Information Technologies

Lectures 30 hours

Labs 30 hours

Exam

Lecturers: Assoc. Prof. Mariana Goranova, PhD
Assoc. Prof. Rajna Pavlova, PhD
Department of Programming and Computer Technologies
Technical University of Sofia
Room: 2304
E-mail: mgor@tu-sofia.bg

URL: pct.tu-sofia.bg/Moodle001/
Username: student Password: pktt

Database Systems

Bibliography

1. C. Y. Date. An Introduction to Database Systems, Addison-Wesley, Philippines, 1987.
2. Jeffrey D. Ullman, Principles of Database and Knowledge Base Systems, Volume I, Computer Science Press, 1988.
3. John C. Shepherd, Database Management Theory and Application, Richard Irwin, Inc., Boston, 1990.
4. Thomas Connolly, Carolyn Begg and Anne Strachan, Database Systems, A Practical Approach to Design, Implementation and Management, Addison-Wesley Publishing Company, Inc., 1996.
5. Евлоги Георгиев, Научете сами SQL, Ръководство за работа с бази данни, част I и II, Express Design, София, 1998.

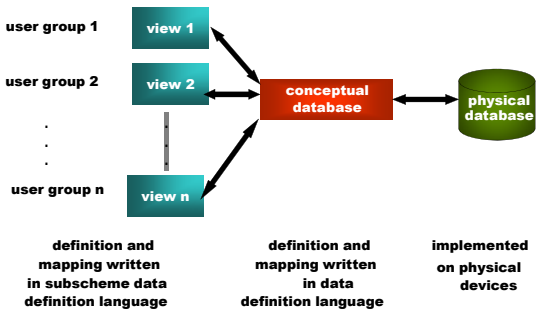
A **database system** is an important type of programming system, used today on the biggest and the smallest computers. There are two qualities that distinguish database systems from other sorts of programming systems:

1. The ability to manage persistent data – there is a **database** which exists permanently and the system accesses and manages the contents of this database.
2. The ability to access large amounts of data **efficiently** – the file system also manages persistent data, but does not provide fast access to the data; simple access techniques such as linear scans of the data, are usually not adequate.

Other capabilities of database systems:

1. Support for at least one **data model** – mathematical abstraction through which the user can view the data.
2. Support for certain **high-level languages** that allow the user to define the structure of data, access data, and manipulate data.
3. **Transaction management**, the capability to provide correct, concurred access to the database by many users at once.
4. **Access control**, the ability to limit access to data by unauthorized users, and the ability to check the validity of data.
5. **Flexibility**, the ability to recover from system failures without losing data.

Levels of Abstraction in a Database System



• The Physical Database Level

A collection of files and the indices or other storage structures used to access them efficiently is termed a physical database. The physical database resides permanently on secondary storage devices, such as disks, and many different physical databases can be managed by the same database management system software.

• The Conceptual Database Level

The conceptual database is an abstraction of the real world as it pertains to the users of the database. A database system provides a data definition language, or DDL, to describe the conceptual scheme and the implementation of the conceptual scheme by the physical scheme (in terms of a "data model").

The conceptual database is intended to be a unified whole, including all the data used by a single organization. The advent of database management systems allowed an enterprise to bring all its fields of information together.

The fundamental data models are:

- **an entity-relationship model**
- **relational data model**
- **network data model**
- **hierarchical data model**
- **object-oriented data model.**

• The View Level

A view or subscheme is a portion of the conceptual database or an abstraction of part of the conceptual database. Most systems provide a facility for declaring views, called a subscheme data definition language and a facility for expressing queries and operations on the views, which would be called a subscheme data manipulation language.

In a sense, the construction of views is the inverse of the process of database integration; for each collection of data that contributed to the conceptual database, we may construct a view containing just that data. Views are also important for enforcing security in a database system, allowing subsets of the data to be seen only by those users with a need or privilege to see it.

Example: We examine arrays – an analogy from the programming languages world.

1. **Conceptual level – we might describe an array by a declaration such as**

```
#define N 100
#define M 50
int A[N][M];
```

2. **Physical level – we might see the array A as stored in a block of consecutive storage locations, by the rule:**
   A[i][j] is in location $a_o+size_{element}(Mi+j)$
3. **A view of the array A might be formed by declaring a function f(i) to be the sum from j=0 to M-1 of A[i][j]. In this view, we not only see A in a related but different form, as a function rather than an array, but we have obscured some of the information, since we can only see the sums of rows, rather than the rows themselves.**

Schemes and Instances

When the database is designed, we are interested in plans for the database; when it is used, we are concerned with the actual data present in the database.

