Database Management Systems

Examination scheme: Marks-50 [Continuous Assessment]

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.

Module 1- Introduction to Database Management Systems and Data Modeling

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, Armstrong Axiom's, Functional Dependency, Normal Forms (1 NF—5 NF)

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.

MIT-WPU
॥ विश्वशान्तिर्ध्रुवं ध्रुवा ॥

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 A program that controls the execution of application programs
- **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 ?

MIT-WPU
॥ विश्वशान्तिर्ध्रुवं ध्रुवा ॥

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.

MIT-WPU
॥ विश्वशान्तिर्ध्रुवं ध्रुवा ॥

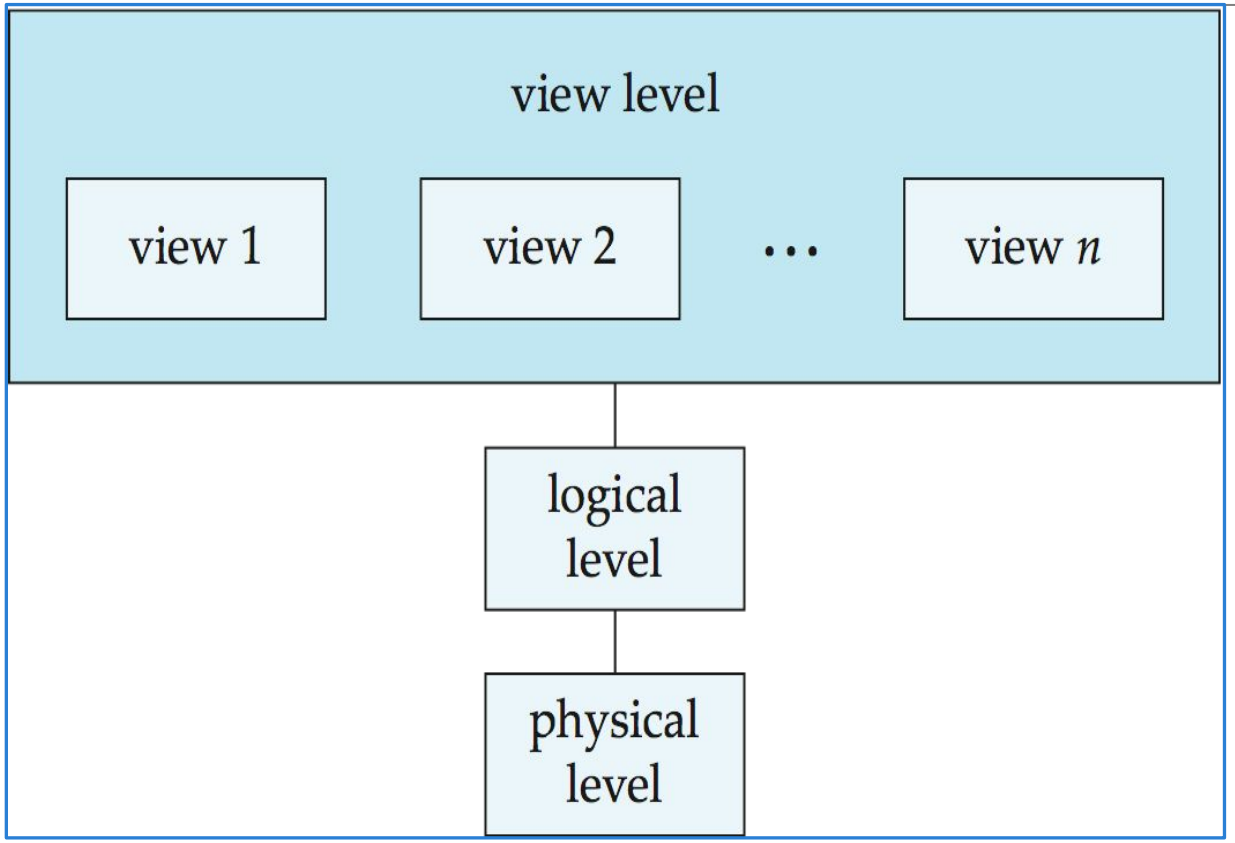
Motivation for Database Management Systems

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.

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.

