j

**Chapter 1:**

What is difference between database system and file system?

A database-management system (DBMS) is a collection of interrelated data (database) and a set of programs to access those data. The primary goal of a DBMS is to provide a way to store and retrieve database information that is both convenient and efficient.

File-processing system is supported by a conventional operating system. The system stores permanent records in various files, and it needs different application programs to extract records from, and add records to, the appropriate files.

**What are the drawbacks of using file systems to store data?**

1- Data redundancy and inconsistency

 Multiple file formats, duplication of information in different files

2- Difficulty in accessing data

 Need to write a new program to carry out each new task

3- Data isolation — multiple different files and formats, new application programs to retrieve data is difficult.

4- Integrity problems

 Integrity constraints become “buried” in program code rather than

being stated explicitly

 Hard to add new constraints or change existing ones

5- Atomicity of updates

 Failures may leave database in an inconsistent state with partial updates carried out

 Concurrent access by multiple users

6-Concurrent access needed for performance

 Uncontrolled concurrent accesses can lead to inconsistencies

7- Security problems

 Hard to provide user access to some, but not all, data

**Level of abstraction?**

**Physical level**: The lowest level of abstraction describes how the data are actually stored. The physical level describes complex low-level data structures in detail.

**Logical level:** Describes what data are stored in the database, and what relationships exist among those data. Describes the entire database in terms of a small number of relatively simple structures. The user of the logical level does not need to be aware of this complexity. This is referred to as physical data independence

**View level**: (The highest level of abstraction) describes only part of the entire database. Application programs hide details of data types. The view level of abstraction exists to simplify their interaction with the system.

**What are the issues of storage management?**

1. Storage access

2. File organization

3. Indexing and hashing

**Storage manager** is a program module that provides the interface between the low-level data stored in the database and the application programs and queries submitted to the system.

The storage manager is responsible to the following tasks:

 Interaction with the file manager

 Efficient storing, retrieving and updating of data

The storage manager components include:

1. **Authorization and integrity manager**, which tests for the satisfaction of integrity constraints and checks the authority of users to access data.

2. **Transaction manager**, which ensures that the database remains in a consistent

(correct) state despite system failures, and that concurrent transaction executions proceed without conflicting.

3. **File manager**, which manages the allocation of space on disk storage and the

data structures used to represent information stored on disk.

4. **Buffer manager**, which is responsible for fetching data from disk storage into main memory, and deciding what data to cache in main memory.

**Briefly talking about Database Architecture?**

The architecture of database systems is greatly influenced by the underlying computer system on which the database is running:

1. Centralized

2. Client-server

3. Parallel (multi-processor)

4. Distributed

Database applications are usually partitioned into two or three parts:

1. **Two-tier architecture**, the application resides at the client machine, where it invokes database system functionality at the server machine through query language statements.

2. **Three-tier architecture**, the client machine acts as merely a front end and does not contain any direct database calls. Instead, the client end communicates with an application server

What is the difference between procedural and non-procedural languages?

1. Procedural – user specifies what data is required and how to get those data.

2. Declarative (nonprocedural) – user specifies what data is required without

specifying how to get those data.

**What are the types of binary relationship?**

z

1. One to one

2. One to many

3. Many to one

4. Many to many

**Data Manipulation Language (DML)**

Language for accessing and manipulating the data organized by the appropriate data model. DML also known as query language

 Retrieval of information stored in the database

 Insertion of new information into the database

 Deletion of information from the database

 Modification of information stored in the database

**Data Definition Language (DDL)**

 DDL specify a database schema by a set of definitions. It is used to specify additional properties of the data.

 DDL compiler generates a set of table templates stored in a data dictionary

 Data dictionary contains metadata (i.e., data about data)

 Database schema

**Transaction Management**

 A transaction is a collection of operations that performs a single logical function in a database application

 Transaction-management component ensures that the database remains in a consistent (correct) state despite system failures (e.g., power failures and operating system crashes) and transaction failures.

 Concurrency-control manager controls the interaction among the concurrent transactions, to ensure the consistency of the database.

 The transaction manager consists of the concurrency-control manager and the recovery manager.

**Chapter 2**

 A relational database consists of a collection of  **tables**, each of which is

assigned a unique name.