The current contents of a database we term as instance of the database (or extension of the database or database state).

Plans for a database tell us of the types of entities that the database deals with, the relationship among these types of entities, and the ways in which the entities and relationships at one level of abstraction are expressed at the next lower (more concrete) level.

The term scheme is used to refer to plans, so we talk of a conceptual scheme as the plan for the conceptual database, and we call the physical database plan a physical scheme. The plan for the view is often referred to simply as a subscheme (or intention).

Example: We can continue with the array analogy. The description of arrays and functions given in that example was really schema information.

1. **Physical scheme is the statement, that the array A is stored beginning at location** a_o**, and that A[i][j] appears in word**
   $a_o+size_{element}(Mi+j)$**.**
2. **Conceptual scheme is the declaration**
   int A[N][M].
3. **Subscheme is the definition of the function f(i), that is,**

$$f(i)=\sum_{j=0}^{M-1} A[i][j]$$

As an example of an instance of this conceptual scheme, we could let n=m=3 and let A be the "magic square" matrix:

8 1 6
3 5 7
4 9 2

Then, the physical instance would be the nine words starting at location , containing, in order,

8 1 6 3 5 7 4 9 2.

Finally, the view instance would be the function

f(1)=f(2)=f(3)=15.

Data Independence

The chain of abstraction, from view to conceptual to physical database, provides two levels of "data independence". Most obviously, in a well-designed database system the physical scheme can be changed without altering the conceptual scheme or requiring a redefinition of subschemes. This independence is referred to as physical data independence. It implies that modifications to the physical database organization may affect the efficiency of application programs, but it will never be required that we rewrite those programs just because the implementation of the conceptual scheme by the physical scheme has changed.

An illustration, references to the array A should work correctly whether the physical implementation of arrays is row-major (row-by-row, as in the examples) or column-major (column-by-column).

The relationship between views and the conceptual database also provides a type of independence called logical data independence. As the database is used, it may become necessary to modify the conceptual scheme, for example, by adding information about different types of entities or extra information about existing entities. Many modifications to the conceptual scheme can be made without affecting existing subschemes, and other modifications to the conceptual scheme can be made if we redefine the mapping from the subscheme to the conceptual scheme.

Again, no change to the application programs is necessary. The only kind of change in the conceptual scheme that could not be reflected in a redefinition of a subscheme in terms of the conceptual scheme is the deletion of information that corresponds to information present in the subscheme. Such changes would naturally require rewriting or discarding some application programs.

Data Models

A data model is a mathematical formalism with two parts:
1. A notation for describing data, and
2. A set of operations used to manipulate that data.

The Entity-Relationship Model

The purpose of the entity-relationship model is to allow the description of the conceptual scheme of an enterprise to be written down without the attention to efficiency or physical database design. The entity-relationship diagram will be turned later into a conceptual scheme in one of the other models, e.g., the relational model.

Entity

An entity is a thing that exists and is distinguishable; that is, we can tell one entity from another. For example, each person is an entity, and each automobile is an entity.

Entity Set

A group consisting of all "similar" entities forms an entity set. Examples of entity sets are: 1) All persons; 2) All red-haired persons; 3) All students. Entity set is the current subset of all members of a given entity set that are present in the database. Example: The Technical University of Sofia may design its database scheme to have an entity set STUDENTS. In the current instance of that entity set are all students presently studied in TU, not all students in Bulgaria or all the students that could ever exist.

Attributes and Keys

Attributes are entity sets properties, which associate with each entity in the set a value from a **domain** of values for that attribute. The domain for an attribute is a set of integers, real numbers, or character strings.

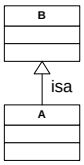
Example: The entity set of students may be declared to have attributes such as name (a character string), faculty number (an integer number), and so on.

An attribute or set of attributes whose values uniquely identify each entity in an entity set is called a **key** for that entity set. Often an arbitrary serial number is supplied as an attribute to serve as a key.

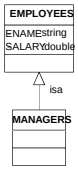
Example: An entity set that included only students from the TU could use the single attribute "faculty number" as a key. However, suppose we wished to identify uniquely members of an entity set including students of many universities. We could not be sure that two universities do not use the same faculty numbers, so an appropriate key would be the pair of attributes **FACULTY_NO** and **UNIVERSITY**.

Isa Hierarchies

We say A isa B, read "A is a B", if entity set B is a generalization of entity set A, or equivalently, A is a special kind of B. The primary purpose of declaring isa relationship between entity sets A and B is so A can inherit the attributes of B, but also have some additional attributes. Each entity a in set A is related to exactly one entity b in set B, such that a and b are really the same entity. No b in B can be so related to two different members of A, but some members of B can be related to no member of A. The key attributes for entity set A are actually attributes of entity set B, and the values of those attributes for an entity a in A are taken from the corresponding b in B.