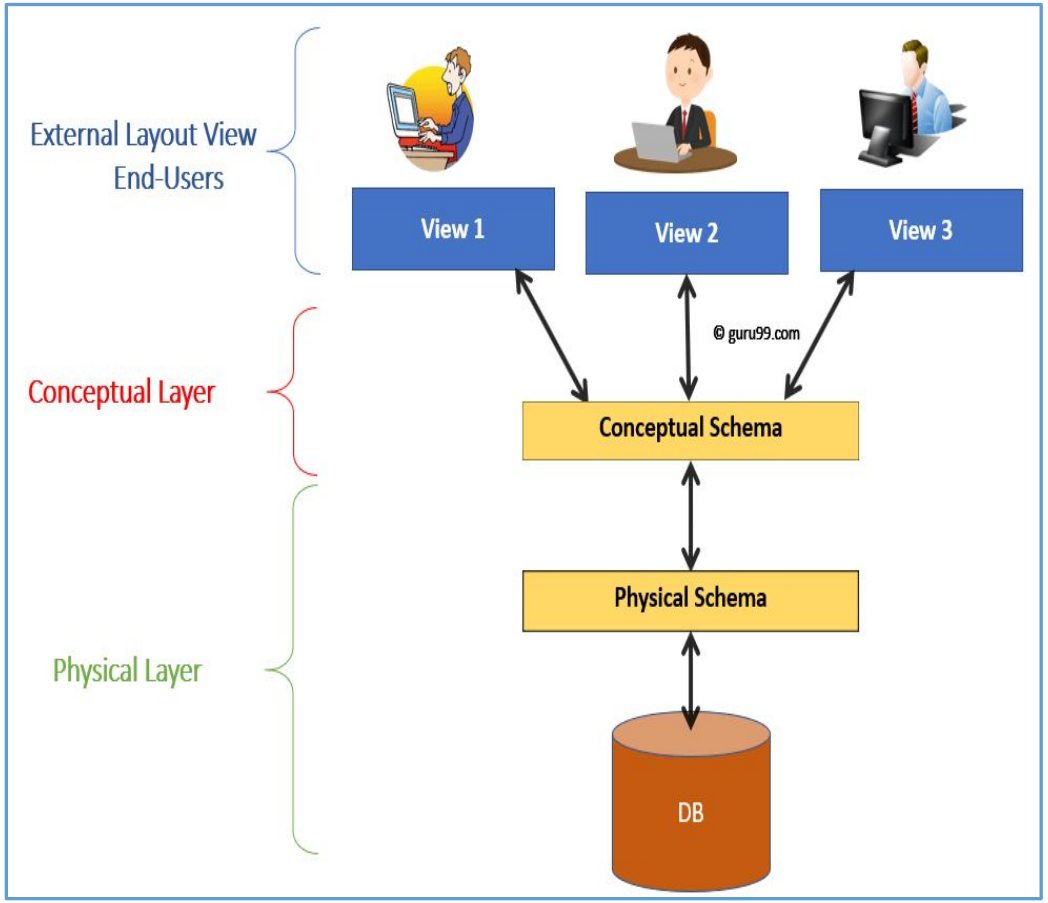
  type *instructor* = **record**
  ID : string;
  name : string;
  dept_name : string;
  salary : integer;
  end;

- **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

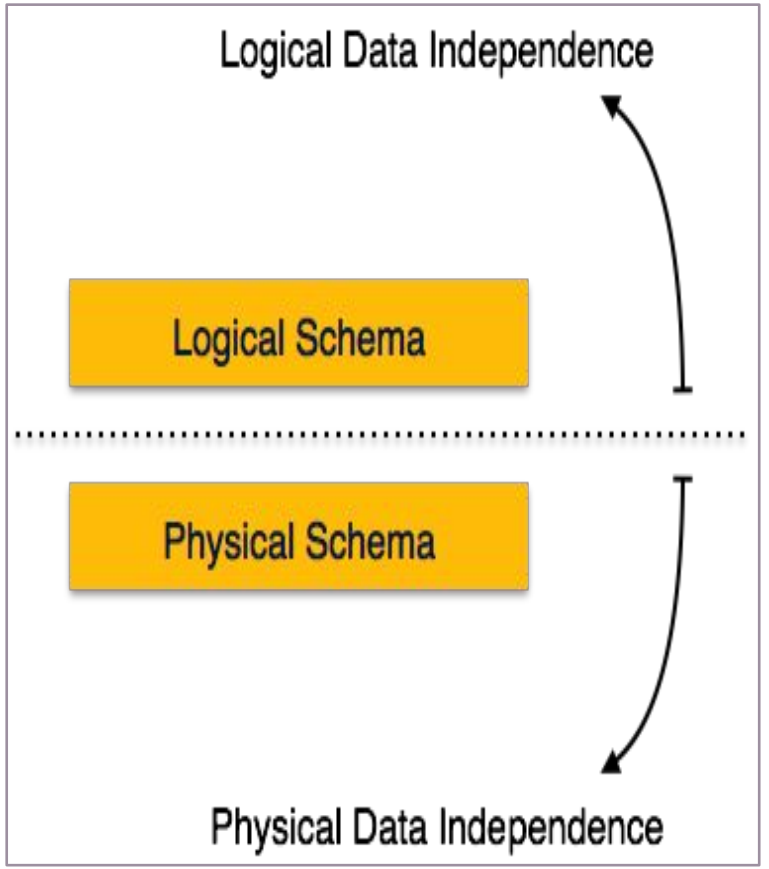
- **Instance – the actual content of the database at a particular point in time**
  - Analogous to the **value of a variable.**