 A row in a table represents a  *relationship* among a set of values. Since a table is a collection of such relationships, there is a close correspondence between the concept of  *table* and the mathematical concept of *relation*, from which the relational data model takes its name.

 In the relational model the term **relation** is used to refer to a table, while the term  **tuple** is used to refer to a row. Similarly, the term  **attribute** refers to a

column of a table.

 For each attribute of a relation, there is a set of permitted values(allowed

values), called the **domain** of that attribute

  **A domain** is **atomic** if elements of the domain are considered to be indivisible units

 The **database schema**, which is the logical design of the database,

 The **database instance**, which is a snapshot of the data in the database at a given instant in time.

 a **relation schema** corresponds to the programming-language notion of type definition. relation schema consists of a list of attributes and their corresponding domains

  **Relation instance** to refer to a specific instance of a relation, i.e., containing a specific set of rows.

**keys**

 No two tuples in a relation are allowed to have exactly the same value for all attributes

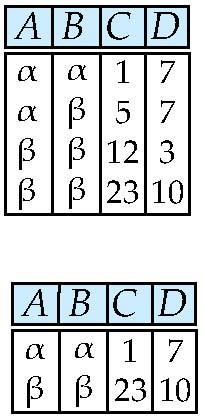
 A **superkey** is a set of one or more attributes that, taken collectively, allow us to identify uniquely a tuple in the relation. For example, the *ID* attribute of the relation *instructor* is sufficient to distinguish one instructor tuple from another. Thus, *ID* is a superkey.

 No proper subset is a superkey. Such  *minimal* superkeys are called  **candidate keys.**

  **Primary key** to denote a candidate key that is chosen by the database designer as the principal means of identifying tuples within a relation.

 A **foreign key** is a set of attributes in a referencing relation, such that for each tuple in the referencing relation, the values of the foreign key attributes are guaranteed to occur as the primary key value of a tuple in the referenced relation.

 a **referential integrity constraint**; a referential integrity constraint requires that the values appearing in specified attributes of any tuple in the referencing relation also appear in specified attributes of at least one tuple in the referenced relation.

**Selection of tuples**

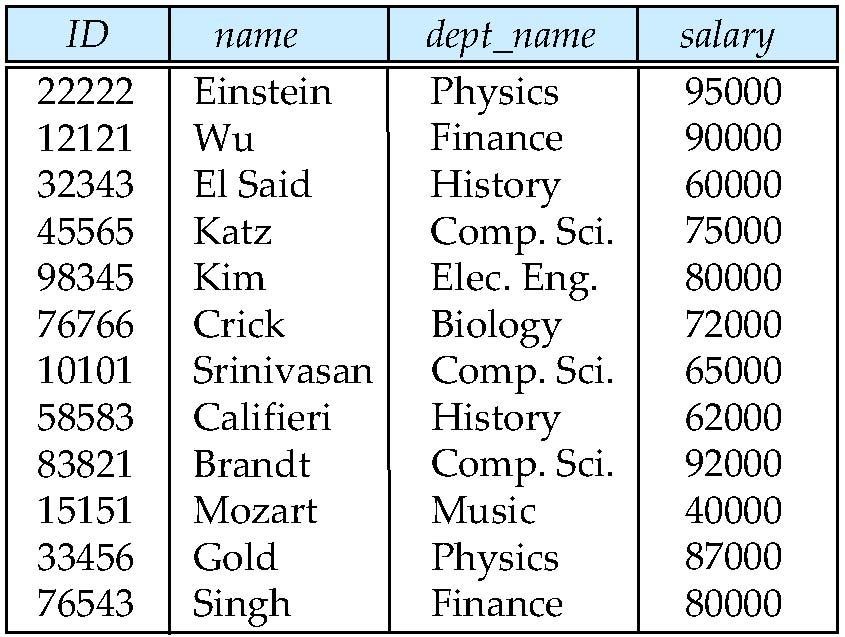
The selection of specific tuples from a single relation (r )

Select tuples with A=B and D > 5

σ A=B and D > 5 (r)

Exampl e

**σ salary >= 85000 (instrcutor)**



**Result**



**Selection of Columns (Attributes)**

**The result is a new relation having only those selected attri**

Select A and C(Projection)

