Database systems I

Radim Bača

radim.baca@vsb.cz

dbedu.cs.vsb.cz

VSB TECHNICAL UNIVERSITY OF OSTRAVA
FACULTY OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE
DEPARTMENT OF COMPUTER SCIENCE

EVROPSKÁ UNIE
Evropské strukturální a investiční fondy
Operační program Výzkum, vývoj a vzdělávání

MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY

Content

- Introduction
- Conceptual model
    - Tools for conceptual modeling
    - E-R model
    - UML model
- Typical conceptual modeling situations
    - Codebooks
    - Historical data
    - Tree and graph data
    - Fact table

Information sysems

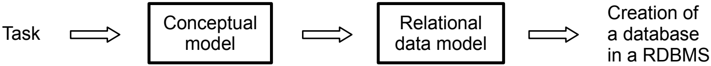
- An information system (IS) is a special type of a software work
- When developing an IS, we use the recommended software development techniques
- However, we focus only on a part of an IS development that concerns **databases** in this subject

Information sysems

- An information system (IS) is a special type of a software work
- When developing an IS, we use the recommended software development techniques
- However, we focus only on a part of an IS development that concerns **databases** in this subject

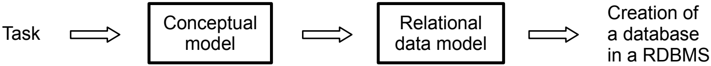
Data analysis

- Three steps for a database development:
    - **Task** - written specification of the task
    - **Conceptual modeling** (conceptual model) - a logical description of a database
    - **Database scheme design** (relational data model) - a description of a database defined for a conrete database system
    - **Physical design** - a concrete implementation of data files (CREATE TABLE . . . , CREATE INDEX . . .)



Data analysis

- Three steps for a database development:
  - **Task** - written specification of the task
  - **Conceptual modeling** (conceptual model) - a logical description of a database
  - **Database scheme design** (relational data model) - a description of a database defined for a conrete database system
  - **Physical design** - a concrete implementation of data files (CREATE TABLE . . . , CREATE INDEX . . .)



Conceptual Modeling

- is a process of a development of a system description that is used to design and implement a database application
- is independent of database
- defines restrictions put on data

Linear Notation & Basic Concepts

- Linear notation:

  `Student(stID, name, birth_year)`

  `Subject(suID, name, study_year)`

Linear Notation & Basic Concepts

- Linear notation:

  `Student(stID, name, birth_year)`

  `Subject(suID, name, study_year)`

- **Entity type**

Linear Notation & Basic Concepts

- Linear notation:

  `Student(stID, name, birth_year)`

  `Subject(suID, name, study_year)`

- **Entity type**
- **Attribute**

Linear Notation & Basic Concepts

- Linear notation:

  `Student(stID, name, birth_year)`

  `Subject(suID, name, study_year)`

- **Entity type**
- **Attribute**
- **Key**

Linear Notation & Basic Concepts

- Linear notation:

  `Student(stID, name, birth_year)`

  `Subject(suID, name, study_year)`

- **Entity type**
- **Attribute**
- **Key**

- **Entity** – object of a reality (one instance of entity type)

Relationship

- **Relationship** - describes a relationship among entity types
  - Linear notation:
    `RELATIONSHIP(`$\texttt{EntityType}_1$`, … ,` $\texttt{EntityType}_n$`)`
  - Two or more entity types can be in a relationship
- **Relationship with attributes** - a relationship containing also attributes specifying properties of the relationship
- *Example of a relationship (the relationship Studies)*

  `STUDIES(Student, Subject)`
- *Example of a relationship with attributes (the relationship Studies)*

  `STUDIES(Student, Subject, gained_points)`

Relationship

- **Relationship** - describes a relationship among entity types
  - Linear notation: `RELATIONSHIP(`$\texttt{EntityType}_1$`, ... ,` $\texttt{EntityType}_n$`)`
  - Two or more entity types can be in a relationship
- **Relationship with attributes** - a relationship containing also attributes specifying properties of the relationship
- *Example of a relationship (the relationship Studies)*

  `STUDIES(Student, Subject)`
- *Example of a relationship with attributes (the relationship Studies)*

