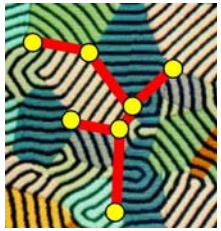


METRO
MEtallurgical TRaining On-line

Fundamentals of Relational Databases

Romuald Kotowski
IPPT PAN

Education and Culture

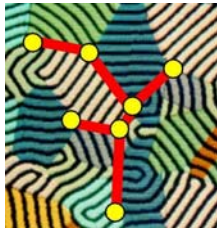
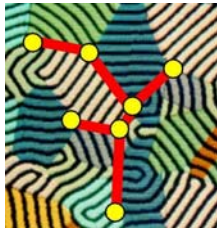


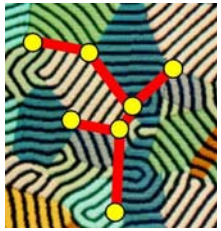
Table of Contents



Introduction

Database is a model of a certain part of the real world. The fundamental components of the database are called classes or entities. These classes have certain properties or attributes. E.g. metals have names and a set of physical and chemical properties like density, melting temperature, chemical activity etc.

The every database consists of two parts: one part is the collection of definitions defining the structure of the database – it is called the schema of the database. The second part of the database is the cumulative collection of data.



DataBase Management System

Talking nowadays about the data bases we have on mind the **DBMS** – the **D**ata **B**ase **M**anagement **S**ystem – in fact. It is not enough to have data, we have to know how to find it and then how to use it. Usually the well developed data bases contain the huge amount of data, so it is quite not a trivial job to find the needed data as soon as possible and even faster.

