C# Application Development

Database Design (IIO30100 Tietokantojen suunnittelu)
Jyväskylä University of Applied Sciences, School of IT, 2008

Michal Zábovský

Department of Informatics
Faculty of Management Science and Informatics
University of Zilina Slovak Republic

Something to think about…

"Economist Paul Seabright reminds us of the pleas of the Soviet official trying to comprehend the Western system: 'Tell me… who is in charge of the supply of bread to the population of London?' The question is comical, but the answer – nobody – is dizzying."

Tim Harford, The Undercover Economist

Database design

Introduction [1]

- Information system
- Information management
- Information and data

Database design and data modeling

- Data modeling using CASE tools
- Entity-relationship models
- Toad Data Modeler

Practical issues

- Real examples
- Data models for different database systems

Information system

Definition [1]

- **Component of an organization that manages (gets, processes, stores, communicates) the information of interest**
  - each organization has an information system, possibly not made explicit in its structure
  - usually, the information system operates in support to other components of the organization

- The very notion of information system is partly independent of its computerization; however, we are mainly interested in information systems that are, to a large extent, computerized.

Information and data

Information and **data**

In most computer-based systems information is represented by means of **data.**

Why **data**?

This is the best than can be done to a large extent.

In most cases data are a valuable resource, with a very long lifecycle: banking applications have had data with the same structure for centuries, well before computers were invented!

Data and **information**

Data are raw facts, to be interpreted and correlated in order to provide **information.**

- *A502, 6* – only data (string and number)
- If they form answer for the question "*What is the code and number of credits for course Database systems?*" then we get information out of them.

Database

A collection of data, used to represent information of interest to an information system. (generic definition)

Technical definition – A collection of data managed by Database Management Systems (DBMS).

Database Management System

Software system able to manage **collections of data** that are

- **large** bigger, often much bigger, than the main memory available
- **shared** used by various applications and users
- **persistent** with a lifespan that is not limited to single executions of the programs that use them

and to ensure their reliability (so preserving the database in case of hardware or software failure) and privacy (controlling accesses and authorizations).

Like any software product, a DBMS must be **efficient** (using the appropriate amount of resources, such as time and space) and **effective** (supporting the productivity of its users).

DBMS vs. file systems

- The management of large and persistent sets of data can be done by means of simpler tools: file systems
- File systems provide also rough support for sharing
- There is no sharp line between DBMSs and non-DBMSs: DBMSs provide many features, that extend those of file systems
- The crucial issue is **effectiveness**, take advantage of these features

- In traditional programs that make use of files, each program includes a description of the organization of the file, which is often just a stream of bytes; there are chances of incoherence between the file and its description (or descriptions, if the file is shared)
- In DBMSs, there is a portion of the database (called the **dictionary** or **catalogue**) that describes the database itself, which is shared

Data model

- set of constructs used to organize data
- basic feature: **structuring mechanism** (or **type constructor**), as in programming languages; in Pascal we have array, record, set, file constructors
- in the **relational database model** we have the **relation** constructor, to organize data as sets of homogeneous records

