
  - *Y = attributes both on the right hand and the left hand side*
  - *Z = attributes on the right hand never on the left hand side*
  - *X must be in every key, Y might be in a key, Z cannot be in any key*
- To normalize

- address(street, postcode, city)

- *address1(street, postcode)* and
- *address2(postcode, city)*

- inclusion dependency
  - *address1*[*postcode*] ⊆ *address2*[*postcode*]

Normalization and the “Forms”

- First Normal Form (1NF)
  - none of its domains have any sets as elements, attributes contains only single values from the domain (each column contains atomic values, and there are not repeating groups of columns)
- Second Normal Form (2NF)
  - 1NF+all non-key attributes are fully functional dependent on the primary key (the columns are dependent on the primary key)

Normalization and the “Forms”

- Third Normal Form (3NF)
  - 2NF+there is **no** transitive functional dependency (the columns are dependent on the primary key (2NF) and no other columns in the table)
    - *Student_id*, Student_name, Street, City, State, Zip
      - (Student_id as primary key)
    - But now Street, City, State are dependent on Zip not Student_id -> transitive functional dependency -> not in 3NF

Normalization and the “Forms”

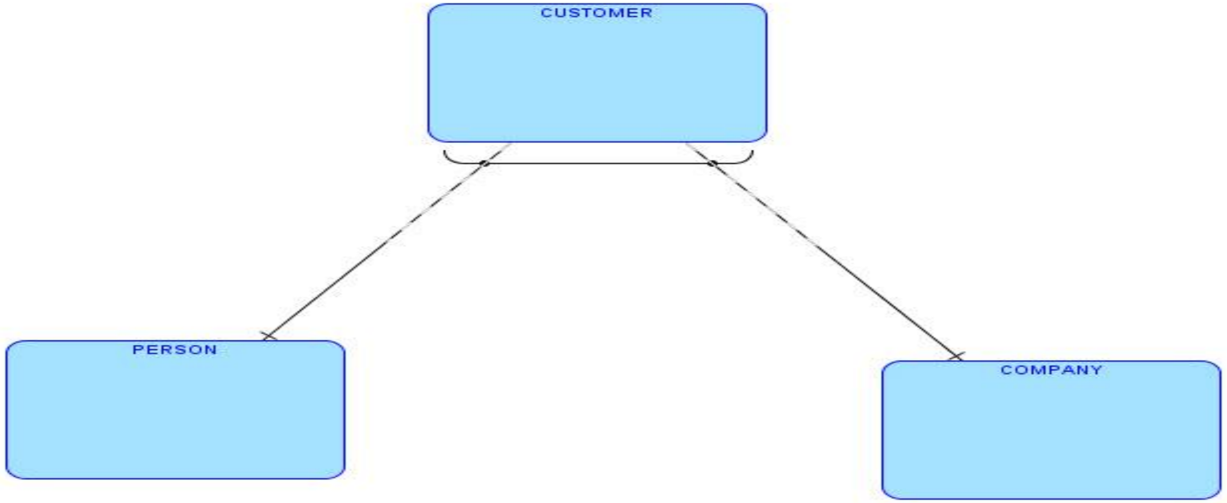
- Boyce–Codd Normal Form (BCNF)
  - Must be in 3NF and for each functional dependency (X -> Y), X should be a super Key.
  - if and only if there are no non-trivial functional dependencies of attributes on anything other than a superset of a candidate key. (Each attribute must represent a fact about the key, the whole key, and nothing but the key)
- Fourth Normal Form (4NF), Fifth Normal Form (5NF).

An entity, a relationship or an attribute?

- There are no foreign keys (and FK columns) in ER!
  - They are **relationships**
- An entity or an attribute?
  - Example, phonenumber (how many? Other attributes?)
- An entity or a relationship?
  - If it is a relationship in real world it is most likely a relationship in the data model too.

Specialization

- Specialization: A customer -> Person or Company



Specialization

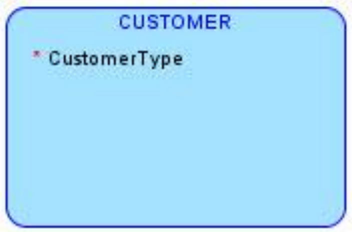
- Can Person change to a Company or vice versa?
- Do you always know which one the customer is?
- …

Generalization

- Generalization: A Vehicle (<- a car, a bus, a lorry, …)
  - A bus might have some attributes a car does not have etc. NULLs.
  - The Customer example:

Generalization

- Usually a discriminator to identify the ”type”



- Now only the attributes all the ”types” have can be defined mandatory and there will be penty on NULLs

