MIT-WPU

॥ विश्वशान्तिर्ध्रुवं ध्रुवा ॥

Dr. Vishwanath Karad

MIT WORLD PEACE UNIVERSITY | PUNE

TECHNOLOGY, RESEARCH, SOCIAL INNOVATION & PARTNERSHIPS

Database Management Systems

SCHOOL OF COMPUTER ENGINEERING AND TECHNOLOGY

Database Management Systems

Course Objectives:

1.Understand and successfully apply logical database design principles, including E-R diagrams and database normalization.

2. Learn Database Programming languages and apply in DBMS applications.

3. Understand transaction processing and concurrency control in DBMS.

4. Learn database architectures, DBMS advancements and its usage in advance applications.

Course Outcomes: Upon completion of the course, the students will be able to:

1.Design ER-models to represent simple database application scenarios and Improve the database design by normalization.

2. Design Database Relational Model and apply SQL , PLSQL concepts for database programming.

3. Describe Transaction Processing and Concurrency Control techniques for databases.

4. Identify appropriate database architecture for the real world database applications.

Introduction to Database Management Systems and Data Modeling

Syllabus :

DBMS Vs File Systems, Database System Architecture, Data Abstraction, Data Independence, Data Definition and Data Manipulation Languages, Database System Internals-Components of a database system,

Data Models , E-R diagram: Components of E-R Model, Conventions, Keys, EER diagram Components, E-R diagram into tables, Relational Model, Relational Integrity, Referential Integrities, Enterprise Constraints, Schema Diagram, Relational Algebra- Basic Operations, Normalization.

Database Management System Basics

- It contains information about a particular enterprise.
- It provides :
  - ➢ Collection of interrelated data.
  - ➢ Set of programs to access the collected data.
  - ➢ An environment that is both convenient and efficient to use.
  - ➢ Databases touch all aspects of our lives.

Database Applications

- Banking: Transactions
- Airlines: Reservations, Schedules
- Universities : Student Registration.
- Sales: Customers, Orders, Products
- Online retailers : Order tracking, recommendations
- Manufacturing :Production, Inventory, Supply Chain
- Human Resources :Employee payroll.

Motivation for Database Management Systems

Traditional file Systems have following drawbacks to store data :

- **Data Redundancy and Inconsistency**
  - Multiple file formats, duplication of information in different files
- **Difficulty in accessing data**
  - Need to write a new program to carry out each new task.
- **Data Isolation**
  - Multiple files and formats.

Question : What can be other drawbacks related to usage of file systems ?

Motivation for Database Management Systems

Answer : Traditional file Systems have following drawbacks to store data:

- **Integrity Problems**
  - Integrity constraints (e.g., account balance > 0) become "buried" in program code rather than being stated explicitly
  - Difficult to add new constraints or change existing ones
- **Atomicity of Updates**
  - Failures may leave database in an inconsistent state with partial updates carried out.

Motivation for Database Management Systems

Traditional file Systems have following drawbacks to store data :

- **Concurrent Access by Multiple users**
  - Concurrent access needed for performance.
  - Uncontrolled concurrent accesses can lead to inconsistencies.
- **Security Problems**
  - Hard to provide user access to some, but not all, data

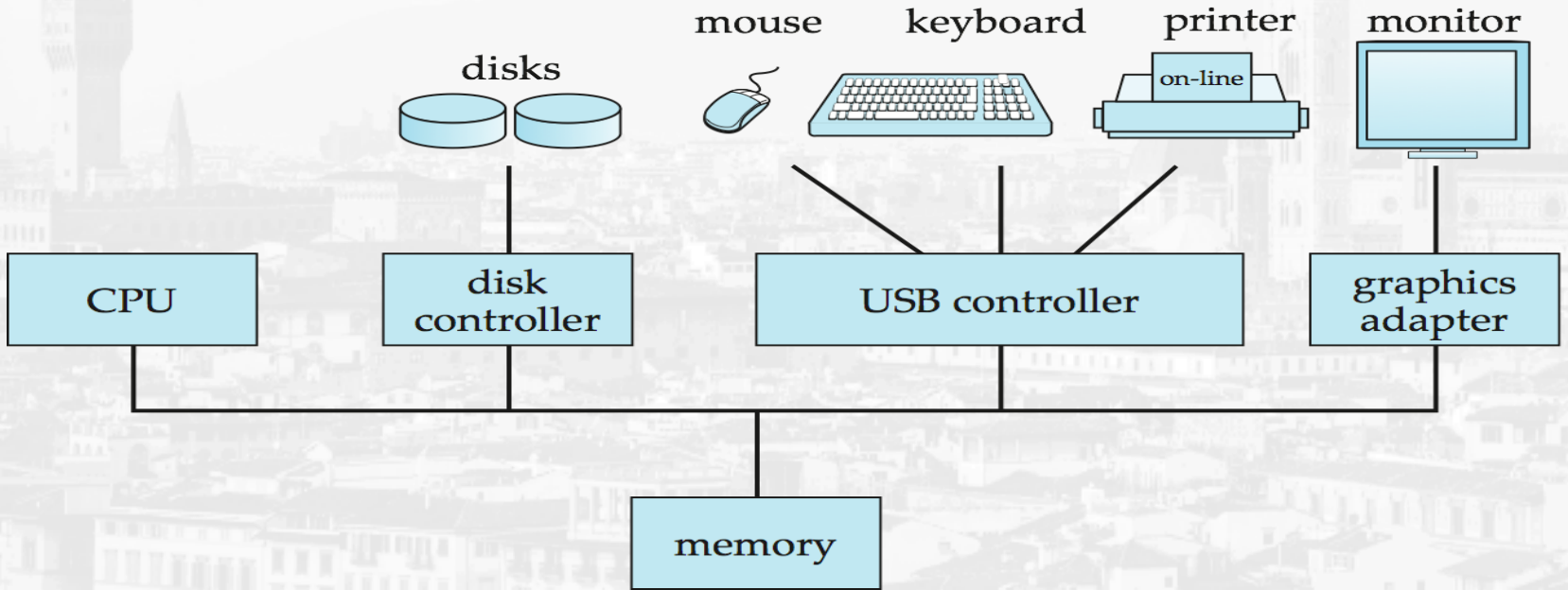
Database Management Systems offers solutions to all the above problems/limitations of traditional file systems.

Database System Architectures

Centralized and Client-Server Systems
Server System Architectures
Parallel Systems
Distributed Systems

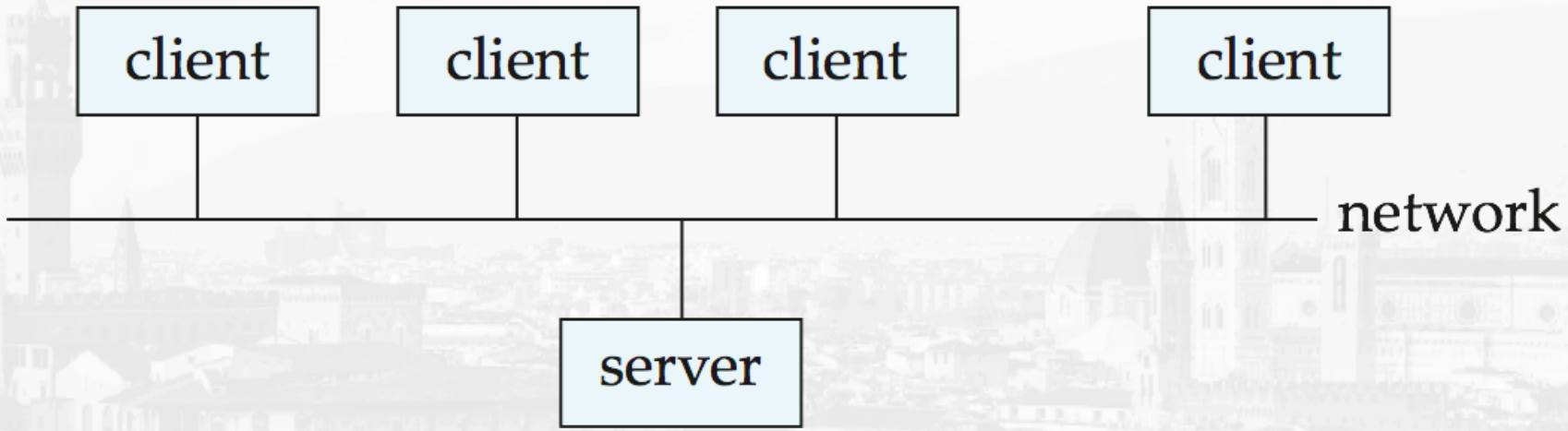
Centralized Systems

Run on a single computer system and do not interact with other computer systems.
General-purpose computer system: one to a few CPUs and a number of device controllers that are connected through a common bus that provides access to shared memory.
Single-user system (e.g., personal computer or workstation): desk-top unit, single user, usually has only one CPU and one or two hard disks; the OS may support only one user.
Multi-user system: more disks, more memory, multiple CPUs, and a multi-user OS. Serve a large number of users who are connected to the system vie terminals. Often called *server* systems.



Client-Server Systems

Server systems satisfy requests generated at *m* client systems, whose general structure is shown below:

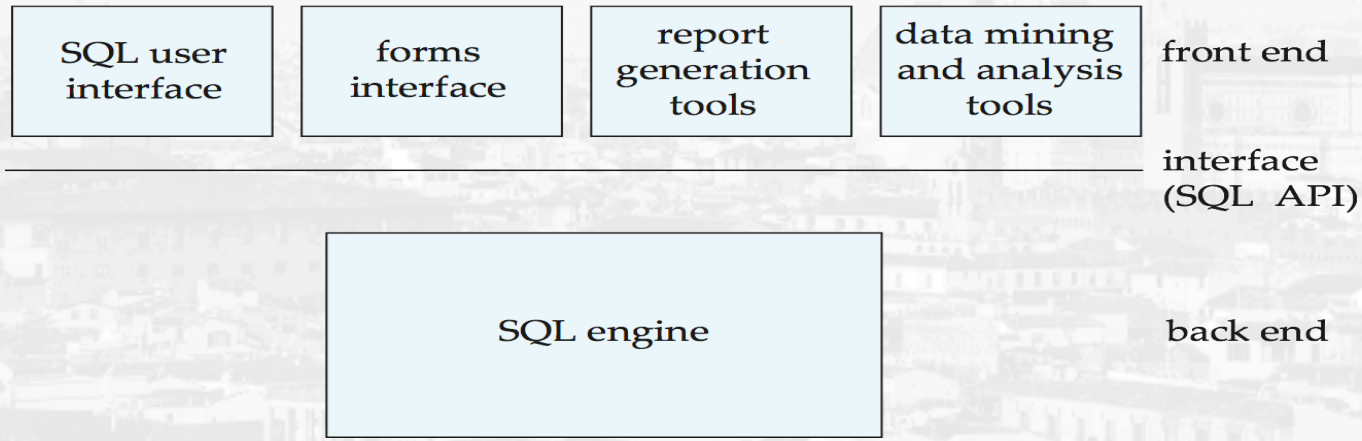


Client-Server Systems (Cont.)

Database functionality can be divided into:

- **Back-end**: manages access structures, query evaluation and optimization, concurrency control and recovery.
- **Front-end**: consists of tools such as *forms*, *report-writers*, and graphical user interface facilities.

The interface between the front-end and the back-end is through SQL or through an application program interface.



Client-Server Systems (Cont.)

Advantages of replacing mainframes with networks of workstations or personal computers connected to back-end server machines:

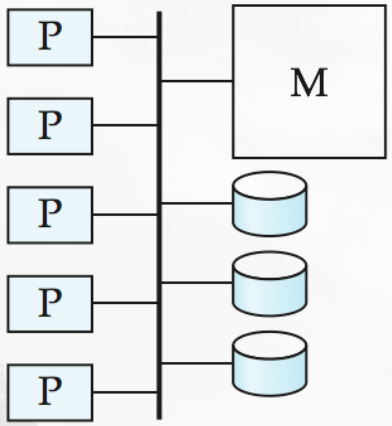
- better functionality for the cost
- flexibility in locating resources and expanding facilities
- better user interfaces
- easier maintenance

Parallel Systems

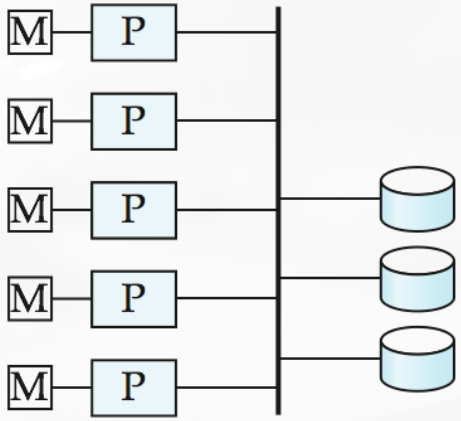
Parallel database systems consist of multiple processors and multiple disks connected by a fast interconnection network.
A **coarse-grain parallel** machine consists of a small number of powerful processors
A **massively parallel** or **fine grain parallel** machine utilizes thousands of smaller processors.
Two main performance measures:

- **throughput** --- the number of tasks that can be completed in a given time interval
- **response time** --- the amount of time it takes to complete a single task from the time it is submitted

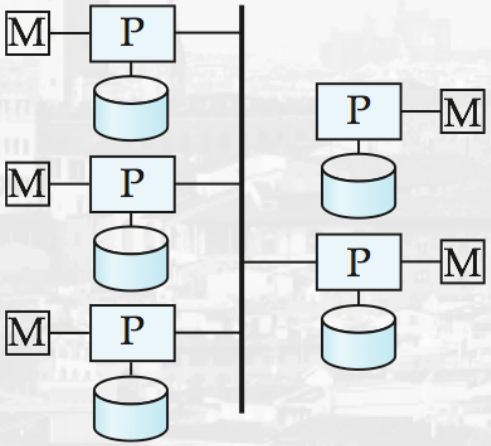
Parallel Database Architectures



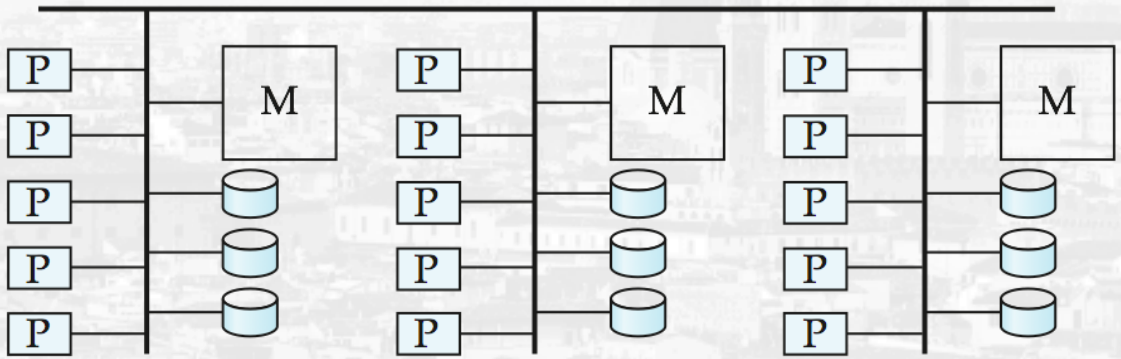
(a) shared memory



(b) shared disk



(c) shared nothing



(d) hierarchical

Shared Memory

Processors and disks have access to a common memory, typically via a bus or through an interconnection network.
Extremely efficient communication between processors — data in shared memory can be accessed by any processor without having to move it using software.
Downside – architecture is not scalable beyond 32 or 64 processors since the bus or the interconnection network becomes a bottleneck
Widely used for lower degrees of parallelism (4 to 8).

Shared Disk

All processors can directly access all disks via an interconnection network, but the processors have private memories.

- The memory bus is not a bottleneck
- Architecture provides a degree of **fault-tolerance** — if a processor fails, the other processors can take over its tasks since the database is resident on disks that are accessible from all processors.

Examples: IBM Sysplex and DEC clusters (now part of Compaq) running Rdb (now Oracle Rdb) were early commercial users

Downside: bottleneck now occurs at interconnection to the disk subsystem.

Shared-disk systems can scale to a somewhat larger number of processors, but communication between processors is slower.

Shared Nothing

Node consists of a processor, memory, and one or more disks. Processors at one node communicate with another processor at another node using an interconnection network. A node functions as the server for the data on the disk or disks the node owns.

Examples: Teradata, Tandem, Oracle-n CUBE

Data accessed from local disks (and local memory accesses) do not pass through interconnection network, thereby minimizing the interference of resource sharing.

Shared-nothing multiprocessors can be scaled up to thousands of processors without interference.

Main drawback: cost of communication and non-local disk access; sending data involves software interaction at both ends.

Hierarchical

Combines characteristics of shared-memory, shared-disk, and shared-nothing architectures.

Top level is a shared-nothing architecture – nodes connected by an interconnection network, and do not share disks or memory with each other.

Each node of the system could be a shared-memory system with a few processors.