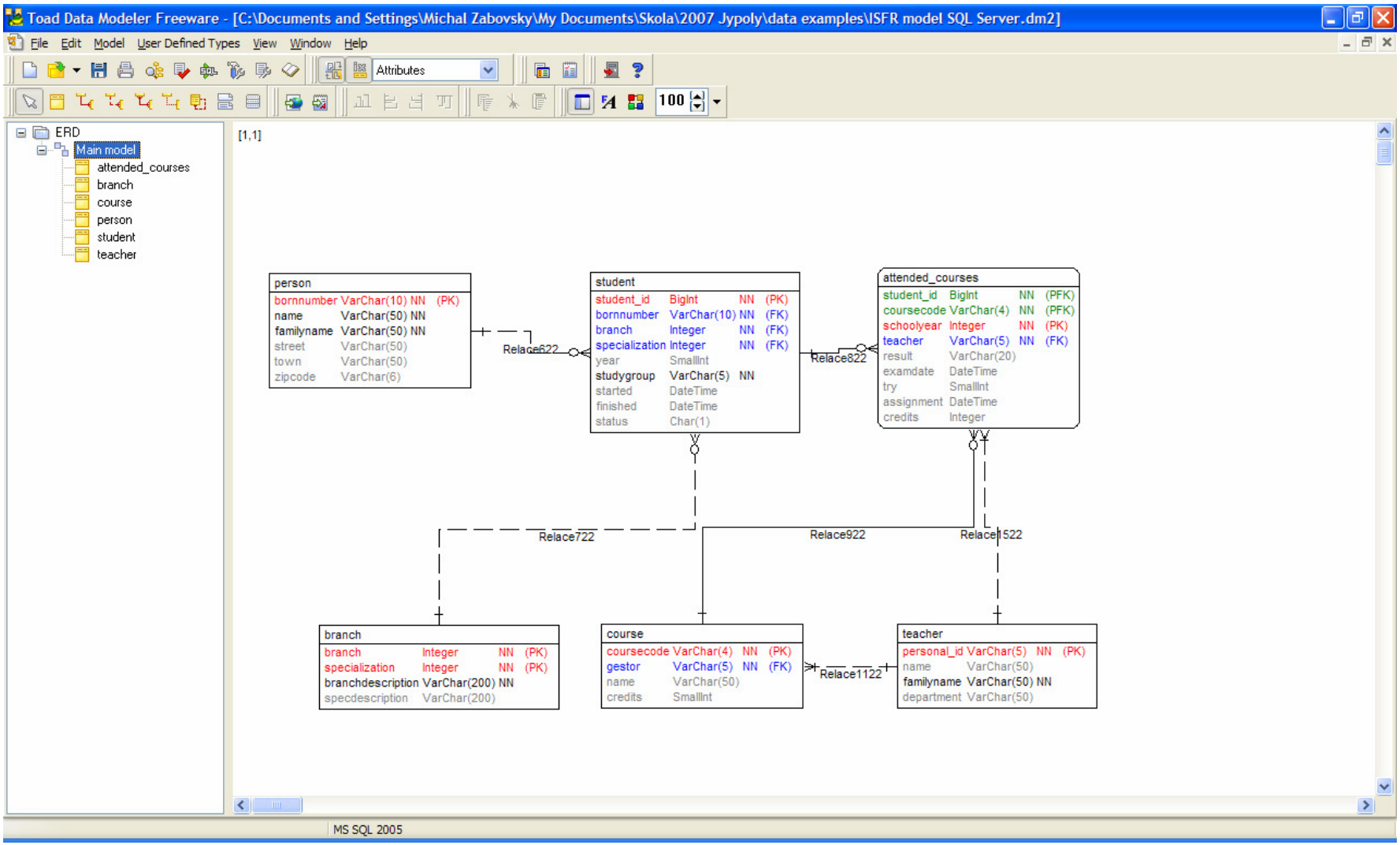
Database design

To build good application systems, design must proceed in a set of orderly steps using proper techniques at each step. Such techniques must be chosen so that the output of one step can be used as the input for the following step.

Design methodologies are collections of techniques organized into set of steps that supports an orderly way for designing database.

Design methodologies also provide the documentation necessary to record any design decisions. Many methodologies now provide software support or CASE (Computer Assisted Software Engineering) tools to support designers and maintain the documentation.

Toad Data Modeler



Data representation

Overall efficiency of RDBMS is the essential requirement of database design. Hence decision, which data type would be used for data representation is critical and should be retained on the database designer consideration.

We should consider about:

- numerical data representation (often is necessary to define accuracy or range)
- character data representation (different encoding can be used)
- data encoding (e.g. 1 = Red, 2 = Green etc.)
- units for numeric data
- data materialization
- structure of stored records
- structure of stored files

Data modeling

Data modeling is the act of exploring data-oriented structures. Like other modeling artifacts data models can be used for a variety of purposes, from high-level conceptual models to physical data models.

Two main types of models:

- **Logical models** - used in DBMSs for the organization of data at a \level that abstracts from physical structures. Examples: relational, network, hierarchical, object
- **Conceptual models**: used to describe data in a way that is completely independent of any system, with the goal of representing the concepts of the real world; they are used in the early stages of database design. The most popular is the **Entity-Relationship** model

Data models used in practice:

- Conceptual data models (CDM)
- Logical data models (LDM)
- Physical data models (PDM)

Conceptual model

One of the most difficult problem during the design phase is transformation of data to information stored in the database. The problem grows with the very weak process formalization. Data itself provides just partial information, hence additional description must be created by the conceptual model. This description must follow conceptual view of users to the particular real world part.

Conceptual design works with terms like **entity**, **object**, **relation**, **attribute**, **property** etc. Since the terms are close to real world the models are sometimes called *object-oriented models*.

Each conceptual model solves three problems:

- Data structure
- Data manipulation
- Data integrity

Logical model

Logical model is used to explore:

- Domain concepts
- Relationships

This could be done for the scope of a single project or for entire enterprise.

Logical model identifies:

- Logical entity types, typically referred to simply as entity types
- Data attributes describing those entities
- Relationships between the entities

Physical model

Physical model is used to design the internal schema of database.

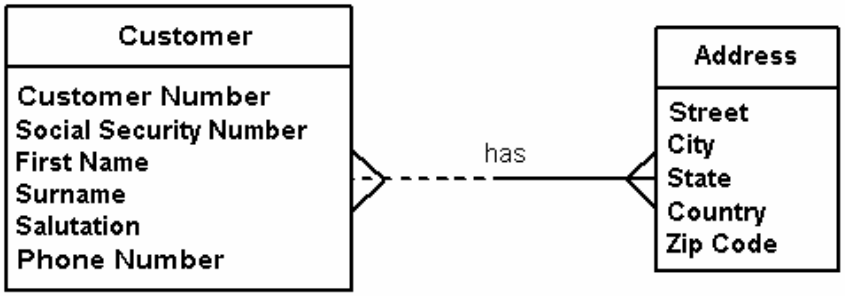
Physical model depicts:

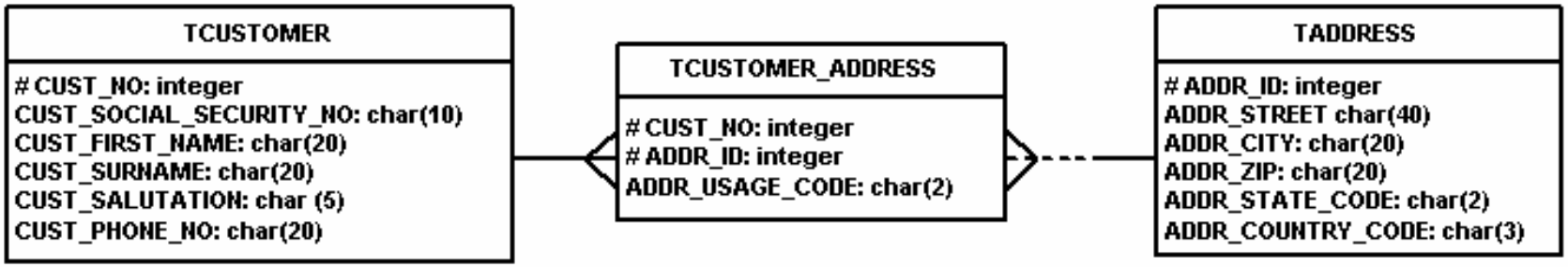
- Data tables
- Columns
- Relationships

Physical data model shows greater detail, including an associative table required to implement the association as well as the keys needed to maintain the relationships.

PDM should also reflect database naming standards and indicate the data types for the columns, such as integer and char(5).

Logical and physical data model





Entity-Relationship model

Entity-relationship model (E-R model) helps to describe user's application on the conceptual level - defines top-level data semantics. It identifies the most important entities and relationships in the data.

