***Relational database and database management system***

**Abstract**

In lecture we explain the fundamental concepts of databases. In particular we present *the relational data model*. Modern commercial databases are based on this model. We have also enclosed some information on *the client-server architecture* which is used in database applications. At the end of the lecture you will find a brief summary.

None of the exercises for this lecture require using any software.

**Database**

Here are some general facts that drive the whole domain of databases.

1. *The data* is an asset of the company just like the others (employees, machines, stock). The data must be *managed* just like the other assets.
2. *The information* is the data with semantics (defined by the meaning of the data in the company).
3. The management of data is realized by *the information system* that caters for the need for information inside the company.
4. *The database* has become the standard means to structure the process of data management.
5. The database is a part of the information system of the company.

**Why is this lecture important and interesting?**

* Information is ubiquitous:
  + from simple services in web browsers
  + to scientific applications.
* There are more and more collections of data varying in the degree of complexity and size.
  + electronic libraries, multimedia databases, interactive video, the exploration of the human genome, space exploration (e.g. NASA).
* The development of a database requires expertise in many areas of information technology:
  + operating systems, information theory, artificial intelligence, programming languages, multimedia, software engineering.
* Database systems constitute one of the largest and most active parts of the market.

**Information system**

Some aspects of an information system highlight its implementation nature:

* *The database application* is considered, when we want to pay closer attention to the software and to emphasize the remarkable role of the database.
* *The computer system* is considered, when we want to pay closer attention to all computer-related artifacts, i.e. both hardware and software.

**Relational data model**

Relation data model was invented by Edgar Codd and presented for the first time in a paper in 1970. From the mathematical point o view *a database* is a set of relations. This is why we talk about *the relational data model* and *relational databases*. Mathematicians define a relation as a subset of the Cartesian product of some sets. A relation may be represented as *a two-dimensional table* that consists of *rows* and *columns*. We assume that:

1. The number of columns is fixed.
2. Each column is assigned *a name* and *a domain*.
3. There is exactly one atomic value at the intersection of a row and a column. This value belongs to the domain of this column.
4. A row represents a single chunk of information, e.g. a person.
5. We take no account of the order of rows (records) and columns (fields of records).

**Table *Teachers***

|  |  |  |  |
| --- | --- | --- | --- |
| **TeacherId** | **FirstName** | **LastName** | **Title** |
| 2137 | Robert | Brown | Prof. |
| 3245 | Alex | Sten | Dr. |
| 8976 | John | Smith | Mr. |

**Table *Lectures***

|  |  |  |
| --- | --- | --- |
| **LectureName** | **Code** | **TeacherId** |
| Databases | DBA | 2137 |
| Information systems | INS | 3245 |
| Internet technologies | INT | 3245 |
| Object-oriented programming | OOP | 8976 |
| Expert systems | EXS | 2137 |

|  |
| --- |
| Column *TeacherId* of table *Lectures* is specific, because:   1. Its values do not describe properties of lectures. 2. It represents *the relationship* between the lecture and a teacher, whose data is stored in a separate table. In order to retrieve this data we have to use the identifier and find appropriate row in the related table. 3. Therefore, it is important that this identifier uniquely determines the teacher. In the relational data model there is no other means of setting the identity of a row. You can do it by means of the stored values only. |

**Primary key and unique keys**

1. In every table there must be a unique identifier. We call it *the primary key* It is a subset (may be a singleton) of the columns of the table such that the values in these columns uniquely identify the row.
2. *A unique key* (or *alternative key* or simply *key*) has the same property as the primary key. However, there is only one primary key, whereas the number of unique keys may be grater than one.

|  |
| --- |
| The primary key of the table *Lectures* is the column *Code*. The column *LectureName* is a unique key.  The primary key of the table *Teacher* is the column *TeacherId*. The column *SecondName* need not be a key. |

Please answer the following simple question:

|  |
| --- |
| [How](javascript:popUp('ok1.html',380,50)) many primary keys may a table have? |

**Foreign key**

*A foreign key* is the set of one or more columns whose values are also the values of the primary key or a unique key of the related table (it can be the same table). The values of these columns are interpreted as pointers to the rows of the related table.