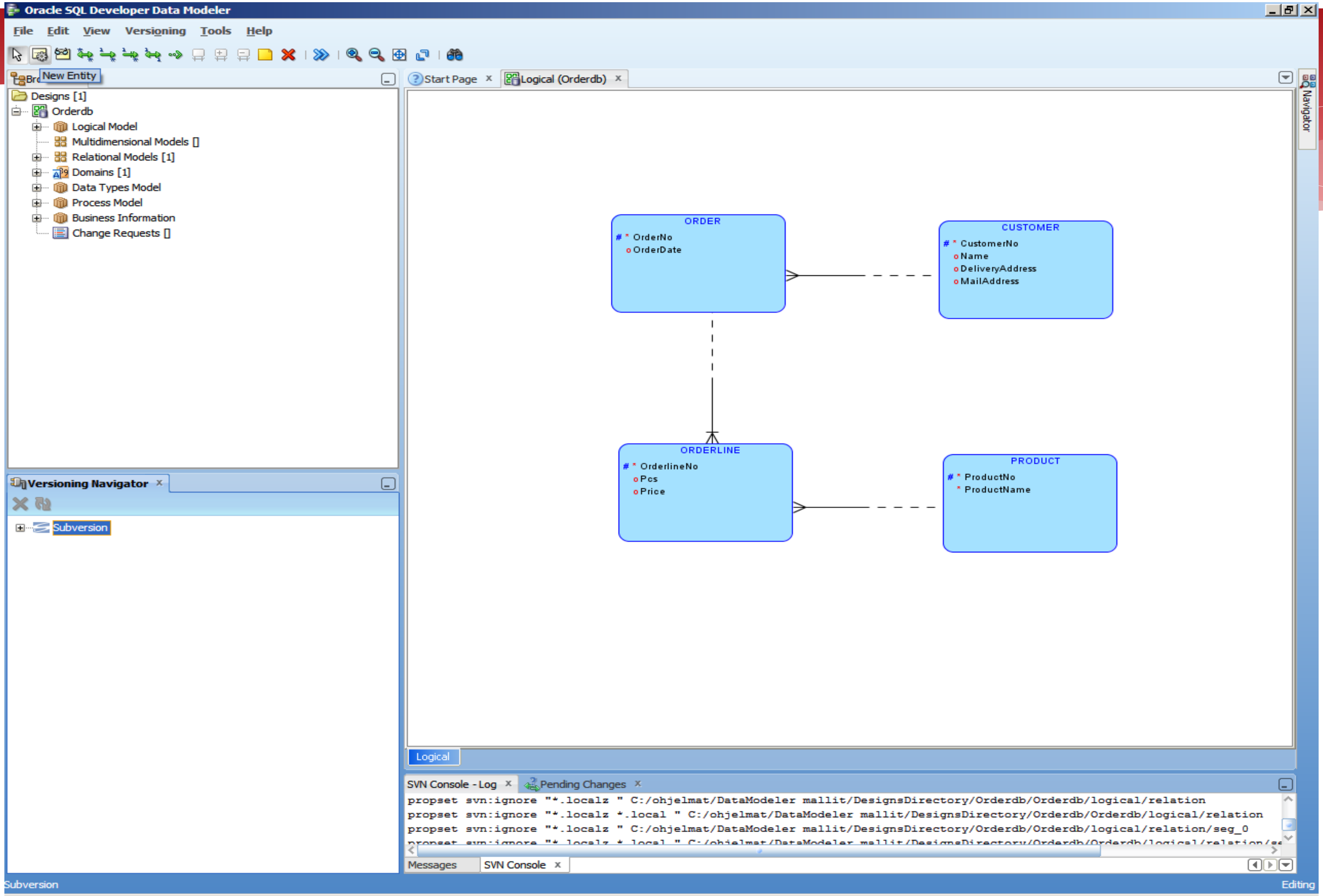
Specialization, generalization

- **Participation** constraint
  - Defines whether every member in the superclass must also be a member of a subclass or not
  - mandatory or optional
- **Disjoint** constraints
  - The disjoint constraint only applies when a superclass has more than one subclass. If the subclasses are disjoint, then an entity occurrence can be a member of only one of the subclasses.

Data Modeler

- Let’s design a database
- https://github.com/Mavihe1/DBworkshop

Testing the conceptual model



```
propset svn:ignore "*.localz " C:/ohjelmat/DataModeler mallit/DesignsDirectory/Orderdb/Orderdb/logical/relation
propset svn:ignore "*.localz *.local " C:/ohjelmat/DataModeler mallit/DesignsDirectory/Orderdb/Orderdb/logical/relation
propset svn:ignore "*.localz " C:/ohjelmat/DataModeler mallit/DesignsDirectory/Orderdb/Orderdb/logical/relation/seg_0
propset svn:ignore "*.localz *.local " C:/ohjelmat/DataModeler mallit/DesignsDirectory/Orderdb/Orderdb/logical/relation/se
```