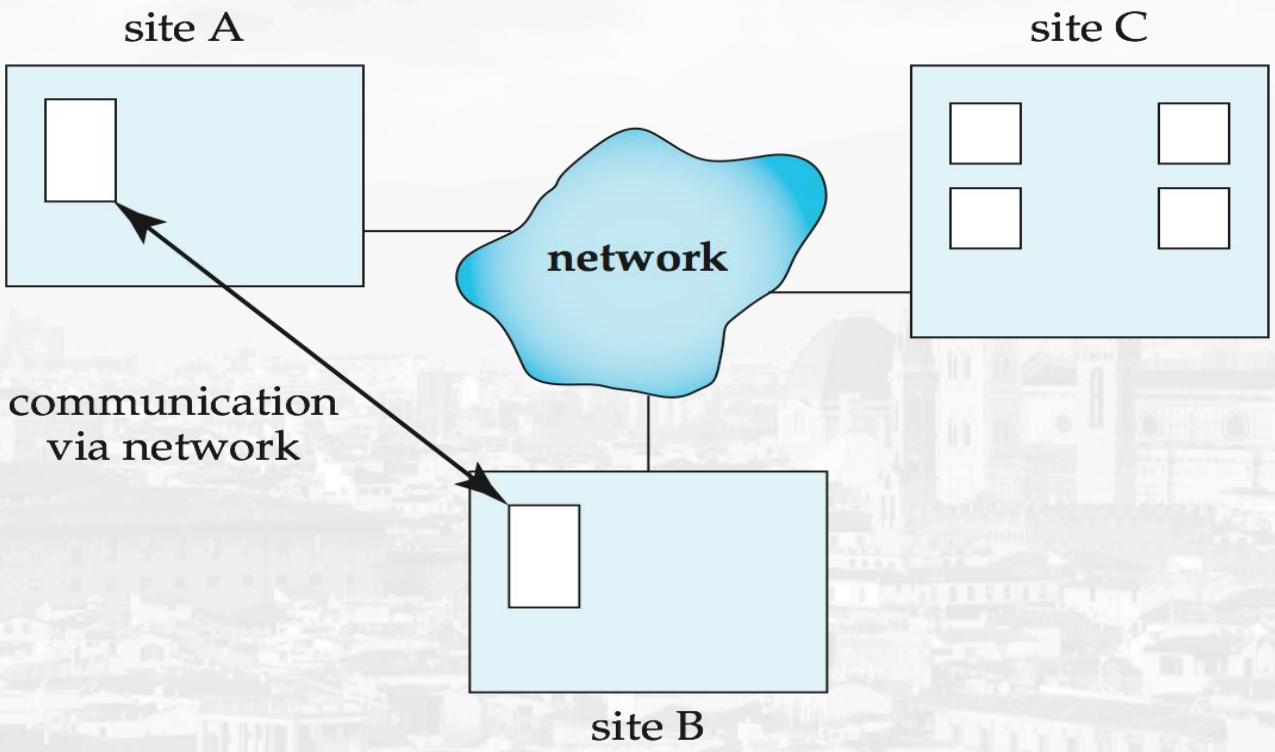
Alternatively, each node could be a shared-disk system, and each of the systems sharing a set of disks could be a shared-memory system.

Reduce the complexity of programming such systems by **distributed virtual-memory** architectures

Also called **non-uniform memory architecture (NUMA)**

Distributed Systems

Data spread over multiple machines (also referred to as **sites** or **nodes**).
Network interconnects the machines
Data shared by users on multiple machines



Distributed Databases

Homogeneous distributed databases

- Same software/schema on all sites, data may be partitioned among sites
- Goal: provide a view of a single database, hiding details of distribution

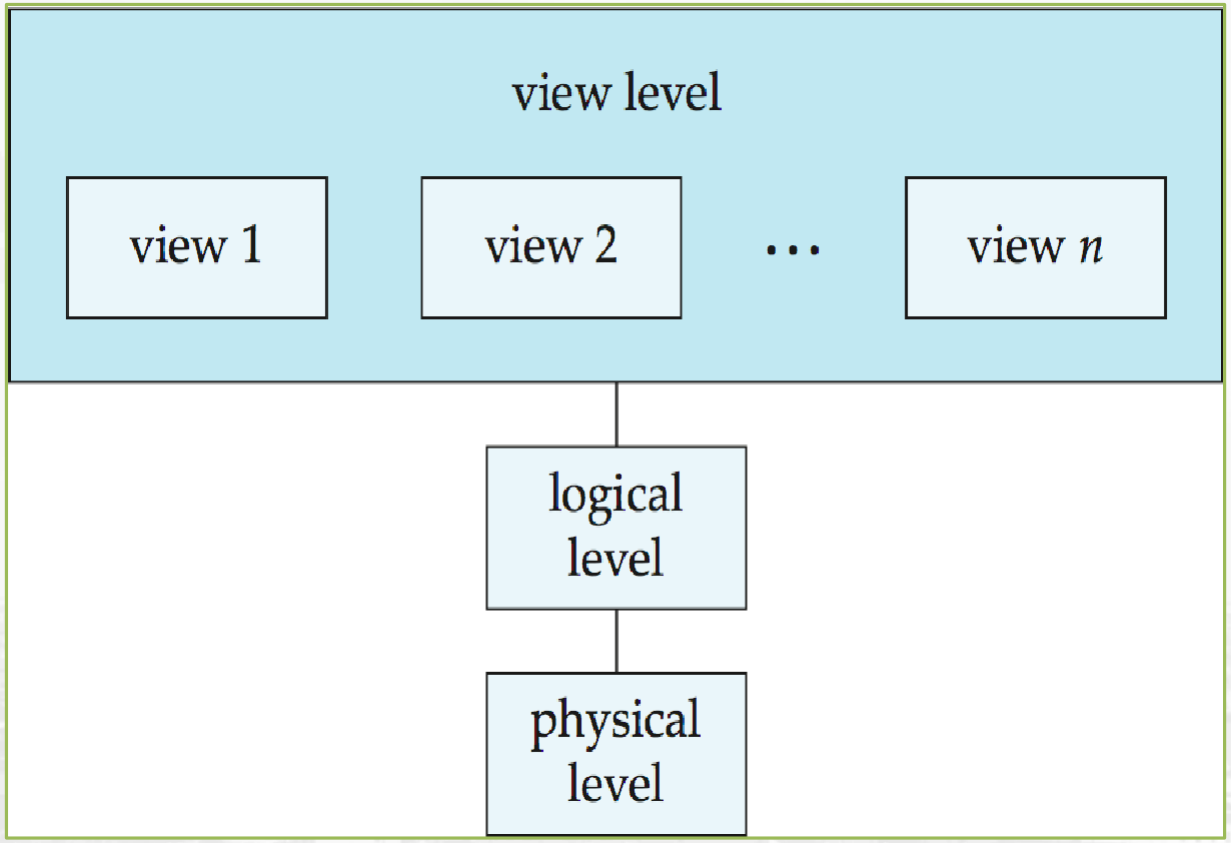
Heterogeneous distributed databases

- Different software/schema on different sites
- Goal: integrate existing databases to provide useful functionality

Differentiate between *local* and *global* transactions

- A **local transaction** accesses data in the *single* site at which the transaction was initiated.
- A **global transaction** either accesses data in a site different from the one at which the transaction was initiated or accesses data in several different sites.

Levels of Data Abstraction in Database System



Levels of Abstraction in a Database System

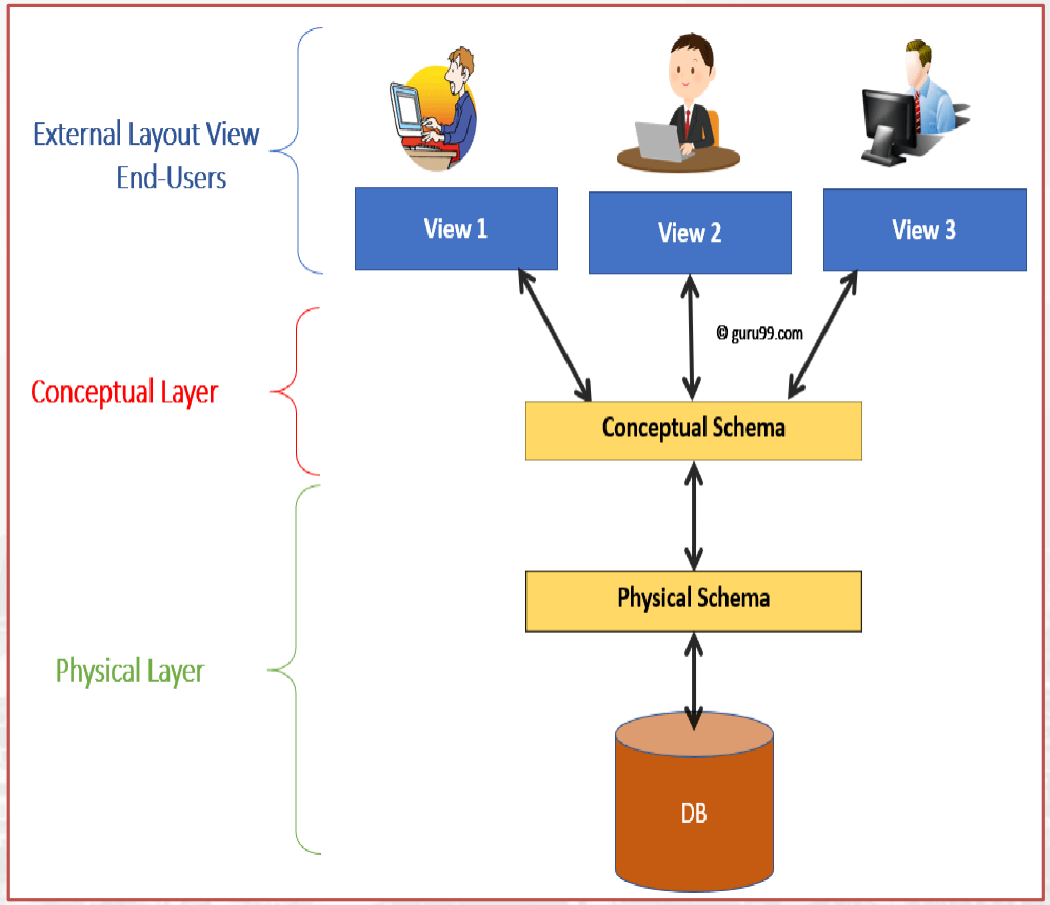
- **Physical level:** describes how a **record** (e.g., customer) is stored.
- **Logical level:** describes data stored in database, and the relationships among the data.

```
type instructor = record
        ID : string;
        name : string;
        dept_name : string;
        salary : integer;
    end;
```

- **View level:** application programs hide details of data types. Views can also hide information (such as an employee's salary) for security purposes.

Instances and Schema

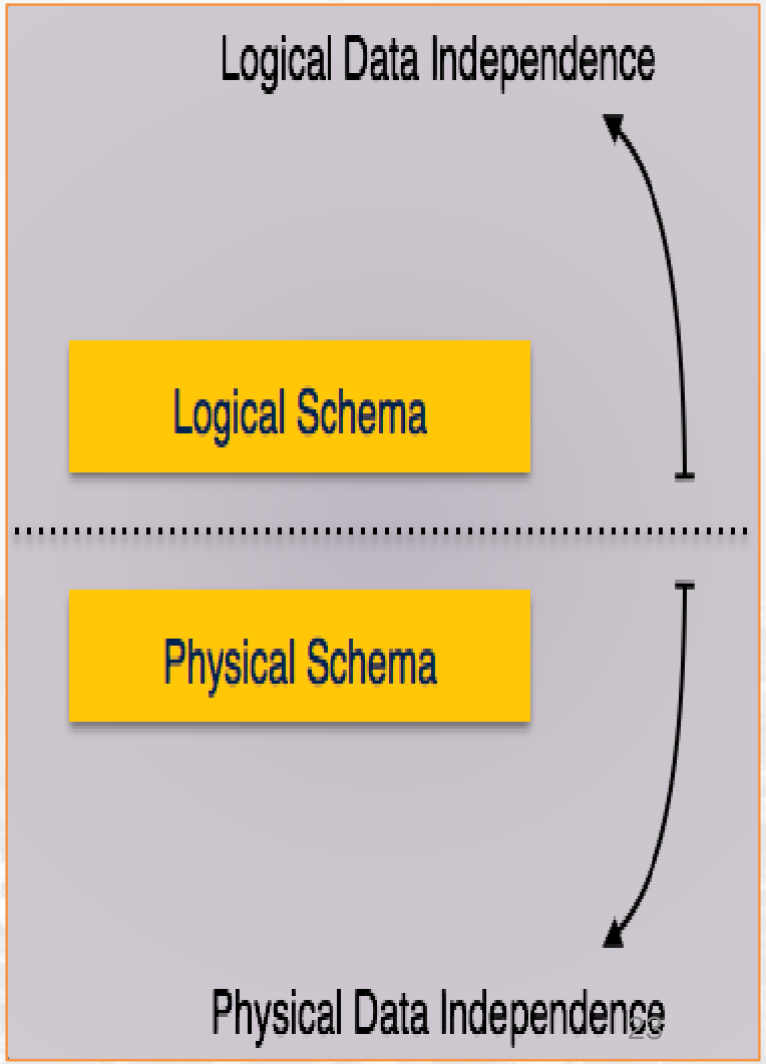
- **Schema – the logical structure of the database**
  - Example: The database consists of information about a set of customers and accounts and the relationship between them
  - ➢ **Physical schema**: database design at the physical level
  - ➢ **Logical schema**: database design at the logical level
- **Instance – the actual content of the database at a particular point in time**
  - ➢ Analogous to the **value of a variable.**



Data Independence

Types of Data Independence :

- **Physical Data Independence** : the ability to modify the physical schema without changing the logical schema
  - Applications depend on the logical schema
  - In general, the interfaces between the various levels and components should be well defined so that changes in some parts do not seriously influence others.
- **Logical Data Independence :** the ability to change the conceptual scheme without changing
  - External views
  - External API or programs
  - Any change made will be absorbed by the mapping between external and conceptual levels.
  - When compared to Physical Data independence, it is challenging to achieve logical data independence.



Database System Languages

Data Definition Language(DDL)

Specification notation for defining the database schema

Example: **create table** *instructor* (
ID **char**(5),
name **varchar**(20),
dept_name **varchar**(20),
salary **numeric**(8,2))

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

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

- Database schema
- Integrity constraints
  - Primary key (ID uniquely identifies instructors)
  - Referential integrity (**references** constraint in SQL)
    - e.g. *dept_name* value in any *instructor* tuple must appear in *department* relation

Data Manipulation Language(DML)

Language for accessing and manipulating the data organized by the appropriate data model

- DML also known as query language

Two classes of languages

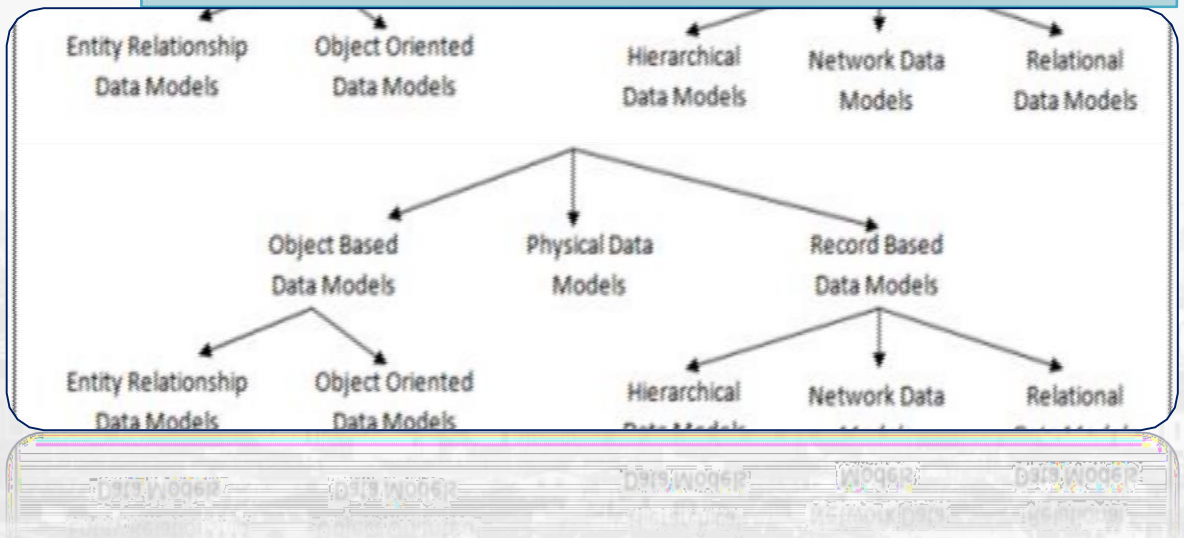
- **Procedural** – user specifies what data is required and how to get those data
- **Declarative (nonprocedural)** – user specifies what data is required without specifying how to get those data