Please answer the following simple question:

|  |
| --- |
| [How](javascript:popUp('ok2.html',420,50)) many foreign keys may a table have? |

**Null = lack of value**

Domains of all columns are extended by the special value called Null. The meaning of Null is the lack of values. It could be:

* temporary or
* permanent (Null is something else than the empty string or zero).

All comparisons with Null yield Null as the result (expression Null=Null also evaluates Null). It concerns almost all operations that have Null as the argument - their value is Null.

Therefore, Null is the third logical value besides True and False. Here are the truth tables for three major logical connectives:

**Disjunction (OR)**

|  |  |  |  |
| --- | --- | --- | --- |
| **OR** | **True** | **False** | **Null** |
| **True** | True | True | True |
| **False** | True | False | Null |
| **Null** | True | Null | Null |

**Conjunction (AND)**

|  |  |  |  |
| --- | --- | --- | --- |
| **AND** | **True** | **False** | **Null** |
| **True** | True | False | Null |
| **False** | False | False | False |
| **Null** | Null | False | Null |

**Negation (NOT)**

|  |  |  |  |
| --- | --- | --- | --- |
| **NOT** | **True** | **False** | **Null** |
|  | False | True | Null |

It is time for a simple question:

|  |
| --- |
| [Is](javascript:popUp('ok3.html',400,50)) it possible that a logical expression is true, when one if its arguments is Null? |

**Predicates Is Null and Is Not Null**

They allow recognizing null values.

(X Is Null) evaluates to True, when X is Null  
(X Is Null) evaluates to False, when X is not Null

However, be careful, because

(X = Null) **always** evaluates to Null (even for X being Null)

Here is another simple question:

|  |
| --- |
| [What](javascript:popUp('ok4.html',350,50)) is the logical value of expression (Null=Null) Is Null? |

**Integrity constraints**

*Integrity constraints* are conditions that must be fulfilled by data in a database, e.g.

* a constraint for a single column:

0 < Age < 140

* a constraint for several columns:

birthDate < hireDate

* the uniqueness (the primary keys, unique keys),
* NOT NULL constraint,
* *the referential integrity*: each value of a foreign key is either Null or occurs also in the appropriate column of the associated primary (or unique) key.
* a more complex *business rule* that can be programmed in a more sophisticated language, e.g.
* The sum of the salaries of all employees of a department
* equals

the wage fund of this department.

**View**

*A view* is a virtual table to be used by the users, e.g.

**Lectures-Teachers**

|  |  |
| --- | --- |
| **LectureName** | **Teacher** |
| Databases | Brown |
| Information systems | Sten |
| Internet technologies | Sten |
| Object-oriented programming | Smith |
| Expert systems | Brown |

The content of a usual view is calculated from the base tables at each usage. It is not stored in the database. However, sometimes we need to save this content in the database. Then we use this frozen image and do not recompute it again and again. Such view is called *materialized*.

**Levels of a relational database**

The same data of a database may be viewed and processed in many different ways. The point of view depends on the level of abstraction. The basic levels of abstraction are:

1. *The user level* consists of the views for users.
2. *The logical (conceptual) level* consists of views, tables and indexes.
3. *The physical level* consists of files with data and indexes.

Views are defined on the logical level and used on the user level. Indexes are defined on the logical level and used on the physical level. Tables are defined on the logical level and used on the physical level and the user level.

The user level is intended mainly for *the end users*. The logical level is used by *the system data administrator*, while *the database administrator* (shortly *dba*) works with the physical level. Of course, *the designer of the database* defines and works with all three levels.

Each level is somewhat independent. For example, you can change the location of the data on the disk as well as the storage parameters without any impact on the logical structure of the tables. Such a possibility is called *the physical data independence*. You can also change tables without any change of the applications, provided these are based on views and not directly tables. This feature is called *the logical data independence*.

Consider our sample database. Its application can consist in displaying the information on who delivers lectures, i.e. displaying the content of a view [Lectures-Teachers](https://edux.pjwstk.edu.pl/mat/2398/lec/wyklad01/wyklad.html#lect-teach). In the future it may turn out that some lectures are given by more than one teacher. The schema of the database with two tables *Teachers* and *Lectures* is no longer sufficient. A new associative table is required. It will store the relationship between teachers and lectures. The definition of view [Lectures-Teachers](https://edux.pjwstk.edu.pl/mat/2398/lec/wyklad01/wyklad.html#lect-teach) must be changed as well. All these changes apply to the logical level of the database. The application itself is based on this view and thus it need not be changed. It is an example of the meaning of the logical data independence.