Designing the database

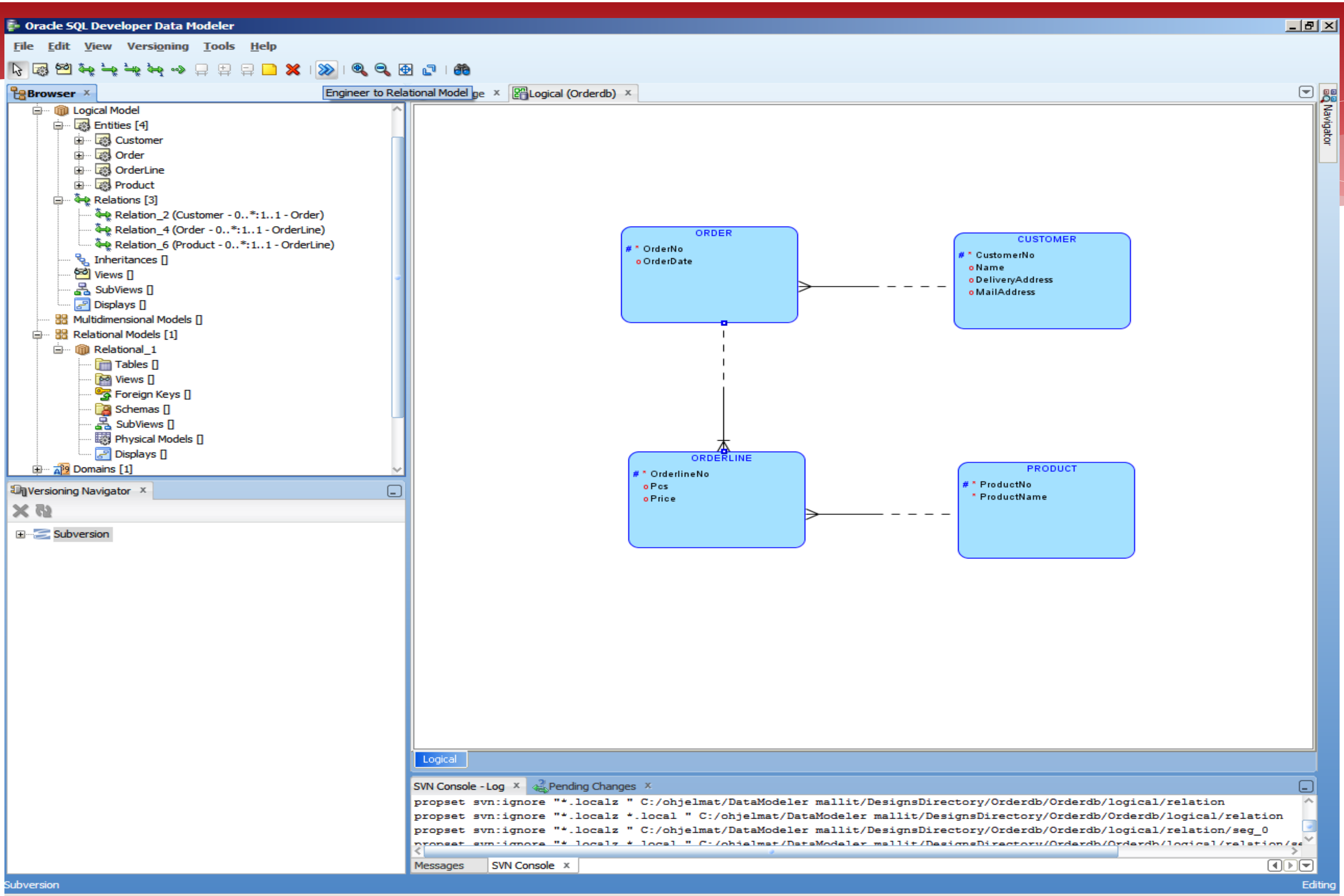
3. **Logical design:** transforming the conceptual model into a logical data model and a logical schema that the RDMS understands

Result: relational-database schema

(relational schemas and constraints)

To logical (relational) model

- The conceptual data model is refined and mapped to a logical data model.
- The logical data model defines the DBMS used
  - In our case relational data model for RDBMS
- But we do not define the ”brand” yet (Oracle, MySQL, MS SQL Server,…)
- The easiest is if you have a tool that automatically creates tables and foreign keys based on the ER