  `STUDIES(Student, Subject, gained_points)`

Relationship, Example 1

- *We would like to model a situation where one student coordinate another student. What is the proper way using linear notation?*

Relationship, Example 1

- *We would like to model a situation where one student coordinate another student. What is the proper way using linear notation?*

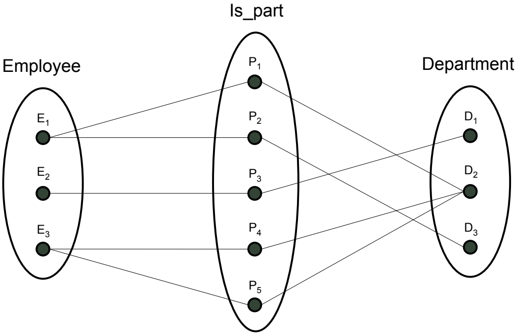
  `COORDINATE(Student, Student)`

Cardinality of a Relationship

- We distinguish relationships according to number of entities entering a relationship
- A relationship between two entity types (a so-called binary relationship) can be of the following types:
  - 1 : 1, 1 : N, M : N
- *Consider a company with the entity types Department and Employee.*
  - Cardinality 1 : 1 - an employee can be a chief only of one department and every deparment has at most one chief
  - Cardinality 1 : N - an employee can belong only to one department and every department can have many employees
  - Cardinality M : N - an employee can belong to several departments every department can have many employees

Cardinality

- In order to determine cardinality, it can be useful to draw the following diagram:



Mandatory/Obligatory Relationships

- Some entities has to have a relationship and some does not:
  - Mandatory relationship - each entity has to have a relationship
  - Obligatory relationship - there can be entities without relationship

- Linear notation:

```
RELATIONSHIP(EntityType1:(min,max),
             EntityType2:(min,max))
```

Relationships, Example 2

- *A teacher does not have to teach, but a subject has to have a teacher. A teacher is teaching many subjects and a subject can be taught by many teachers.*

Relationships, Example 2

- *A teacher does not have to teach, but a subject has to have a teacher. A teacher is teaching many subjects and a subject can be taught by many teachers.*

`TEACH(Teacher:(0,M), Subject:(1,N))`

Constraints

- **Constraints** provide additional information for conceptual model
- They are invariants of the database which have to be always satisfied
- Their typically concern:
    - an attribute value (e.g., the format of email)
    - relationship among entities (e.g., a department has to have its chief)

Description of Data model in Project

- Table containing a more detailed description of attributes
- Every table corresponds to a single entity type

User

	Data type	Lenght	Key	Null	Index	AC	Meaning
login	varchar	10	Y	N	Y		user's login
fname	varchar	20	N	N	N		user's first name
lname	varchar	20	N	N	Y		user's last name
phone	number	12	N	Y	N		phone number
type	varchar	10	N	N	N	1	user's cathegory
last_visit	Timestamp		N	Y	N		date of user's last login to the IS

1: data type must be one of these: admin, bidder, or user

E-R diagram (ERD)

- Graphic representation of conceptual model
- Unfortunately, there is no standard for it, therefore, one can come across many different notations of ERDs
- Let us mention just of them:
    - Chen’s notation
    - Crow’s foot notation
        - Oracle data modeler
        - Toad data modeler

Types of ERD

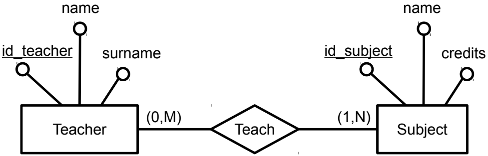


FIGURE: Chen's notation

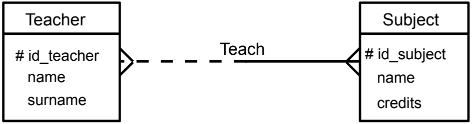


FIGURE: Crow's foot notation - Oracle

Types of ERD

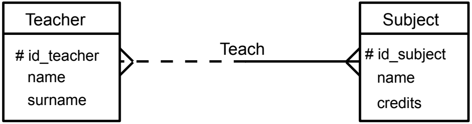


FIGURE: Crow's foot notation - Oracle

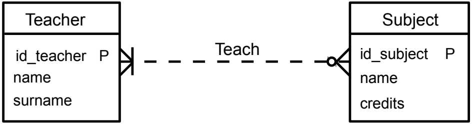


FIGURE: Crow's foot notation - Toad

Cardinality in ERD

- Cardinality is represented by a number or by crow’s foot
- See previous slide

Mandatory/Obligatory relationships

- It is solved by many different ways, but there are two basic categories:
  - Information is contained in pair (min, max), which determine the cardinality as well
  - We use some graphical symbol that determine mandatory and obligatory relationships

