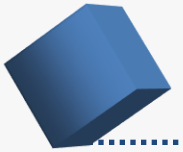
Organization of Data bases

semester 4

lector: Lyudmyla Rozova, associate professor, phd

email: lyudmyla.rozova@khpi.edu.ua

Lecture 2.



Relational Data Model

Relational Data Model

Mathematician E.F. Codd first formulated the basic concepts and limitations of the **relational model** in 1970.

The goals were formulated as follows:

- To provide a higher degree of independence from data. Application programs should be independent of changes to the internal representation of data. Such as changes in file organization, reordering of records and access paths;
- To create a solid foundation for solving semantic issues, as well as problems of **data consistency and redundancy**. Codd presented the concept of **normalized relations** - relations without repeating groups;
- extension of data management languages by including operations on mathematical **sets**.

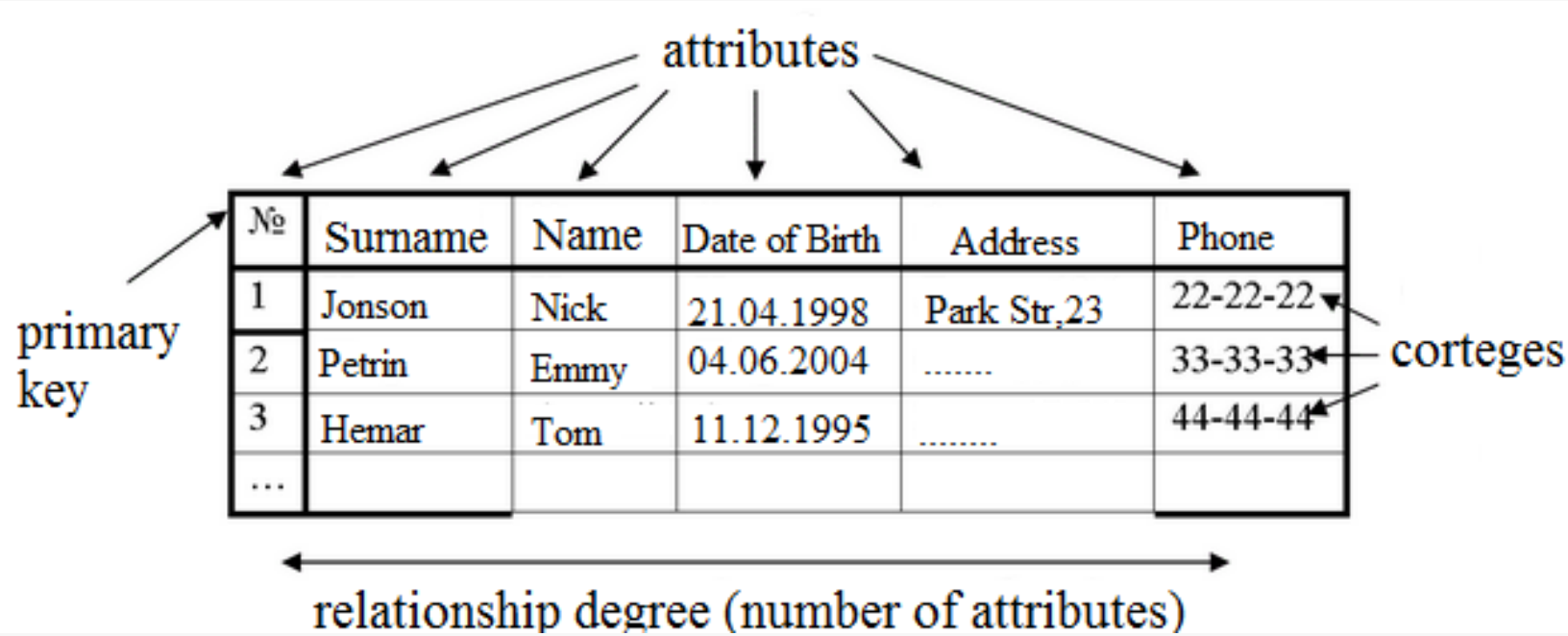
Commercial systems based on the relational data model began to appear in the late 70s and early 80s.