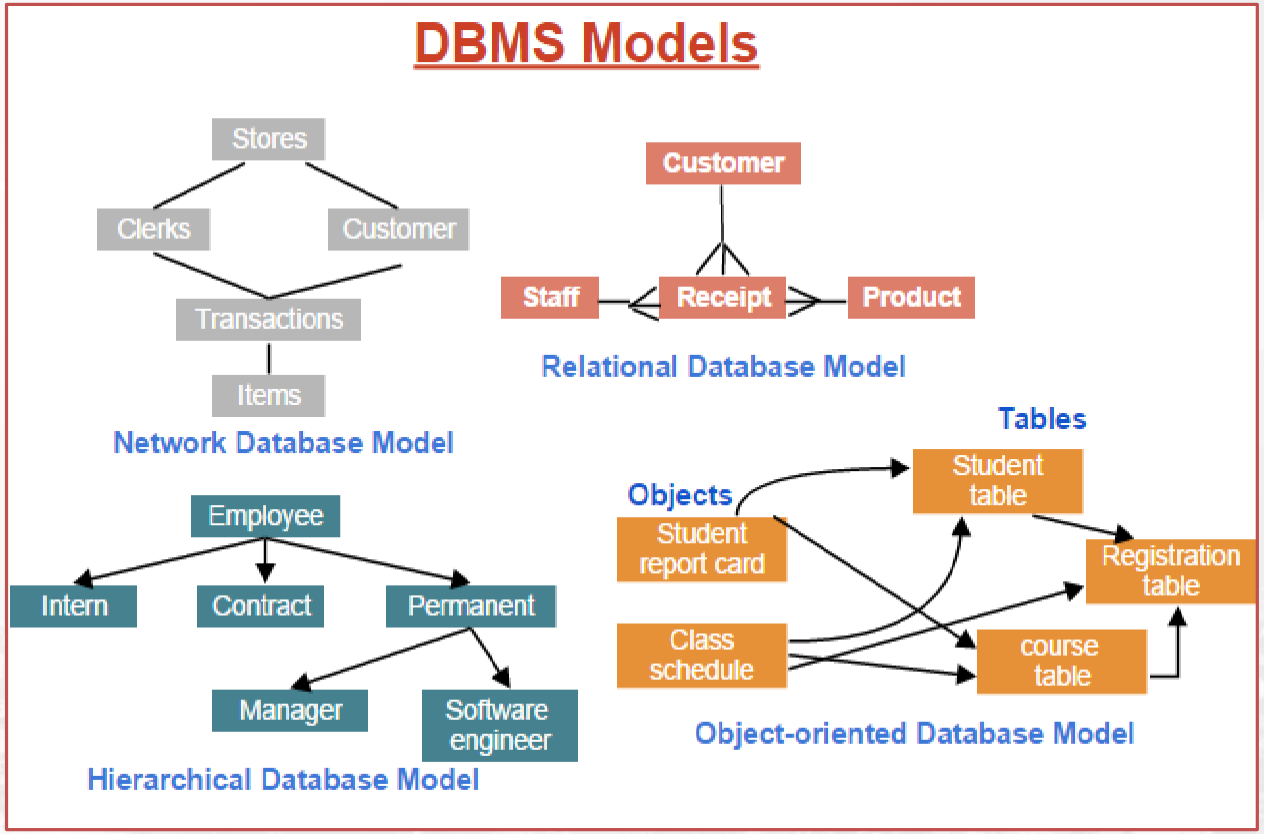
SQL is the most widely used query language

Data Models

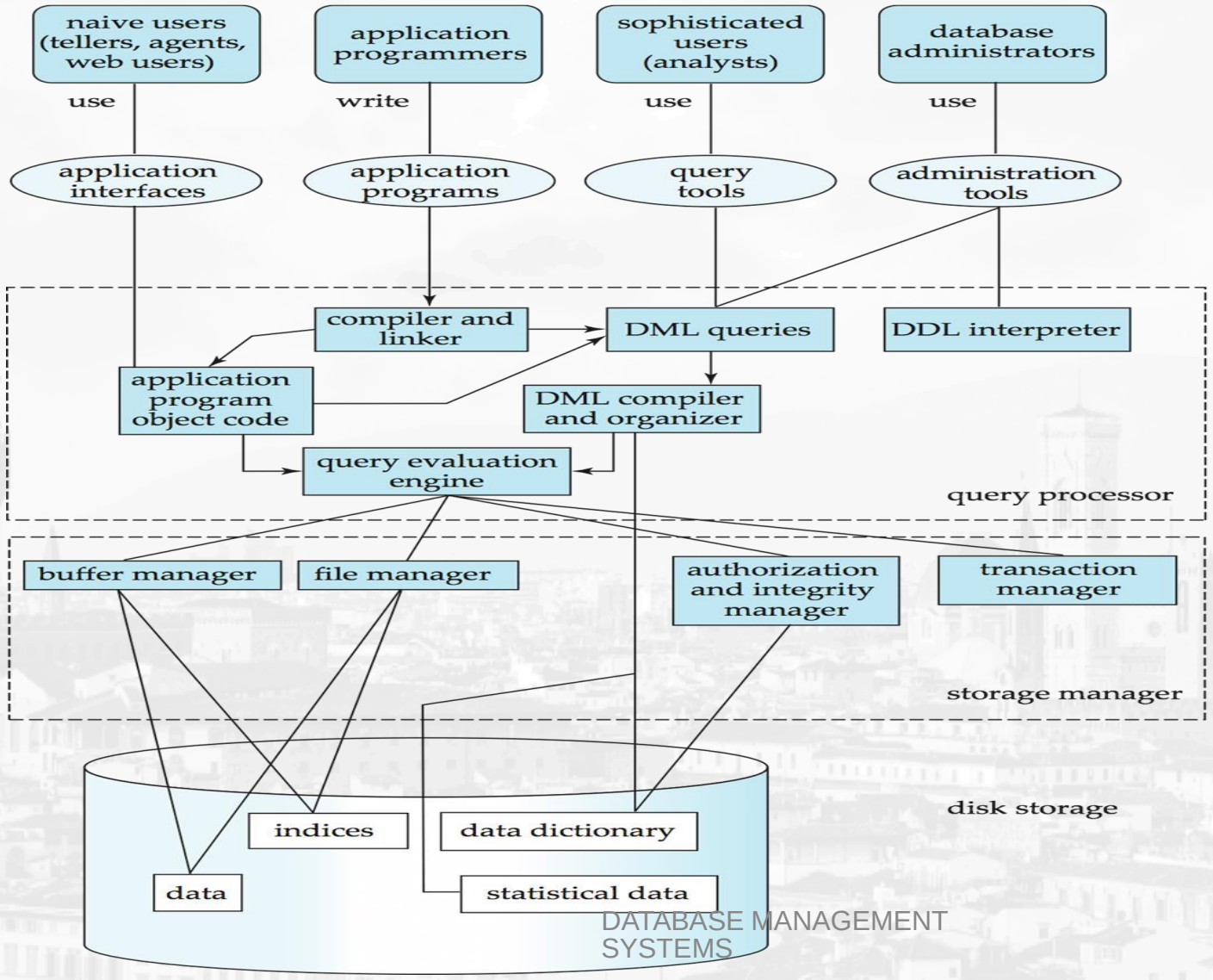
A collection of tools for describing :

- Data
- Data relationships
- Data semantics
- Data constraints





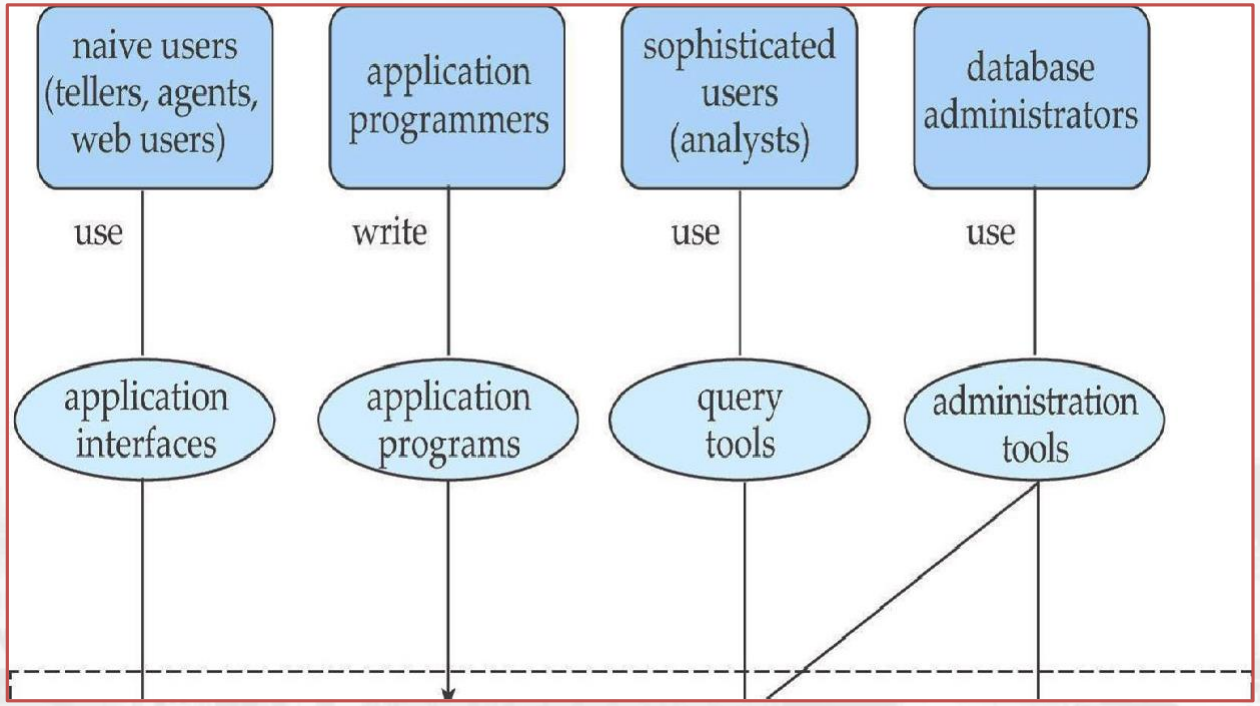
Database System Structure/Architecture



Important Components of Database System :

- **Database Users**
- **Query Processing**
- **Storage Management**
- **Transaction Management**

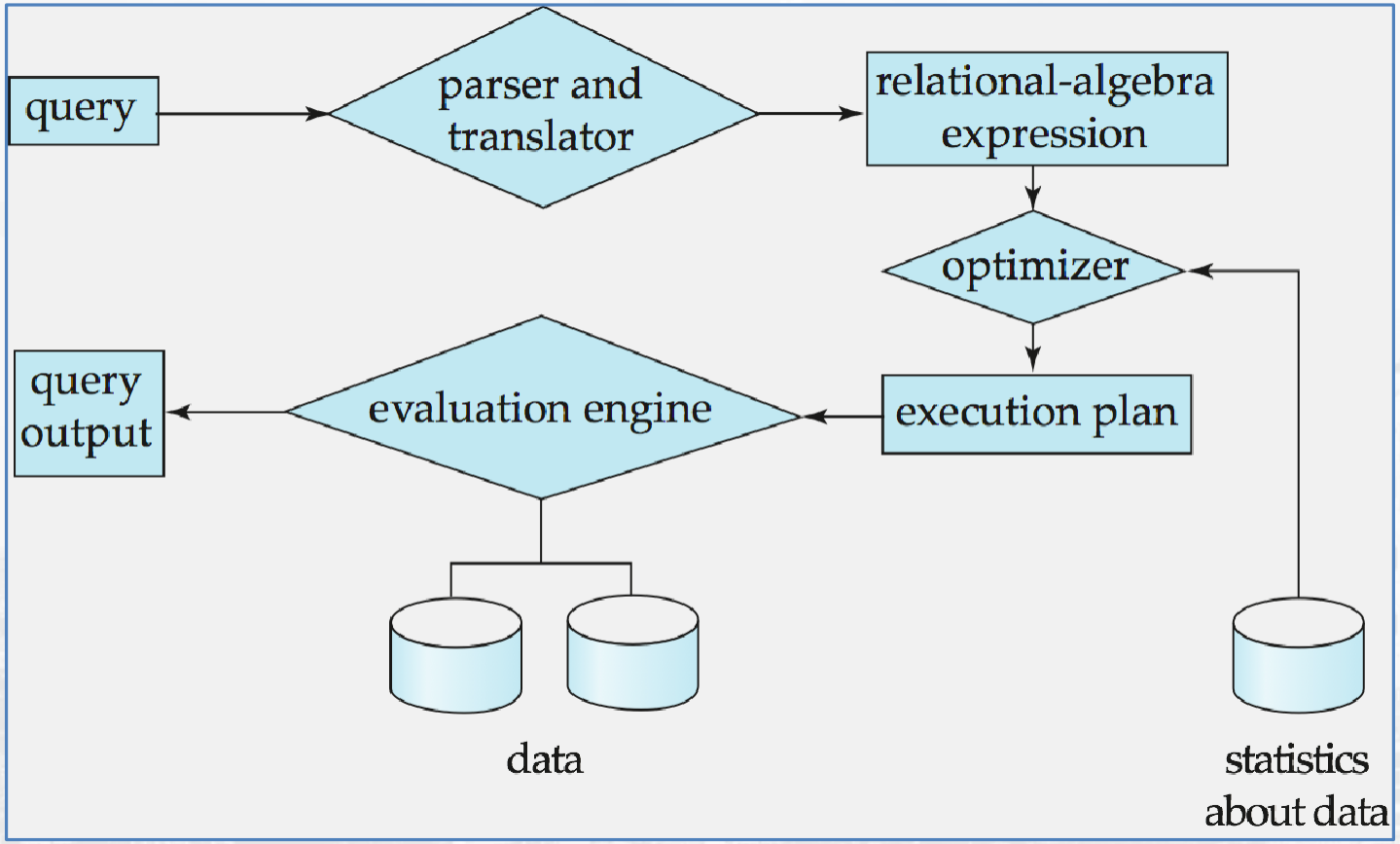
Database System Components : Database Users



Types of Database Users :

- Naive Users
- Application Programmers
- Sophisticated Users
- Database Administrators

Database System Components : Query Processing



Query Processing Steps :

1.Parsing and Translation

2.Optimization

3.Evaluation

Database System Components :Storage Management

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.

It is responsible for the following tasks:

- Interaction with the file manager
- Efficient storing, retrieving and updating of data

Issues:

- Storage access
- File organization
- Indexing and hashing

Database System Components :Transaction Management

What if the system fails?

What if more than one user is concurrently updating the same data?

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

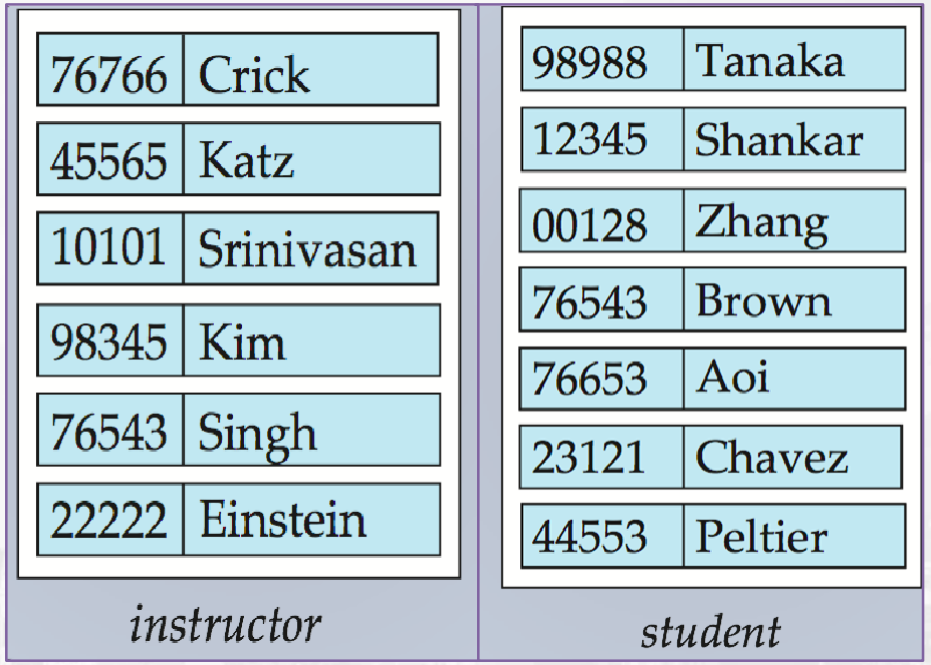
Two Important Components related to Transactions:

- **Transaction Manager**
- **Concurrency Control Manager**

- **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.