Mandatory/Obligatory relationships - (min, max)

- It is used in UML
- Pair `(min, max)` determine maximum and minimum number of entities that are in the relationship
- Having the `TEACH` rel. between `Subject` and `Teacher`
    - `Subject:(0,N)` - Teacher does not have to have a subject
    - `Subject:(1,N)` - Teacher has to have a subject
    - `Subject:(0,1)` - Teacher does not have to have a subject
    - `Subject:(1,1)` - Teacher has to have exactly one subject

M/O relationships - graphical symbol

- *Subject has to have at least one teacher and teacher does not have to teach anything*

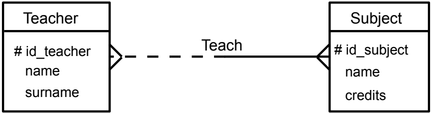


FIGURE: Crow's foot notation - Oracle

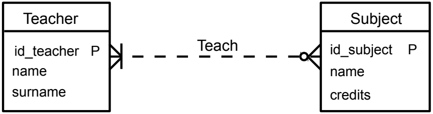


FIGURE: Crow's foot notation - Toad

M/O relationships - graphical symbol

- *Subject has to have at least one teacher and teacher does not have to teach anything*

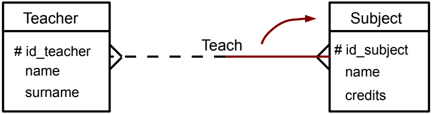


FIGURE: Crow's foot notation - Oracle

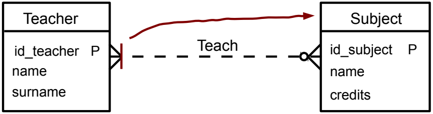


FIGURE: Crow's foot notation - Toad

M/O relationships - graphical symbol

- *subject has to have at least one teacher and teacher does not have to teach anything*

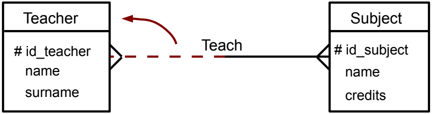


FIGURE: Crow's foot notation - Oracle

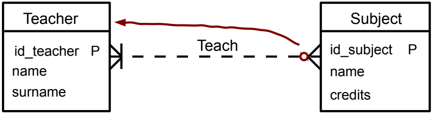


FIGURE: Crow's foot notation - Toad

Weak entity types

- Sometimes a key is formed by attributes belonging to another entity types
- The entity make sense only with respect to a different entity
- Then we speak about a so-called *weak entity type*

- *Diploma thesis is determined by both its title and its supervisor.*

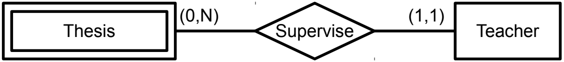


FIGURE: Chen's notation

Weak entity types - Oracle, TOAD

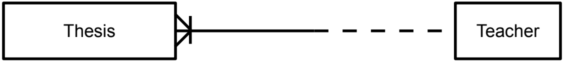


FIGURE: Crow's foot notation - Oracle

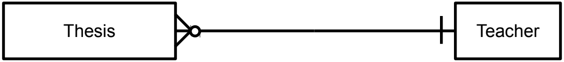


FIGURE: Crow's foot notation - Toad

Weak entity types - Oracle, TOAD

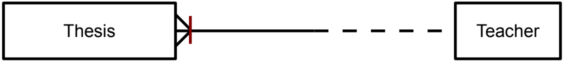
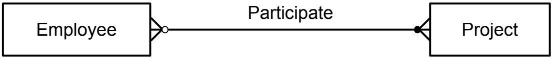