The physical data independence becomes important, when the disk is already full and we have to buy a new one. New rows will be stored on the new disk, but neither the schemata of the tables nor the applications have to be changed.

**Catalog**

*The catalog* (or *data dictionary* or *metadata*) is the collection of tables and views that describe the schema of the database. In other words, the catalog contains definitions of all the objects of all the three levels of the database. The name and the data type of a column are examples of metadata.

It is important to use the relational data model for the catalog. This ensures that the metadata are stored and processed the same way as the data.

**Client-server architecture**

Database applications usually consists of at least two parts:

*the client side*

that operates on the user's workstation,

*the server side*

that operates on the computer that runs the database server, i.e. the database itself and the database management system (DBMS).

The functions performed at the server side are:

1. storing data and facilitating access to it,
2. executing statements of the database language (usually it is SQL that will be presented during the next lecture),
3. enforcing data integrity,
4. managing the resources of the database, e.g. the users' accounts.

The functions performed at the client side are:

1. interacting with the user (running the user interface),
2. explaining the state of the computation to the user (it includes also errors and exceptional events),
3. accepting users' queries and then executing them or sending them to the server as SQL statements.

**Summary**

During this lecture we explained the fundamental notions that apply to databases. We also emphasized the importance of databases for information systems. In particular, we presented the primary data model of modern databases. We call it *the relational data model*. We described the following concepts of this model: *table*, *key*, *primary key*, *foreign key*, *view*, *pseudo-value Null* and *integrity constraints*.

**Dictionary**

[foreign key](https://edux.pjwstk.edu.pl/mat/2398/lec/wyklad01/wyklad.html#Klucze)

A set of one or more columns whose values are also values of the primary key or a unique key of the related table (it can be the same table). The values of these columns are interpreted as pointers to the rows of the related table.

[integrity constraint](https://edux.pjwstk.edu.pl/mat/2398/lec/wyklad01/wyklad.html#Wiezy)

A correctness condition for the data in a database.

[key](https://edux.pjwstk.edu.pl/mat/2398/lec/wyklad01/wyklad.html#Znaczenie)

A subset (may be a singleton) of the columns of a table such that the values in these columns uniquely identify the row of this table.

[Null](https://edux.pjwstk.edu.pl/mat/2398/lec/wyklad01/wyklad.html#NULL)

The pseudo-value which means that the data is missing.

[primary key](https://edux.pjwstk.edu.pl/mat/2398/lec/wyklad01/wyklad.html#Znaczenie)

The distinguished key which is used to identify objects.

[referential integrity constraint](https://edux.pjwstk.edu.pl/mat/2398/lec/wyklad01/wyklad.html#ref-cons)

An integrity constraint which states that each value of a foreign key is either Null or occurs also in the apropriate column of the associated primary (or unique) key.

[table](https://edux.pjwstk.edu.pl/mat/2398/lec/wyklad01/wyklad.html#tabela-def)

A two-dimensional structure which consists of *rows* and *columns*. At the intersection of a row and a column there is only one atomic data item. A row stores a record of data on an object (e.g. a person or a company) or a relationship between objects. Each column contains a set of atomic data items which describe one of the attributes of an object (e.g. the name of the company or the last name of a person).

[unique key](https://edux.pjwstk.edu.pl/mat/2398/lec/wyklad01/wyklad.html#Znaczenie)

A key that is not primary.

[view](https://edux.pjwstk.edu.pl/mat/2398/lec/wyklad01/wyklad.html#Perspektywy)

A virtual table created for users. It is defined on the logical level and used on the user level. If it is physically saved in the form of a relational table, we call it *a materialized view*.

**Data modeling notation IDEF1X**

We are going to present two alternative notations for entity-relationship diagrams. In practice many different notations are used and one best standard does not exist. All these notations are based on the notions *entity*-*attribute*-*relationship*. The choice of the notation is really a secondary (or even tertiary) issue. It is usually determined by the CASE tool used by the project team.

We will start from IDEF1X. It is available in Visio. Simply choose: "Database -> Options -> Document :: Symbol set = IDEF1X" (instead of default "Relational").

Obraz zawierający tekst, diagram, Plan, Rysunek techniczny

Zawartość wygenerowana przez sztuczną inteligencję może być niepoprawna.