Entity Relationship Model

- A *database* can be modeled as:
  - a collection of entities,
  - relationship among entities.
- An **entity** is an object that exists and is distinguishable from other objects.
  - Example: specific person, company, event, plant
- Entities have **attributes**
  - Example: people have *names* and *addresses*
- An **entity set** is a set of entities of the same type that share the same properties.
  - Example: set of all persons, companies, trees, holidays



Relationship Sets

A **relationship** is an association among several entities

Example:

44553 (Peltier) *advisor* 22222 (Einstein)

***student* entity** **relationship set** ***instructor* entity**

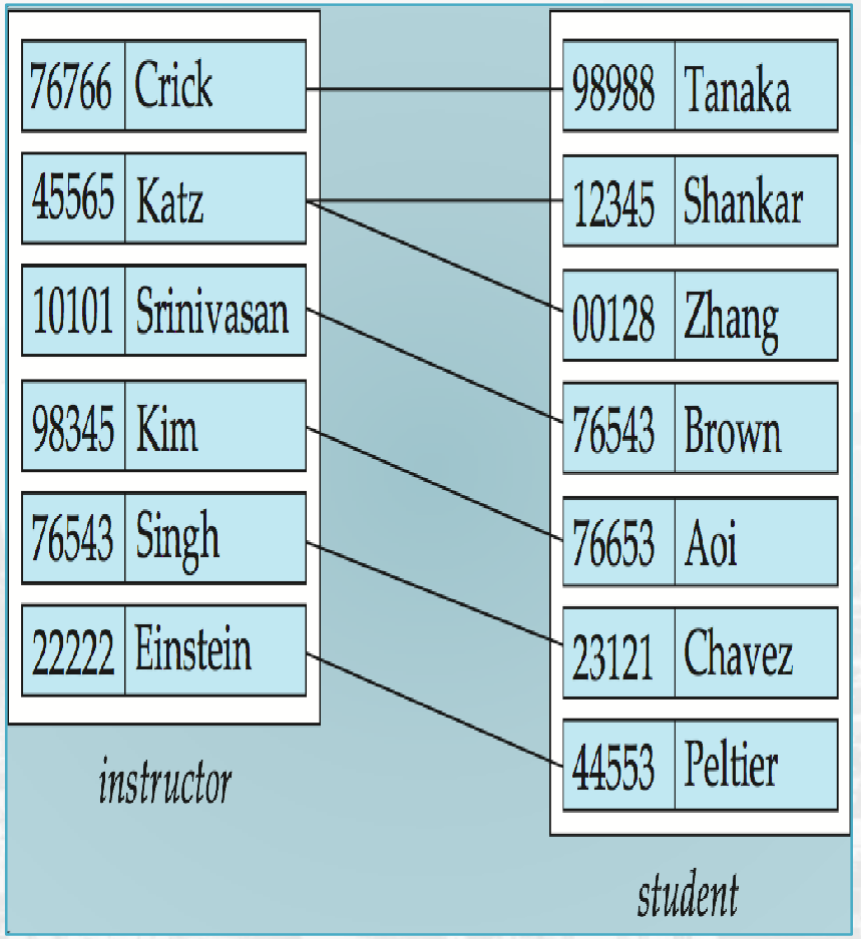
A **relationship set** is a mathematical relation among $n \geq 2$ entities, each taken from entity sets

$$\{(e_1, e_2, \ldots e_n) \mid e_1 \in E_1, e_2 \in E_2, \ldots, e_n \in E_n\}$$

where $(e_1, e_2, \ldots, e_n)$ is a relationship

- Example:

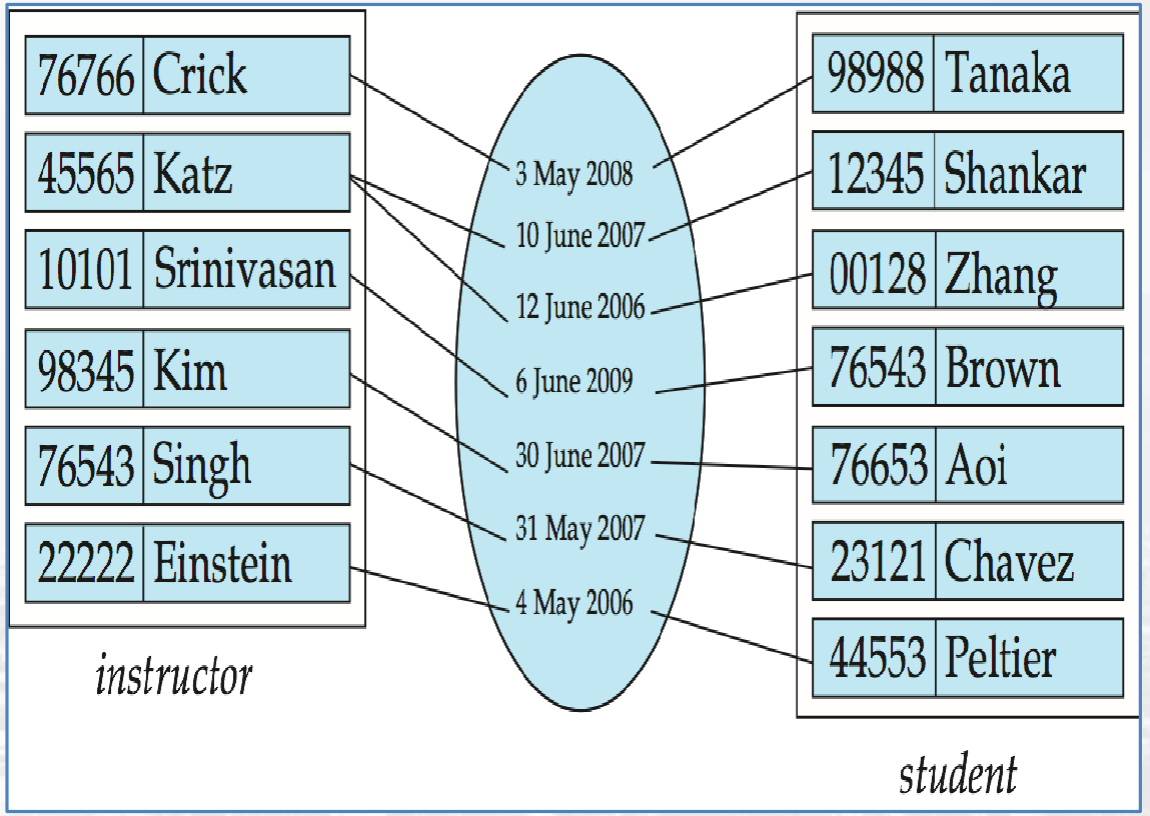
$$(44553,22222) \in advisor$$



Relationship Sets

- An **attribute** can also be property of a relationship set.

➢For instance, the *advisor* relationship set between entity sets *instructor* and *student* may have the attribute *date* which tracks when the student started being associated with the advisor

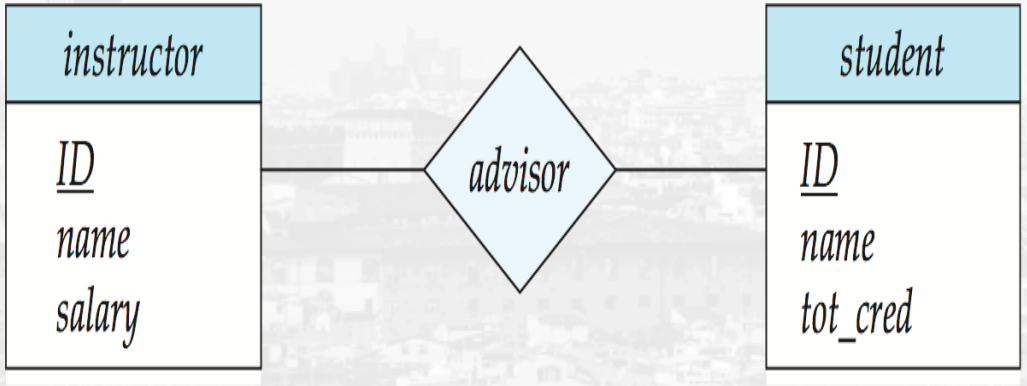


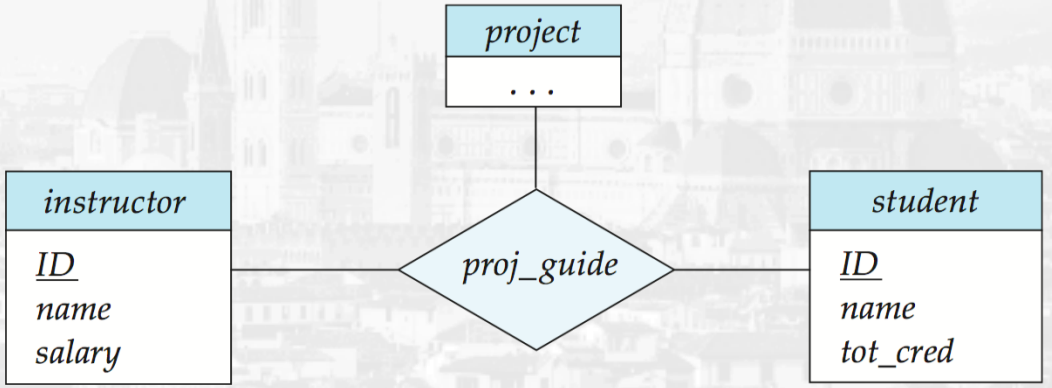
Degree of a Relationship Set

- **Binary relationship**
  - involve two entity sets (or degree two).
  - most relationship sets in a database system are binary.

Relationships between more than two entity sets are rare. Most relationships are binary.

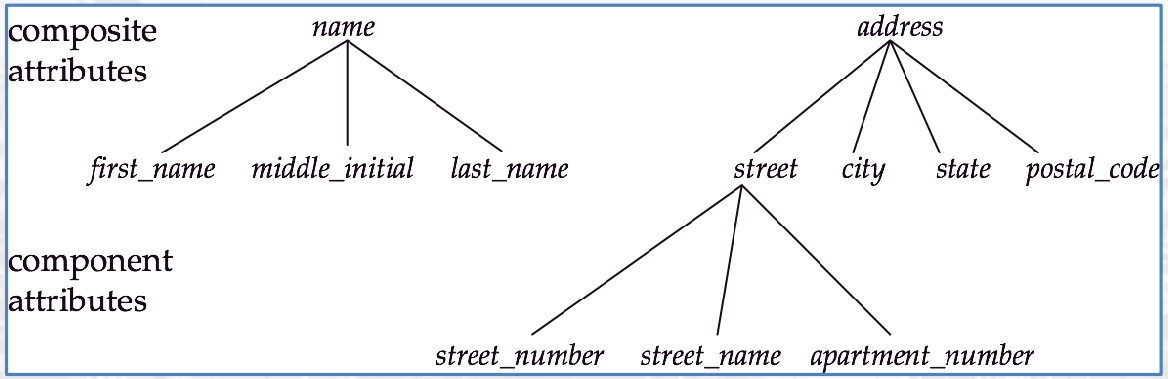
- **Ternary relationship**
  - Example: *students* work on research *projects* under the guidance of an *instructor*.
  - relationship *proj_guide* is a ternary relationship between *instructor*, *student*, and *project*





Attributes

- **Attribute types**
  - **Simple** and **composite** attributes.
  - **Single-valued** and **multivalued** attributes.
    Example: multivalued attribute: *phone_numbers*
  - **Derived** attributes
    Can be computed from other attributes
    Example: age, given date_of_birth



- An entity is represented by a set of attributes, that is descriptive properties possessed by all members of an entity set.
  - Example:

    instructor = (*ID*, *name*, *street*, *city*, *salary*)

    course= (*course_id*, *title*, *credits*)

- **Domain** – the set of permitted values for each attribute

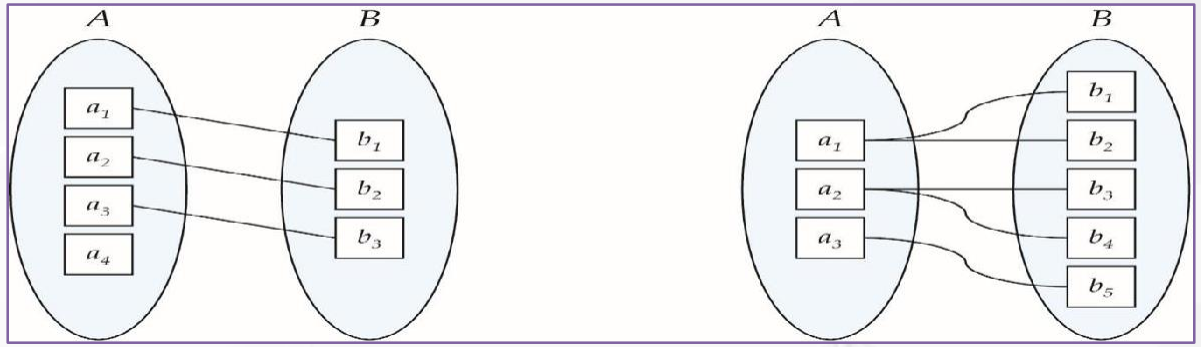
Mapping Cardinality Constraints

Express the number of entities to which another entity can be associated via a relationship set.

Most useful in describing binary relationship sets.

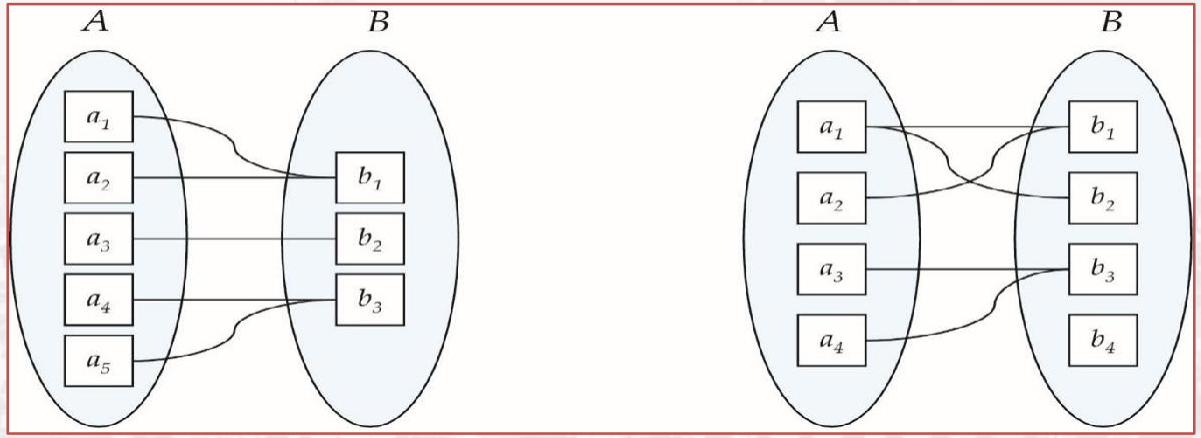
For a binary relationship set the mapping cardinality must be one of the following types:

- ***One to one***
- ***One to many***
- ***Many to one***
- ***Many to many***



One to one

One to many



Many to one

Many to many

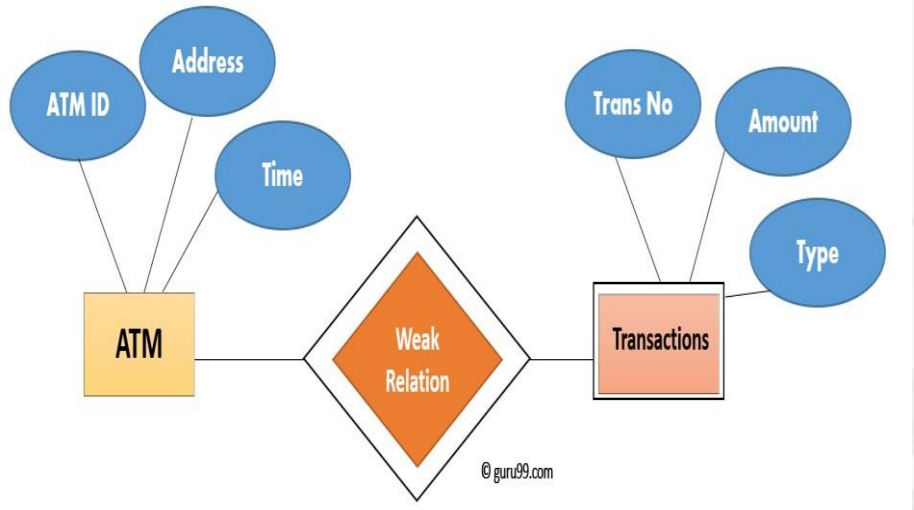
Keys for Relationship Sets

The combination of primary keys of the participating entity sets forms a super key of a relationship set.