DataBase Management System

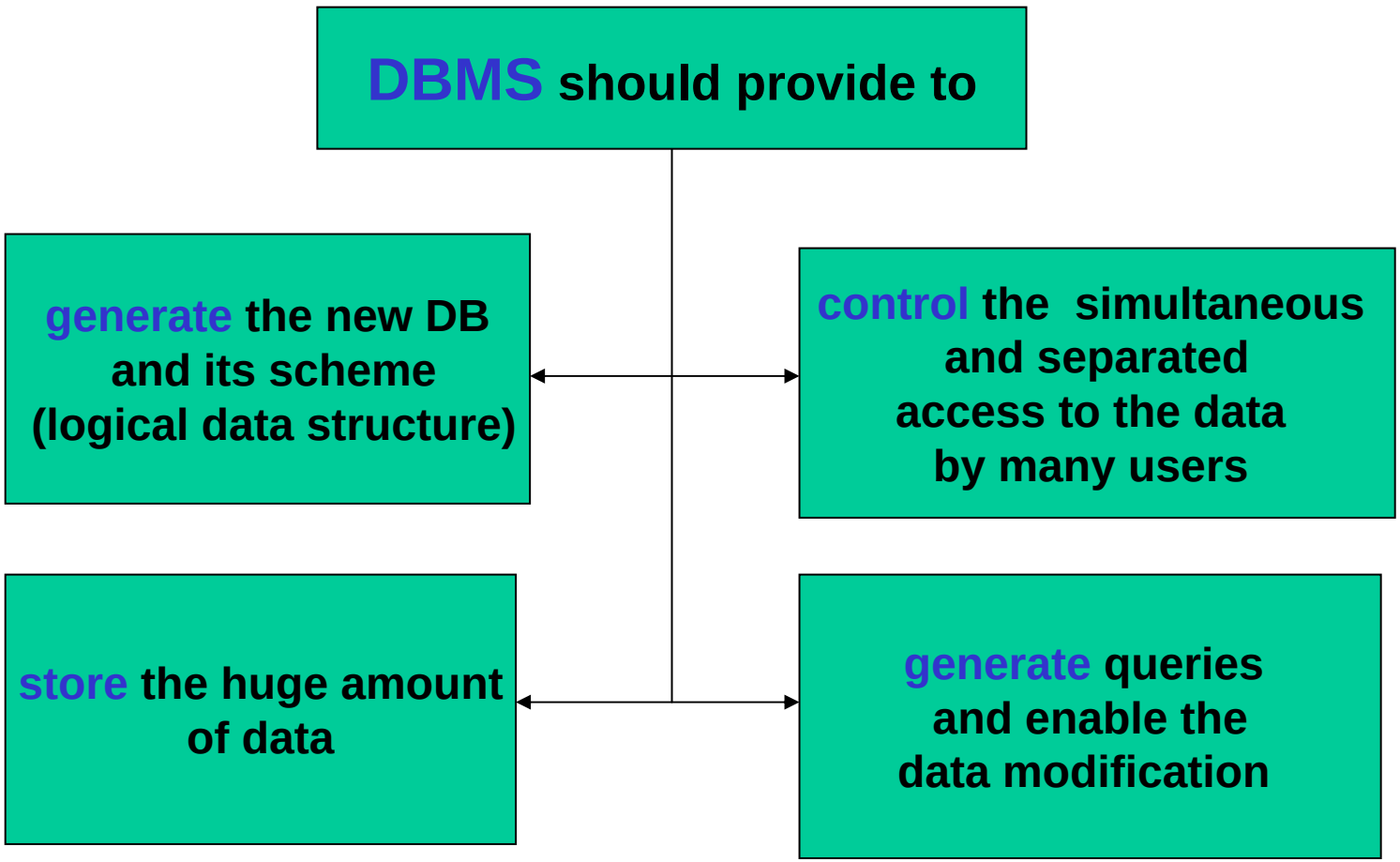


Fig. 1. Objectives of the DBMS

DataBase Management System

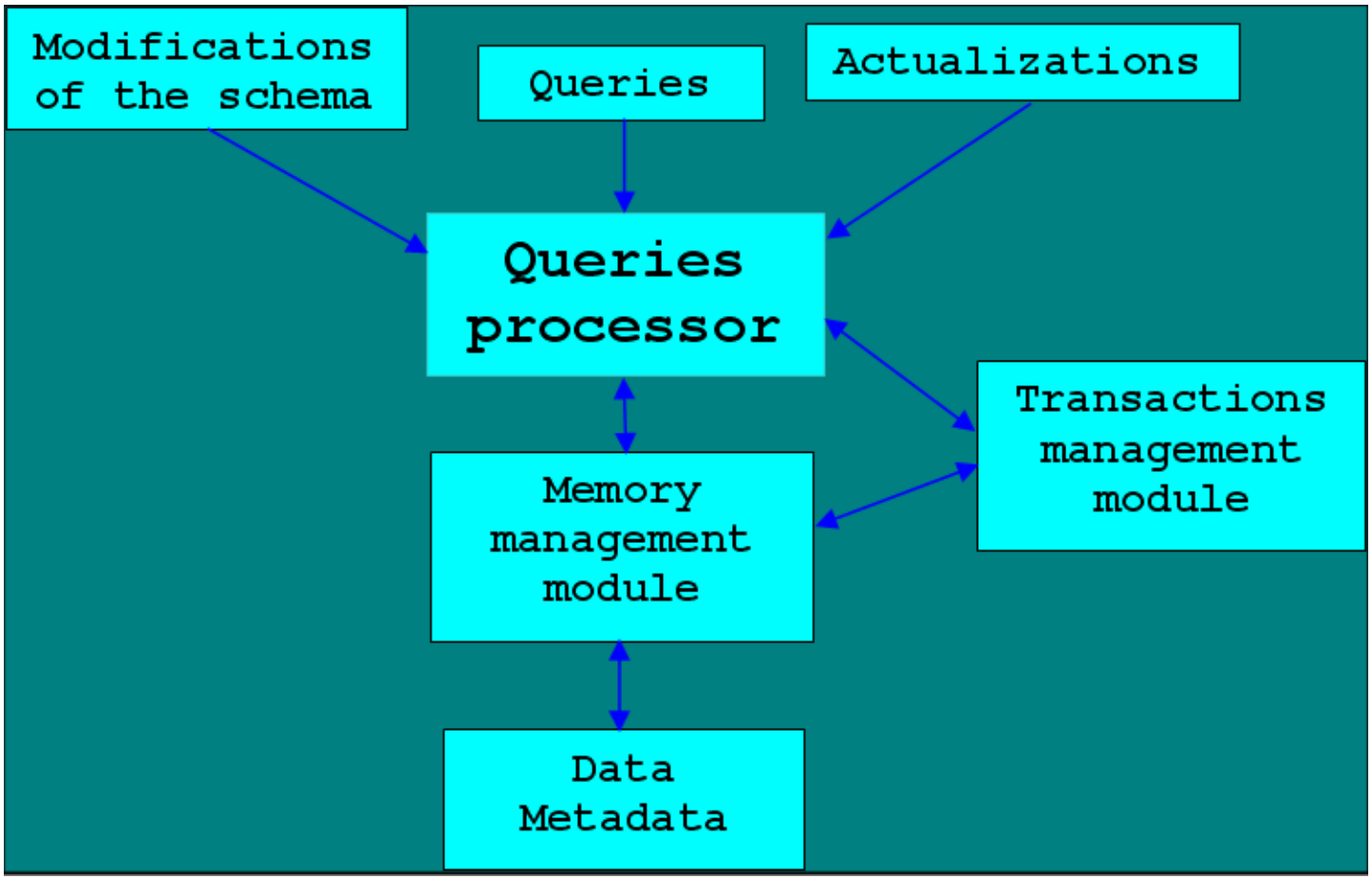
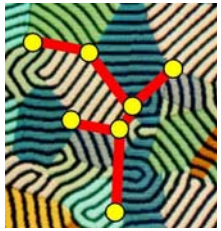


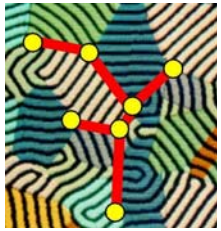
Fig.2. Modules of DBMS



Databases

The first professional data bases appeared on the market at the late sixties of the twentieth century. At the beginning there were the systems of files only. From the practical point of view the systems of files have some very important disadvantages:

- approach to them is essentially restricted,
- queries are not possible,
- the system of the data base (logical scheme of the data base) is simply the tree of files,
- there are no mechanisms to control the access of many users to the files at the same time (there are no ways to prohibit two users to modify the file simultaneously).



