Database Systems I

Radim Bača

Department of Computer Science, FEECS

radim.baca@vsb.cz

dbedu.cs.vsb.cz

Outline

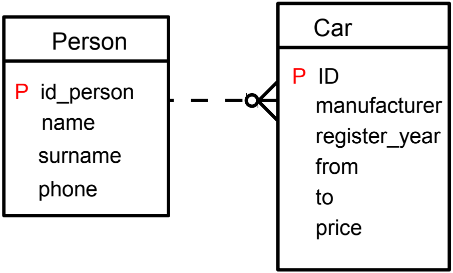
- Introduction
- Conceptual modeling situations
  - Codebooks
  - Historical data
  - Tree and graph data
  - Fact table

Úvod

- We described the conceptual modeling tools perspective in the previous lecture
- 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 ourself

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?



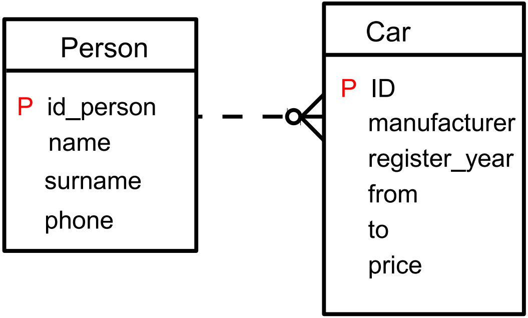
Person

P id_person
name
surname
phone

Car

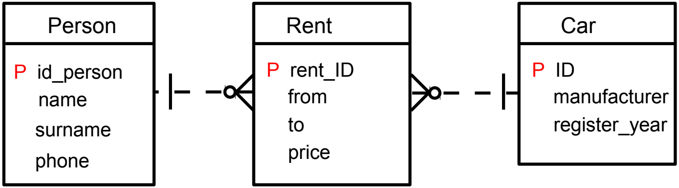
P ID
manufacturer
register_year
from
to
price

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

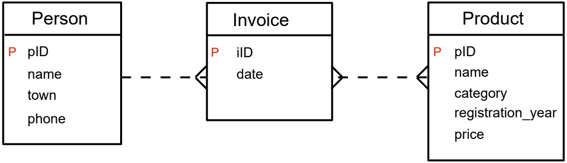
- Codebook is a entity type that describes some categories
- The most significant feature of a codebook is that we do not expect many DML operations on a result table
- Other examples of codebooks:
  - Car model in a car rent IS
  - Types of payment in a e-shop
  - List of towns or countries
  - List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

Codebook

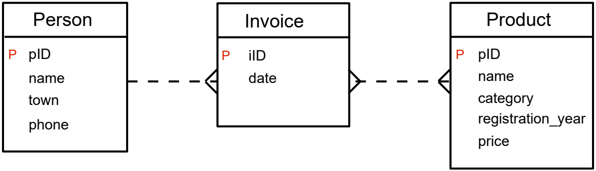
- Codebook is a entity type that describes some categories
- The most significant feature of a codebook is that we do not expect many DML operations on a result table
- Other examples of codebooks:
  - Car model in a car rent IS
  - Types of payment in a e-shop
  - List of towns or countries
  - List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



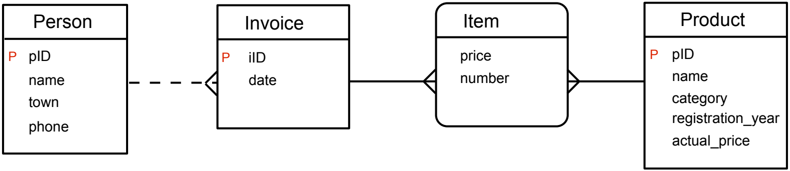
Historic Data



- 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

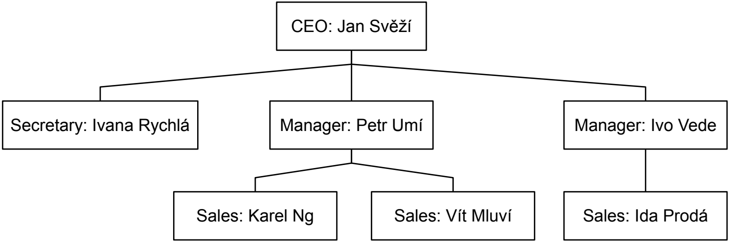
- What about the other attributes?
- We may ask whether any change in the person's phone (or address) should be reflected in all invoices of that person, or whether the old invoices should remain unchanged
- Obviously, the problem is related to whether it is necessary to preserve the history of the entities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