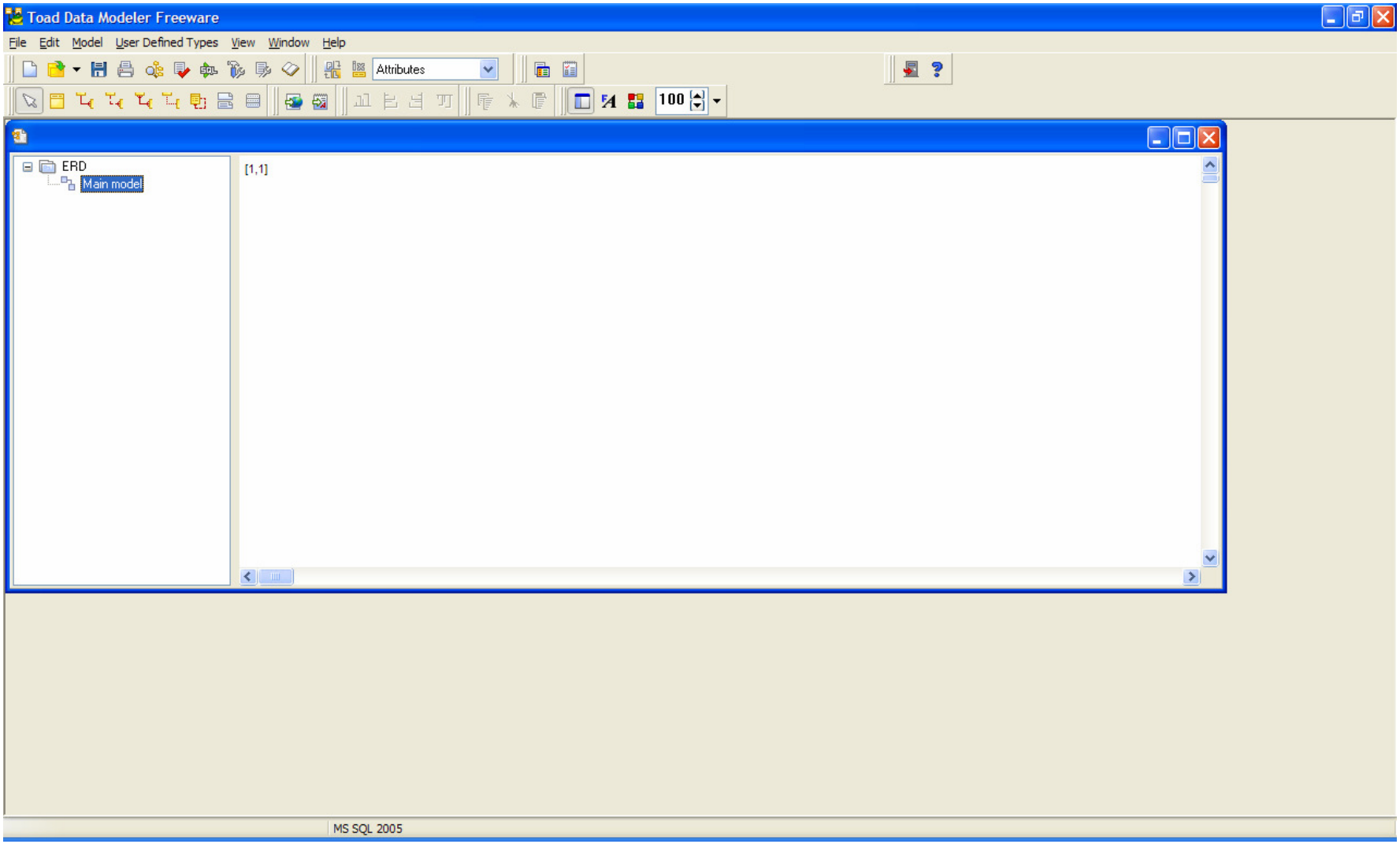
The next step is to construct a set of normalized relations. This set of relations removes any unnecessary redundancy. Then the logical definition is created.

The form of the definition depends on the database management software that is to be used to implement the database. It defines the database schema using the DBMS data definition language.

How to model data?

1. Identify entity types
2. Identify attributes
3. Apply naming conventions
4. Identify relationships
5. Apply data model patterns
6. Assign keys
7. Normalize to reduce data redundancy
8. Denormalize to improve performance

Toad – new model



E-R model elements

Entity is the real life object instance that is considered to be important in a system.

```
Example: “student Peter Novák, born number 123456/7890“
```

Entity set is a collection of objects with the same properties.