Due to the popularity of the relational model, many non-relational systems now provide a relational user interface.

Later, some extensions of the relational data model were proposed, used for the most complete and accurate expression data meaning, to support object-oriented, and also to support deductive capabilities.

1.Relational data objects

The **relational model** is based on the mathematical concept of a ***relation***, the physical representation of ***relation*** is ***table***.
The fact is that Codd, was experienced mathematician, so he used the mathematical terminology, especially from set' theory and predicate logic.



№	Surname	Name	Date of Birth	Address	Phone
1	Jonson	Nick	21.04.1998	Park Str,23	22-22-22
2	Petrin	Emmy	04.06.2004	.....	33-33-33
3	Hemar	Tom	11.12.1995	.....	44-44-44
…					

The **relation (table)** is characterized by the following concepts:
The attribute corresponds to the column of this table, to the properties of the objects, information about which is stored in it. In specific DBMSs, attributes are often called fields.

1.Relational data objects

In the ***relational model***, relations are used to store information about the objects represented in the database.

Relation - table

Attribute - Field (columns)

Cortege – record (rows)

The attributes can be located in any order, regardless of their reordering, the relation will remain the same, and therefore have the same meaning.

The elements of a relations are **corteges,** or rows of tables. A **cortege** is a relation string.

Corteges can be in any order, and the relation will remain the same, and therefore have the same meaning. In DBMSs, **Cortege**s are called records. The degree of a relation is the number of its attributes.

A **domain** is a general collection of values from which specific values for a specific attribute are taken.

2.Domains

This is a semantic concept.

Domains can be more accurately defined as **set of scalar values** of the same type. These scalar values are called ***scalars***.

In fact, it is the smallest semantic unit of data.

Scalars have ***no internal*** *structure*, they are ***not decomposable*** in the given relational model.

For example, if there is an attribute (object property) **"full name"**, it provides scalars containing the **last name, first name**. Of course, these scalars can still be broken down into letters, but then the desired meaning will be lost. That is, for this model, the smallest semantic data units will be exactly the surname, name.

The attribute values are taken from the domains. In practice, domains are often not described, but specified by the type, format, and other properties of data.

Each attribute must be defined on a ***single domain***.

The main purpose of domains is to limit the comparison of attributes that are different in meaning. A domain can be viewed as a subset of the values of some data type that have a specific meaning.

3.Relations

Relation variable is an ordinary variable whose value can change over time (in fact, it is a **set of specified attributes** of a given relation).

Relation value is the value of a relation variable at a ***particular point in time*** (in fact, these are stored relation corteges).

Let's give a more precise, formal, definition of the relation.

The relation R_1, defined on the set of domains D_1, D_2, …, D_n (not necessarily different), consists of two parts: a header and a body:

- **The header of the relation** contains a fixed set of pairs $\{A_i: D_i\}$,

where A_i is the name of the attribute, D_i is the name of the domain.

- **The body of a relation** contains many corteges of a relation

Each cortege of a relation is a set of pairs of the form <**Attribute_Name: Attribute_Value**>: (<A1: Val1>, <A2: Val2>,…, <An: Valn>) such the value Val_i value of the attribute A_i belongs to the domain D_i.