- **(*s_id, i_id*) is the super key of *advisor***
- *NOTE: this means **a pair of entity sets can have at most one relationship in a particular relationship set.***
  - Example: if we wish to track multiple meeting dates between a student and her advisor, we cannot assume a relationship for each meeting. We can use a multivalued attribute though

Must consider the mapping cardinality of the relationship set when deciding what are the candidate keys

Entity Relationship Diagram

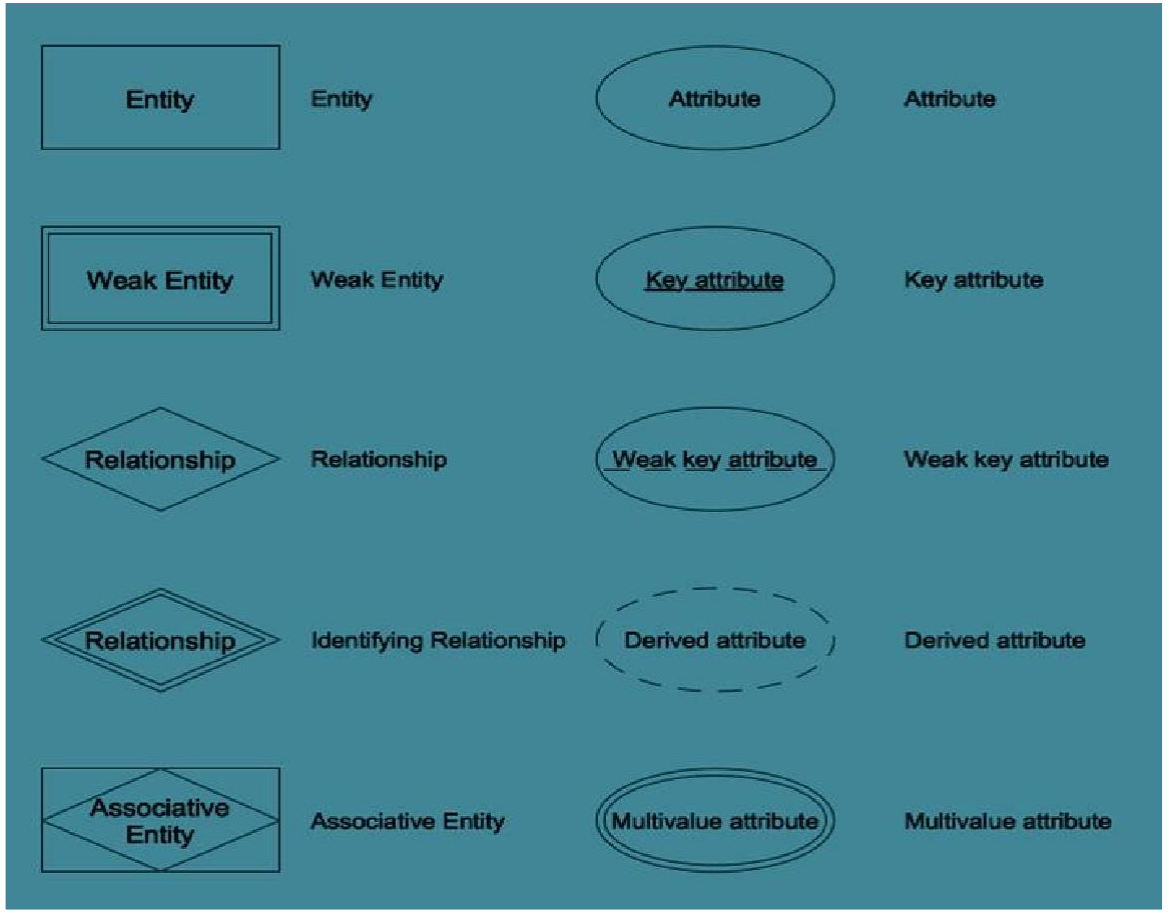


Entity-Relationship Diagram

It is a graphical Representation of ER Model

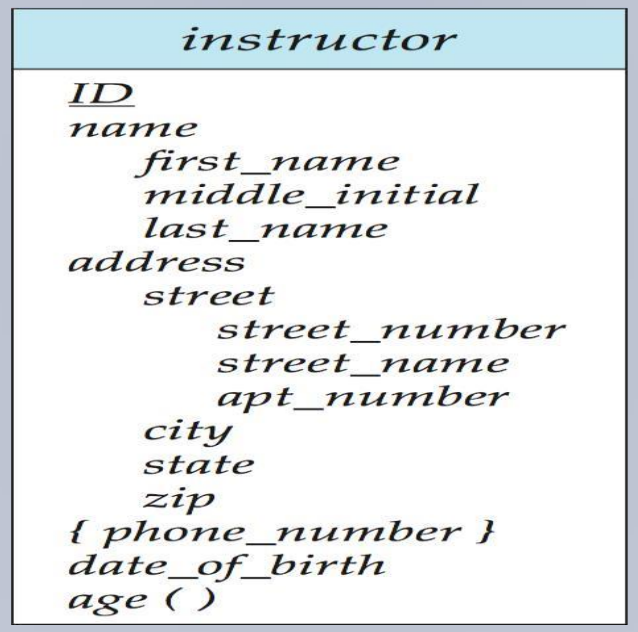
Basic Notations :

- Rectangles represent entity sets.
- Diamonds represent relationship sets.
- Attributes listed inside entity rectangle
- Underline indicates primary key attributes

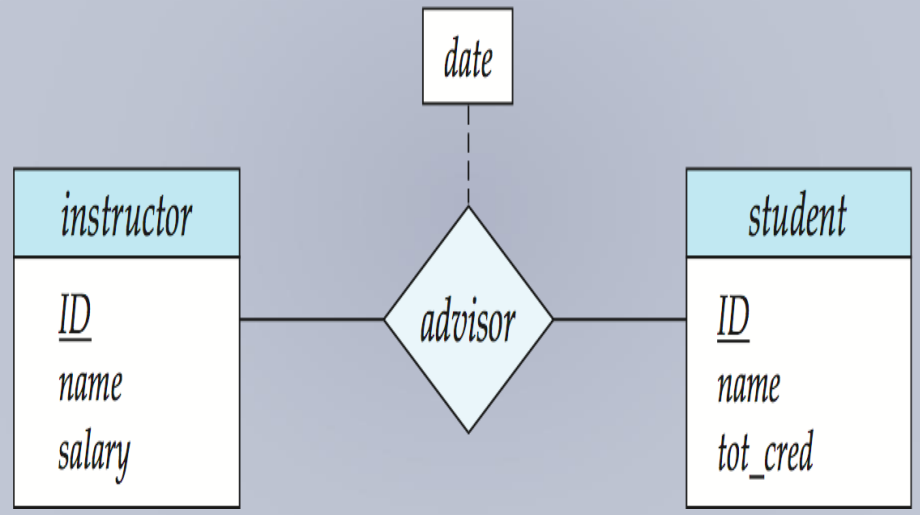


Entity Relationship Diagram Continued

- **Entity With Composite, Multivalued, and Derived Attributes**



- **Relationship Sets with Attributes**

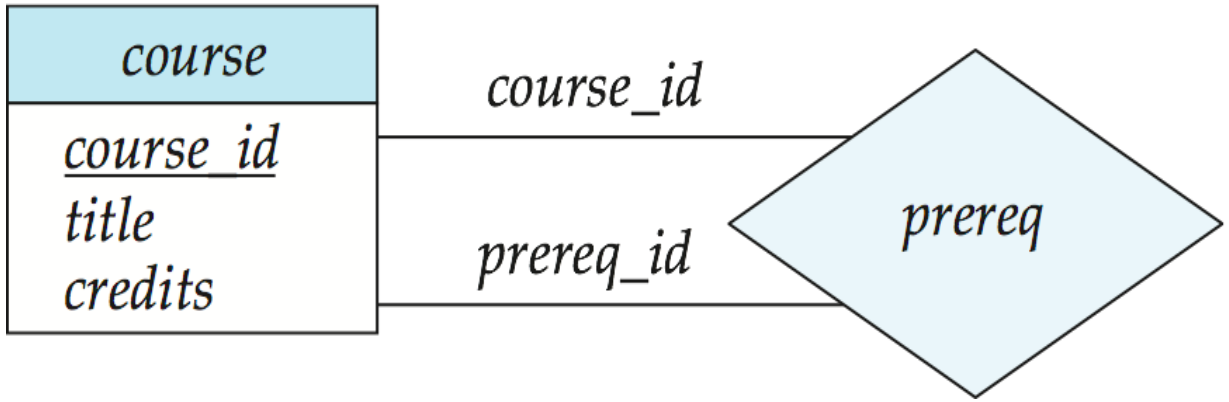


Entity Relationship Diagram Continued

Roles

- Entity sets of a relationship need not be distinct.
- Each occurrence of an entity set plays a "role" in the relationship.

The labels "*course_id*" and "*prereq_id*" are called **roles**.



Entity Relationship Diagram Continued

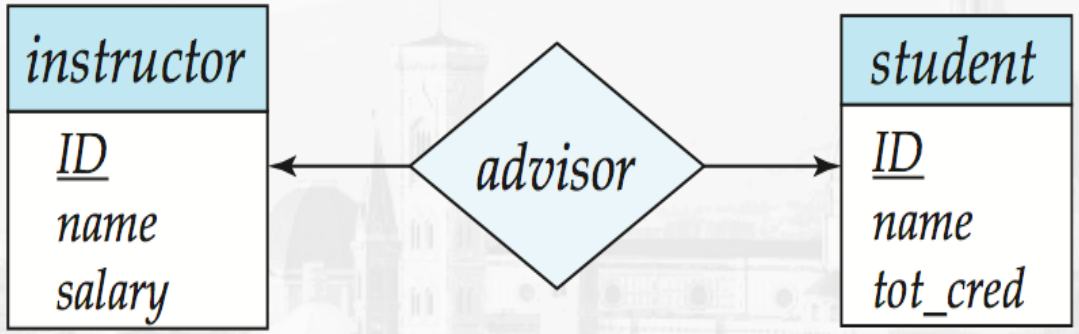
Cardinality Constraints

➢We express cardinality constraints by drawing either a directed line (→), signifying "one," or an undirected line (—), signifying "many," between the relationship set and the entity set.

1. One-to-One Relationship

one-to-one relationship between an *instructor* and a *student*

- an instructor is associated with at most one student via *advisor*
- and a student is associated with at most one instructor via *advisor*

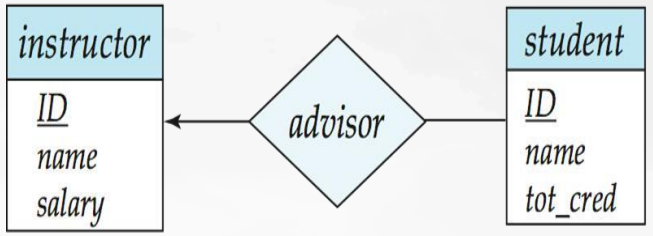


Entity Relationship Diagram Continued

b. One-to-Many Relationship

A one-to-many relationship between an *instructor* and a *student*

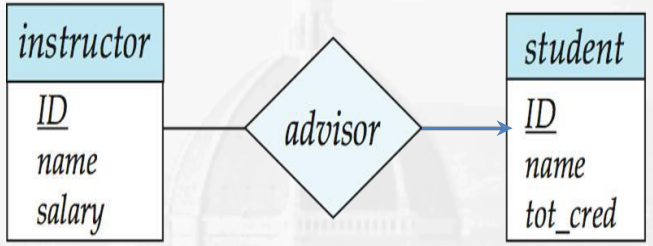
- a an instructor is associated with several (including 0) students via *advisor*
- student is associated with at most one instructor via advisor



c. Many-to-One Relationship

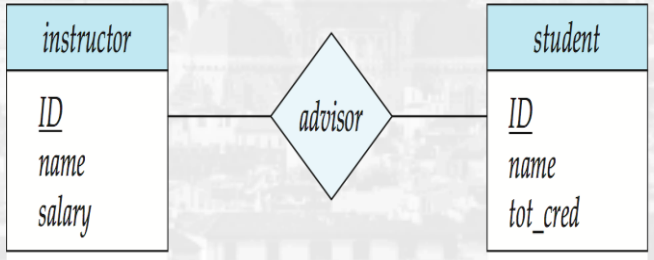
In a many-to-one relationship between an *instructor* and a *student*,

- an instructor is associated with at most one student via *advisor*,
- and a student is associated with several (including 0) instructors via *advisor*



d. Many-to Many Relationship

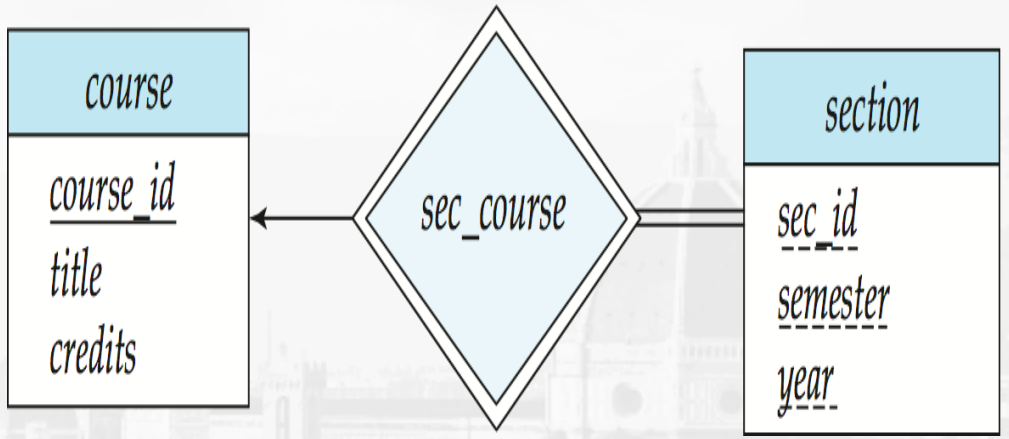
- An instructor is associated with several (possibly 0) students via *advisor*
- A student is associated with several (possibly 0) instructors via *advisor*



Entity Relationship Diagram Continued

Participation of an Entity Set in a Relationship Set

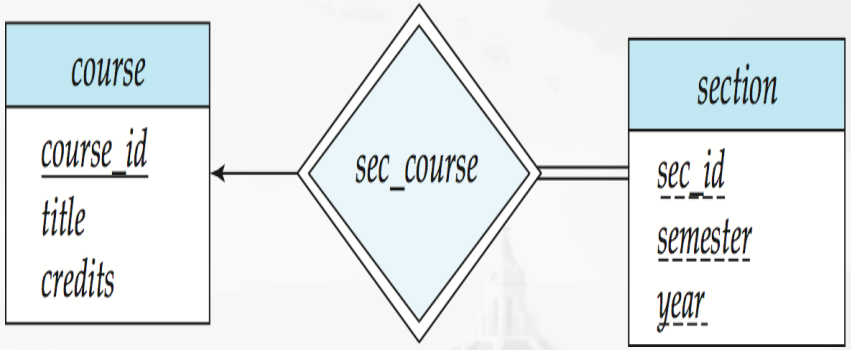
- **Total participation** (**indicated by double line**): every entity in the entity set participates in at least one relationship in the relationship set
  - E.g., participation of *section* in *sec_course* is total
    - every *section* must have an associated course
- **Partial participation**: some entities may not participate in any relationship in the relationship set
  - Example: participation of *instructor* in *advisor* is partial



Entity Relationship Diagram : Weak Entity Sets

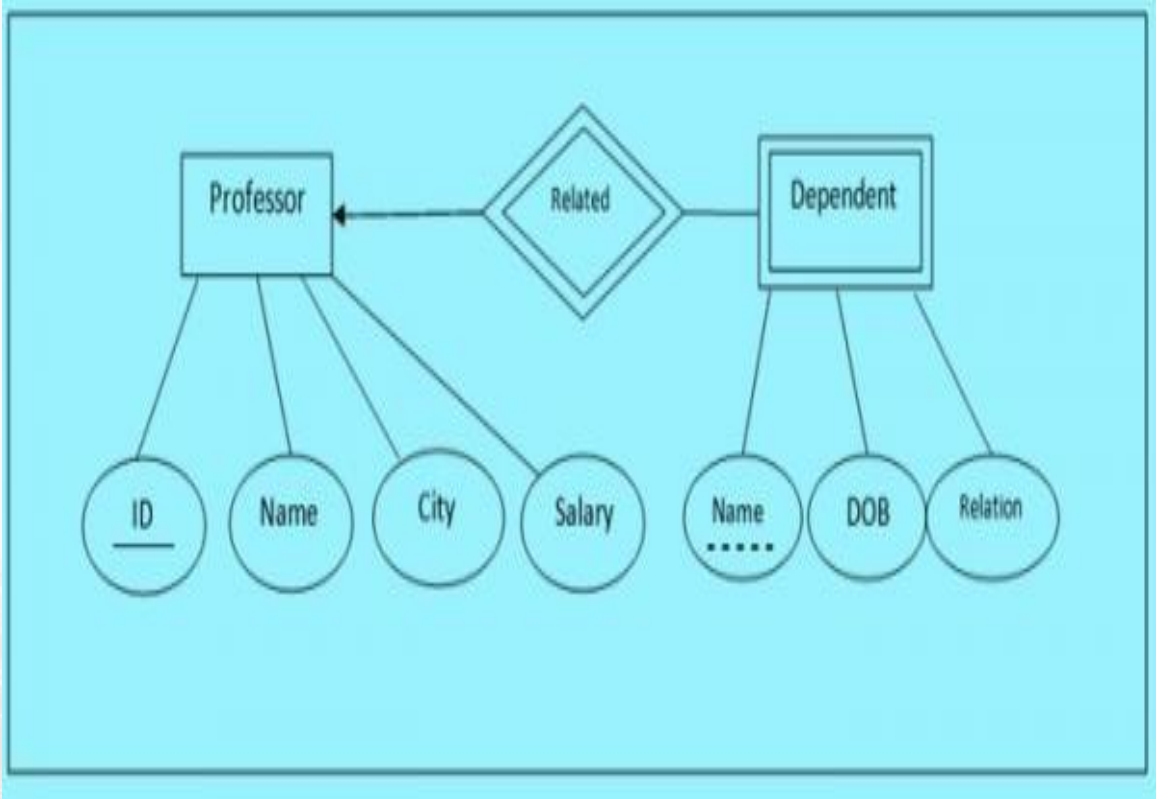
- An entity set that does not have a primary key is referred to as a **weak entity set**.
- The existence of a weak entity set depends on the existence of a **identifying entity set**
  - It must relate to the identifying entity set via a total, one-to-many relationship set from the identifying to the weak entity set
  - **Identifying relationship depicted using a double diamond**
- The **discriminator** (*or partial key)* of a weak entity set is the set of attributes that distinguishes among all the entities of a weak entity set.