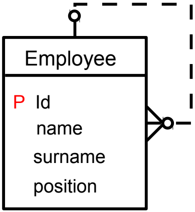
Hierarchical Data

- 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

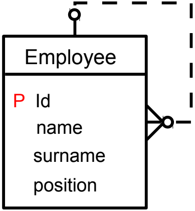


Hierarchical Data

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

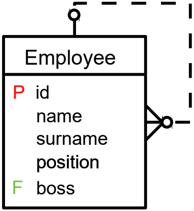


Hierarchical Data



- 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

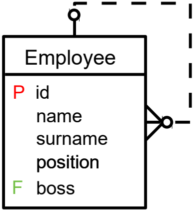
Hierarchical data - Descendants Query



- *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 e2.surname = 'Pan'
```

Hierarchical data - Descendants Query



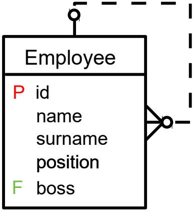
Employee

P id
name
surname
position
F boss

- *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 e2.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 e2.surname = ’Pan’
```

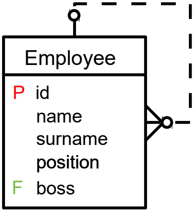
Hierarchical data - Descendants Query



- *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 e2.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 e2.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 e2.surname = 'Pan'
   UNION
  SELECT e2.* FROM rcte e1
  JOIN Employee e2 ON e1.id = e2.boss
)
SELECT * FROM rcte
```

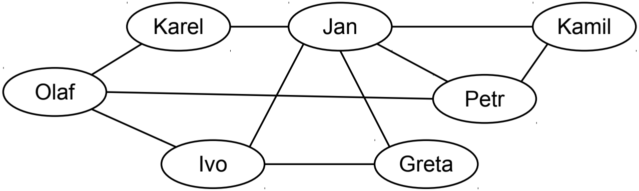
- Which may be quite difficult task (not only for a SQL developer)

Hierarchical data - Other Examples

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

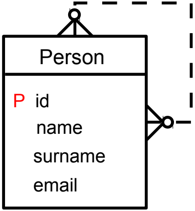
Graph Data

- 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



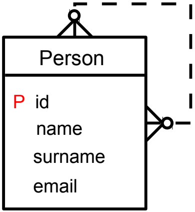
Graph Data

- 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



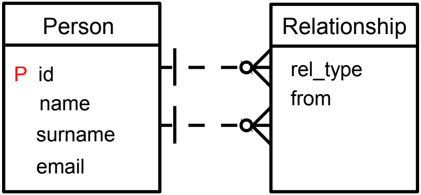
Graph Data with Labeled Edges

- 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

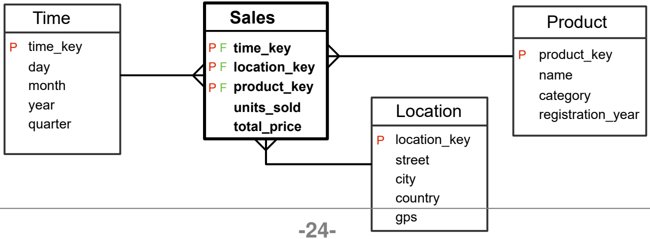
- The data model has its limitations (similarly to hierarchical data model)
- For example let us consider the following queries:
    - Shortest path between two vertexes
    - Centrality of an vertex
    - Find any path between vertexes where the edges has labels from set L
- Most of these queries are 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



Time
- P time_key
- day
- month
- year
- quarter

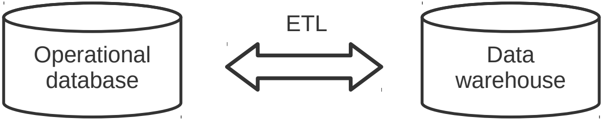
Sales
- P F **time_key**
- P F **location_key**
- P F **product_key**
- **units_sold**
- **total_price**

Product
- P product_key
- name
- category
- registration_year

Location
- P location_key
- street
- city
- country
- gps

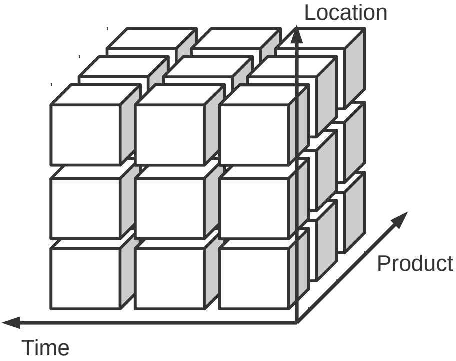
Star Schema

- 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)



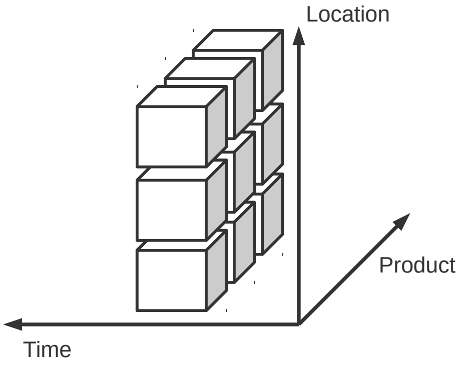
Star Schema

- 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

Reference

- http://dbedu.cs.vsb.cz