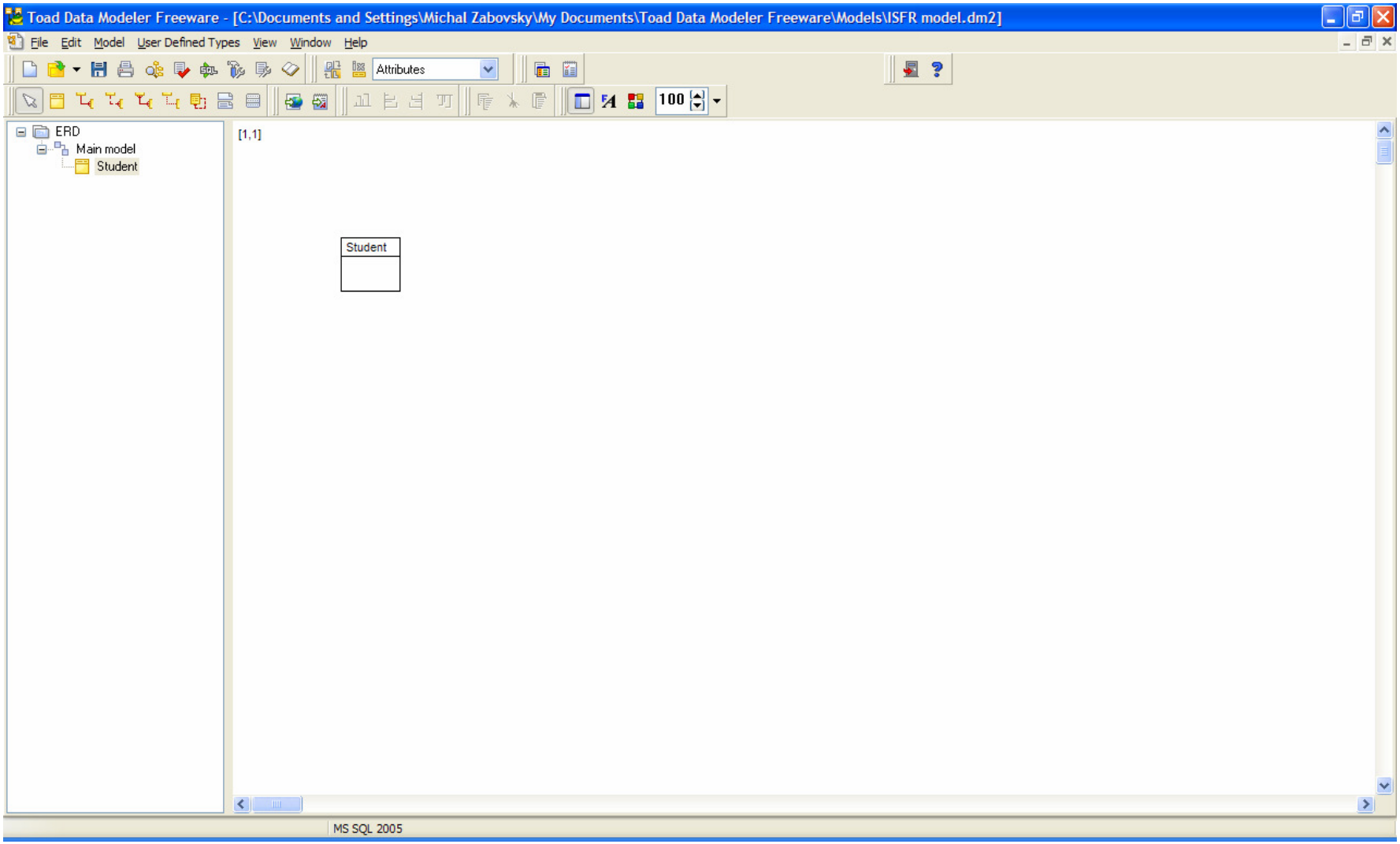
Relationship models connection between one or more entities (way, how entities interact with each other).

```
Example: entity „student Peter Novák, born number 123456/7890“
interacts  (“attend”) with entity “course Database Systems“
```

Attributes describe the properties of entities and relationships.

An entity **identifier** is and attribute (or set of attributes) whose values identify a unique entity. Identifiers are either unique (single attribute) or composite (made up of more than one attribute).

Toad – new entity



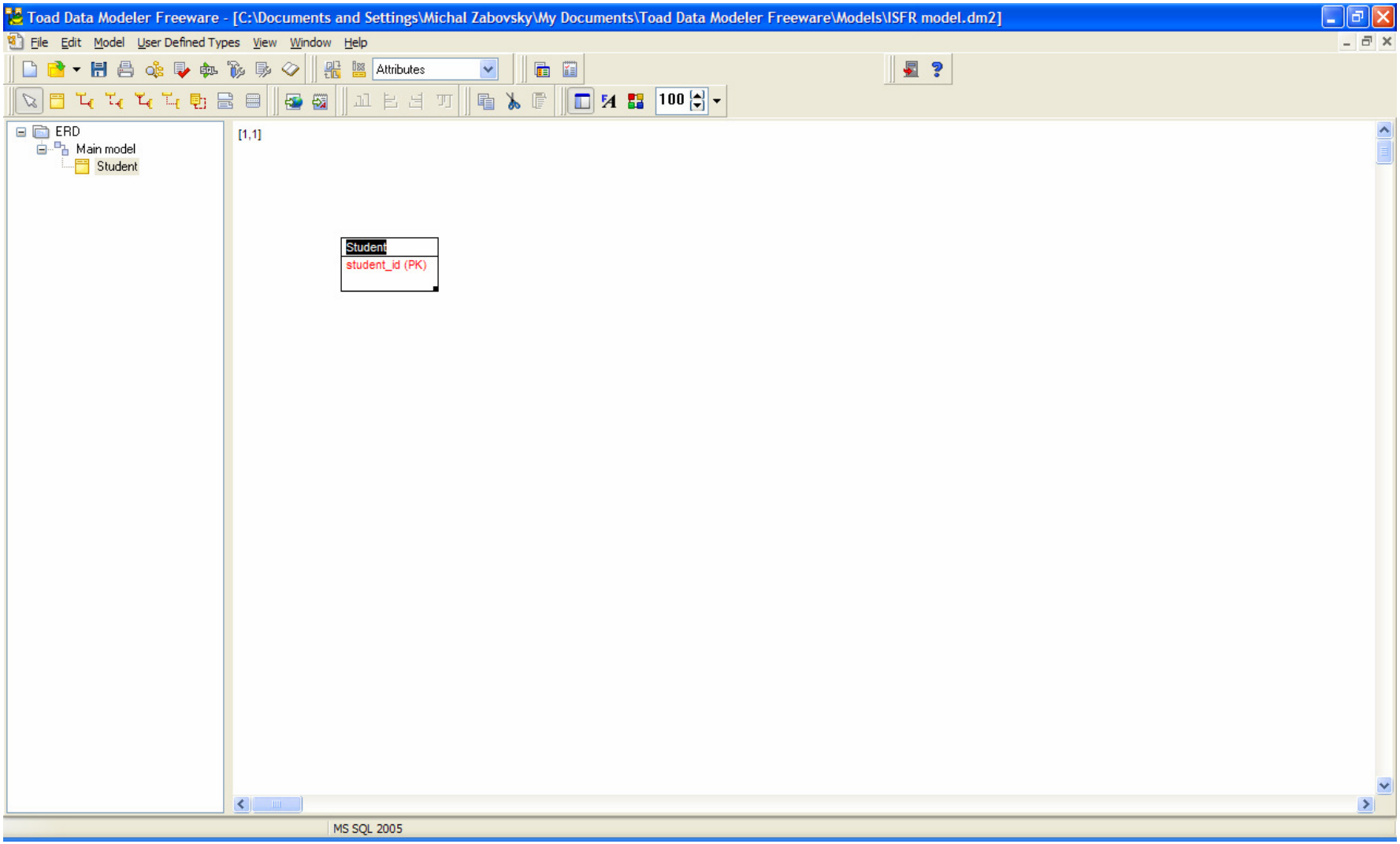
Identifiers

So far, we have made a distinction between entities and entity sets. E-R diagrams only include sets but not individual entities. However, one requirement of all systems is to be able to identify individual entities without necessarily showing them on the E-R diagram. Such identification is needed to determine the activities carried out by the entities. The usual practice is to identify entities by an entity identifier.