The relation is usually written as: ***R (<A1: D1>, <A2: D2>,…, <An: Dn>) or R (A1, A2,…, An). RelationName (AttributeName1, AttributeName2, …, AttributeNameN)***

For example, *Relation header*: {Surname - text, Name-text, Date of Birth - data, Address - text, Phone - integer}

Relation body: {Surname - Jonson, Name- Nick, Date of Birth - 04.21.1998, Address - Park Str, 23, Phone - 111111}

4.Properties of relations

1) ***There are no identical corteges***.
This follows from the fact that the body of a relation is defined as a mathematical set of corteges, which does not contain identical elements by definition. →a consequence of this property is that there is always a ***primary key in relation.***

2) C***orteges*** **are unordered.**
This also follows from the fact that relation body is defined as a mathematical set of corteges. By definition, mathematical set is not ordered. That is why there are no such concepts as “next”, “previous”, “second tuple”, etc. in relation.

3) ***Attributes are not ordered.***
This follows from the fact that the title of the relations is defined as a mathematical set of attributes. And the set is not ordered by definition. Those, again there are no concepts of "first attribute", "next attribute", etc.

4) ***All attribute values are indivisible (atomic).***
This is a consequence of the fact that each attribute is defined on its own domain, and the domain is a set of indivisible scalars.

5.Relational data integrity

At any time, any database contains some specific attribute values that express a specific state of a real world object.

→Consequently, the **database needs to define the integrity rules** necessary to ensure that the ***data does not conflict with the real world.***

These integrity rules are specific to each database. This is, in fact, informing the DBMS about the limitations of the real world.

For example, the name is only a text value, the values of weight, height are only positive, etc. But such integrity rules are not enough. It is more important that the data inside the database ***does not contradict*** each other.

For example, it was accidentally indicated in the database that Nick Jonson was born on 04.21.1998 and 05.01.2000, etc.

To prevent such situations, there are general rules for the integrity of relational data. These rules are related to **primary** and **foreign keys**.

6. Potential, primary, secondary keys

Potential key of relations is a single attribute (simple key) or a set of attributes (composite key) of a relation that has **uniqueness** and **non-redundancy** properties.

Each cortege is accessed by the value of the *potential key*.

Let ***R*** - relation, ***K*** - the potential key for ***R***

K is a subset of the set of attributes ***R,*** for which the following properties are satisfied:

1) **Uniqueness -** there are no two different corteges in the current value of the relation variable ***R*** with the same ***K*** values;

2) ***non-redundancy*** - no subset of ***K*** possesses the uniqueness property.

Potential keys are intended to provide a basic corteges-level addressing mechanism, the value of a potential key can be used to uniquely find a cortege. In Microsoft Access, potential keys are also called **indexed fields** (they are set to *Yes* in the *Indexed Field property*).

6. Potential, primary, secondary keys

- The relation can have ***multiple potential keys***, but one of them must be selected as the ***primary key***. In Access, for the primary key, the value of the Indexed Field property is set to Yes (no matches allowed).
- Any relation must have a ***primary key***, →so there must be at least one ***potential key***. It can be one field, or it can be a set of several or even all fields of the relation

If there are no natural potential keys in the relation or they are inconvenient for use, then artificial keys are created, for example, *the number of something.*

When choosing a primary key, it is recommended to select an attribute whose value does not change during the entire lifetime of the instance (in this case, the personnel number is preferable to the surname, since it can be changed by marriage).

- **Secondary keys** are created for the fields that are often used in searching and sorting data: they will help the system to find the required data much faster. Unlike primary keys, fields for indexes (secondary keys) can contain non-unique values.

Example, database “Organization”

In our example, database “Organization” according to the logical modeling, at the first step, it is proposed to store data in one relation with the following attributes:

EMPLOYEES_DEPARTMENTS_PROJECTS (NEmp, Surname, NDep, TEL, NPr, Project, NTask), where

NEmp - employee personnel number,

Surname - employee surname ,

NDep – the number of the department in which the employee is listed,

TEL – department phone,

NPr – the number of the project the employee is working on,

Project – the name of the project the employee is working on

NTask – the number of the job the employee is working on.

Each employee in each project performs exactly one task, so a pair of attributes **{NEmp, NPr}** should be selected as a ***potential relation key***. It is customary to highlight the attributes included in a potential key with an underscore.