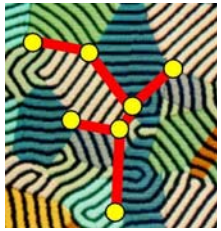
Databases

The early attempts to manage the problems relied on dividing the files into small pieces.

For example, if we wished to collect all the data from the Mendeleyev table of chemical elements, the good idea was not to put all the data into the one file but to store the information about any particular element into the separate file.
There were two types of data bases using the systems of files:

- hierarchical model (tree – type),
- network model (graph model),

and today they have the historical meaning only.

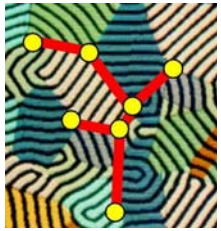


Databases and RDBS

The situation has changed radically, when **Ted Codd** published in 1970 his famous paper on Relational Data Base Systems (RDBS).
The fundamental objects of RDBS are relations (tables).

ElementId	Name	Symbol	AtomicNumber	Density	MeltTemp
1	Kalium	K	19	1.55	637
2	Scandium	Sc	21	2.99	1539
3	Titanium	Ti	22	4.50	1725
4	Vanadium	V	23	6.11	1710
5	Chromium	Cr	24	7.17	1875
6	Manganum	Mn	25	7.47	1244
7	Ferrum	Fe	26	7.87	1537

Table 1. Elements - example of the relation



Databases and SQL

There exists the special computer programming language to

operate with databases. Its name is

SQL – **S**equential **Q**uery **L**anguage.

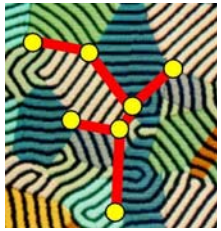
Below we have the first example how to use it.

Let us suppose we wish to now the melting temperatures of all elements. It is enough to write

SELECT Name, MeltTemp

FROM Elements;

Name	MeltTemp
Kalium	637
Scandium	1539
Titanium	1725
Vanadium	1710
Chromium	1875
Manganum	1244
Ferrum	1537

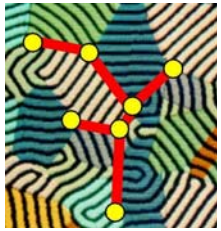


Databases and SQL

In the case we wish to know the melting temperature of the one element only, e.g. Chromium, we simply write

```
SELECT Name, MeltTemp
FROM Elements
WHERE Name= ’Chromium’;
```

Name	MeltTemp
Chromium	1875

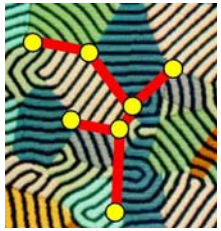


Databases and SQL

It is seen that SQL is very similar to a natural language (in a certain sense, of course).

The system, while executing these commands, has to:

1. check all the entities of the relation Elements given in the clause **FROM**;
2. find all the entities fulfilling the condition given in the clause **WHERE**;
3. formulate answer for all the attributes listed in the clause **SELECT**.



Databases and SQL

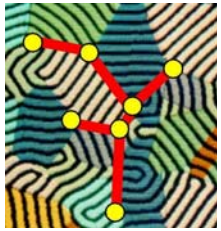
The system has of course to optimize its job:

it is extremely important for huge databases and for big and complex queries.

Big databases need a lot of computer memory. E.g. systems operating with millions of pictures form the Universe need petabytes

(1 petabyte = 1000 terabytes = 10^6 Gb)

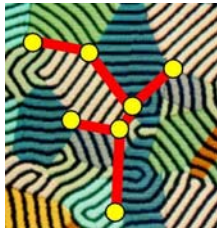
of the disc memory. To operate with such big memories, discs are organized in disc-matrix systems (so called second level memory). The next very important factor in the database systems is the access time to the needed item. In this case the parallel computing can be very helpful.



Databases and SQL

In the contemporary information systems the **client-server** architecture is frequently used. It means that the one process, called the client process, is send to another process, called server, and there is completed. DBMS are not the exceptions and are mostly accomplished in that architecture and are organized as one server and many clients.