**A, C**

**Π**

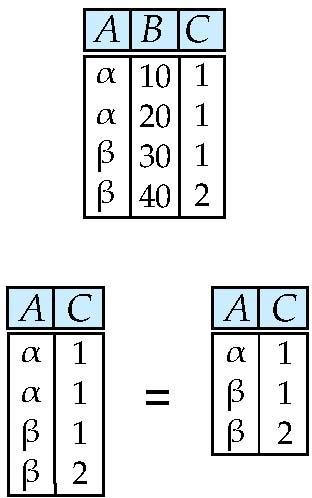
**(r)**

**Example**

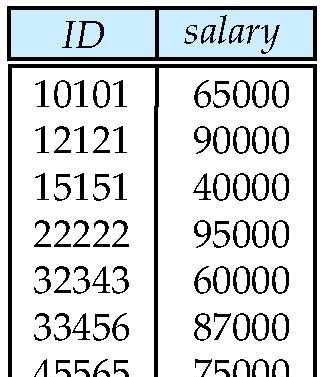
**ID, salary**

**Π**

**(instructor)**

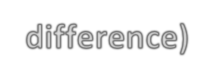


**utes**



**Set operators ( union , intersection an difference)**

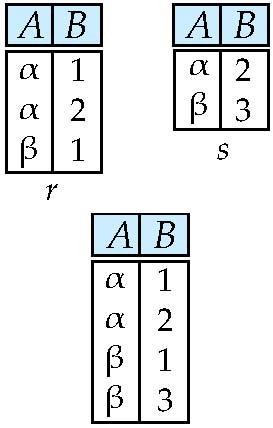
**d**



**Union of two relations**

 **The *union* operation performs a set union of two “similarly structured” tables**

 **Relations *r, s***



**As**

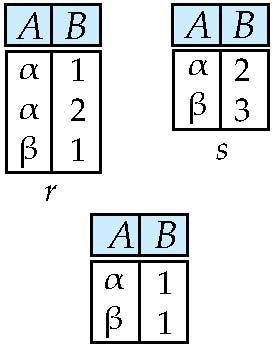
**r**  **s**

**Example**

**Set difference of two relations**

**For Relations *r*,s**

**As:**

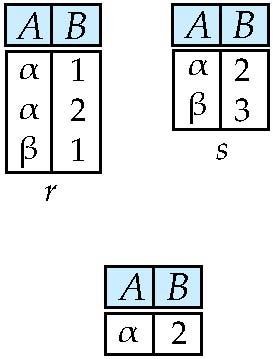


***r – s***

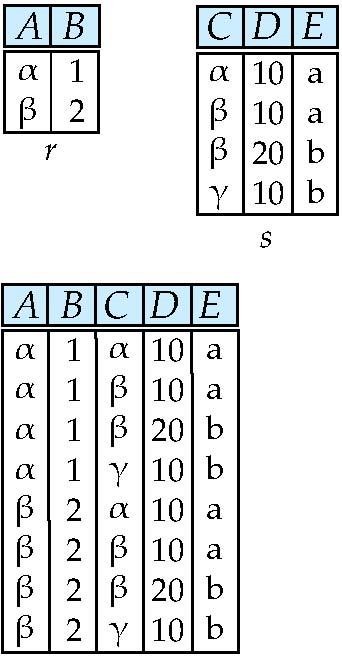
**Set Intersection of two relations**

**As:**

***r***  ***s***

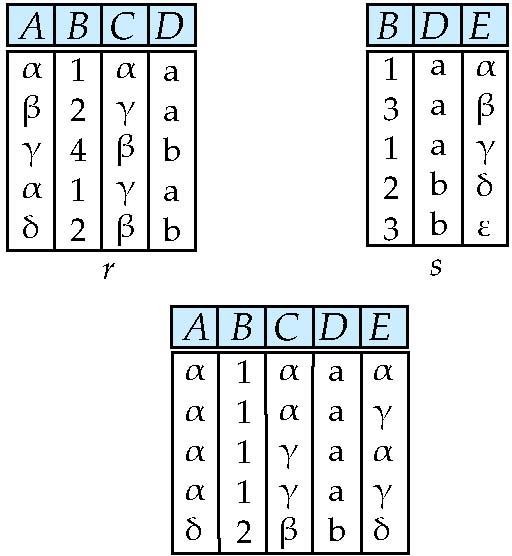
**Joining two relations – Cartesian Product**

The ***Cartesian product*** operation combines tuples from two relations its result contains *all* pairs of tuples from the two relations, regardless of whether their attribute values match.