* Solid line = identifying relationship
* Dashed line = non-identifying relationship
* Black circle = detail entity
* Hollow diamond = optional relationship
* IE = index (Inversed Entry)

IDEF1X is the notation used by Erwin - a tool used earlier. IDEF1X allows to model many-many relationships directly, but this feature is not available directly in MS Visio. You can simulate such a relationship by the item "Dynamic connector".

Obraz zawierający tekst, zrzut ekranu, linia, numer

Zawartość wygenerowana przez sztuczną inteligencję może być niepoprawna.

|  |
| --- |
| A *person* can work on many *projects*. A *project* is conducted by many *persons*.  Connection IS WRONG. |

**Chen's data modeling notation**

Peter Chen invented the modeling method based on entities and relationships in 1976. Here we present this very first notation for entity-relationship diagramming. It is more general than the two methods described previously, because it allows defining multi-argument and many-many relationships. In this notation:

* Entity = rectangle
* Attribute = circle
* Relationship = diamond

If you want to use Chen's notation in Visio, use option "File -> New -> Database -> Chen ERD" instead of "File -> New-> Database -> Database Model Diagram".

Obraz zawierający diagram, tekst, linia, Wykres

Zawartość wygenerowana przez sztuczną inteligencję może być niepoprawna.

|  |
| --- |
| A person participates in a projects in a role. |

As an "exploratory" task examine MS Visio:

|  |
| --- |
| Find more notations for entity-relationship diagrams offered by MS Visio. Perhaps some other notation is more suitable for you? |

**Views**

A view is a user-oriented presentation of the data from a database. It is created to cater for the needs of a user of the database. The first example shows a view with names of persons (it is an attribute of the entity *Person*) and the locations of their departments (it is an attribute of the entity *Department*). When you define a view, you have to specify the join condition for tables participating in the view.

Obraz zawierający tekst, zrzut ekranu, diagram, numer

Zawartość wygenerowana przez sztuczną inteligencję może być niepoprawna.

The second example shows view *ProductOrder* with products ordered by customers. It contains two attributes: *Customer name* (an attribute of entity *Customer*) and *Product name* (an attribute of entity *Product*). Note that entities *Customer* and *Product* that provide attributes for this view are not connected directly. When you define the join condition (tab "Join Criteria"), specify a sequence of join conditions that starts from entity *Customer*, goes through entities *Order* and *Line item*, and stops at entity *Product*. This example shows that the definition of a view can include other entities than those that provide attributes for the view.

Obraz zawierający tekst, zrzut ekranu, diagram, numer

Zawartość wygenerowana przez sztuczną inteligencję może być niepoprawna.

When you generate this model in MS Access, you will get a select query for each view.

**Subcategories**

There is one more relationship that frequently appears in entity-relationship models. This relationship can connect any number of relationships. In such cases one entity is distinguished as the *super-entity* (or *general entity*), while all the other related entities are the *sub-entities* (or *specific entity*). Such relationships are called *category relationships*. They allow specifying the inheritance of properties of the general entity by the specific entities. For example *Person* is the super-entity for sub-entities *Designer*, *Analyst* and *Secretary*.

|  |
| --- |
| A person may be a designer, an analyst or a secretary. Common properties are grouped in the entity *Person*. Specific attributes of sub-entities fall into one of entities *Designer*, *Analyst* and *Secretary*. |

Obraz zawierający tekst, zrzut ekranu, wyświetlacz, diagram

Zawartość wygenerowana przez sztuczną inteligencję może być niepoprawna.

The attribute *Job* is called *the discriminator* because it indicates to which sub-entity belongs an instance of the entity *Person*. This categorization has been designated to be *complete*, i.e. every person belongs to one subcategory.

Here is the full tool for the option "Entity Relationship" in MS Visio:

Obraz zawierający tekst, zrzut ekranu, wyświetlacz, Czcionka

Zawartość wygenerowana przez sztuczną inteligencję może być niepoprawna.

A category relationship can be replaced by a set of one-to-one relationships between the super-entity and each of the the sub-entities. During lecture 3 we presented three ways to implement a one-to-one relationship in a relational database. They can be used to implement a category relationship:

1. separate tables for the super-entity and all the sub-entities,
2. separate tables for all the sub-entities,
3. one common table.