7. Foreign key

The foreign key exists to ensure data consistency in database.

Primary keys are used to establish relations between tables in a relational database. In this case, the **primary key** of one table (parent) corresponds to the **foreign key** of another table (child).

A foreign key contains the values of its associated *primary key* field. Foreign key values can be non-unique, but must not be empty. The primary and foreign keys must be of the same type.

A number of other terms are associated with the concept of a foreign key.

We can say that the foreign key value is a reference to a cortege containing the corresponding potential key value. This cortege is called the **reference (target) cortege**.

A relation that contains a foreign key is called a referencing relation.

There is one of the basic rules of integrity associated with foreign keys:

Referential integrity rule:

The database should not contain inconsistent foreign key values (inconsistent values are those values that are not present for a potential key in a reference relationship). In essence, this rule is equivalent to a foreign key definition.

8. Foreign key rules

The referential integrity rule answer the question “How to avoid temporary incorrect situations that may arise when updating data in the database? “
The easiest way is to prohibit any operations that violate the referential integrity rule. But when updating the database, a temporary integrity violation cannot be avoided.

Therefore, there are compensating operations:

- ✓ 1)What to do in the action when trying ***to delete a foreign key reference object***?

(For example, remove all data about a *Department*).
There are at least two possible answers to this question:
a**) restrict,** - don't delete until the user deletes the referencing corteges, postpone deletion;
b) **cascade,** - delete, removing all matching referencing corteges.

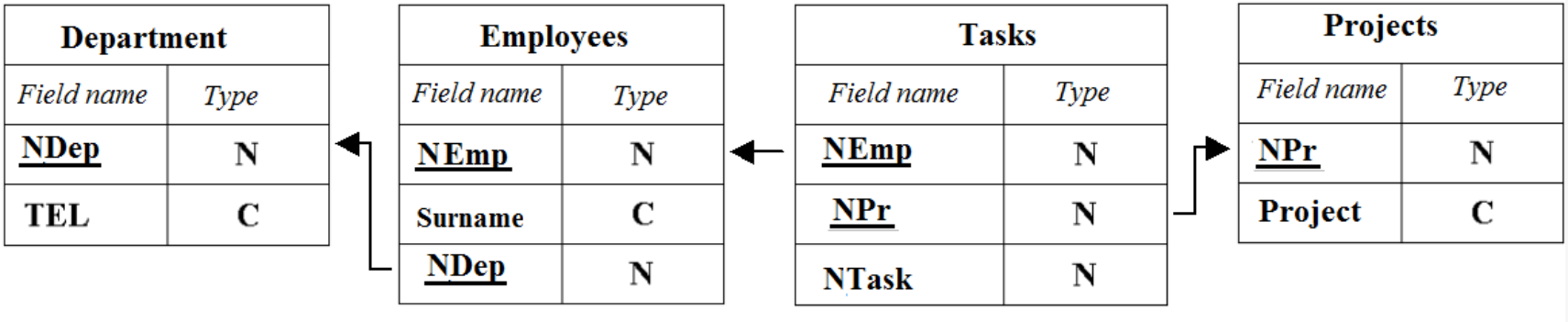
- ✓ 2) What to do when you try ***to change (update) the value of the potential key*** that is referenced? (For example, replace the names of departments in the Departments table). There are also two options:

a) **restrict,** - defer until the values of the referencing corteges are removed;
b) **cascade,** - update in all referencing corteges.

8. Foreign key rules

The choice of answers to these two questions is the task (or definition) of the foreign key rules.
In MS Access, the definition of foreign key rules is carried out when creating relations between tables:
if the ***Cascading update of related fields*** option is checked, then the cascading operation is selected for updating; if not, it is a restriction; similar to the parameter ***Cascade delete related records***

The scheme of database "ORGANIZATION"



Department	
Field name	*Type*
NDep	N
TEL	C

Employees	
Field name	*Type*
NEmp	N
Surname	C
NDep	N

Tasks	
Field name	*Type*
NEmp	N
NPr	N
NTask	N

Projects	
Field name	*Type*
NPr	N
Project	C

9.Null values in data

Data integrity complications can be caused by undefined or missing values.