Data Independence

Types of Data Independence :

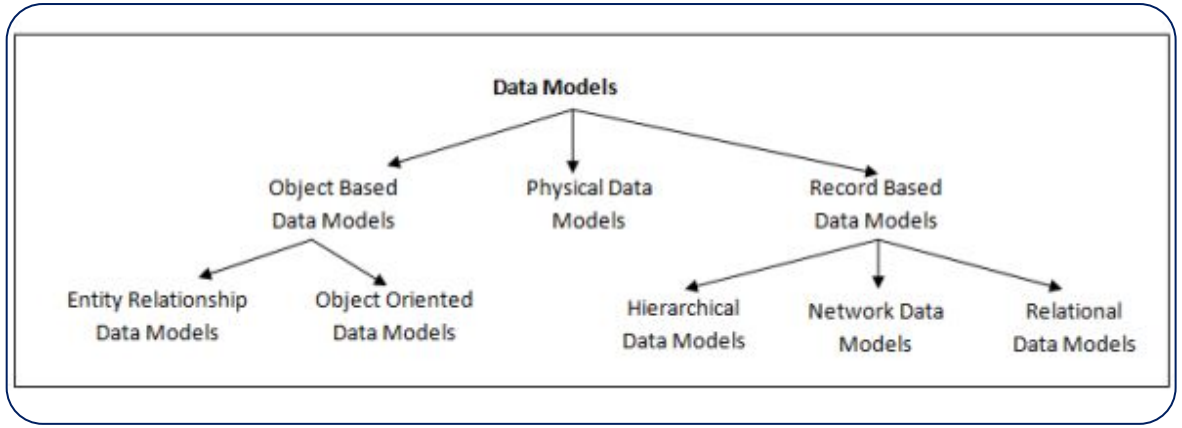
- **Physical Data Independence** : the ability to modify the physical schema without changing the logical schema
  - Applications depend on the logical schema
  - In general, the interfaces between the various levels and components should be well defined so that changes in some parts do not seriously 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.