In the simplest case only the query interface there is on the client side and the rest of the system there is on the server side. In the **Relational Data Base Systems** (**RDBS**) instructions in SQL are directed to the server, and the server sends to the client the answers like tables or forms. The restriction is the number of clients – in the case when there are a very big numbers of clients the danger of the bottleneck is very serious.



Relational Database Systems

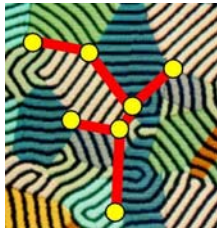
In the mathematical terminology the **database** (DB) it is the set of relations. **Relation** R it is a subset of the Cartesian product

$$R = \{(x, y) \in X \times Y \mid x \in X, y \in Y\}$$

A **table** is a representation of the relation in the RDBS. A table consists of

rows and columns. The following assumptions are made:

- number of columns is set to constant;
- every column has its name and domain;
- on the intersection of the column and the row there is a simple (atomic) value having the property of the column;
- the row represents one record of information, e.g., one chemical element;
- the order of rows and the order of columns are unimportant.



Relational Database Systems

Let us have a look at our example of the relation once again.

ElementId	Name	Symbol	AtomicNumber	Density	MeltTemp
1	Kalium	K	19	1.55	637
2	Scandium	Sc	21	2.99	1539
3	Titanium	Ti	22	4.50	1725
4	Vanadium	V	23	6.11	1710
5	Chromium	Cr	24	7.17	1875
6	Manganum	Mn	25	7.47	1244
7	Ferrum	Fe	26	7.87	1537