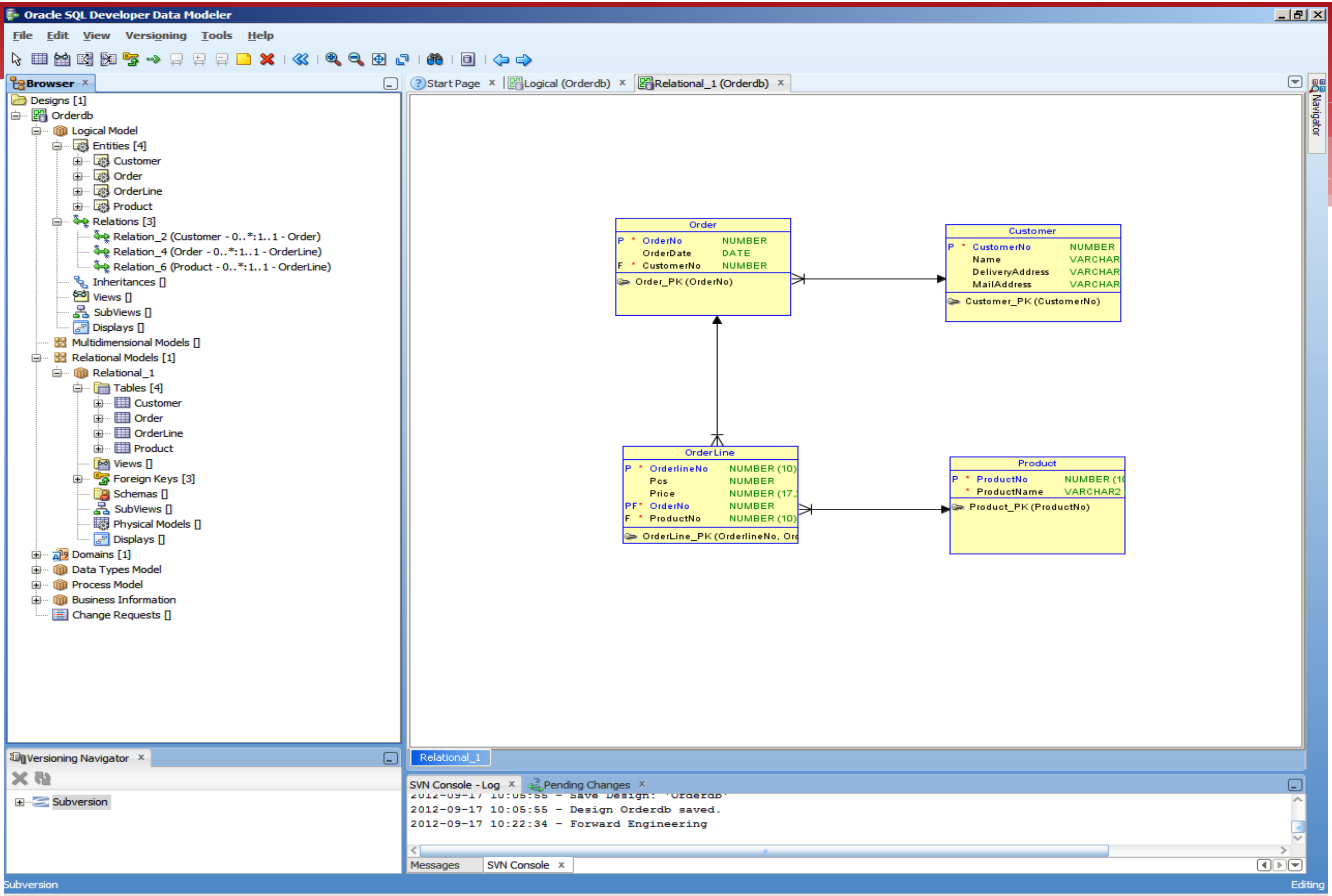


- Logical Model
  - Entities [4]
    - Customer
    - Order
    - OrderLine
    - Product
  - Relations [3]
    - Relation_2 (Customer - 0..*:1..1 - Order)
    - Relation_4 (Order - 0..*:1..1 - OrderLine)
    - Relation_6 (Product - 0..*:1..1 - OrderLine)
  - Inheritances []
  - Views []
  - SubViews []
  - Displays []
- Multidimensional Models []
- Relational Models [1]
  - Relational_1
    - Tables []
    - Views []
    - Foreign Keys []
    - Schemas []
    - SubViews []
    - Physical Models []
    - Displays []
- Domains [1]

```
propset svn:ignore "*.localz " C:/ohjelmat/DataModeler mallit/DesignsDirectory/Orderdb/Orderdb/logical/relation
propset svn:ignore "*.localz *.local " C:/ohjelmat/DataModeler mallit/DesignsDirectory/Orderdb/Orderdb/logical/relation
propset svn:ignore "*.localz " C:/ohjelmat/DataModeler mallit/DesignsDirectory/Orderdb/Orderdb/logical/relation/seg_0
propset svn:ignore "*.localz *.local " C:/ohjelmat/DataModeler mallit/DesignsDirectory/Orderdb/Orderdb/logical/relation/se
```

MIRACLE
Miracle Finland Oy

Relational model



Order		
P *	OrderNo	NUMBER
	OrderDate	DATE
F *	CustomerNo	NUMBER
	Order_PK (OrderNo)	

Customer		
P *	CustomerNo	NUMBER
	Name	VARCHAR
	DeliveryAddress	VARCHAR
	MailAddress	VARCHAR
	Customer_PK (CustomerNo)	

OrderLine		
P *	OrderlineNo	NUMBER (10)
	Pcs	NUMBER
	Price	NUMBER (17,
PF*	OrderNo	NUMBER
F *	ProductNo	NUMBER (10)
	OrderLine_PK (OrderlineNo, Orc	

Product		
P *	ProductNo	NUMBER (1
*	ProductName	VARCHAR2
	Product_PK (ProductNo)	

```
2012-09-17 10:05:55 - Save Design: 'Orderdb'
2012-09-17 10:05:55 - Design Orderdb saved.
2012-09-17 10:22:34 - Forward Engineering
```

Designing the database

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

And all of these phases over and over again…

(this is different, we have always done that but not so many times and in such a short cycles)