**Joining two relations – Natural Join**

 Output pairs of rows from the two input relations that have the same value on all attributes that have the same name.

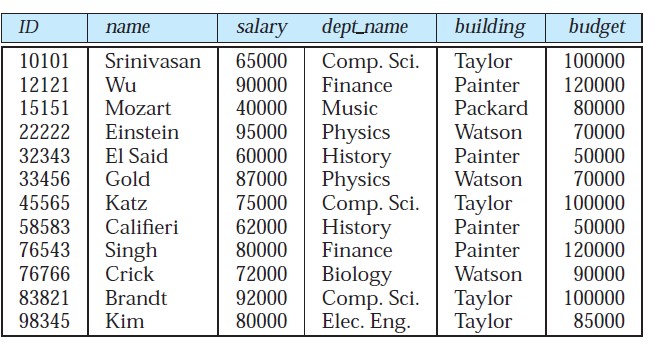
**Natural Join Example**

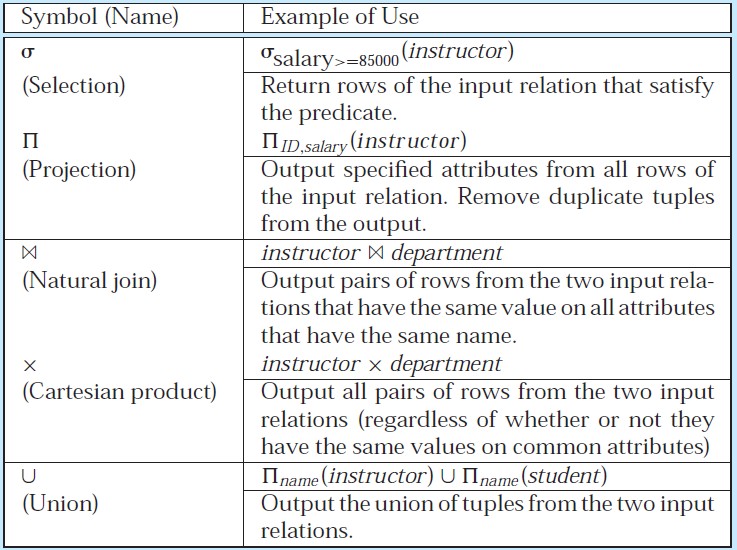
Natural Join

r s

**Example**







**Exercise 2.6**

**classroom(building, room number, capacity) department(dept name, building, budget) course(course id, title, dept name, credits) instructor(ID, name, dept name, salary)**

**section(course id, sec id, semester, year, building, room number, time slot id)**

**teaches(ID, course id, sec id, semester, year)**

**student(ID, name, dept name, tot cred)**

**takes(ID, course id, sec id, semester, year, grade)**

For each expression, explain in words what the expression does.

**a. σ *year*≥2009(*takes*) *student***

**b. σ *year*≥2009(*takes student*)**

**c. *ID,name,course id* (*student takes*)**

**Answer:**

**a.**

**For each student who takes at least one course in 2009, display the students information along with the information about what courses the student took. The attributes in the result are:**

**ID, name, dept name, tot cred, course id, section id, semester, year, grade b. Same as (a); selection can be done before the join operation.**

**c. Provide a list of consisting of**

**ID, name, course id of all students who took any course in the university**

**Exercise 2.7**

***employee* (*person\_name*, *street*, *city*)**

***works* (*person\_name*, *company name*, *salary*)**

***company* (*company name*, *city*)**

Give an expression in the relational algebra to express each of the following queries:

a. Find the names of all employees who live in city “Miami”.

b. Find the names of all employees whose salary is greater than $100,000.

c. Find the names of all employees who live in “Miami” and whose salary is greater

than $100,000.

Answer a.

**Π *person\_name* ( σ**

b.

**Π *person\_name* ( σ**

**city = “Miami”**

**(*employee*) )**

**(*employee*) )**

c.