MS Visio generates an Access database with the first method, i.e. separate tables are created for the super-entity and all the sub-entities:

Obraz zawierający tekst, zrzut ekranu, linia, diagram

Zawartość wygenerowana przez sztuczną inteligencję może być niepoprawna.

**Hierarchical data**

One of the most frequent patterns of data models are *hierarchies*. For example let us consider the structure of a company:

Obraz zawierający tekst, zrzut ekranu, Czcionka, linia

Zawartość wygenerowana przez sztuczną inteligencję może być niepoprawna.

There is an alternative way to model it. All units of the company can be instances of one entity. The link to the master unit becomes the recursive relationship around this only entity.

Obraz zawierający tekst, zrzut ekranu, Czcionka, diagram

Zawartość wygenerowana przez sztuczną inteligencję może być niepoprawna.

This model is simpler and more flexible. When the structure of the company changes, there will be no need to change the schema of the database. Note that the recursive relationship that represents the hierarchy must be optional. Otherwise there will be no end of the hierarchy. If the relationship is optional, we can walk up this tree and we will eventually reach the *root of the hierarchy*.

The entity *Unit* has an attribute *Type* that indicates the type of the unit, e.g. "Department" or "Branch". The set of these type is small and rarely changed. To facilitate the integrity checks for this attribute we can introduce a so called *dictionary entity*. On the diagram below *UnitType* is a dictionary entity.

Obraz zawierający tekst, zrzut ekranu, diagram, Czcionka

Zawartość wygenerowana przez sztuczną inteligencję może być niepoprawna.

|  |
| --- |
| The set of the components of a car constitutes a hierarchy based on the containment relationship (a component aggregates other components). Draw a data-relationship diagram of this hierarchy. |

**Modeling changes in time**

Designers of databases often face the problem of data that changes in time. The question is how to include the aspect of time in the data model. We are not only interested in the current salary, job and department of an employee but we want to know also his/her prevoius salaries, jobs and departments.

Obraz zawierający tekst, zrzut ekranu, linia, diagram

Zawartość wygenerowana przez sztuczną inteligencję może być niepoprawna.

In such cases we employ the same technique that was used to handle many-many relationships. We add a so called *temporal entity* that will represent the changes in time. It can store the log of values of the attributes or instances of the relationships with other entities. First, we solve the problem how to enrich the data model so that it contains the history of salaries (attribute *Salary*) and jobs (attribute *Job*). We add new dependent entities *SalaryHist* (the history of salaries) and *JobHist* (the history of jobs). We also add an attribute (*From*) to mark the time the change occurred. This attribute should be an element of the primary key. Perhaps you will want to create another attribute *To* which is to store the time when the value becomes invalid.

Obraz zawierający tekst, zrzut ekranu, linia, Czcionka

Zawartość wygenerowana przez sztuczną inteligencję może być niepoprawna.

Now we are going to solve the problem how to enrich the data model so that it contains the history of employees' assignments to departments (this is a relationship between entities *Person* and *Department*). We add a new dependent entity *AssigmentHist* (the history of employee's assignments to departments). We also add an attribute (*From*) to mark the time the assignment occurred. This attribute should be an element of the primary key. As before, you may want to create another attribute *To* that is to store the time when the assignment becomes invalid.

Obraz zawierający tekst, zrzut ekranu, linia, Czcionka

Zawartość wygenerowana przez sztuczną inteligencję może być niepoprawna.

This way we have described the changes to attributes and relationships that happen in time. We can also ask whether it is reasonable to consider changes of an entity itself. The changes of attributes of entities could be represented as before but it does not concern the primary key, because a change of its value means the change of the identity of the entity. This change can be represented as replacement of one instance by another, i.e. deletion and insertion with probable movement of some attributes.

Sometimes it could be only the deletion with the history of values of the attributes. In fact, databases of banks and companies usually store information on accounts and employees although they are no longer customers or employees. Here is one possible solution that consists in adding attribute *Status*. It indicates whether a person is currently an employee of the company.

Obraz zawierający tekst, zrzut ekranu, diagram, numer

Zawartość wygenerowana przez sztuczną inteligencję może być niepoprawna.

We could also move the data from deleted objects to separate entities that are a kind of historical archive of the database. It would be reasonable especially when the number of instances of the entities is huge, because it speeds up database searches.