Table 1. Example of the relation Elements

Relational Database Systems

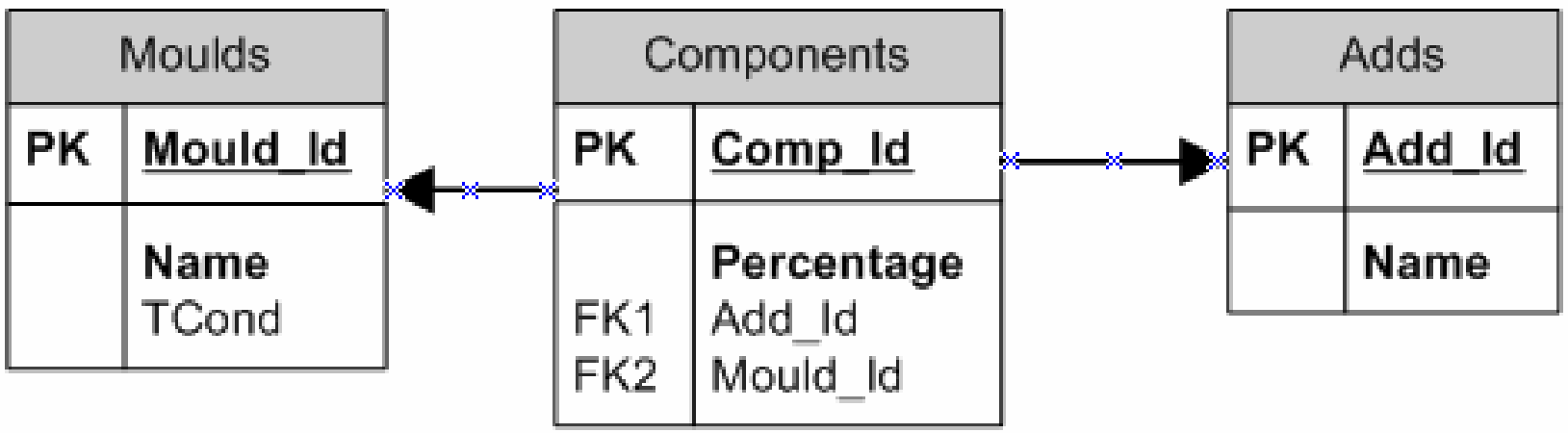


Fig. 3. Diagram of entities relations for the moulds database

Relational Database Systems

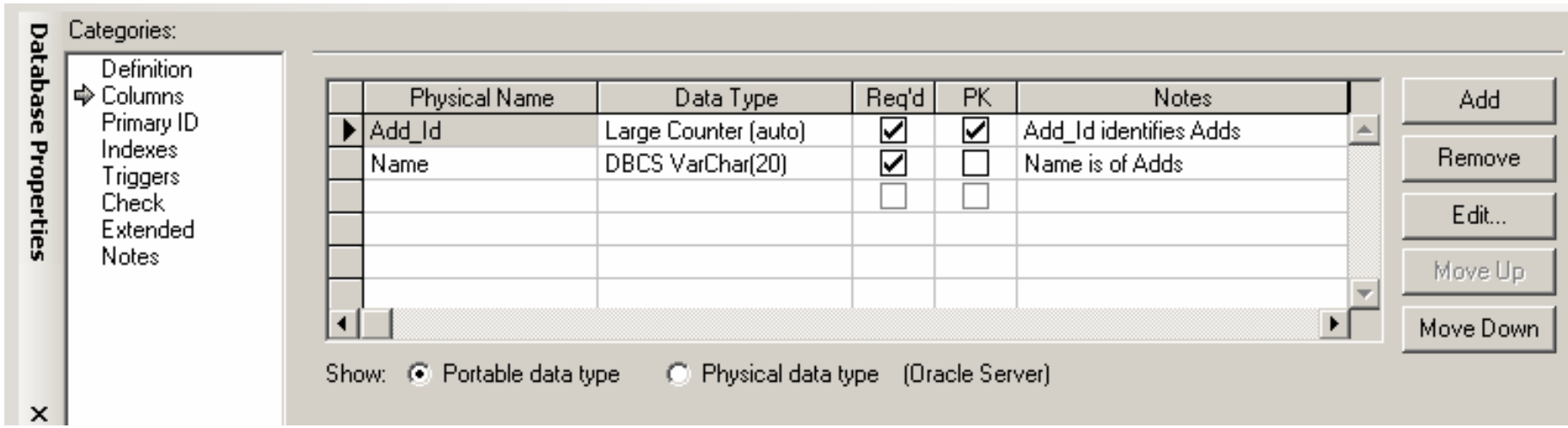


Fig. 4. Properties of the table **Adds**

Relational Database Systems

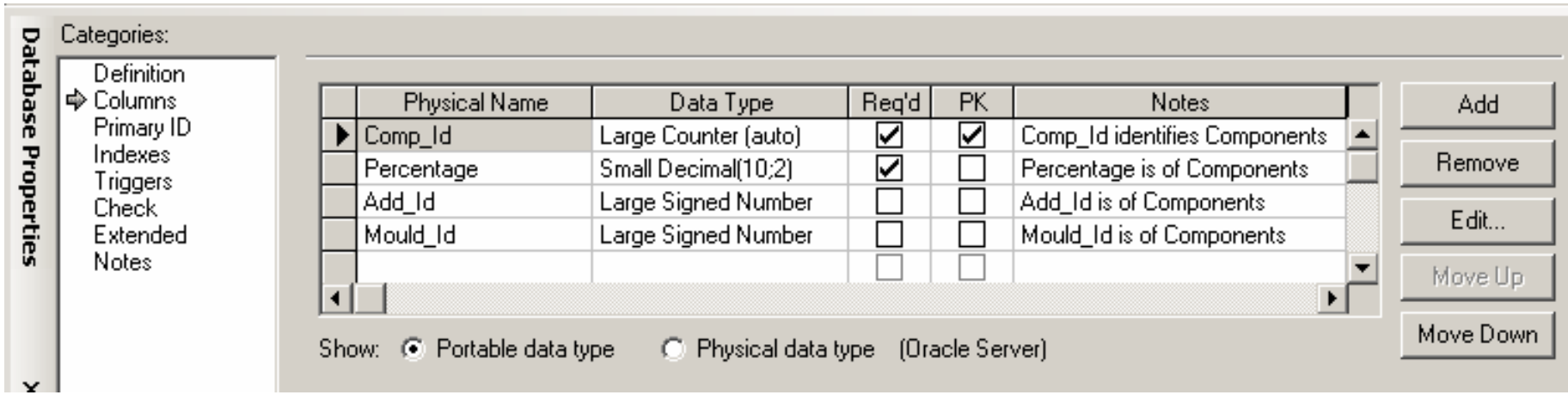


Fig. 5. Properties of the table **`Components`**

Relational Database Systems

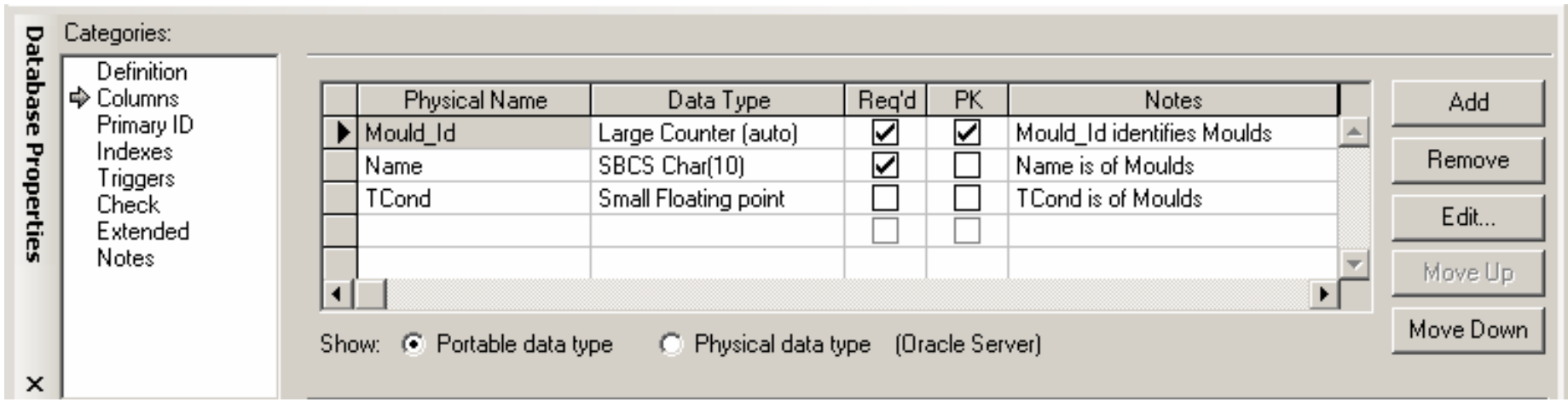
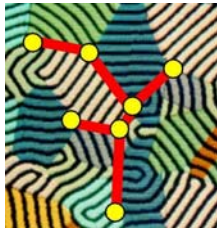