An entity identifier is an attribute (or set of attributes) whose value identifies a unique entity.

Since entity can have more then one candidate for identifier, is necessary to choose such identifier which is the best for efficiency or memory consumption.

Toad – primary key



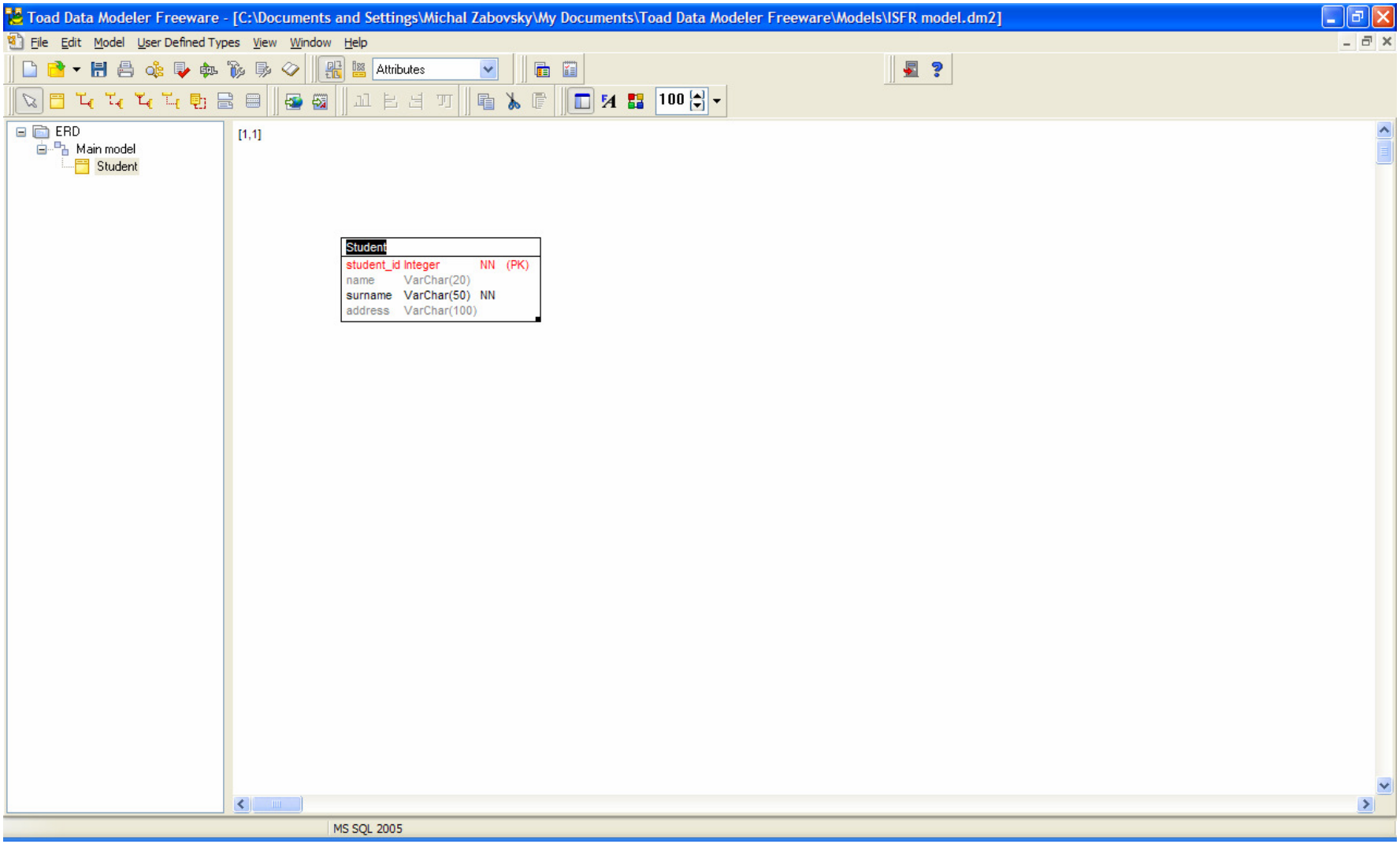
Attributes

Attributes describe the properties of entities and relationships. Each such attribute is given a name which refer to a particular property.

Each attribute is characterized by:

- name
- type
- flag, whether the attribute is part of identifier
- flag, whether the value could be undefined (NULL)
- flag, which specifies the unique value for attribute

Toad – attributes



Cardinality

Cardinality shows the number of relationships in which one entity can appear.

Cardinality 1:1 - entity can participate in only one relationship. In the case that both entities are of the same type relationship is called *recursive.*

```
Example: person <-drives-> car
```

Cardinality 1:N – entity from one set can participate in more than one relationship.

```
Example: person <-use|-> car
```

Cardinality M:N – each entity from one set can participate in more than one relationship.

```
Example: teacher <-|teaching|-> subject
```

Participation

Participation is an additional information that is often stored about relationship. This describes whether each entity in a set must participate in a relationship. Participation can be either **mandatory** or **optional**.

Department must have teacher and each teacher must be assigned to department.

```
teacher <-|has|-> department
```

Teacher must have department and department need not to have teacher.

```
teacher <-|has|-O> department
```

Department need not to have teacher and teacher need not to have department.

```
teacher <O-|has|-O> department
```

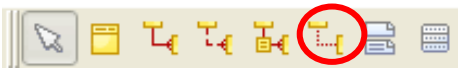
Relationships

- **Identifying relationship (1:1)** - Primary key migrates from the parent entity to the child entity and there becomes a part of the primary key. It is used when the primary key of the child entity is unable to provide definite identification. An entity, connected with a parent entity through an identifying relationship, is called a *dependent* entity and is shown in a model with rounded corners.