FIGURE: Crow's foot notation - Oracle

FIGURE: Crow's foot notation - Toad

Decomposition of a relationship



- In the relational scheme, we decompose the relationship M : N by using a table

UML

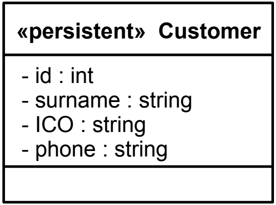
- UML is another tool wich enables us to design conceptual model of a system
- It represents an alternative to E-R diagrams (ERDs)
- It is a visual, object-oriented language modeling structural and dynamic aspects of a software work
- Unlike ERD, UML is a collection of modeling techniques that are applied to various aspects of software development
- Every UML technique provides different static or dynamic perspective of an application (a so-called model)

UML versus ERD

UML	*ERD*
class	entity type
object	entity
attribute	attribute
association	relationship

Class

- describes a structure and a behavior of an object representing an instance of the class
- Three parts: class name, attributes, and methods

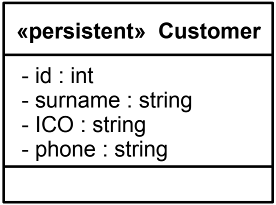


«persistent» Customer

- id : int
- surname : string
- ICO : string
- phone : string

Class

- describes a structure and a behavior of an object representing an instance of the class
- Three parts: class name, attributes, and methods

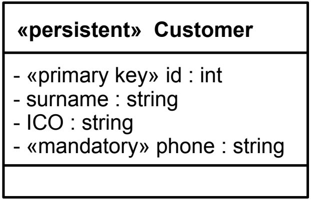


«persistent» Customer

- id : int
- surname : string
- ICO : string
- phone : string

Stereotypes

- extend the description (of attributes and class)
- are written between the symbols ≪ ≫
- The notation ≪ Persistent ≫ means that the class (attribute) will be mapped to the database, i.e., to the relational database scheme
- Furthermore, we can specify a primary key, mandatory attributes, etc.

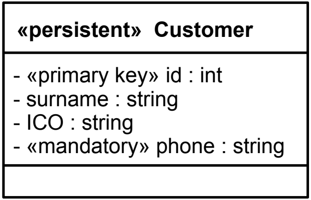


«persistent» Customer

- «primary key» id : int
- surname : string
- ICO : string
- «mandatory» phone : string

Stereotypes

- extend the description (of attributes and class)
- are written between the symbols ≪ ≫
- The notation ≪ Persistent ≫ means that the class (attribute) will be mapped to the database, i.e., to the relational database scheme
- Furthermore, we can specify a primary key, mandatory attributes, etc.

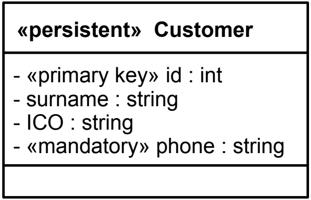


«persistent» Customer

- «primary key» id : int
- surname : string
- ICO : string
- «mandatory» phone : string

Stereotypes

- extend the description (of attributes and class)
- are written between the symbols $\ll$ $\gg$
- The notation $\ll$ Persistent $\gg$ means that the class (attribute) will be mapped to the database, i.e., to the relational database scheme
- Furthermore, we can specify a primary key, mandatory attributes, etc.

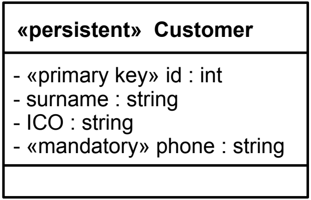


«persistent» Customer

- «primary key» id : int
- surname : string
- ICO : string
- «mandatory» phone : string

Stereotypes

- extend the description (of attributes and class)
- are written between the symbols $\ll$ $\gg$
- The notation $\ll$ Persistent $\gg$ means that the class (attribute) will be mapped to the database, i.e., to the relational database scheme
- Furthermore, we can specify a primary key, mandatory attributes, etc.