**Π *person\_name* ( σ**

**salary >= 100000**

**salary >= 100000** ∧ **city = “Miami”**

**(*employee*) )**

**2.8**

branch(branch name, branch city, assets)

customer (customer name, customer street, customer city)

loan (loan number, branch name, amount)

borrower (customer name, loan number)

account (account number, branch name, balance)

depositor (customer name, account number)

Give an expression in the relational algebra for each of the following queries. a. Find the names of all branches located in “Chicago”.

b. Find the names of all borrowers who have a loan in branch “Downtown”.

Answer:

a.

**Π** branch\_name (**σ** branch\_city = “Chicago” (branch) )

b.

**Π** customer\_name (**σ** branch\_name = “Downtown” (borrower loan))

**Chapter 7**

An  **entity** is a “thing” or “object” in the real world that is distinguishable from all other objects. For example, each person in a university is an entity.

An  **entity set** is a set of entities of the same type that share the same properties, or attributes.

**binary relationship** : Relationships between more than two entity sets. A **relationship** is an association among several entities.

A **relationship set** is a set of relationships of the same type. Formally, it is a

mathematical relation on **n ≥ 2** entity sets

The **association** between entity sets is referred to as  **participation**;

**Attribute types**

1-

**Simple**: they have not been divided into subparts

**composite** :can be divided into subparts . For example, an attribute name could be structured as a composite attribute consisting of first name, middle initial, and last name.

*2-*

**Single-valued :** have a single value for a particular entity, the *student ID* attribute

for a specific student entity refers to only one student *ID*.

**multivalued:** There may be instances where an attribute has a set of values for a specific entity.

**3-**

**Derived attribute**: The value for this type of attribute can be derived from the values of other related attributes or entities.

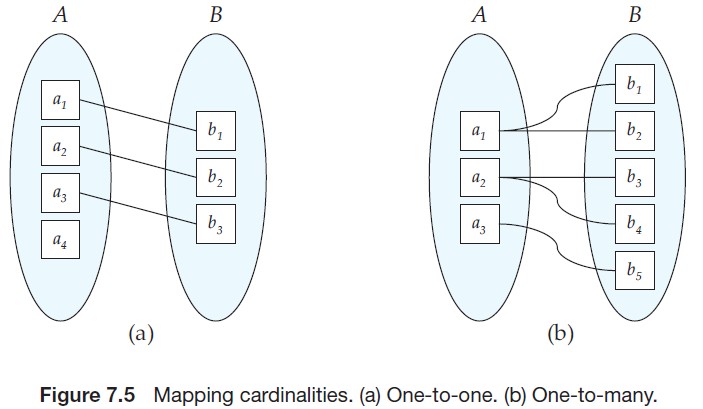
**Mapping cardinalities**, or **cardinality ratios**, express the number of entities to which another entity can be associated via a relationship set.

For a binary relationship set *R* between entity sets *A* and *B*, the mapping cardinality must be one of the following:

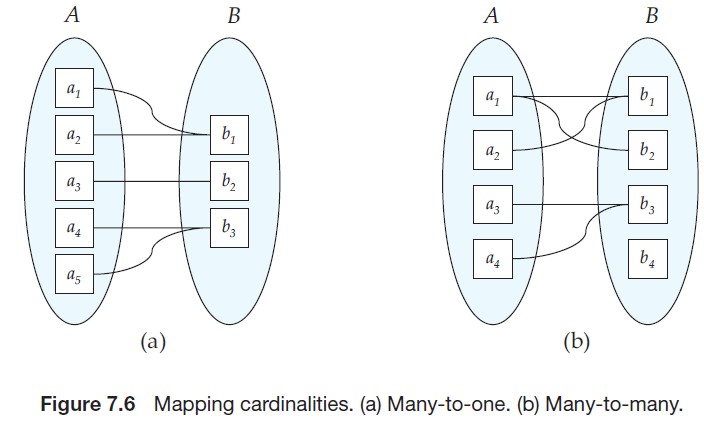
• One-to-one. An entity in *A* is associated with *at most* one entity in *B*, and an entity in *B* is associated with *at most* one entity in *A*. (Figure 7.5a.)