- **Nonidentifying relationship (1:N)** - Primary key migrates from the parent entity to the child entity and does not become a part of the primary key. This kind of relationship is often called **master-detail** relationship.

- **Many-to-many relationship (M:N)** - Most of the RDBMS is not able to represent relationship M:N directly. Also other reasons lead to decomposition to the two relationships of 1:N type.

- **Informative relationship** - Primary key does not migrate at all, no referential integrity or triggers are generated, it is only information about the existence of some relationship.



- **Self relationship** - In most cases this kind of relationship is used to model hierarchy – e.g. relationship children – parents.

Denormalization

Denormalization is process, which is from the theoretical point of view unacceptable. Common argument is that price for denormalization is very high (especially risks attended to this process). But in the real life is process of denormalization used since significant overall system efficiency improvement.

Example

Objectives:

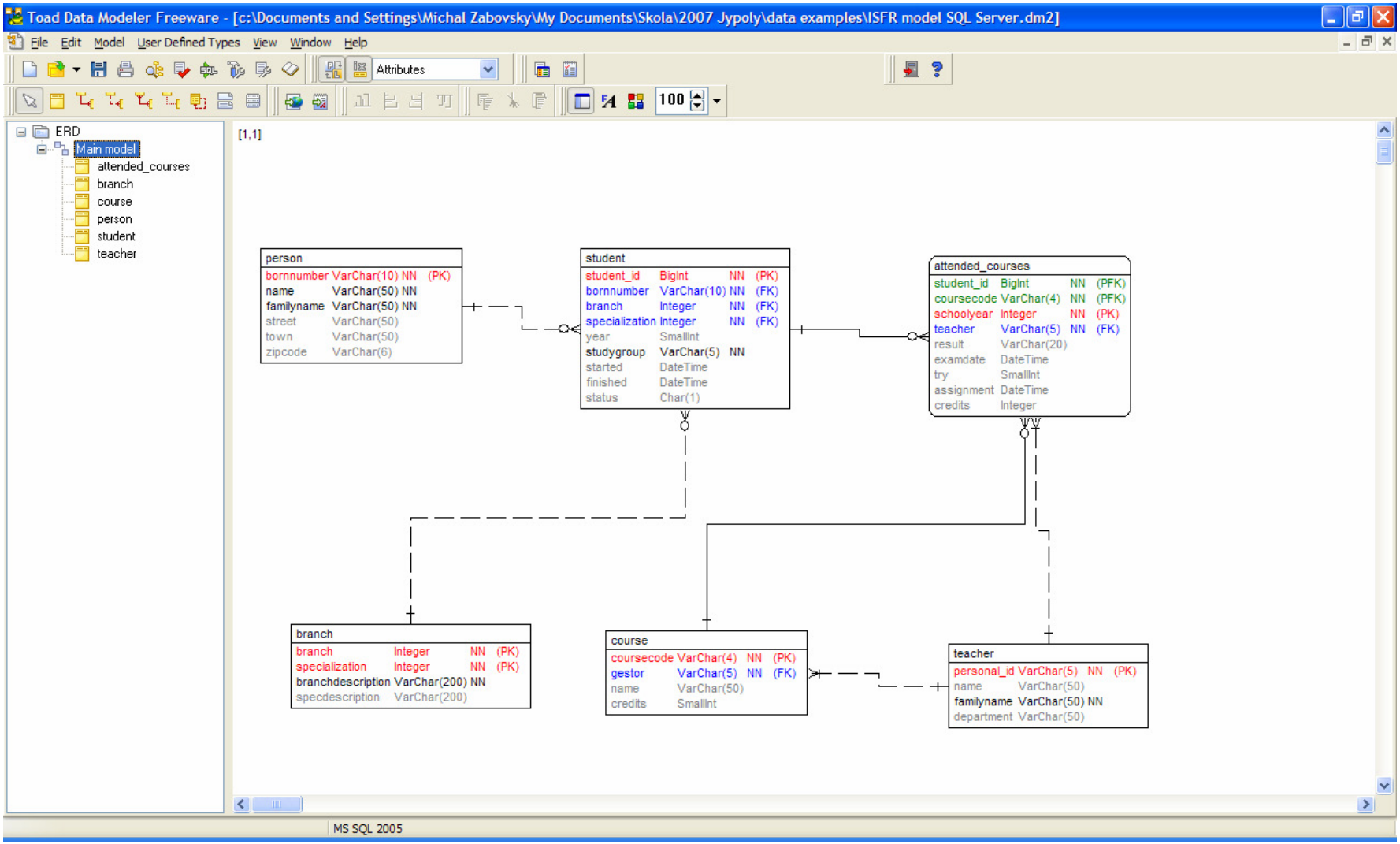
Create data model for the information system of faculty. In the IS we want to store necessary information about:

- students
- teachers
- attended courses
- specializations

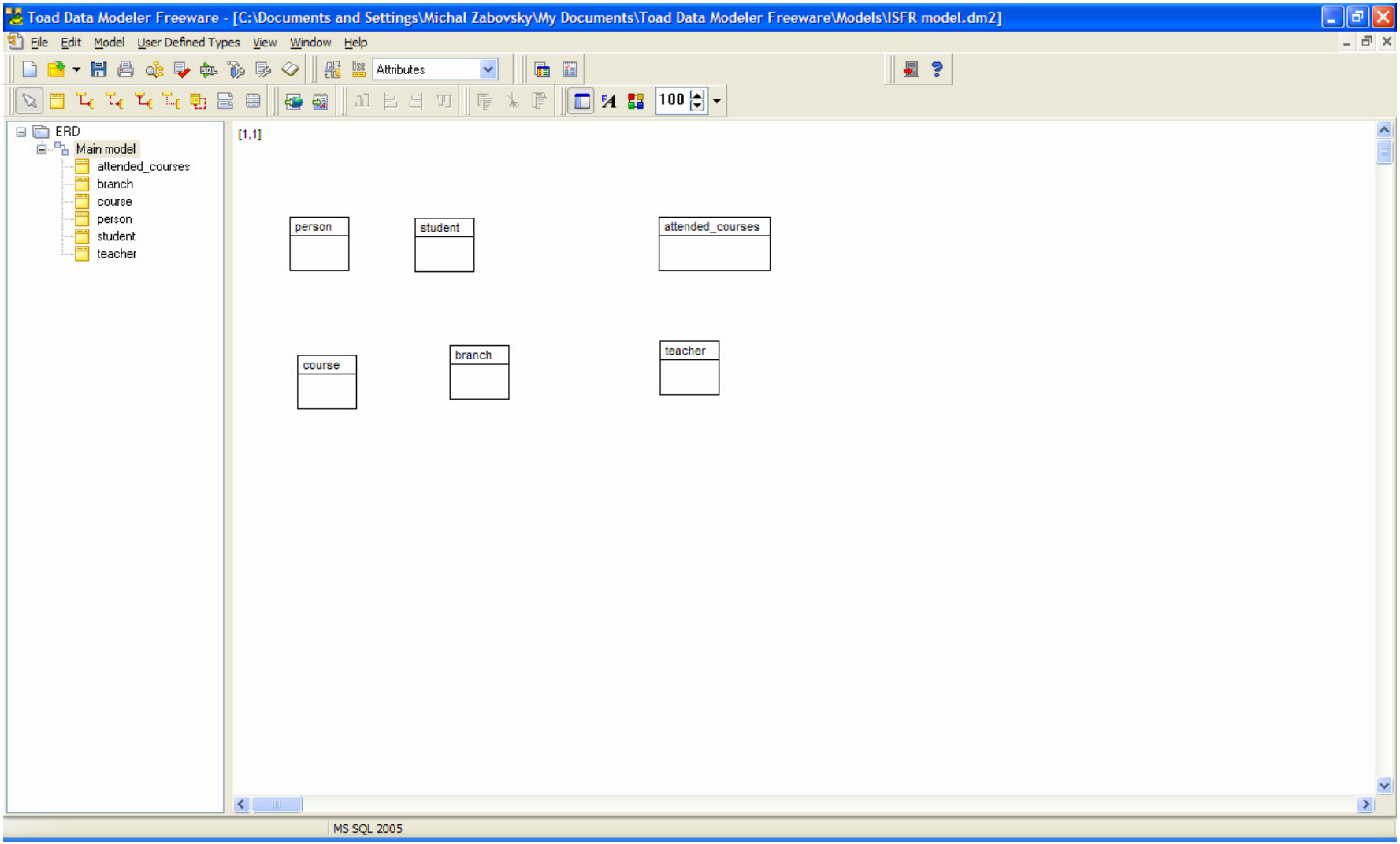
Steps:

1. Create entities person, student, attended_courses, branch, course and teacher
2. Define attributes for entities
3. Create relationships, define participation
4. Define physical data types for attributes

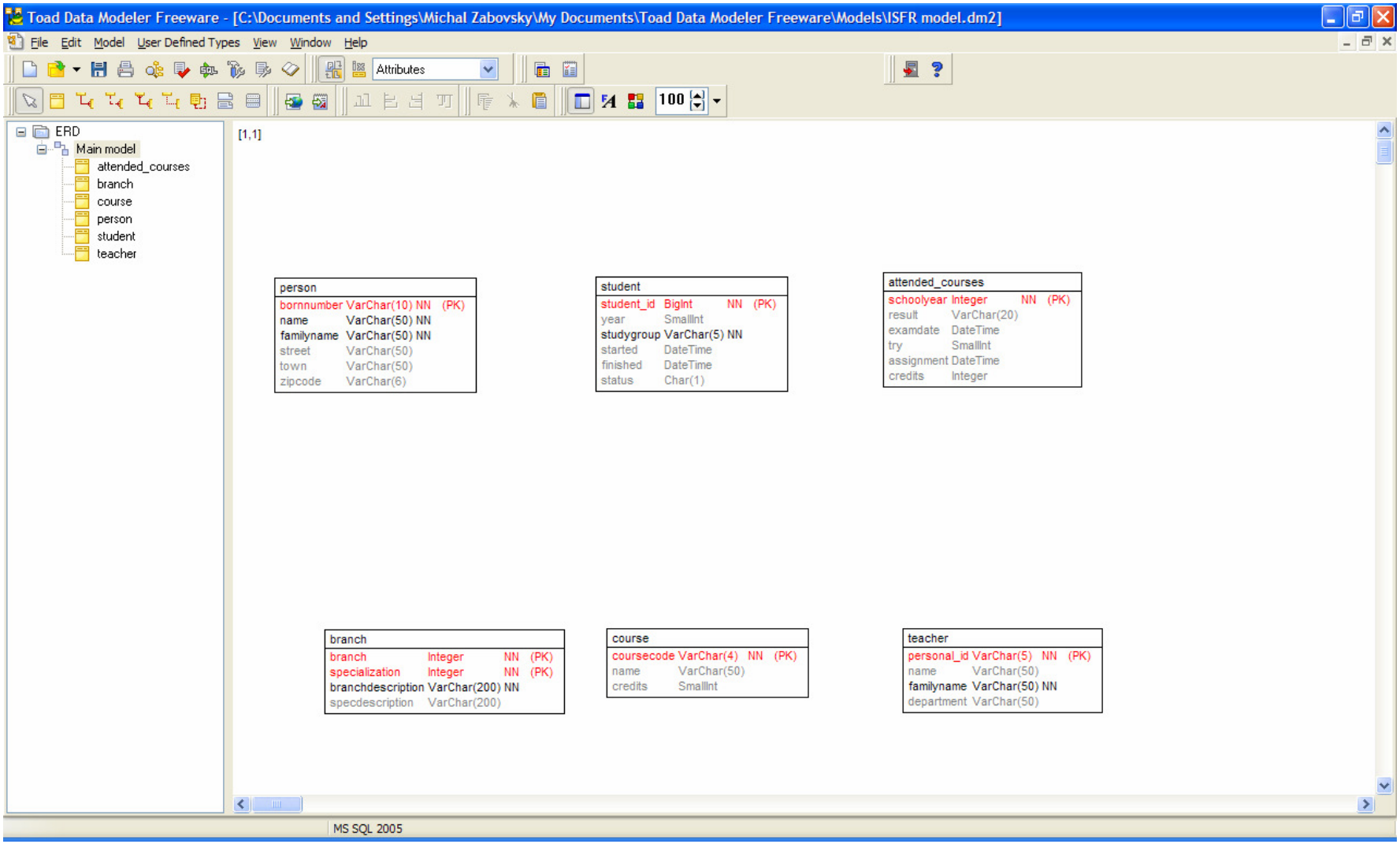
What will be done?