The primary key of a weak entity set is formed by the primary key of the strong entity set on which the weak entity set is existence dependent, plus the weak entity set's discriminator.



- We underline the discriminator of a weak entity set with a dashed line.
- We put the identifying relationship of a weak entity in a double diamond.
- Primary key for *section* – (*course_id*, *sec_id*, *semester*, *year*)

Example of Strong and Weak Entity



Strong Entity :
Professor(ID,Name,City,Salary)

Weak Entity :
Dependent(Name,DOB,Relation)

The Dependent Entity will share the ID attribute of Professor.

Resultant Schema :

Dependent(ID,Name,DOB,Relation)

The primary key for Weak Entity Dependent will be ID + Name as Name is the discriminator attribute.

E-R Diagram Example

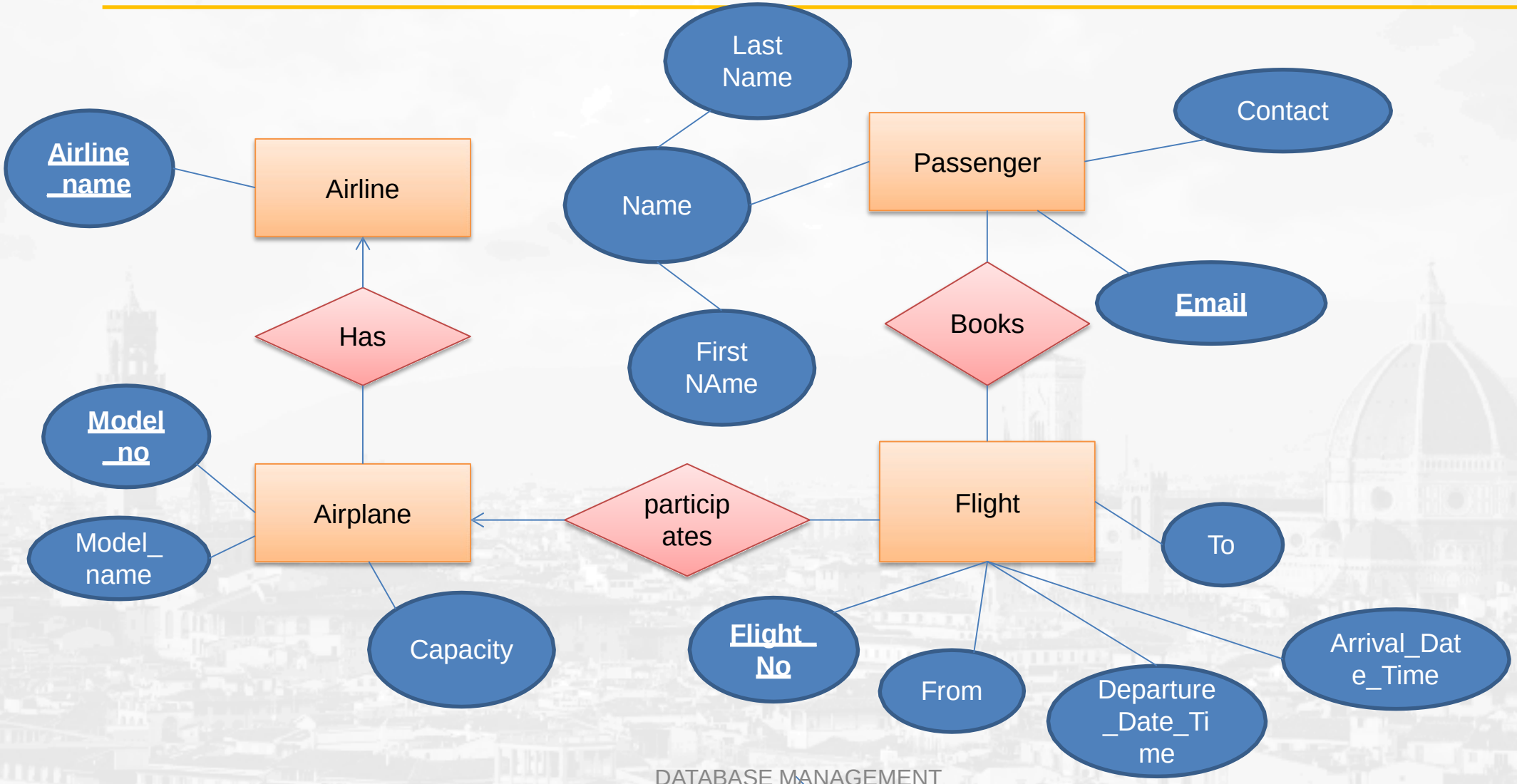
Question : **Design an ER Diagram for Airline Reservation scenario given below :**

The flight database stores details about an airline's fleet, flights, and seat bookings.

Consider the following scenario:

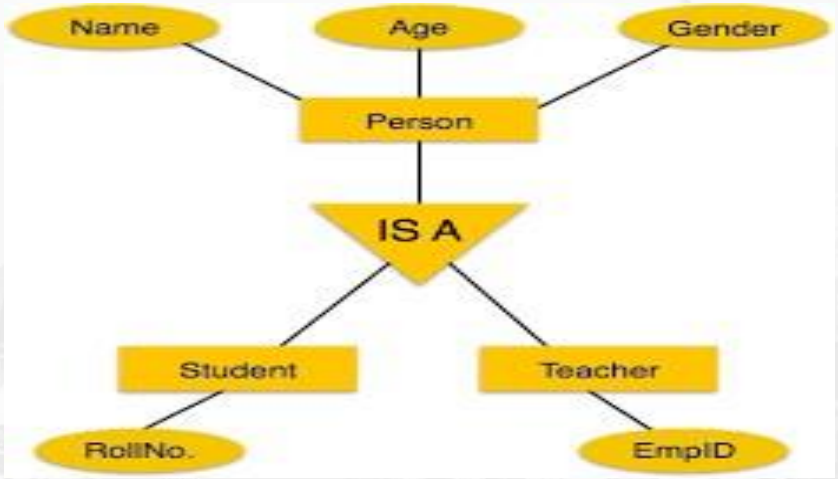
- The airline has one or more airplanes.
- An airplane has a model number, a registration number, and the capacity to take one or more passengers.
- An airplane flight has a unique flight number, a departure airport, a destination airport, a departure date and time, and an arrival date and time.
- Each flight is carried out by a single airplane.
- A passenger has given names(first name, last name),contact and a unique email address.
- Passengers can book seats on flights.

Flight Reservation ERD



Extended ER Features :

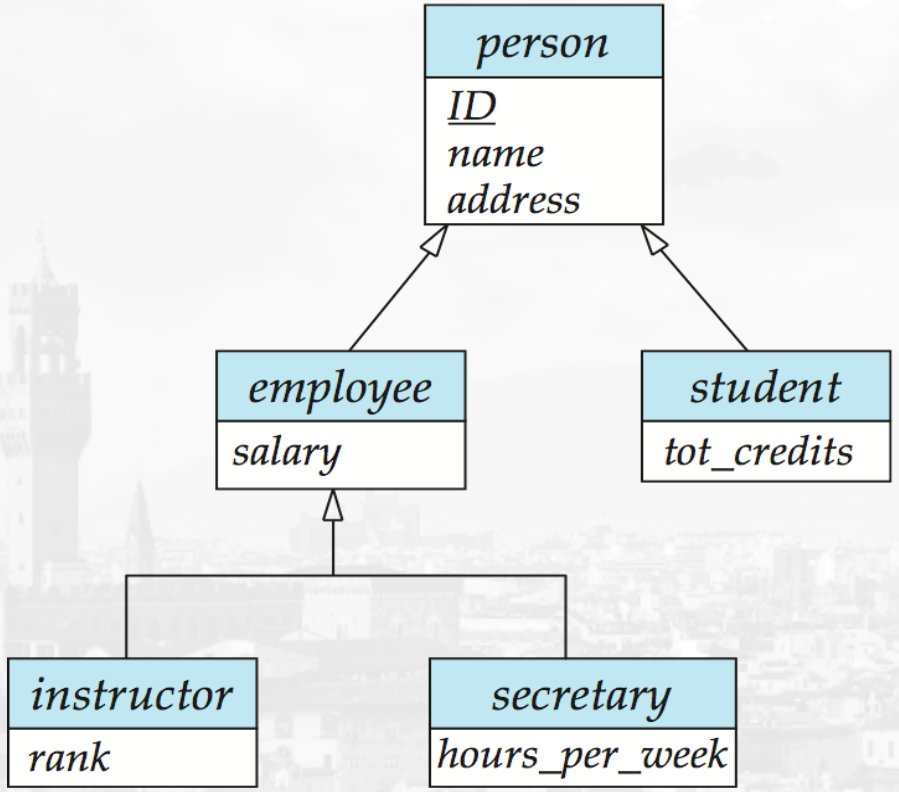
- Specialization
- Generalization
- Aggregation

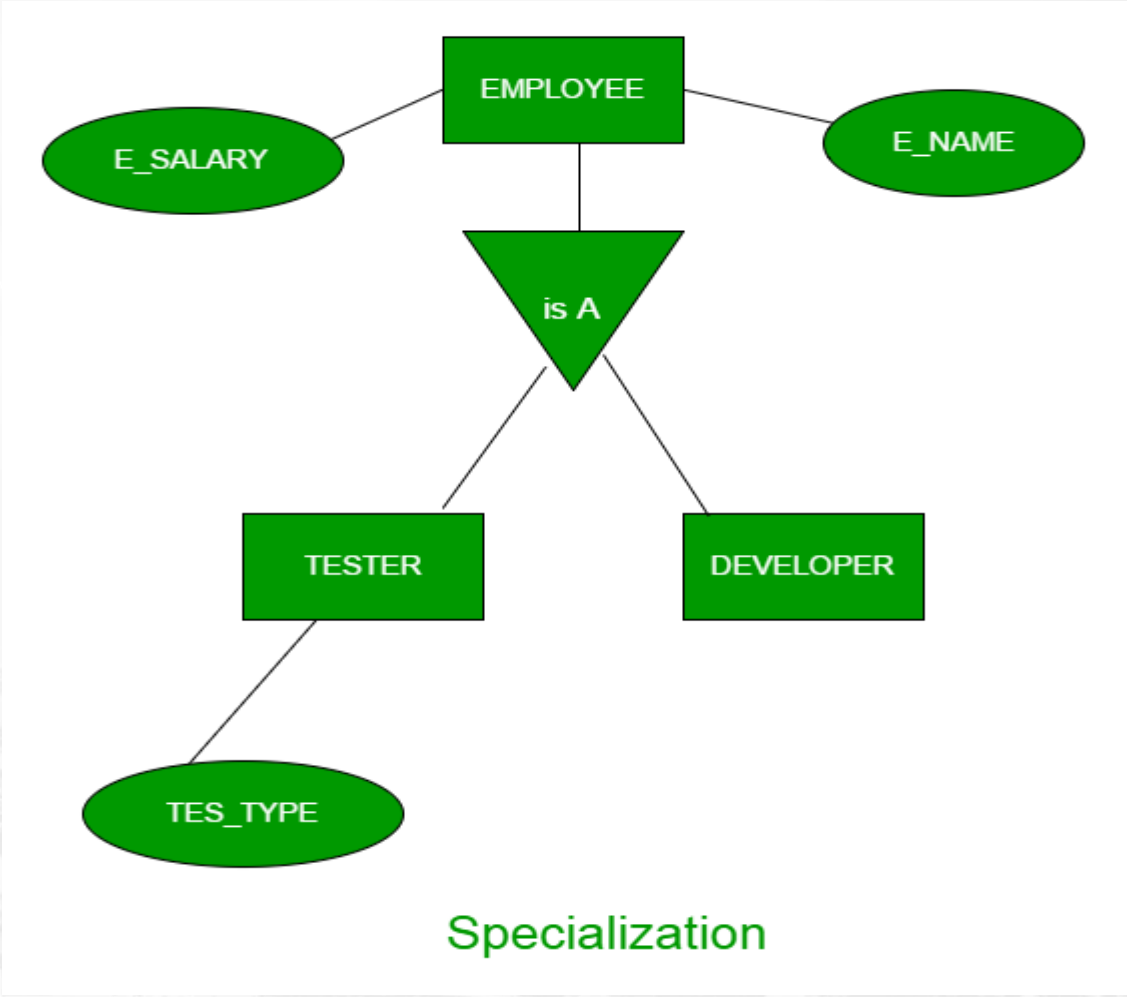


Extended ER Features : Specialization

- Top-down design process; we designate subgroupings within an entity set that are distinctive from other entities in the set.
- These subgroupings become lower-level entity sets that have attributes or participate in relationships that do not apply to the higher-level entity set.
- Depicted by a *triangle* component labeled IS A (E.g., *instructor* "is a" *person*).
- **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.

Specialization : Example

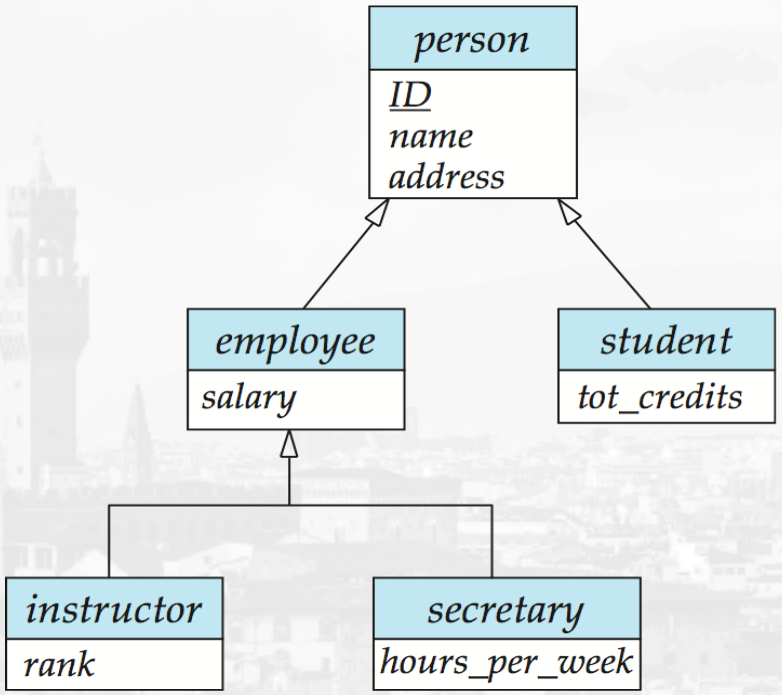


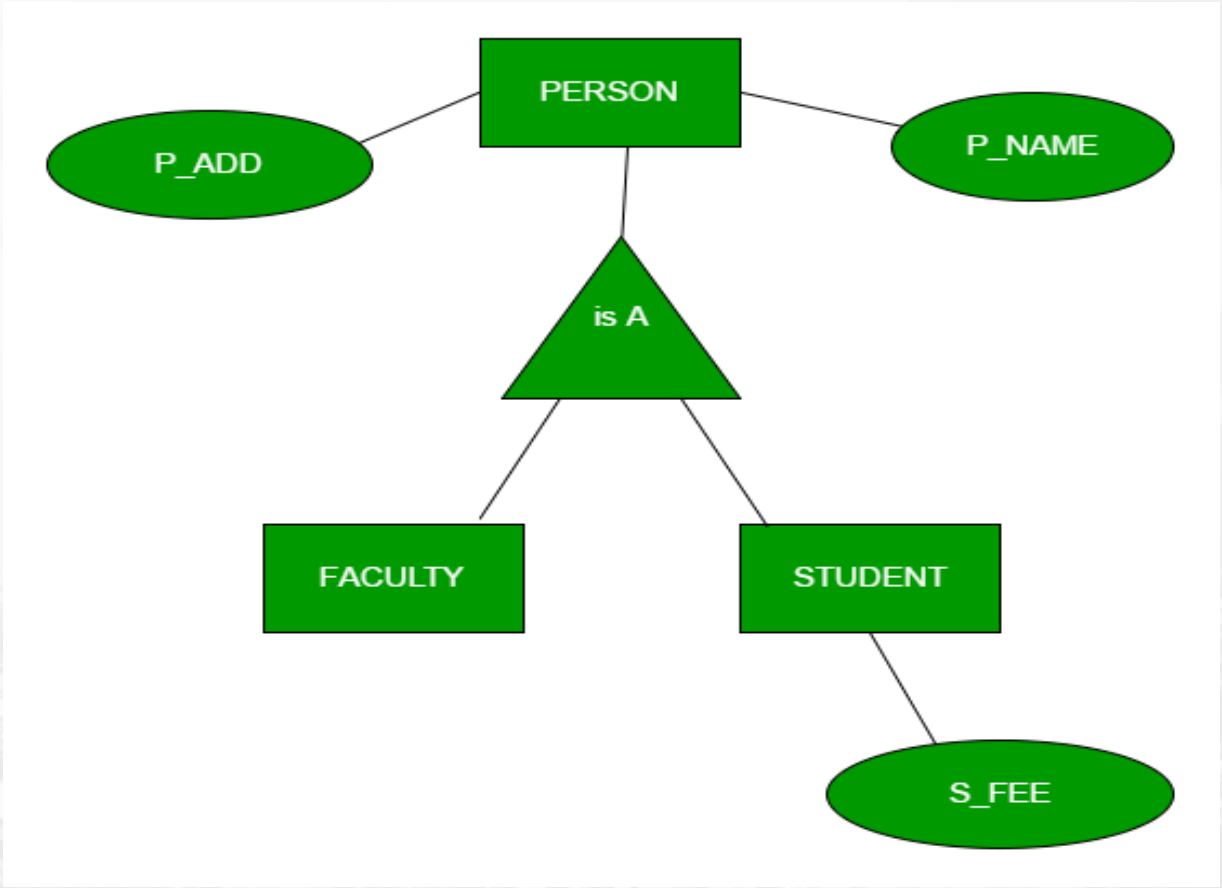


Extended ER Features : Generalization

- **A bottom-up design process** – combine a number of entity sets that share the same features into a higher-level entity set.
- Specialization and generalization are simple inversions of each other; they are represented in an E-R diagram in the same way.
- The terms specialization and generalization are used interchangeably.