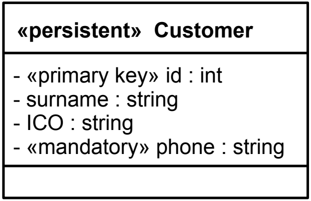
«persistent» Customer

- «primary key» id : int
- surname : string
- ICO : string
- «mandatory» phone : string

Syntax of attributes

`[visibility] [≪stereotype≫] attribute : [type]`

- `visibility` - gains the value
  - + for a public attribute
  - # for a protected attribute
  - – for a private attribute
- `≪stereotype≫` - adds more semantics to an attribute
- `attribute` - name of an attribute

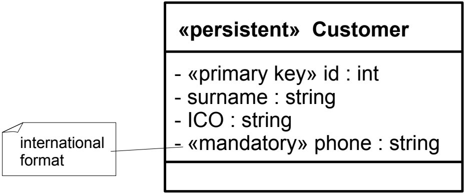


«persistent» Customer

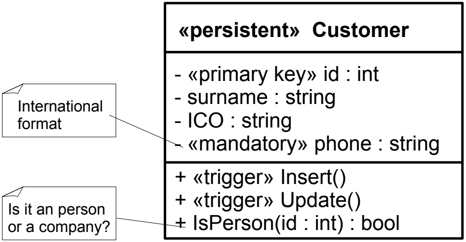
- «primary key» id : int
- surname : string
- ICO : string
- «mandatory» phone : string

Example: attributes and notes

Notes are used to specify an attribute in more detail, e.g., for further specification of format of the telephone number



Example: Operations [1/2]



- The stereotype `trigger` indicates that a method is a trigger (i.e., a code which runs automatically during a DML operation)
- The method `IsPerson` returns an information, whether the customer is a person or a company; we should also decide, whether the method will be a stored procedure or whether it will be a part of the application

Association

- is an equivalent to a relationship in ERD
- Again, the cardinality can be 1:1, 1:N, M:N
- When describing an association, we mention: name, role, cardinality, and whether the association is mandatory or not
- By an arrow we can determine the direction of an association

Conceptual modeling summary

- Linear notation of entity types and linear notation of relationships among entity types
- Graphic illustration of data model:
  - E-R diagram or UML diagram
  - transformed diagram for database scheme (see the next lecture)
- Data model
- List of constraints

Software for conceptual modeling

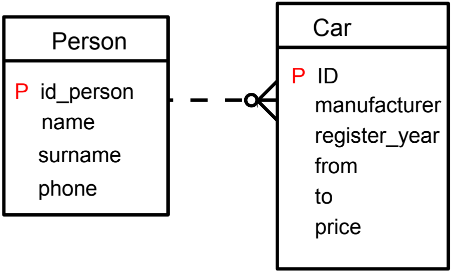
- Microsoft SQL Server 2008 Management Studio
- Oracle SQL Developer Data Modeler
- MySQL Workbench, (previously MySQL GUI Tools)
- Toad Data Modeler
- And a lot more:
  `http://www.databaseanswers.org/modelling_tools.htm`

Introduction

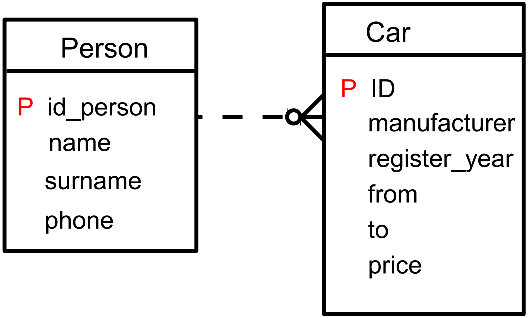
- We described the conceptual modeling tools
- However, knowledge about the modeling tools is not enough
- We need to
    - learn a typical modeling situations,
    - and we need to to create several database conceptual models by ourselves

Codebook

- Let us have a task: *We need to keep track of borrowing information for customers in a car rental company*
- What problems can you observe in a following solution?

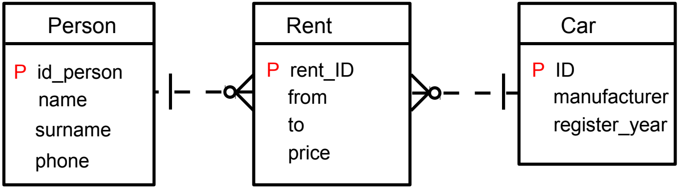


Codebook



- What exactly is one entity of the `Car` entity type?
- The entity is one car or it is a borrow?
- What about duplicity? Can we have a car in many entities?