Fig. 6. Properties of the table **Moulds**



Relational Database Systems

The above screenshots are from the Microsoft Visio program with components especially designed to make projects of databases. It is seen that every attribute has a prescribed Data Type. The properties shown are the default values and sometimes should be improved. The Primary Keys are Counters and the corresponding Foreign Keys are Signed Numbers. It is not a reason that this attribute should be a signed number – number is enough. Primary key as a small counter is not enough for big data bases. It is also seen that the values of the Foreign Keys are not required. It means that here can be either empty places or the special value called NULL – value unknown. Null is different from zero or empty place. This value plays a very important role in the database design, and we show the logic tables only.

Relational Database Systems

AND	True	False	Null
True	True	False	Null
False	False	False	False
Null	Null	False	Null

Table 3. Conjunction operator **AND**

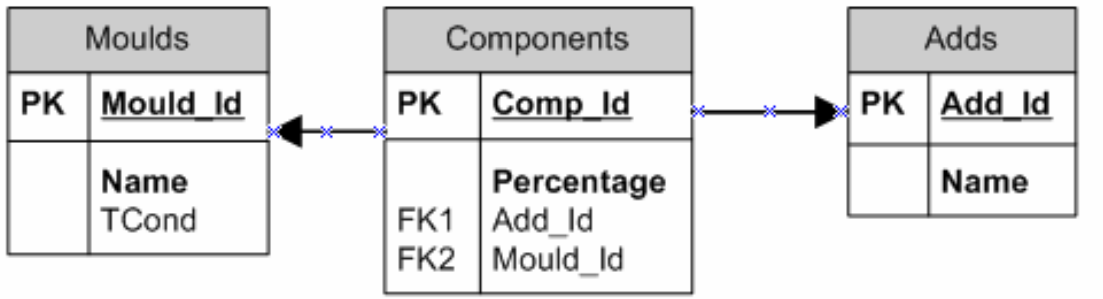
OR	True	False	Null
True	True	True	True
False	True	False	Null
Null	True	Null	Null

Table 2. Alternative operator **OR**

NOT	True	False	Null
	False	True	Null

Table 4. Negation operator **NOT**

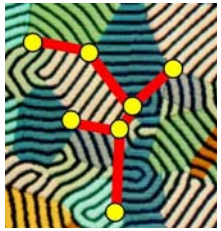
Relational Database Systems



Comp_Id	Percentage	Add_Id	CM_Id
1	8	1	1
2	16	1	2
3	24	1	3
4	10	3	5
5	15	3	6
6	20	2	6

Add_Id	Name
1	clay
2	graphite
3	bentonite

Mould_Id	Name	TCond
1	quartz	0,696
2	quartz	0,616
3	quartz	0,89
5	quartz	0,716
6	quartz	0,542



Relational Database Systems

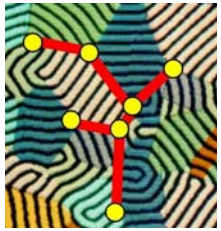
The tables can be coupled in the following way:

- one-to-one,
- one-to-many,
- many-to-one or
- many-to-many.

These couplings are realized by the direct joins of columns with two tables. In the relational data bases couplings many-to-many are not supported. The typical example of the many-to-many coupling is the so called phone problem: one person can have many phone numbers, and one phone number can belong to many people (e.g. in the office or in the family). This problem is solved with the help of the so called associative table, like in the example below.

Relational Database Systems

The table PersonsPhones is the connecting table of the tables Persons and PhoneNumbers. The primary key of table PersonsPhones consists of two foreign keys coming from the neighbor tables Persons and PhoneNumbers. In this way we have changed the one coupling many-to-many of tables Persons and PhoneNumbers into two one-to-many couplings: Persons with PersonsPhones and PhoneNumbers with PersonsPhones.