From relational to physical

- how the database will be implemented
- A specific DBMS system

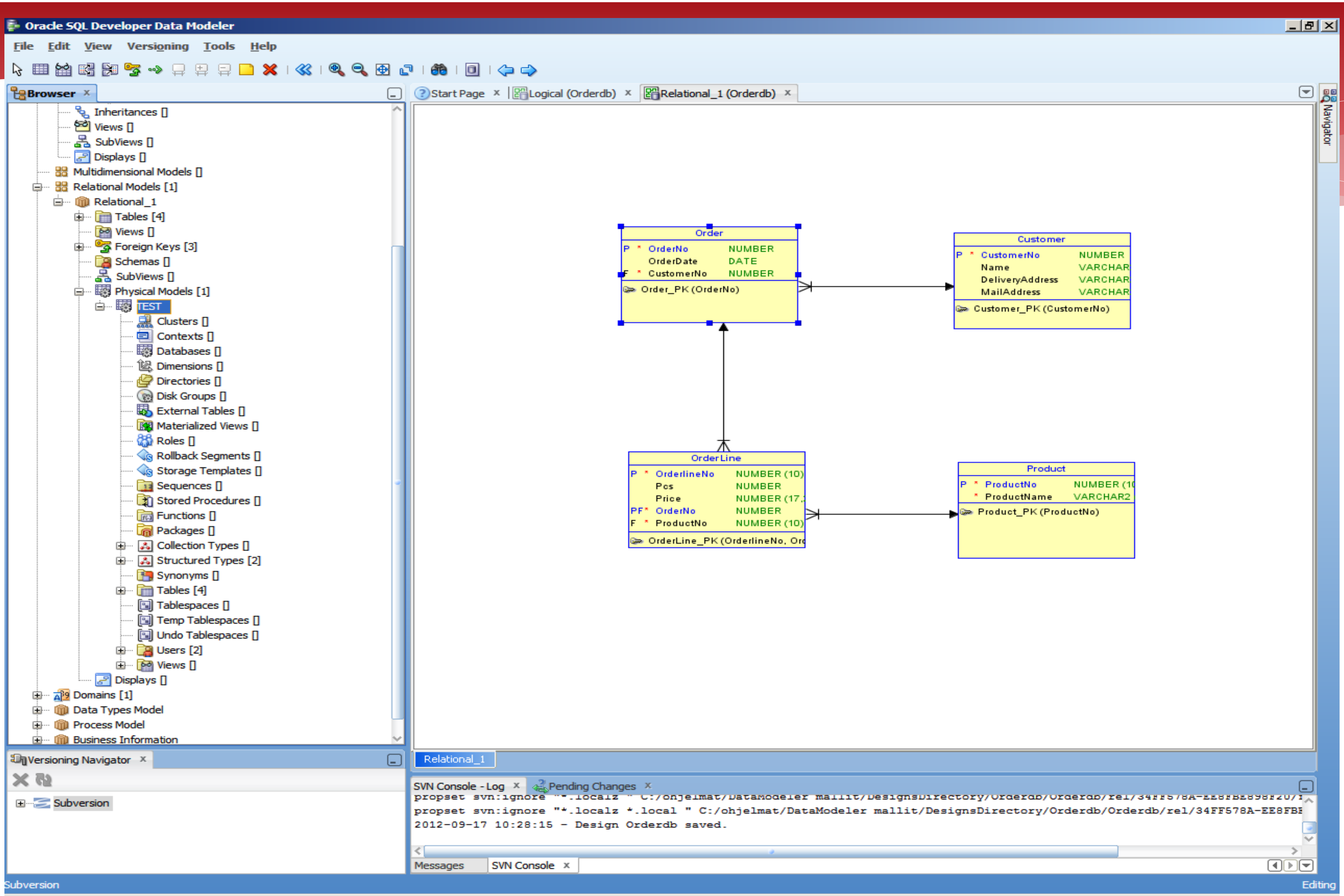
Physical design

- In short:
  - Start with the logical model
  - Design the disc usage, count the need for the space etc.
  - Indexes
  - Schemas
  - Tablespaces, storage definitions, user permissions,...

Physical design

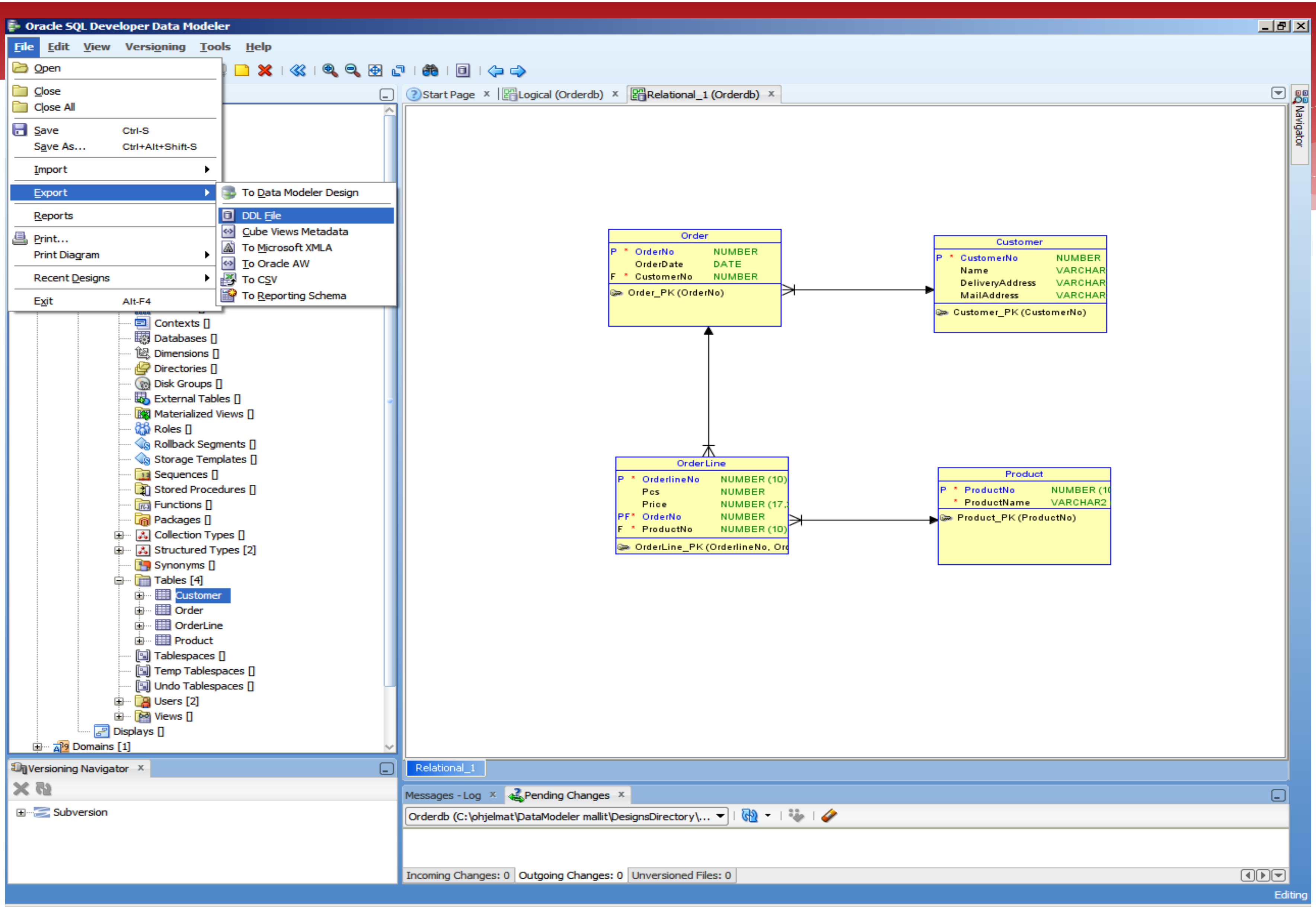
- How, who and where to document?
- How to document any changes (patch, changes in database objects, ...)?
- Backup and recovery (test that is works every now and then!)