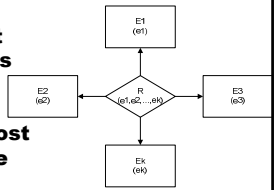
Example: A supermarket might well have an entity set EMPLOYEES, with attributes such as ENAME, and SALARY. A few employees are managers. Since managers are employees, we have an **isa** relationship from MANAGERS to EMPLOYEES. To access the salary or name of a manager, we follow the **isa** relationship to the employee entity that the manager is, and find that information in the attributes SALARY and ENAME of EMPLOYEES.



Relationships

A **relationship** among entity sets is an ordered list of entity sets. A particular entity set may appear more that once on the list. If there is a relationship R among entity sets E_1, E_2, ..., E_k, then the current instance of R is a set of k-tuples. We call such a set a **relationship set**.

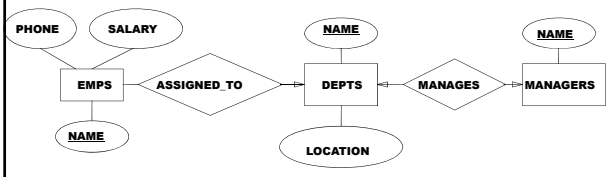
Each k-tuple $(e_1, e_2, \ldots, e_k)$ in relationship set R implies that entities e_1, e_2, ..., e_k, where e_1 is in set E_1, e_2 is in set E_2 and so on, stand in relationship R to each other as a group. The most common case, by far, is where k=2, but lists of three or more entity sets are sometimes related.



Entity-Relationship Diagram

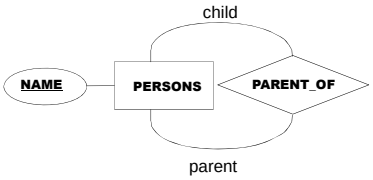
1. **Rectangles** represent entity sets.
2. **Circles** represent attributes. They are linked to their entity sets by edges. Sometimes, attributes that are part of the key for their entity set will be underlined.
3. **Diamonds** represent relationship. They are linked to their constituent entity sets by edges, which can be **undirected edges** or **directed edges** (arcs).

Example: A simple entity-relationship diagram has three entity sets, EMPS, DEPS, and MANAGERS. The first two are related by relationship ASSIGNED_TO and the second and third are related by MANAGES. We show the attributes, NAME, PHONE, and SALARY for EMPS; NAME is taken to be the key. Departments have attributes NAME (of the departments) and LOCATION, while MANAGERS has only the attribute NAME.



a)

Example: Suppose we have an entity set PERSONS and we have a relationship PARENT_OF, whose list on entity sets is PERSONS, PERSONS. The relationship set for relationship PARENT_OF consists of all and only those pairs (p1, p2) such that person p2 is the parent of person p1.
We notice two edges from PARENT_OF to PERSONS; the first represents the child and the second – the parent.



b)

Example: Suppose we have an entity set PERSONS and we have a relationship MOTHER_OF, whose list on entity sets is PERSONS, PERSONS. The relationship set for relationship MOTHER_OF consists of all and only those pairs (p1, p2) such that person p2 is the mother of person p1.

An alternative way of representing this information is to postulate the existence of entity set MOTHERS and relationship MOTHERS isa PERSONS. This arrangement would be more appropriate if the database stored values for attributes of mothers that it did not store for persons in general. Then the relationship MOTHER_OF would be the list of entity sets PERSONS, MOTHERS. To get information about a person's mother as a person, we would compose the relationship MOTHER_OF and isa.

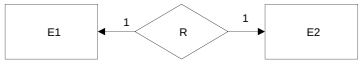
Functionality of Relationships

We classify relationships according to how many entities from one entity set can be associated with how many entities of another entity set.

- **One-to-One Relationship**
- **Many-One Relationships**
- **Many-Many Relationships**

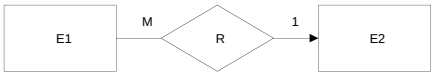
• One-to-One Relationship

One-to-one relationship is the simplest and rarest form of relationship on two sets – for each entity in either set there is at most one associated member of the other set. For example (a), the relationship MANAGES between DEPTS and MANAGERS might be declared a one-to-one relationship. In the database we never can find more than one manager for a department, nor can one person manage two or more departments. It is possible that some department has no manager at the moment, or even that someone listed on the database as a manager currently has no department to manage.