• One-to-many. An entity in *A* is associated with any number (zero or more)

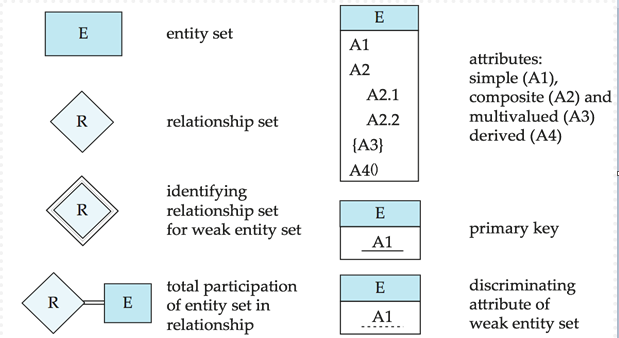
of entities in *B*. An entity in *B*, however, can be associated with *at most* one entity in *A*. (Figure 7.5b.)

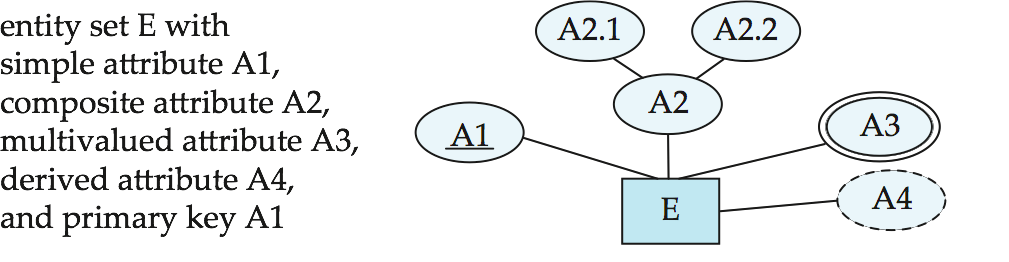


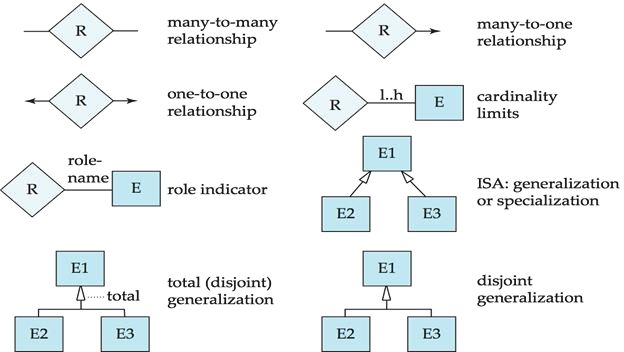
• **Many-to-one**. An entity in *A* is associated with *at most* one entity in *B*. An entity in *B*, however, can be associated with any number (zero or more) of entities in *A*. (See Figure 7.6a.)

 **• M any-to-many**. An entity in *A* is associated with any number (zero or more) of entities in *B*, and an entity in *B* is associated with any number (zero or more) of entities in *A*. (See Figure 7.6b.)

**Summary of Symbols Used in E-R Notation**







**Redudancy of Schemas**

 Many-to-one and one-to-many relationship sets that are  **total** on the many-side can be represented by adding an extra attribute to the “many” side, containing the primary key of the “one” side

 Example: Instead of creating a schema for relationship set ***inst\_dept***, add an attribute ***dept\_name*** to the schema arising from entity set *instructor*

 For **one-to-one** relationship sets, either side can be chosen to act as the “many”

side

 If **participation is *partial*** on the “many” side, replacing a schema by an extra attribute in the schema corresponding to the “ **ma ny” si de** could result in null values

 The schema corresponding to a relationship set linking a weak entity set to its identifying strong entity set  ***is redundant***.

**Extended E-R Features**

**Specialization**

  **ToM p-down design process**; we designate subgroupings within an entity set that

are distinctive from other entities in the set.