Codebook



- This solution avoid the problems of the previous design

Codebook

- Entity type `Car` in the second model is very static and contains limited number of records
- We call such entity type as a *Codebook*
- Questions that we should answer for each entity type in a database design:
    - What is one record in the relation?
    - Is the name of entity type appropriate?
    - Is there a redundancy?

Codebook

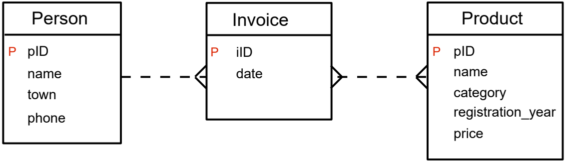
- Codebook is a entity type that describes some categories
- The most significant feature of a codebook is that we do not expect many DML operations on a result table
- Other examples of codebooks:
    - Car model in a car rent IS
    - Types of payment in a e-shop
    - List of towns or countries
    - List of athletic clubs
    - and so on

Codebook

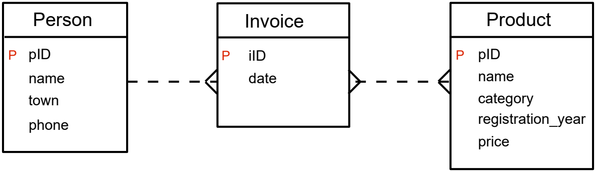
- Codebook is a entity type that describes some categories
- The most significant feature of a codebook is that we do not expect many DML operations on a result table
- Other examples of codebooks:
    - Car model in a car rent IS
    - Types of payment in a e-shop
    - List of towns or countries
    - List of athletic clubs
    - and so on

Historic Data

- Let us have a task: *We would like to store information about customers, products and purchases. Each purchase belongs to an invoice. The invoices must also be retrospectively accessible (we are interested in a history of purchases).*
- What problems can occur?



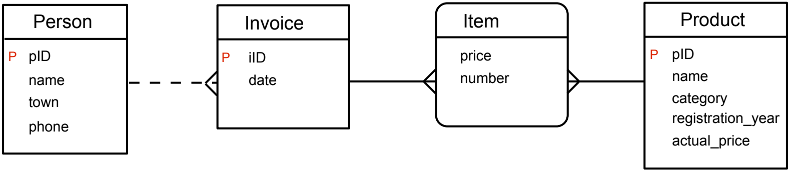
Historic Data



- There are basically two ways how to solve it:
  1. We never update values like price but we insert new records

This solution can be fine if we do not update the price very rarely

Historic Data



- There are basically two ways how to solve it:
  1. We never update values like price but we insert new records
  2. We can add price into M:N relationship

  Much better solution in a case of frequent price updates

Historic Data

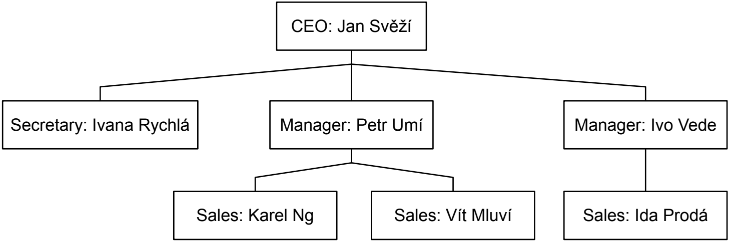
- What about the other attributes?
- We may ask whether any change in the person's phone (or address) should be reflected in all invoices of that person, or whether the old invoices should remain unchanged
- Obviously, the problem is related to whether it is necessary to preserve the history of the entities or whether the current status is sufficient

Historic Data - Other Examples

- We have a car rental company – are we interested about car renting history for each car, or is it enough to know who is currently renting a car?
- Let us have an building monitoring application – is it sufficient to have just an up-to-date list of people in the building, or do we need these lists retrospectively?
- Let us have an application for a pool lockers – is it sufficient to have just an up-to-date list of used lockers, or do we need these lists retrospectively?