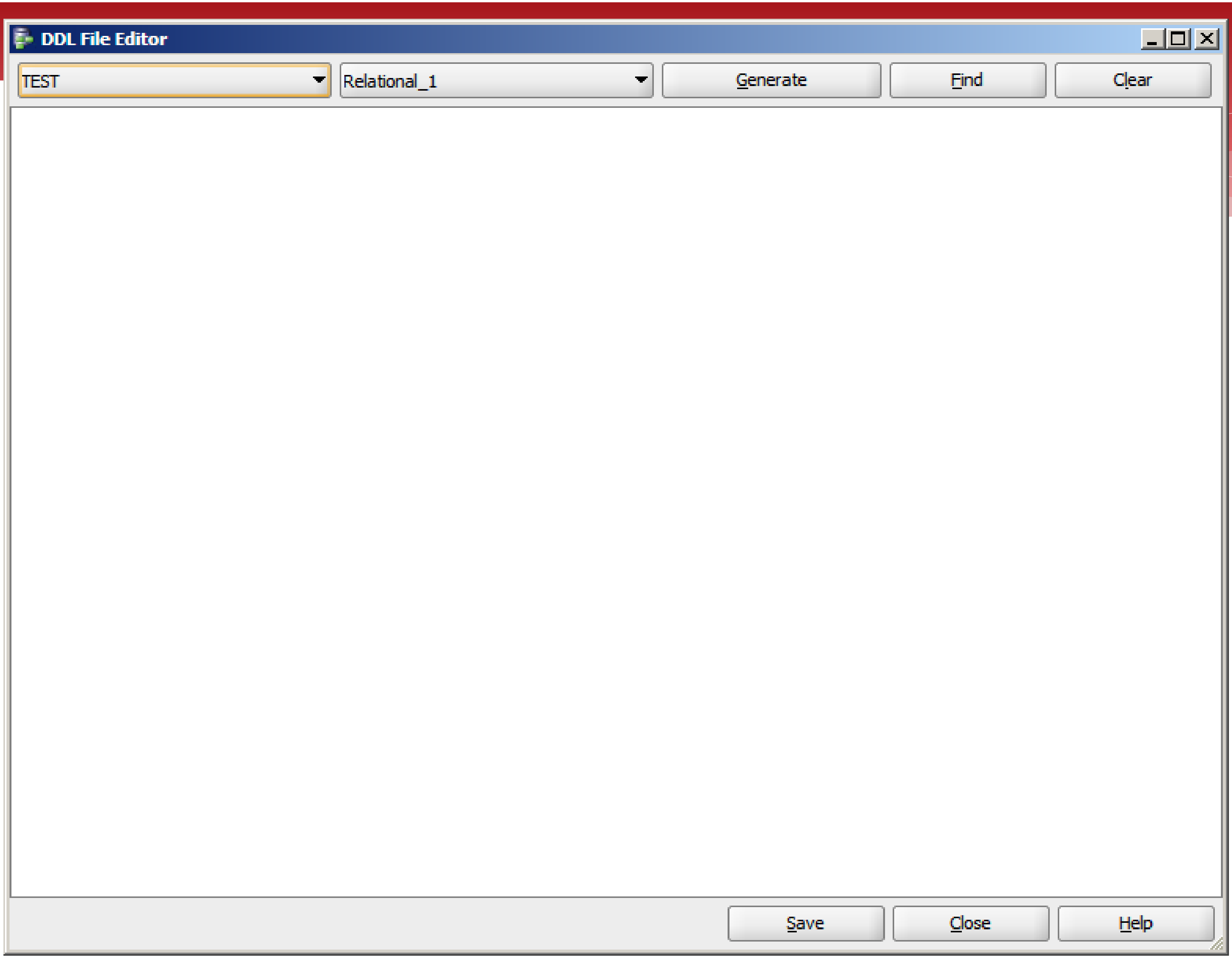
Physical model



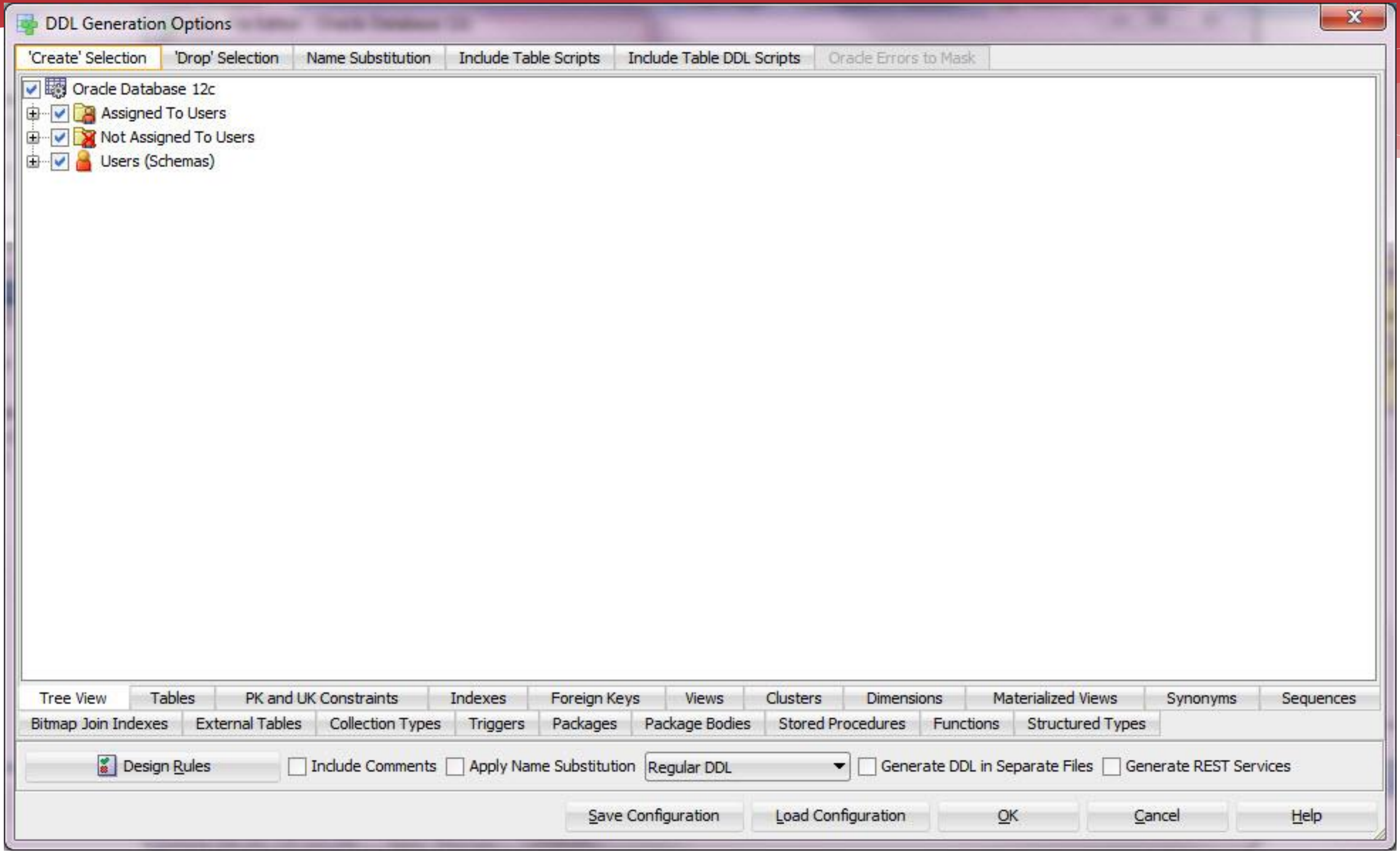
```
propset svn:ignore "*.localz " C:/ohjelmat/DataModeler mallit/DesignsDirectory/Orderdb/Orderdb/rel/34FF578A-EE8FBE898F20/1
propset svn:ignore "*.localz *.local " C:/ohjelmat/DataModeler mallit/DesignsDirectory/Orderdb/Orderdb/rel/34FF578A-EE8FBE
2012-09-17 10:28:15 - Design Orderdb saved.
```

Implementation: Physical to DDLs


Oracle SQL Developer Data Modeler
File
Edit
View
Versioning
Tools
Help
Open
Close
Close All
Save Ctrl-S
Save As... Ctrl+Alt+Shift-S
Import
Export
Reports
Print...
Print Diagram
Recent Designs
Exit Alt-F4
To Data Modeler Design
DDL File
Cube Views Metadata
To Microsoft XMLA
To Oracle AW
To CSV
To Reporting Schema
Contexts []
Databases []
Dimensions []
Directories []
Disk Groups []
External Tables []
Materialized Views []
Roles []
Rollback Segments []
Storage Templates []
Sequences []