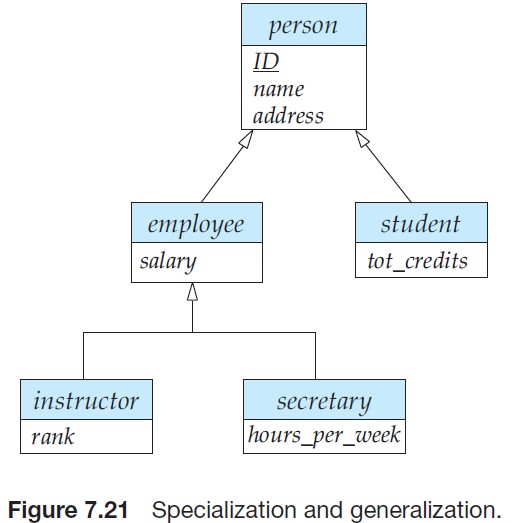
 An entity set may include subgroupings of entities that are distinct in some way from other entities in the set.

 For instance, a subset of entities within an entity set may have attributes that are *not shared* by all the entities in the entity set. The E-R model provides a means for representing these distinctive entity groupings.

 The process of designating subgroupings within an entity set is called

**specialization**.

 The specialization of  ***person*** allows us to distinguish among person entities according to whether they correspond to employees or students: in general, a person could be an employee, a student, both, or neither.



 **Attribute inheritance** – a lower-level entity set inherits all the attributes and

relationship participation of the higher-level entity set to which it is linked.

 The former case (multiple sets permitted) is called  **overlapping specialization**,

while the latter case (at most one permitted) is called  **disjoint specialization.**

 For an **overlapping specialization** (as is the case for *student* and *employee* as specializations of *person*), two separate arrows are used. For  **a disjoint specialization** (as is the case for *instructor* and *secretary* as specializations of *employee*), a single arrow is used. The specialization relationship may also be

referred to as a  **superclass-subclass** relationship.

**Generalization**

 A **bottom-up design process** – combine a number of entity sets that share the same features into a higher-level entity set.

 The refinement from an initial entity set into successive levels of entity subgroupings represents a top-down design process in which distinctions are made explicit.

 Specialization and generalization are simple inversions of each other; they are represented in an E-R diagram in the same way.

 Can have multiple specializations of an entity set based on different features.

 E.g., *permanent\_employee* vs. *temporary\_employee*, in addition to

*instructor* vs. *secretary*

 Each particular employee would be

o a member of one of ***permanent\_employee* or *temporary\_employee***,

o and also a member of one of ***instructor*, *secretary***

 The **ISA** relationship also referred to as  **superclass - subclass** relationship

Constraint on whether or not entities may belong to more than one lower-level

entity set within a single generalization.

1. **Disjoint**. A ***disjointness constraint*** requires that an entity belong to no more than one lower-level entity set. In our example, *student* entity can satisfy only one condition for the *student type* attribute; an entity can be either a graduate student or an undergraduate student, but cannot be both.

2. **Overlapping**. In *overlapping generalizations*, the same entity may belong to more than one lower-level entity set within a single generalization. For an illustration, consider the employee work-team example, and assume that certain employees participate in more than one work team. A given employee may therefore appear in more than one of the team entity sets that are lower level entity sets of *employee*. Thus, the generalization is overlapping.

A the  **completeness constraint**, specifies whether or not an entity in the higher- level entity set must belong to at least one of the lower-level entity sets within the generalization/specialization.

**This constraint may be one of the following:**

1. **Total generalization or specialization**. Each higher-level entity  **must belong**

to a lower-level entity set.

2. **Partial generalization or specialization**. Some higher-level entities  **may not**

belong to any lower-level entity set.

**UML**

Entity-relationship diagrams help model the data representation component of a

software system. Data representation, however, forms only one part of an overall system design. Other components include **models of user interactions** with the system, specification of functional modules of the system and their interaction, etc.

The **Unified Modeling Language** (UML) is a standard developed under the auspices of the Object Management Group (**OMG**) for creating specifications of various components of a software system. Some of the parts of UML are:

 **Class diagram**. A class diagram is similar to an E-R diagram.

 **Activity diagram**. Activity diagrams depict the flow of tasks between various components of a system.

 **Use case diagram**. Use case diagrams show the interaction between users and the system, in particular the steps of tasks that users perform (such as withdrawing money or registering for a course).

 **Implementation diagram**. Implementation diagrams show the system components and their interconnections, both at the software component level and the hardware component level.