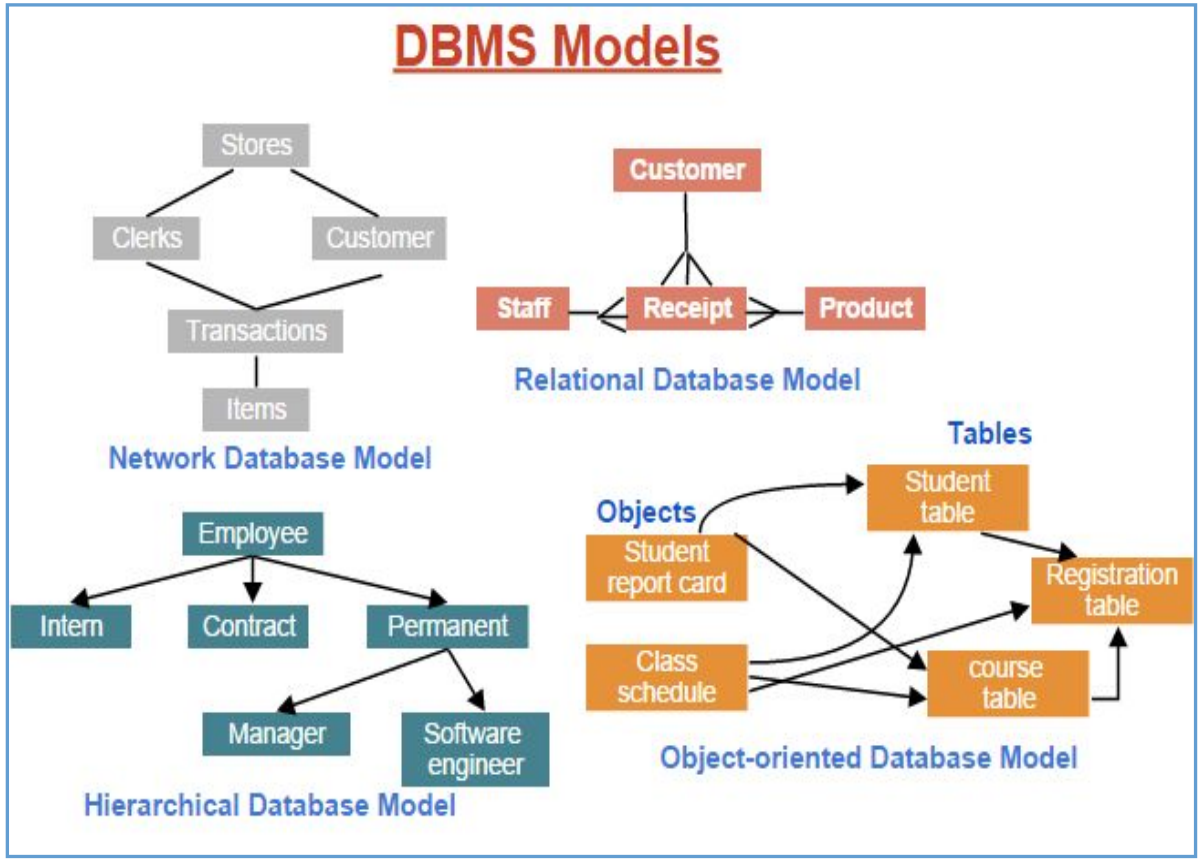


Data Models

A collection of tools for describing :