For example, the author of the painting is not known in the art database; in the DB School some children are orphans (no parents), etc.

To solve the problem of missing values, Codd proposed to introduce special labels, which he called **Null-values,** which he defined as follows:

- if a given cortege has a **Null-value** of a given attribute, then this means that the attribute value is absent in it. This is not the same as a numeric 0 or a space, it is not a value at all, but only a label.

Most modern relational DBMSs support Null values. Access supports Null values.

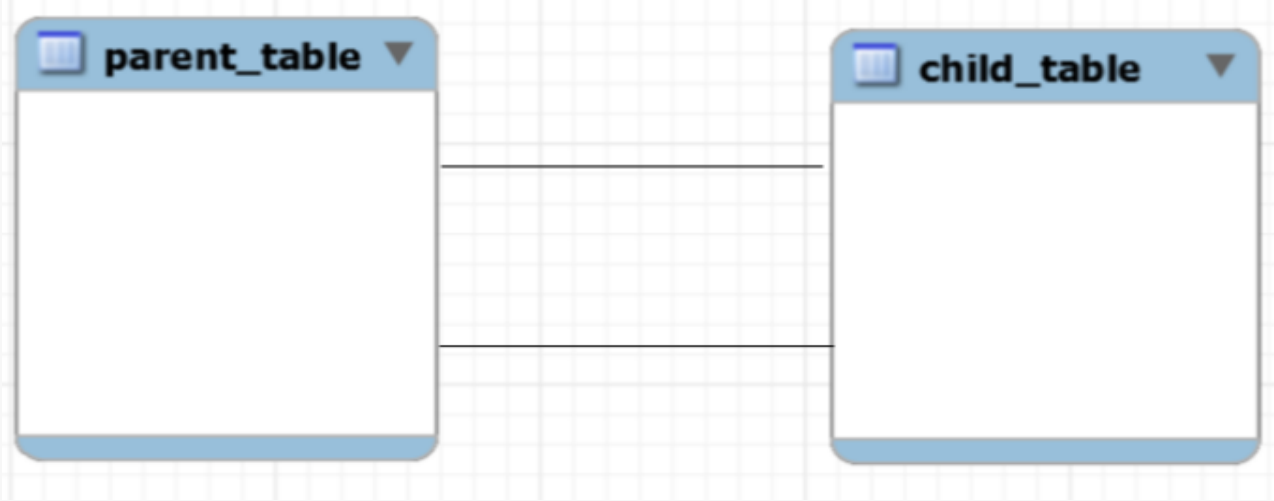
There is a second integrity rule associated with Null values.

Object Integrity Rule: No element of the primary key of the underlying relationship can be Null.

10.Connections between tables

There are three types of connections between tables.