**Chapter 8**

The **functional dependency**

  

**Holds on** *R* if and only if for any legal relations *r*(R), whenever any two tuples *t*1 and

*t*2 of *r* agree on the attributes , they also agree on the attributes *.* That is,

*t*1[] = *t*2 []  *t*1[ ] = *t*2 [ ]

**First Normal Form**(1 NF)

The domains of all attributes are atomic

**Car** (it is not in 1NF)D

|  |  |  |
| --- | --- | --- |
| ID | Color | Price |
| 1 | Green, Blue | 20000 |
| 2 | Red | 15000 |

**2NF: no nonprime attribute depends on part of primary key**

Student ID , Course ID Enrolment Date

Student Id Student Name

|  |  |  |  |
| --- | --- | --- | --- |
| Student ID | Course ID | Student Name | Enrolment Date |
| 1 | Cs140 | Ali | 1-1-2014 |
| 2 | Cs140 | Omar | 3-1-2014 |

**To be in 2NF:**

|  |  |  |
| --- | --- | --- |
| Student ID | Course ID | Enrolment Date |
| 1 | Cs140 | 1-1-2014 |
| 2 | Cs140 | 3-1-2014 |

|  |  |
| --- | --- |
| Student ID | Student Name |
| 1 | Ali |
| 2 | Omar |

**3NF**

|  |  |  |  |
| --- | --- | --- | --- |
| **ID** | **City** | **Meal** | **Price** |
| 1 | 1 | A | 10 |
| 2 | 2 | B | 10 |
| 3 | 1 | C | 20 |

ID  city

ID  price

City  meal

So ID  meal

So we make two tables for to be 3NF:

|  |  |  |
| --- | --- | --- |
| **ID** | **City** | **Price** |
| 1 | 1 | 10 |
| 2 | 2 | 10 |
| 3 | 1 | 20 |

|  |  |
| --- | --- |
| **City** | **Meal** |
| 1 | A |
| 2 | B |
| 1 | C |

**-----------------------------------------------------------------------------------------------------**

 We can find F*+,* the closure of F, by repeatedly applying  **A rms tron g’s Ax i oms :**

1. if   , then    **(reflexivity rule)**

2. if   *,* then      **(augmentation rule)**

3. if   *,* and   , then    **(transitivity rule)**

**Additional rules**  ca n be in fer re d fro m Arm stro ng ’s ax io ms.

4. **Union rule**.    holds and    holds, then →   holds.

5. **Decomposition rule**. If →   holds, then    and    holds.

6. **PseudWWQotransitivity rule**. If    holds and  → ∂ holds, then  →∂ holds. These rules are

1. **sound** (generate only functional dependencies that actually hold), and

2. **complete** (generate all functional dependencies that hold).

]==

**Example**

*R = (A, B, C, G, H, I)*

***F =* { *A***  ***B A***  ***C CG***  ***H CG***  ***I***

***B***  ***H* }**

**some members of *F*+**

 *A*  *H*

 ***AG***  ***I***

 ***CG***  ***HI***

 by transitivity from *A*  *B and B*  *H*

 by augmenting *A*  *C* with G, to get *AG*  *CG*

and then transitivity with *CG*  *I*

 by augmenting *CG*  *I* to infer ***CG***  **CG*I****,*

augmenting of *CG*  *H* to infer ***CGI***  ***HI****,* and then transitivity

**Question**

Compute the closure of the following set F of functional dependencies for relation schema R = ( A, B, C, D, E )

**A → BC**

**CD → E B → D**

**E → A**

**Solution**

Starting with A → BC , we can conclude: A → B and **A → C** .

Since A → B and B → D, **A → D** (decomposition, transitive)

Since A → C D and C D → E , **A → E** (union, decomposition, transitive)

Since A → A , we have (reflexive)

**A → ABCDE** from the above steps (union)

Since E → A , **E → ABCDE** (transitive)

Since C D → E , **C D → ABCDE** (transitive)

Since B → D and BC → C D, **BC →ABCDE** (augmentative, transitive)

**So F+{**

**}**

m

**A → C A → D A → E**

**A → ABC**

**E → ABCDE**

**C D → ABCDE BC →ABCDE**

**------------------------------------------------------------------------------------------- T/F, short answers, long short answers, MCQ , Fill the void**