Stored Procedures []
Functions []
Packages []
Collection Types []
Structured Types [2]
Synonyms []
Tables [4]
Customer
Order
OrderLine
Product
Tablespaces []
Temp Tablespaces []
Undo Tablespaces []
Users [2]
Views []
Displays []
Domains [1]
Start Page
Logical (Orderdb)
Relational_1 (Orderdb)
Order
P * OrderNo NUMBER
OrderDate DATE
F * CustomerNo NUMBER
Order_PK (OrderNo)
Customer
P * CustomerNo NUMBER
Name VARCHAR
DeliveryAddress VARCHAR
MailAddress VARCHAR
Customer_PK (CustomerNo)
OrderLine
P * OrderlineNo NUMBER (10)
Pcs NUMBER
Price NUMBER (17,
PF* OrderNo NUMBER
F * ProductNo NUMBER (10)
OrderLine_PK (OrderlineNo, Or
Product
P * ProductNo NUMBER (1
* ProductName VARCHAR2
Product_PK (ProductNo)
Navigator
Relational_1
Versioning Navigator
Subversion
Messages - Log
Pending Changes
Orderdb (C:\ohjelmat\DataModeler mallit\DesignsDirectory\...
Incoming Changes: 0
Outgoing Changes: 0
Unversioned Files: 0
Editing