Hierarchical Data

- Let us have a task: *We would like to store an information about a fact that an employee can have just one boss*
- In other words, we would like to store the following hierarchy


CEO: Jan Svěží

Secretary: Ivana Rychlá

Manager: Petr Umí

Manager: Ivo Vede

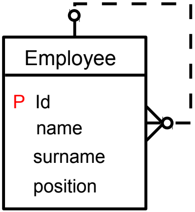
Sales: Karel Ng

Sales: Vít Mluví

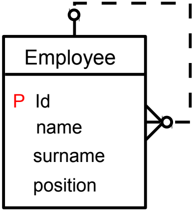
Sales: Ida Prodá

Hierarchical Data

- There is many different approaches to store and query tree data
- In the relational database we can use the following data model:

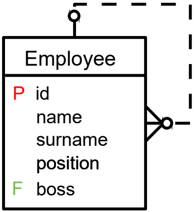


Hierarchical Data



- Why relationship is obligatory from both sides?
- This representation has its limitations:
    - It can be inefficient to work with large data
    - The number of self-joins during a tree traversal is equal to the depth of traversal

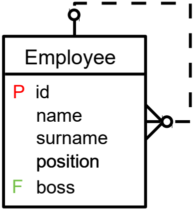
Hierarchical data - Descendants Query



- *Find all employees working under ‘Peter Pan’*
- We may start with the following query

```
SELECT e2.* FROM Employee e1
JOIN Employee e2 ON e1.id = e2.boss
WHERE e1.name = 'Peter'
  and e1.surname = 'Pan'
```

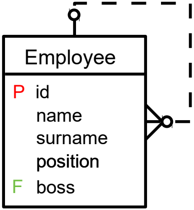
Hierarchical data - Descendants Query



- *Find all employees working under 'Peter Pan'*
- Then use UNION to add another level

```
SELECT e2.* FROM Employee e1
JOIN Employee e2 ON e1.id = e2.boss
WHERE e1.name = 'Peter'
  and e1.surname = 'Pan'
  UNION
SELECT e3.* FROM Employee e1
JOIN Employee e2 ON e1.id = e2.boss
JOIN Employee e3 ON e2.id = e3.boss
WHERE e1.name = 'Peter'
  and e1.surname = 'Pan'
```

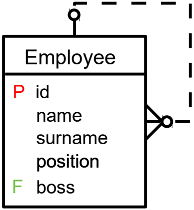
Hierarchical data - Descendants Query



- *Find all employees working under ‘Peter Pan’*
- and another one ...

```
SELECT e2.* FROM Employee e1
JOIN Employee e2 ON e1.id = e2.boss
WHERE e1.name = 'Peter'
  and e1.surname = 'Pan'
  UNION
SELECT e3.* FROM Employee e1
JOIN Employee e2 ON e1.id = e2.boss
JOIN Employee e3 ON e2.id = e3.boss
WHERE e1.name = 'Peter'
  and e1.surname = 'Pan'
  UNION
SELECT e4.* FROM Employee e1
JOIN Employee e2 ON e1.id = e2.boss
...
```

Hierarchical data - Descendants Query



- *Find all employees working under ‘Peter Pan’*
- Generally we need a recursive query

```
WITH rcte AS (
  SELECT e2.* FROM Employee e1
  JOIN Employee e2 ON e1.id = e2.boss
  WHERE e1.name = 'Peter'
    and e1.surname = 'Pan'
   UNION
  SELECT e2.* FROM rcte e1
  JOIN Employee e2 ON e1.id = e2.boss
)
SELECT * FROM rcte
```

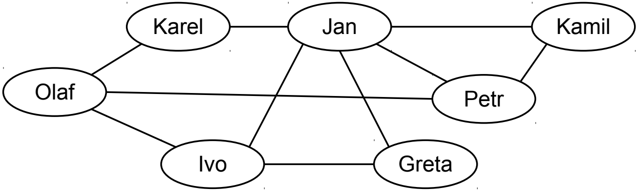
- Which may be quite difficult task (not only for a SQL developer)

Hierarchical data - Other Examples

- Data of a genealogical tree
- Product categories in an internet shop

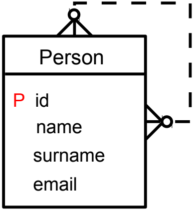
Graph Data

- Let us have a task: *We need to store informations about persons and a fact that two people knows each other*
- The goal is to store a data having a graph structure