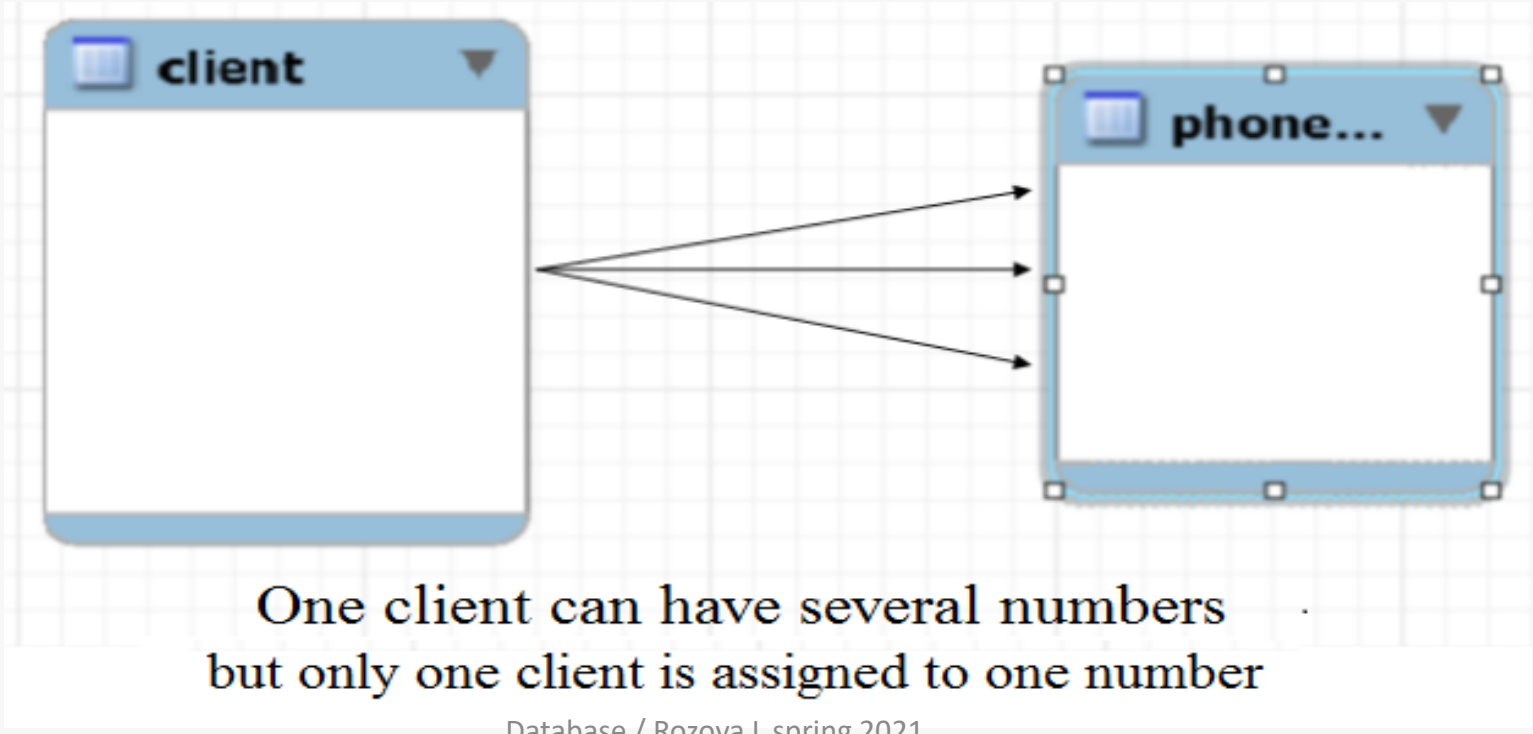
One to one - each record of the parent table is linked to only one record of the child. Such connection is rare in practice. It is implemented by defining a unique foreign key. A **one-to-one** connection is used if you don't want the table to swell up with a large number of fields. Databases containing tables with such connection cannot be considered fully normalized. More often, such tables are combined.



10.Connections between tables

One-to-many — Each record in the parent table is linked to one or more records in the child.

It is implemented when object **A** can own or correspond to several objects **B**, but object **B** can only correspond to one object **A**. One-to-many connection is the most common for relational databases.



10.Connections between tables

Many-to-many - multiple records in one table are linked to multiple records in another.

In the case of such a connection, it is generally impossible to determine which record of one table corresponds to the selected record of another table. It makes impossible to create a physical (at the level of indexes and triggers) implementation of such a connection between the corresponding tables.

Therefore, before moving to the physical model, all **many-to-many** connections must be redefined (some DBMS tools do this automatically). A similar connection between the two tables is achieved by creating a third table and implementing a one-to-many connection for each of the existing tables with the intermediate table