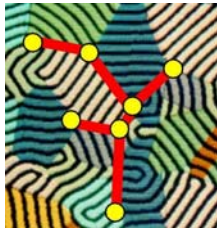
Relational Database Systems

- **Index**

  Index it is an additional data base structure allowing the very quick access to the records on the base of the values of the column. E.g. if the index is based on the column Name in the table Elements allows the rapid search of the names of the elements. It is similar to the index in the book.

- **Views**

  Usually the user of the data base is not allowed to see the original tables. The user can see the views instead. Views are the virtual tables. They are not saved in the data base, they are generated every time the proper query is formulated and executed.



Relational Database Systems

Levels of the relational data base

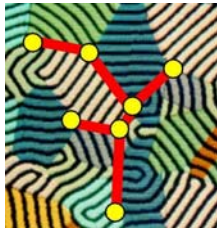
The same data base can be viewed in a different way with respect to the chosen level of abstraction. There are three fundamental levels of abstraction:

- user level – views and programs designed for the user,
- logical (conceptual) level – set of tables, views and indexes,
- physical level – set of files with data and indices.

Views are defined on the logical level and used on the user level.

Indices are defined on the logical level and used on the physical level.

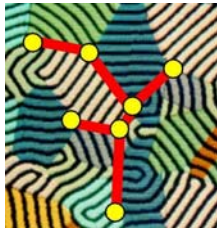
Tables are defined on the logical level and used on the physical and user levels.



Designing the database

The good project of the database should fulfill the following requirements:

- correctness of the model – model properly describes the reality;
- significance of the every element of the project for the functionality of the model;
- fullness of the model – the guarantee that no one element important for the functionality of the model has been omitted.



Relational Database Systems

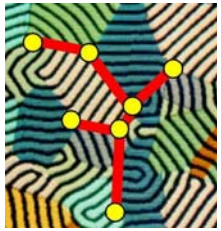
Normalization

The notion of the database normalization can be formulated in the following way:

Every item saved in the database should be expressed in the one way only.

It means in particular that:

- Information about the item should be stored in the one place only;
- Information that can be get from other information should not be stored additionally;
- If one item plays two roles in the database, then it can happen that the information can be stored in two places.



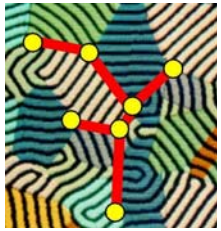
Relational Database Systems

Information should be stored in an atomic form, e.g. name of a person should be stored like:

Person_Id	FirstName	Name
1	John	Smith
2	Helen	Fox

and not like:

Person_Id	Name
1	John Smith
2	Helen Fox



Sequential Query Language

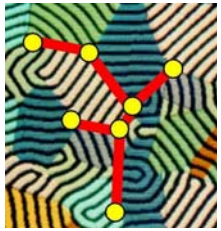
-- Create MOULDS database.

- create database MOULDS;
- create rollback segment MOULDS_r0 tablespace system ;
- alter rollback segment MOULDS_r0 online ;

-- Create Oracle exception file.

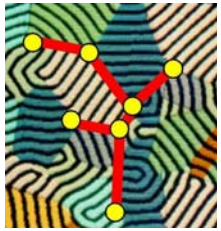
- create table exceptions (row_id rowid, owner varchar2(30),

  table_name varchar2(30), constraint varchar2(30)) ;



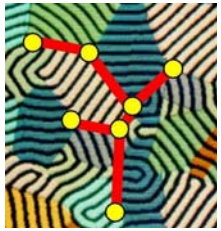
Sequential Query Language

```
--  Create new table COMPONENTS.
--  COMPONENTS : Table of Components
--  COMP_ID : Comp_Id identifies Components
--  PERCENTAGE : Percentage is of Components
--  ADD_ID : Add_Id is of Components
--  MOULD_ID : CM_Id is of Components

create table COMPONENTS (
    COMP_ID NUMBER(10,0) not null,
    PERCENTAGE NUMBER(10,2) not null,
    ADD_ID NUMBER(38,0) null,
    MOULD_ID NUMBER(38,0) null, constraint
    COMPONENTS_PK primary key (COMP_ID) );
```



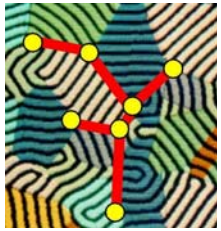
Sequential Query Language

```
-- Create new table ADDS.
-- ADDS : Table of Adds
--  ADD_ID : Add_Id identifies Adds
--  NAME : Name is of Adds

create table ADDS (
     ADD_ID NUMBER(10,0) not null,
     NAME VARCHAR2(20) not null, constraint
    ADDS_PK primary key (ADD_ID) );
```