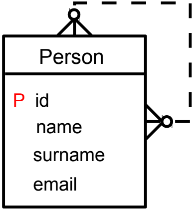
Graph Data

- The solution is very similar to the hierarchical data model
- We need a M:N recursive relationship
- The relationship is obligatory if we allow isolated nodes in a graph



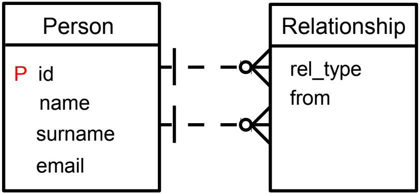
Graph Data with Labeled Edges

- We may want to also store information about the edges of the graph
- Our previous example uses graph with unlabeled graphs
- How to extend this model?



Graph Data with Labeled Edges

- The solution is to do a M:N relationship decomposition and add the necessary information about edges into the binding table



Graph Data

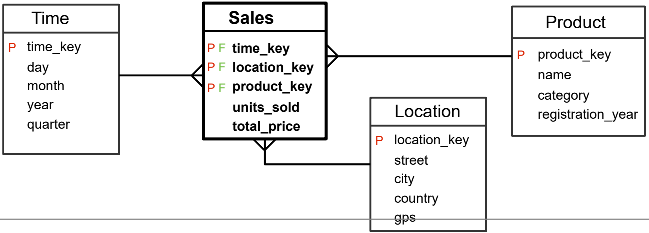
- The data model has its limitations (similarly to hierarchical data model)
- For example let us consider the following queries:
  - Shortest path between two vertexes
  - Centrality of an vertex
  - Find any path between vertexes where the edges has labels from set L
- Most of these queries are extremely difficult to express using SQL (we need recursive queries)
- Moreover their query processing may be slow

Graph Data - Other Examples

- Transportation network (MHD, Trains, ...)
- Mutual matches between sport clubs
- Function calls in a program

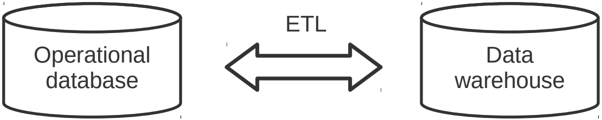
Star Schema

- This schema is often used in a data warehouses/business intelligence
- The main purpose of data warehouse is to provide a tools to analyze data
- The data are organized into two types of tables
    - Fact table - Central table that refers to the dimension tables
    - Dimension table - 'Codebook' tables surrounding the fact table



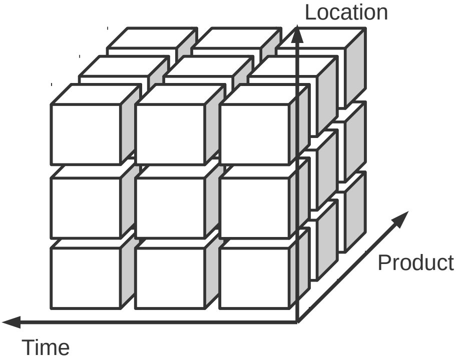
Star Schema

- The data in a data warehouse are usually transfered from a operational database and they are read-only
- The processes responsible for the transfer are called Extract Transform Load (ETL)



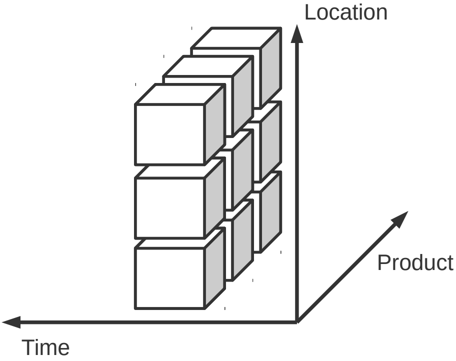
Star Schema

- The data in a data warehouse are usually modeled as cubes



Star Schema

- The data in a data warehouse are usually modeled as cubes
- The queries are subsequently modeled as a slices in the cube



Data Warehouse Database Systems

- Columns datastores - database systems designed especially for these types of data and queries
- The data are stored according to the columns in a data storage
- It makes aggregate queries very efficient

References

- R. Elmasri, S. Navathe. Fundamentals of Database Systems, Addison Wesley, ISBN 0-321-36957-2, 2010.
- UDBS web pages at `http://dbedu.cs.vsb.cz`