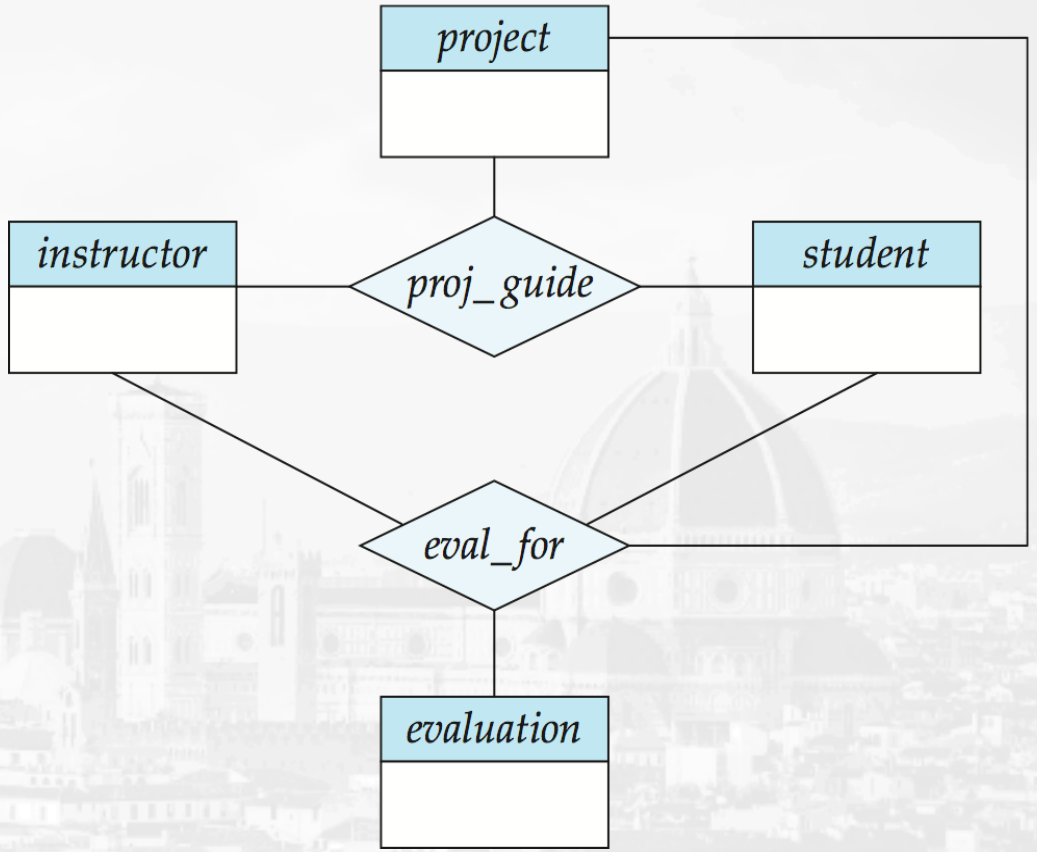
Generalization : Example





Extended ER Features : Aggregation

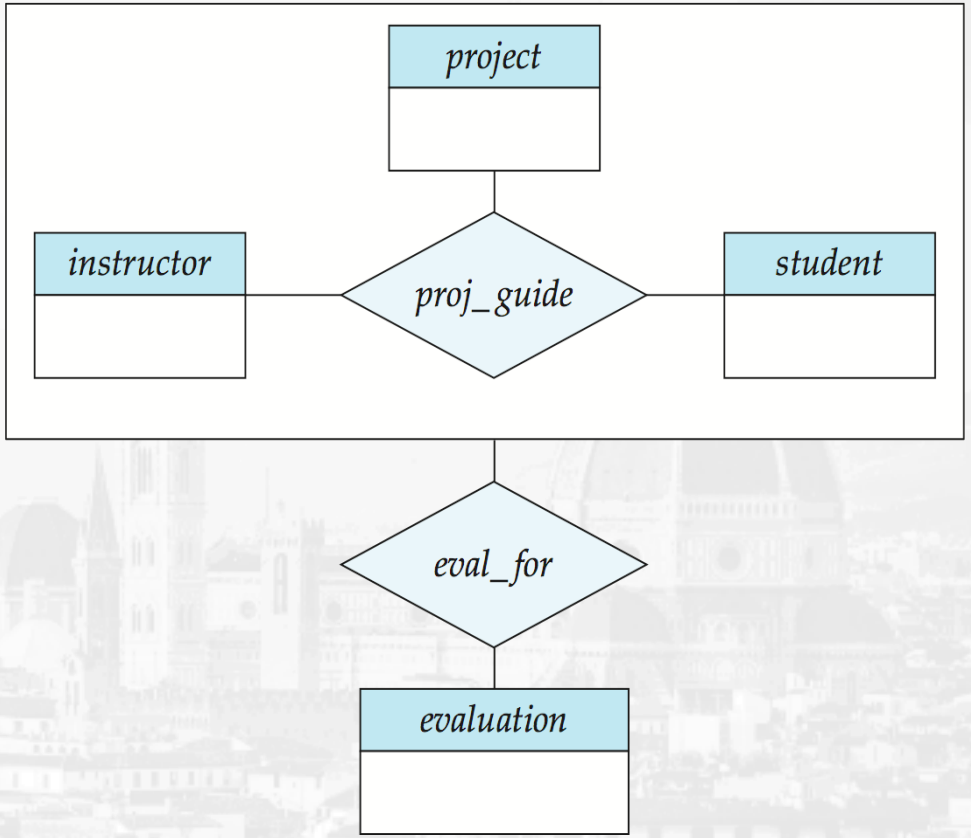
- Consider the ternary relationship *proj_guide*, which we saw earlier
- Suppose we want to record evaluations of a student by a guide on a project.
- Relationship sets *eval_for* and *proj_guide* represent overlapping information
  - Every *eval_for* relationship corresponds to a *proj_guide* relationship
  - However, some *proj_guide* relationships may not correspond to any *eval_for* relationships
    - So we can't discard the *proj_guide* relationship



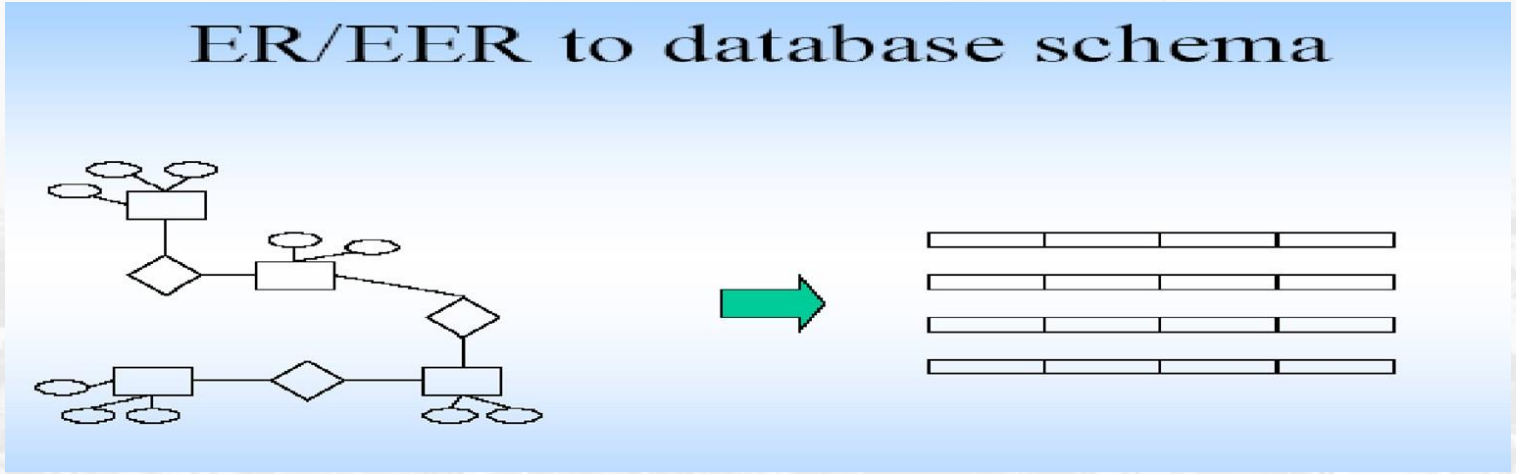
Extended ER Features : Aggregation

- Eliminate this redundancy via *aggregation*
  - Treat relationship as an abstract entity
  - Allows relationships between relationships
  - Abstraction of relationship into new entity

- Without introducing redundancy, the following diagram represents:
  - A student is guided by a particular instructor on a particular project
  - A student, instructor, project combination may have an associated evaluation



Reduction of ER Diagram to Relation Schemas

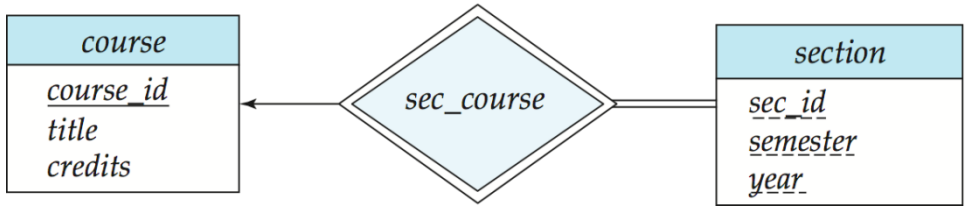


Reduction to Relation Schemas

- Entity sets and relationship sets can be expressed uniformly as *relation schemas* that represent the contents of the database.
- A database which conforms to an E-R diagram can be represented by a collection of schemas.
- For each entity set and relationship set there is a unique schema that is assigned the name of the corresponding entity set or relationship set.
- Each schema has a number of columns (generally corresponding to attributes), which have unique names

Reduction to Relation Schemas

Representing Entity Sets with Simple Attributes



A strong entity set reduces to a schema with the same attributes

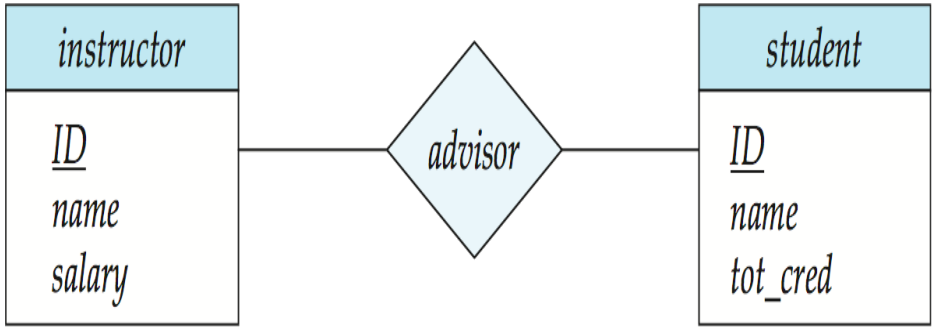
course(course_id,title,credits)

course_id	title	credits

A weak entity set becomes a table that includes a column for the primary key of the identifying strong entity set

Course_id	Sec_id	sem	year

Representing Relationship Sets :



A many-to-many relationship set is represented as a schema with attributes for the primary keys of the two participating entity sets, and any descriptive attributes of the relationship set.

Example: schema for relationship set *advisor*

advisor = (*s_id, i_id*)

S_id	i_id

Reduction to Relation Schema

Redundancy of schemas:

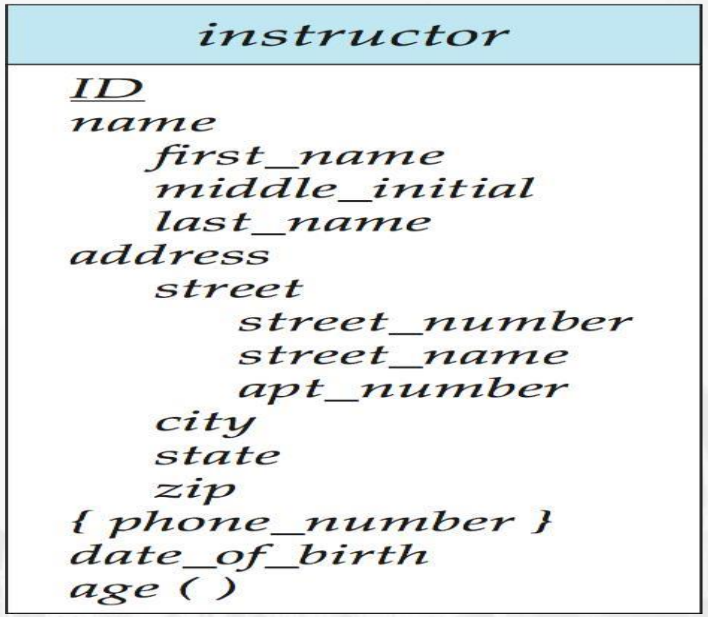
- Many-to-one and one-to-many relationship sets that are total on the many-side can be represented by adding an extra attribute to the “many” side, containing the primary key of the “one” side.