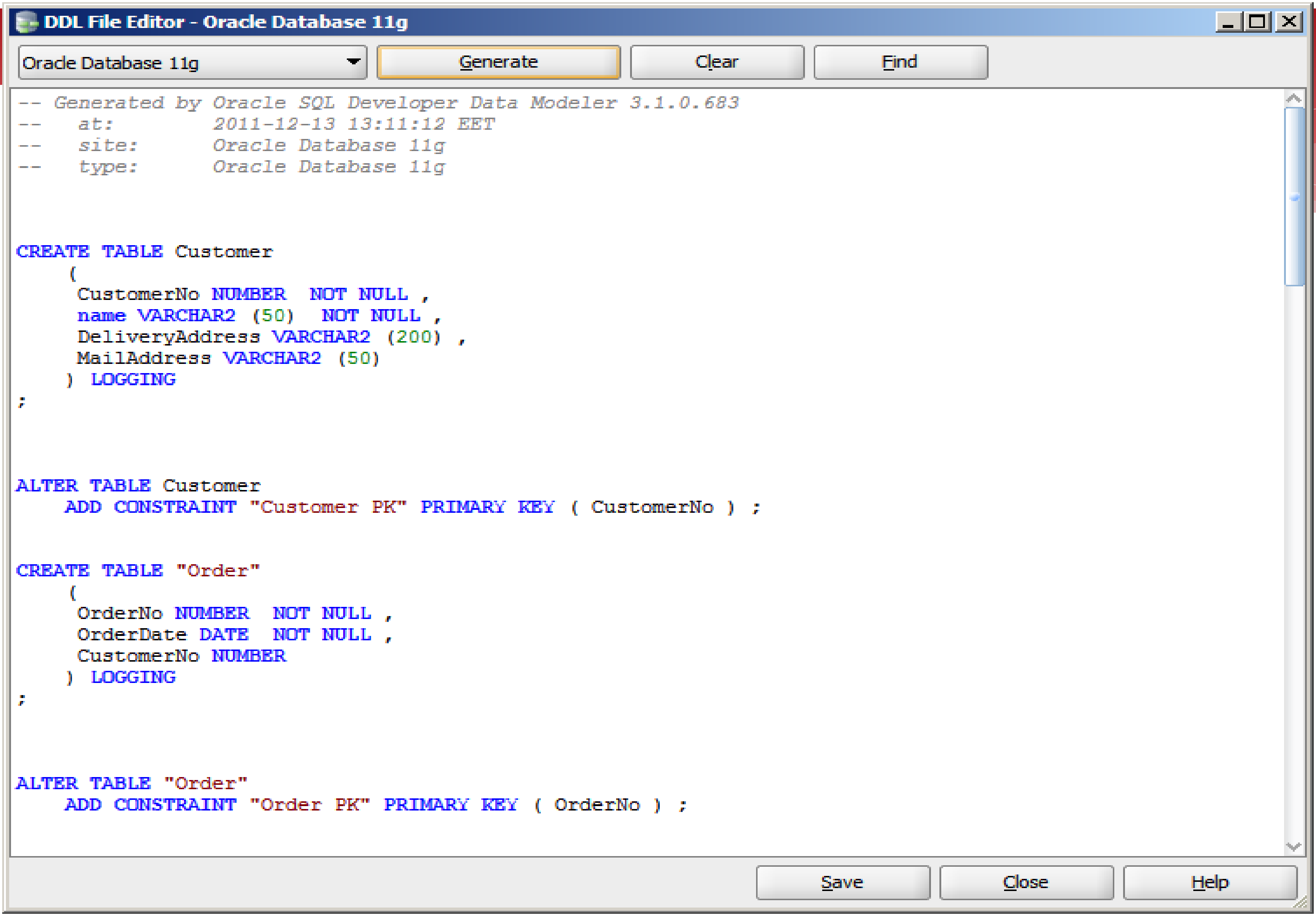


DDL File Editor
TEST
Relational_1
Generate
Find
Clear
Save
Close
Help



- ☑ Oracle Database 12c
  - ☑ Assigned To Users
  - ☑ Not Assigned To Users
  - ☑ Users (Schemas)

□ Include Comments □ Apply Name Substitution Regular DDL □ Generate DDL in Separate Files □ Generate REST Services



```
-- Generated by Oracle SQL Developer Data Modeler 3.1.0.683
--   at:        2011-12-13 13:11:12 EET
--   site:      Oracle Database 11g
--   type:      Oracle Database 11g



CREATE TABLE Customer
    (
     CustomerNo NUMBER  NOT NULL ,
     name VARCHAR2 (50)  NOT NULL ,
     DeliveryAddress VARCHAR2 (200) ,
     MailAddress VARCHAR2 (50)
    ) LOGGING
;



ALTER TABLE Customer
    ADD CONSTRAINT "Customer PK" PRIMARY KEY ( CustomerNo ) ;


CREATE TABLE "Order"
    (
     OrderNo NUMBER  NOT NULL ,
     OrderDate DATE  NOT NULL ,
     CustomerNo NUMBER
    ) LOGGING
;



ALTER TABLE "Order"
    ADD CONSTRAINT "Order PK" PRIMARY KEY ( OrderNo ) ;
```

MIRACLE Miracle Finland Oy

That was the process… but

Miracle Finland Oy

What kind of DB are you designing?

- Different rules for OLTP and DW
  - My ”own” data
  - Somebody else’s data

Maintenance

- Monitoring the performance of the system: tuning if needed.
- Maintaining and upgrading the database system.
- Maintaining the user privileges.
- Testing backups and recoveries.
- New requirements implemented using the design process.

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

Indexing

- Select **might** be faster
- Insert, Update, Delete slower
- Not a silver bullet

Tuning the queries

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

Tuning, the process

- 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

A tool

Oracle SQL Developer Data Modeler

- To be efficient in designing you need a tool: my recommendation is Data Modeler
  - Free of charge
  - Support for many different databases
  - Support for both documenting the existing databases and designing a new one (and mainintaining that)
  - Support for reporting, naming standards, glossaries, design rules, ...
  - Support for version control and multiuser environment
  - Support for everything you need for database design plus more

MiRACLE
Miracle Finland Oy

Conclusions

- Database designing is an important work
- The database must be designed and for the right purpose .
- Data integrity and data quality must be the guiding rules.
- 3NF or Boyce-Codd preferred

Conclusions

- You need a tool!
- Data Modeler is a good tool; good support for iterative processes
- Enables documenting and versioning (and comparing the versions)
- Enables multiuser environment
- Is free to use
- Support for other databases as well

THANK YOU!

QUESTIONS?

heli@miracleoy.fi

Twitter: @HeliFromFinland

Blog: Helifromfinland.com

MIRACLE
Miracle Finland Oy