• Many-One Relationships

In a many-one relationship, one entity in set E_2 is associated with zero or more entities in set E_1, but each entity in E_1 is associated with at most one entity in E_2. This relationship is said to be many-one from E_1 to E_2. For example, the relationship between EMPS and DEPTS in (a) may well be many-one from EMPS to DEPTS, meaning that every employee is assigned to at most one department. It is possible that some employees, such as the company president, are assigned to no department.

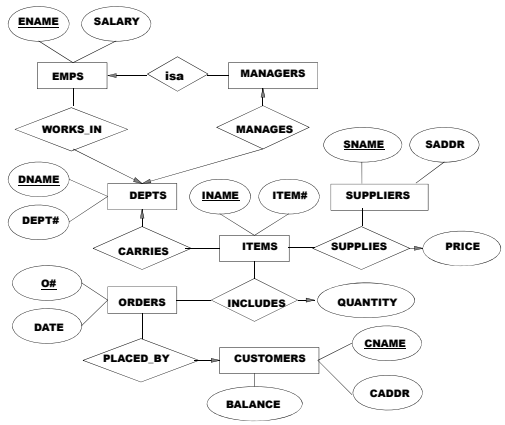


• **Many-Many Relationships**

We also encounter many-many relationship, where there are no restrictions on the sets of k-tuples of entities that may appear in a relationship set. For example, the relationship PARENT_OF in (b) is many-many, because we expect to find two parents for each child, and a given individual may have any number of children.

Many data models do not allow direct expression of many-many relationship, instead requiring that they be decomposed into several many-one relationships.

Example: In the town of Yuppie Valley, a small supermarket, the Yuppie Valley Culinary Boutique (YVCB) has purchased a microcomputer and is about to design a database system that will hold the information the store needs to conduct its business. After due consideration, the database administrator for the system, Sally Hacker, a Sophomore at Calvin Klein Senior High School in Yuppie Valley, who works in the store every Thursday morning, developed the entity-relationship diagram.



One important aspect of the YVCB business is dealing with suppliers. The entity set SUPPLIERS has two attributes, SNAME, the key, and SADDR. One important fact about suppliers that cannot be stored conveniently as an attribute is the set of items that each supplies. Thus, we can specify an entity set ITEMS, with two attributes, INAME and ITEM#, either of which can serve as the key. To connect items and suppliers there is a many-many relationship SUPPLIES, with the intent that each item is related to all the suppliers that can supply the item, and each supplier is related to the items it can supply.

A third entity set, which we call PRICES, is involved in the relationship. Each supplier sets a price for each item it can supply, so we prefer to see the SUPPLIES relationship as a ternary one among ITEMS, SUPPLIERS, and PRICES, with the intent that if the relationship set for SUPPLIES contains the triple (i,s,p), then supplier s is willing to sell item i at price p. PRICES presumably has only one attribute, the price itself. Thus PRICES is an attribute of the relationship SUPPLIES.
We view SUPPLIES as representing item-supplier pairs, and the price as telling something about that pair. SUPPLIES has an arc to PRICES, reminding is that this relationship is many-one from ITEMS and SUPPLIERS to PRICES, i.e., given a supplier and an item, there is a unique price at which the supplier will sell the item.

The YVCB is organized into departments, each of which has a manager and some employees. The attributes of entity set DEPTS are DNAME and DEPT#. Each department is responsible for selling some of the items, and store policy requires that each item can be sold by only one department. Thus, there is a many-one relationship CARRIES from ITEMS to DEPTS. The employees are represented by entity set EMPS, and there is a many-one relationship WOKRS_IN from EMPS to DEPTS, reflecting the policy that employees are never assigned to two or more departments. The managers of departments are represented by another entity set MANAGERS. There is a one-to-one relationship MANAGES between MANAGERS and DEPTS.

The one-to-one-ness reflects the assumption that in the YVCB there will never be more that one manager for a department, nor more than one department managed by one individual. Finally, since managers are employees, we have an isa relationship from MANAGERS to EMPS.

The attributes of the entity set CUSTOMERS of the enterprise are CNAME, CADDR, and BALANCE.

Customers place orders for food items, which are delivered by the YVCB. An order consists of a list of items and quantities placed by one customer. Attributes of the entity set ORDERS are O# (Order number) and DATE, but the actual content of the order is represented by a relationship INCLUDES among ORDERS, ITEMS, and QUANTITY. The later is a entity set whose entities are the integers.

Since a quantity has only its value as an attribute, we show it as a circle attached to the relationship INCLUDES. That relation is many-one from ITEMS and ORDERS to QUANTITY, since each order can have only one quantity of any given item. Finally, the many-one relationship PLACED_BY from ORDERS to CUSTOMERS tells who placed each order.