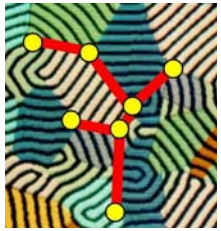
Sequential Query Language

```
-- Create new table MOULDS.
-- MOULDS : Table of CastingMoulds
-- MOULD_ID : CM_Id identifies CastingMoulds
-- NAME : Name is of CastingMoulds
-- TCOND : TCond is of CastingMoulds

create table MOULDS (
  MOULD_ID NUMBER(10,0) not null,
  NAME CHAR(10) not null,
  TCOND NUMBER null, constraint MOULDS_PK primary key
  (MOULD_ID) ) INITRANS 1 PCTFREE 10  PCTUSED 40 ;
```



Sequential Query Language

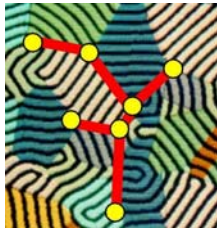
```
-- Add foreign key constraints to table COMPONENTS.
alter table COMPONENTS
    add constraint ADDS_COMPONENTS_FK1 foreign key (
        ADD_ID)
     references ADDS (ADD_ID);

alter table COMPONENTS
    add constraint CASTINGMOULDS_COMPONENTS_FK1 foreign key (
        CM_ID)
     references CASTINGMOULDS (CM_ID);

-- This is the end of the Visio Enterprise
    generated SQL DDL script.
```



Sequential Query Language

The most important command of SQL is the command **SELECT**. It consists of parts called clauses. It has the following syntax:

```
SELECT [Distinct] column_name, ...
FROM table_name, ...
[WHERE condition]
[ORDER BY {column_name|column_nr} {ASC|DESC}];
```



Sequential Query Language

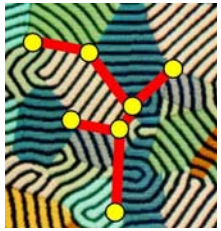
Examples

1. Write names of elements and their melting temperatures:

```
SELECT Name, MeltTemp
FROM Elements;
```

2. Write all information stored in the table Elements:

```
SELECT *
FROM Elements;
```

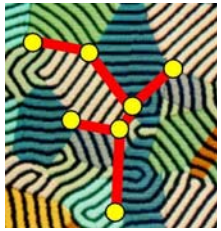


Sequential Query Language

3. Write names of elements with melting temperatures greater then 1500C:

```
SELECT Name, MeltTemp
FROM Elements
WHERE MeltTemp > 1500;
```

It is seen that * substitutes all the names of attributes in the table. Meta-symbol [ ] means the optional parameter. In the clause ORDER BY we can sort lexicographical resulting rows either by the name of the column or by the number of the column_name in the SELECT clause. The sorting can be in the ascending or in the descending order. The operator Distinct in the SELECT clause allows writing the items without repeating them.

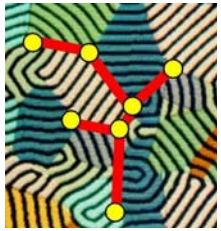


Sequential Query Language

4. Write the names of elements and their melting temperatures in the following form:

The melting temperature of the element element_name is melt_temp

```
SELECT
"The melting temperature of " & Name & " is " & MeltTemp AS
[Melting temperatures]
FROM Elements;
```



Sequential Query Language

The symbol **&** is the concatenation operator and the strings should be enclosed in quotation marks. The alias of the header is enclosed in the square brackets because it consists of several words. The result looks like follows:

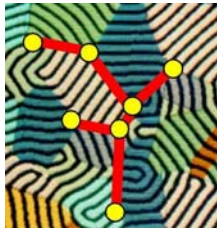
```
Melting temperatures
The melting temperature of Kalium is 637
The melting temperature of Scandium is 1539
The melting temperature of Titanium is 1725
The melting temperature of Vanadium is 1710
.....................................
```

Sequential Query Language

5. Write the names of elements for which the density is unknown:

```
SELECT Name
FROM Elements
WHERE Density IS NULL;
```

There exists also the corresponding operator IS NOT NULL.



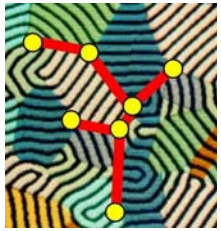
Sequential Query Language

6. Write the names of elements for which the density is in the range from 2 to 7:
(The answer can have two forms!)

```
SELECT Name
FROM Elements
WHERE Density > 2 AND Density < 7;
```

```
SELECT Name
FROM Elements
WHERE Density BETWEEN 2 AND 7;
```

(Which form looks better?)



Fundamentals of relational databases

Literature

1. Ullman J.D., Principles in Database and Knowledge-Base Systems, vol. I, Computer Science Press, New York, 1988
2. Ullman J.D., Principles in Database and Knowledge-Base Systems, vol. II, Computer Science Press, New York, 1989
3. Bowman J., Emerson S.L., Darnovsky M., The Practical SQL, Addison-Wesley, Reading, 1998
4. Beynon-Davies P., Database Systems, MacMillan Press Ltd., 1996