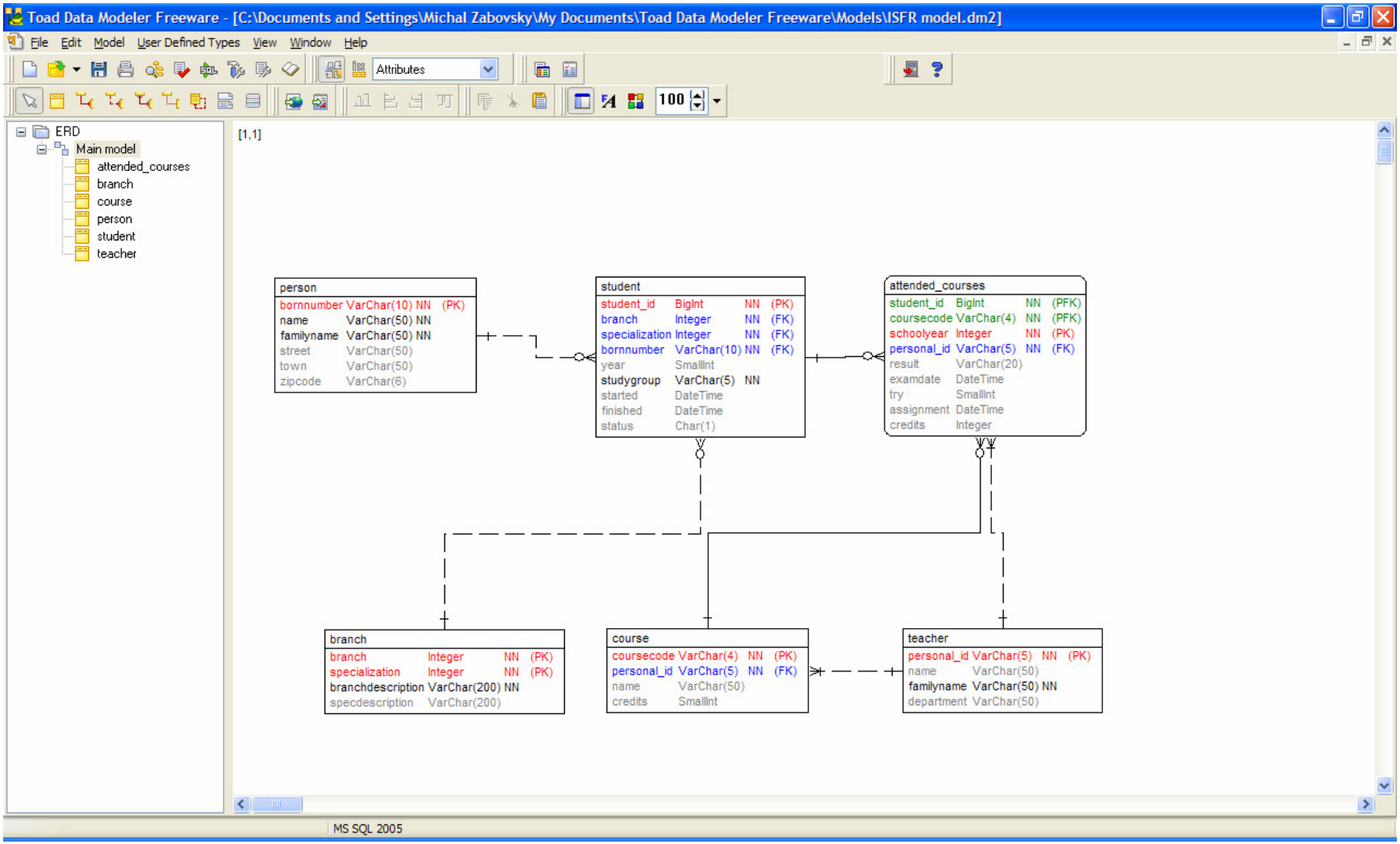
Step 1: Entities



Step 2: Attributes



Step 3: Relationships



What's next?

When the logical and consequently physical model is designed, whole model can be implemented on particular database system. Implementation consists of following steps:

- Database system installation
- Database configuration and tuning based on particular usage
- Database and relations creation
- Data import or creation
- User and access configuration
- Archiving and monitoring system configuration

Bibliography

1. Atzeni, P., Ceri, S., Paraboschi, S., Tarlone, R.: *Database Systems – Concepts, Languages and Architectures*, McGraw-Hill, 1999

Ing. Michal Zábovský, PhD.

michal.zabovsky@fri.uniza.sk

Department of Informatics
Faculty of Management Science and Informatics
University of Zilina

UNIVERSITY OF ZILINA
FACULTY OF MANAGEMENT SCIENCE AND INFORMATICS
DEPARTMENT OF INFORMATICS

Univerzitna 8215/1SK-01026, Zilina, Slovak Republic
Phone: +421-41-513 4181
Fax: +421-41-513 4055
Homepage: http://www.fri.uniza.sk

Introduction

The Department of Informatics comprises around 20 academics and research fellows who form research community in Computer Science. Its complement of people directly involved in research is close to 50. The Department is strongly involved in many practical collaborative industry projects and research projects on national and international level.

Research

Research addresses the fundamentals of computer systems, architectures, database systems and information analysis. The key research topics cover distributed and parallel systems and advanced database systems.