Example: Instead of creating a schema for relationship set *inst_dept*, add an attribute *dept_name* to the schema arising from entity set *instructor*

Reduction to Relation Schema

Representing Composite and Multi-valued Attributes

- **Composite attributes** are flattened out by creating a separate attribute for each component attribute
  - Example: given entity set *instructor* with composite attribute *name* with component attributes *first_name* and *last_name* the schema corresponding to the entity set has two attributes *name_first_name* and *name_last_name*
    - *Prefix omitted if there is no ambiguity*
- Ignoring multivalued attributes, extended instructor schema is :



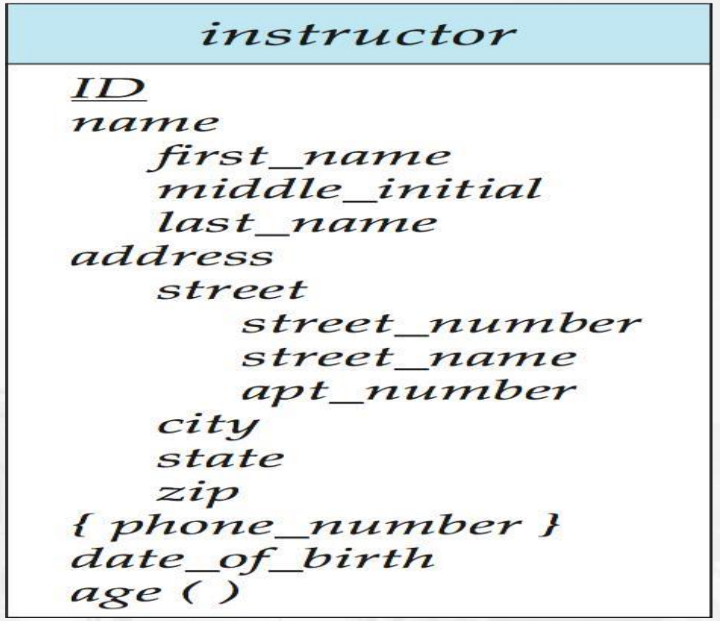
ID	First_name	Middle_initial	Last_name	Street_number	Street_name	Apt_number	city	state	zip	Date_of_birth

Reduction to Relation Schema

Representing Composite and Multi-valued Attributes

A **multivalued attribute** *M* of an entity *E* is represented by a separate schema *EM*

- Schema *EM* has attributes corresponding to the primary key of *E* and an attribute corresponding to multivalued attribute *M*
- Example: Multivalued attribute *phone_number* of *instructor* is represented by a schema:
  inst_phone = (*ID*, *phone_number*)
- Each value of the multivalued attribute maps to a separate tuple of the relation on schema *EM*
  - For example, an *instructor* entity with primary key 22222 and phone numbers 456-7890 and 123-4567 maps to two tuples:
    (22222, 456-7890) and (22222, 123-4567)

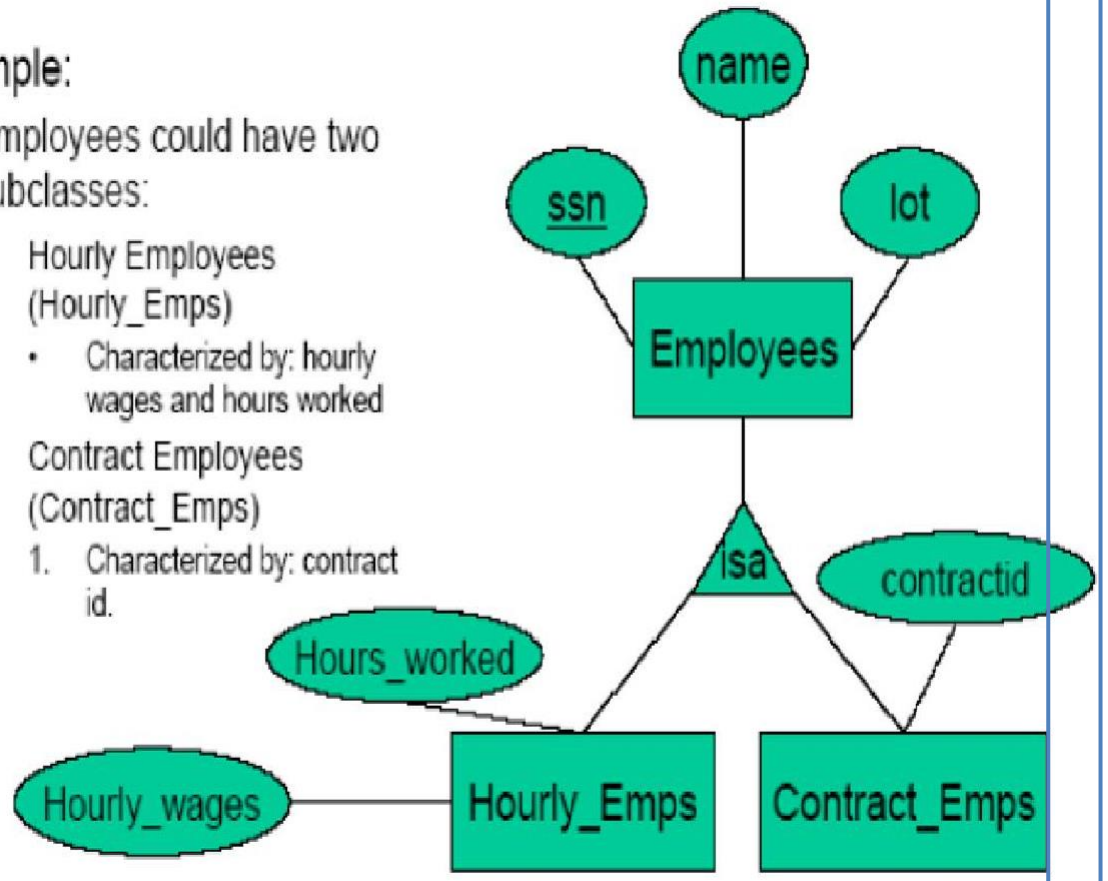


ID	Phone_number
22222	456-7890
22222	123-4567

Extended ER Features Reduction to Relation Schemas

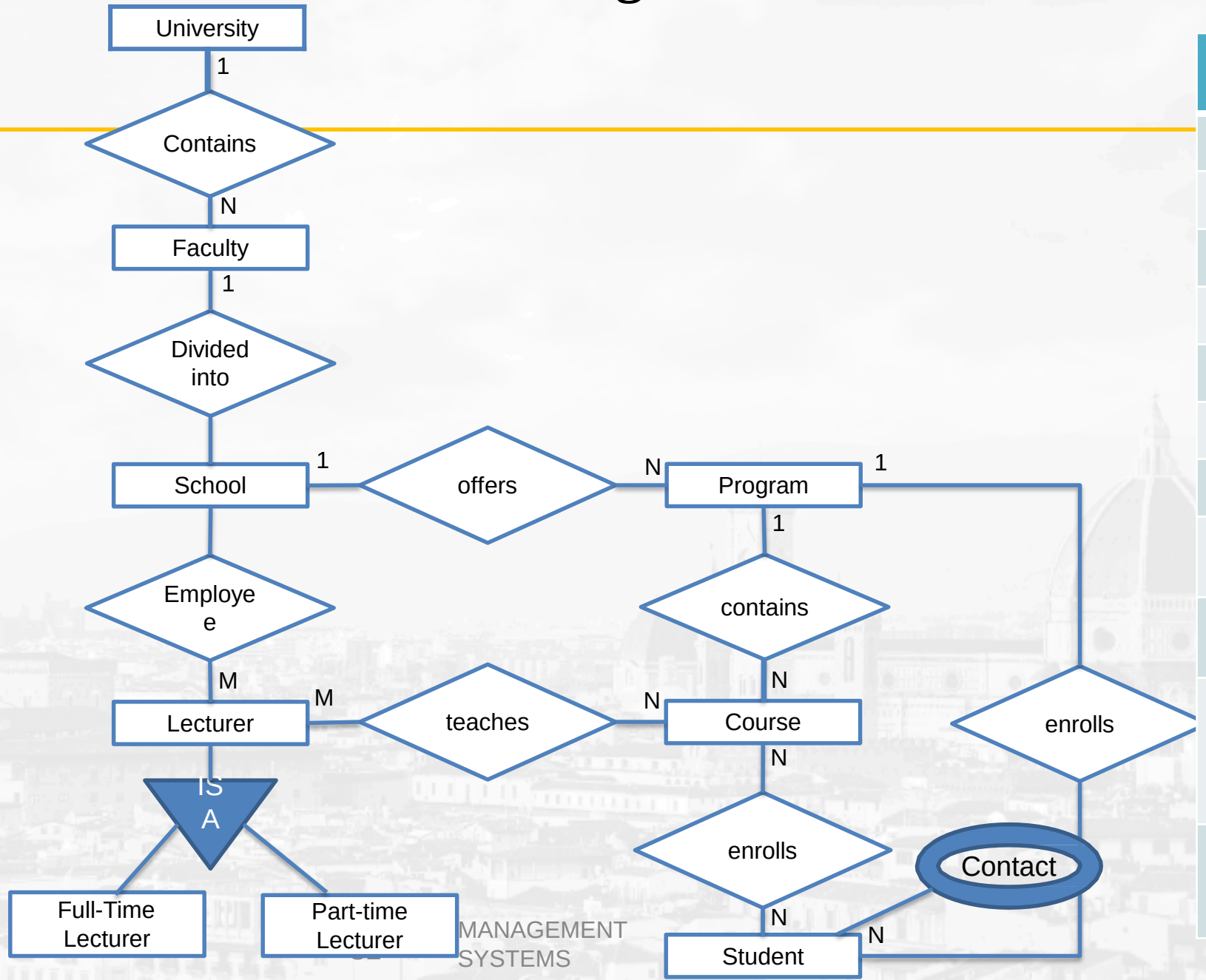
Example:

- Employees could have two subclasses:
  1. Hourly Employees (Hourly_Emps)
     - Characterized by: hourly wages and hours worked
  2. Contract Employees (Contract_Emps)
     1. Characterized by: contract id.



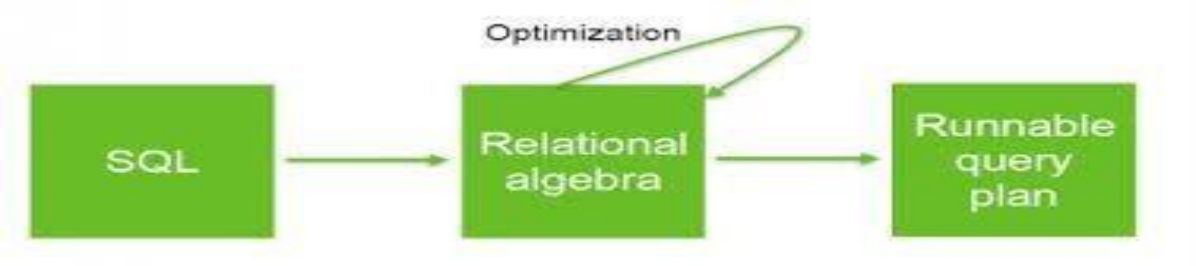
- Two approaches
  - Three tables: Employees, Hourly_Emps and Contract_Emps.
    - *Hourly_Emps*: Every employee is recorded in Employees. For hourly emps, extra info recorded in Hourly_Emps (*hourly_wages*, *hours_worked*, *ssn*);
    - We must delete Hourly_Emps tuple if referenced Employees tuple is deleted).
    - Queries involving all employees easy, those involving just Hourly_Emps require a join to get some attributes.
- Alternative: Just Hourly_Emps and Contract_Emps.
  - *Hourly_Emps*: *ssn*, *name*, *lot*, *hourly_wages*, *hours_worked*.
  - Each employee must be in one of these two subclasses.

Example for Reduction of ER Diagram to Relation schemas



Reduced Schema-Names
University
Faculty
School
Program
Lecturer
Course
Student
Full-time_Lecturer
Part-time_Lecturer
Lect_Course (Associative Entity due to M:M)
Student_Course (Associative Entity due to M:M)

Relational Algebra



Relational Algebra

- What is “algebra ?
- Mathematical model consisting of:
  - *Operands* --- Variables or values;
  - *Operators* --- Symbols denoting procedures that construct new values from a given values
- **Relational Algebra :** is an algebra whose operands are relations and operators are designed to do the most commons things that we need to do with relations

- **Basic Relational Algebra Operations**:
  - Select σ
  - Project $\prod$
  - Union $\cup$
  - Set Difference (or Subtract or minus) –
  - Cartesian Product X
  - Natural Join

Relational Algebra: Select Operation

- **Notation:** $\sigma_p(r)$

p is called the selection predicate

➢ Defined as:

$$\sigma_p(\boldsymbol{r}) = \{t \mid t \in \boldsymbol{r} \textbf{ and } p(t)\}$$

➢ Where p is a formula in propositional calculus consisting of terms connected by : ∧ (**and**), ∨(**or**), ¬(**not**)
Each term is one of:

<attribute> *op* <attribute> or <constant>

where *op* is one of: =, ≠, >, ≥. <. ≤

Example of selection:

Account(account_number, branch_name,balance)

$\sigma_{\textit{branch-name}=\text{“Perryridge”}}(\textit{account})$

A	B	C	D
α	α	1	7
α	β	5	7
β	β	12	3
β	β	23	10

Relation *r*

A	B	C	D
α	α	1	7
β	β	23	10

$\sigma_{A=B \wedge D > 5}(r)$

Relational Algebra: Project Operation

- **Notation:** $\prod_{A1, A2, \ldots, Ak}(r)$

where A_1, A_2 are attribute names and r is a relation.

➢The result is defined as the relation of k columns obtained by erasing the columns that are not listed

➢Duplicate rows removed from result, since relations are sets

E.g. to eliminate the *branch-name* attribute of *account*

$$\prod_{account\text{-}number,\ balance}(account)$$

➢If relation Account contains 50 tuples, how many tuples contains $\prod_{account\text{-}number,\ balance}(account)$?

Example of Project Operation :

A	B	C
α	10	1
α	20	1
β	30	1
β	40	2

Relation *r*

A	C
α	1
α	1
β	1
β	2

$\prod_{A,C}(r)$

=

A	C
α	1
β	1
β	2

That is, the projection of a relation on a set of attributes is a **set of tuples**

Relational Algebra: Union Operation

Notation: $r \cup s$

- Consider relational schemas:

  Depositor(customer_name, account_number)

  Borrower(customer_name, loan_number)

- For $r \cup s$ to be valid.

  1. *r, s* must have the same number of attributes

  2. The attribute domains must be *compatible* (e.g., 2nd column of *r* deals with the same type of values as does the 2nd column of *s*)

- Find all customers with either an account or a loan

  $\prod_{customer\text{-}name}(depositor) \cup \prod_{customer\text{-}name}(borrower)$

Example of Union:

A	B
α	1
α	2
β	1

Relation r

A	B
α	2
β	3

Relation s

A	B
α	1
α	2
β	1
β	3

$r \cup s$

Relational Algebra: Set Difference Operation

- **Notation** : $r - s$

Set differences must be taken between *compatible* relations.

- r and s must have the same number of attributes
- attribute domains of r and s must be compatible

Example of Set Difference:

A	B
α	1
α	2
β	1

Relation r

A	B
α	2
β	3

Relation s

A	B
α	1
β	1

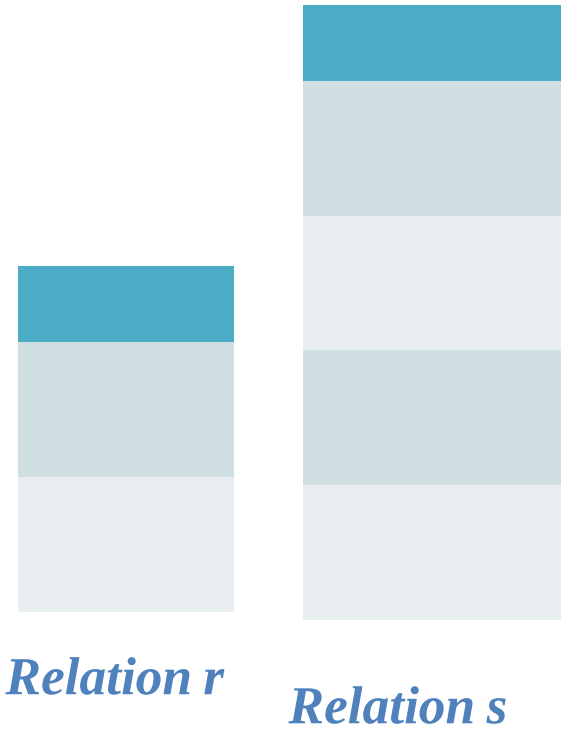
r-s

Relational Algebra: Cartesian Product Operation

- **Notation** : $r \times s$

Assume that attributes of r(R) and s(S) are disjoint. (That is, $R \cap S = \emptyset$).

If attributes of *r(R)* and *s(S)* are not disjoint, then renaming must be used.



$r \times s$

A	B	C	D	E
α	1	α	10	a
α	1	β	10	a
α	1	β	20	b
α	1	γ	10	b
β	2	α	10	a
β	2	β	10	a
β	2	β	20	b
β	2	γ	10	b

Relational Algebra: Natural Join Operation

- **Notation :** $r \bowtie s$
- We can perform a Natural Join only if there is at least one common attribute that exists between two relations
- The common attributes must have the same name and domain.
- Natural join acts on those matching attributes where the values of attributes in both the relations are same.
- **It avoids duplication of columns while providing the result as compared to other joins/cartesian-product.**

A	B
α	1
β	2

Relation r

A	C
α	10
β	30

Relation s

$r \bowtie s$

A	B	C
α	1	10
β	2	30

Relational Algebra Operators

Symbol (Name)	Example of Use
σ (Selection)	$\sigma_{\text{salary}>=85000}(instructor)$
	Return rows of the input relation that satisfy the predicate.
Π (Projection)	$\Pi_{ID,salary}(instructor)$
	Output specified attributes from all rows of the input relation. Remove duplicate tuples from the output.
$\bowtie$ (Natural join)	$instructor \bowtie department$
	Output pairs of rows from the two input relations that have the same value on all attributes that have the same name.
$\times$ (Cartesian product)	$instructor \times department$ Output all pairs of rows from the two input relations (regardless of whether or not they have the same values on common attributes)
$\cup$ (Union)	$\Pi_{name}(instructor) \cup \Pi_{name}(student)$
	Output the union of tuples from the two input relations.

References

Text Books:

- Abraham Silberschatz, Henry F. Korth and S. Sudarshan, Database System Concepts 6th Ed, McGraw Hill, 2010.
- Elmasi, R. and Navathe, S.B., “Fundamentals of Database Systems”, 4th
- Ed., Pearson Education.

Reference Books :

- Ramakrishnan, R. and Gherke, J., “Database Management Systems”, 3rd Ed., McGraw-Hill.
- Connally T, Begg C.,”Database Systems”,Pearson Education

End