- Data
- Data relationships
- Data semantics
- Data constraints





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

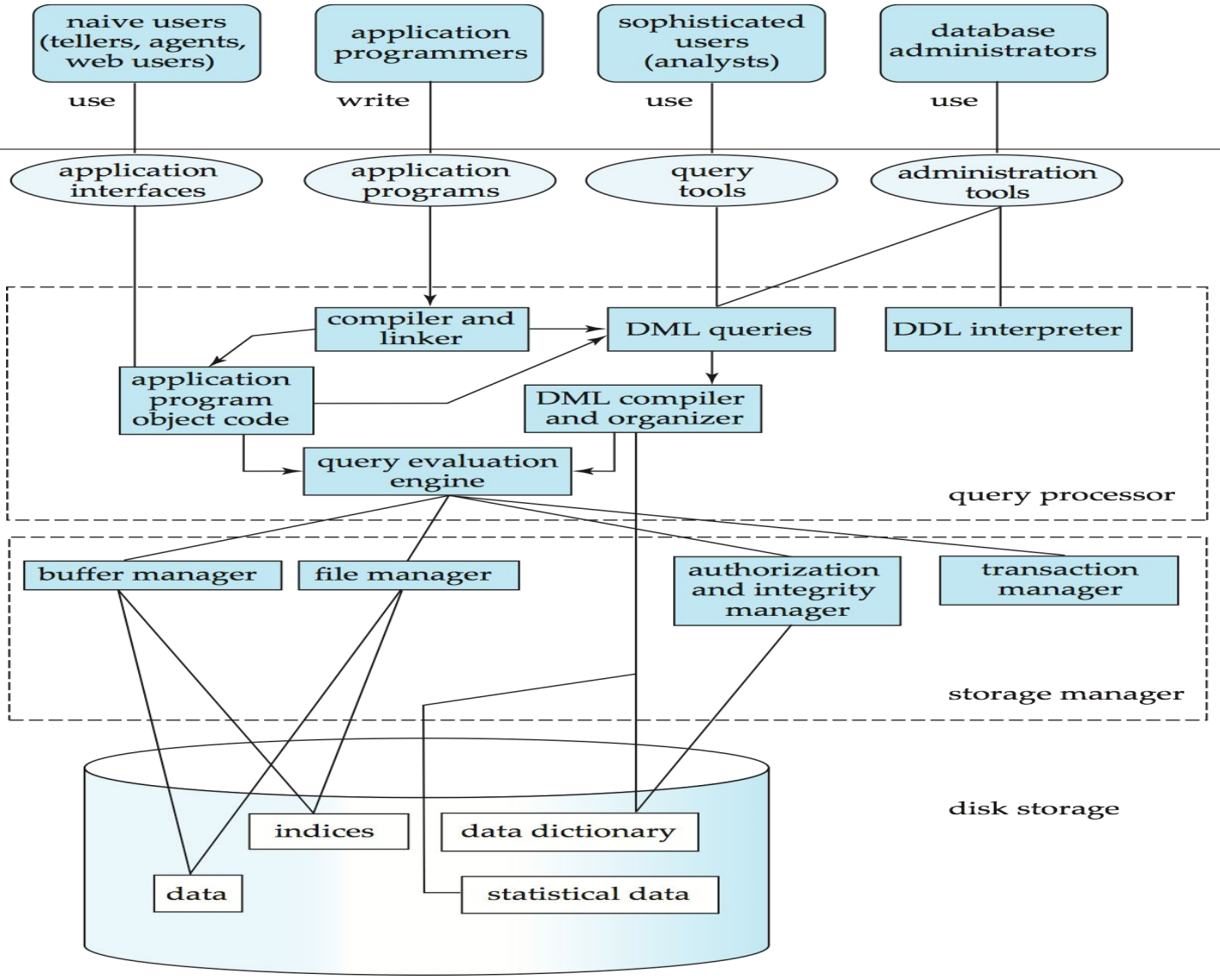
- DML also known as query language

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

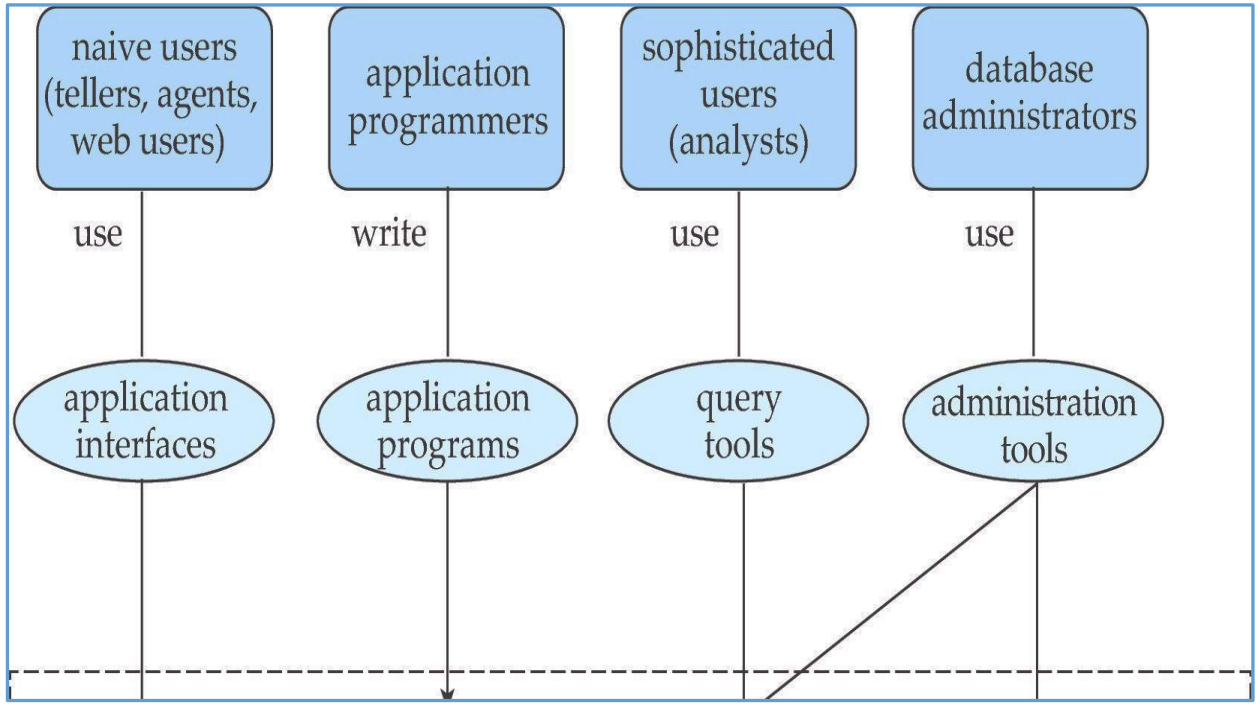
Database System Internals



Important Components of Database System :

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

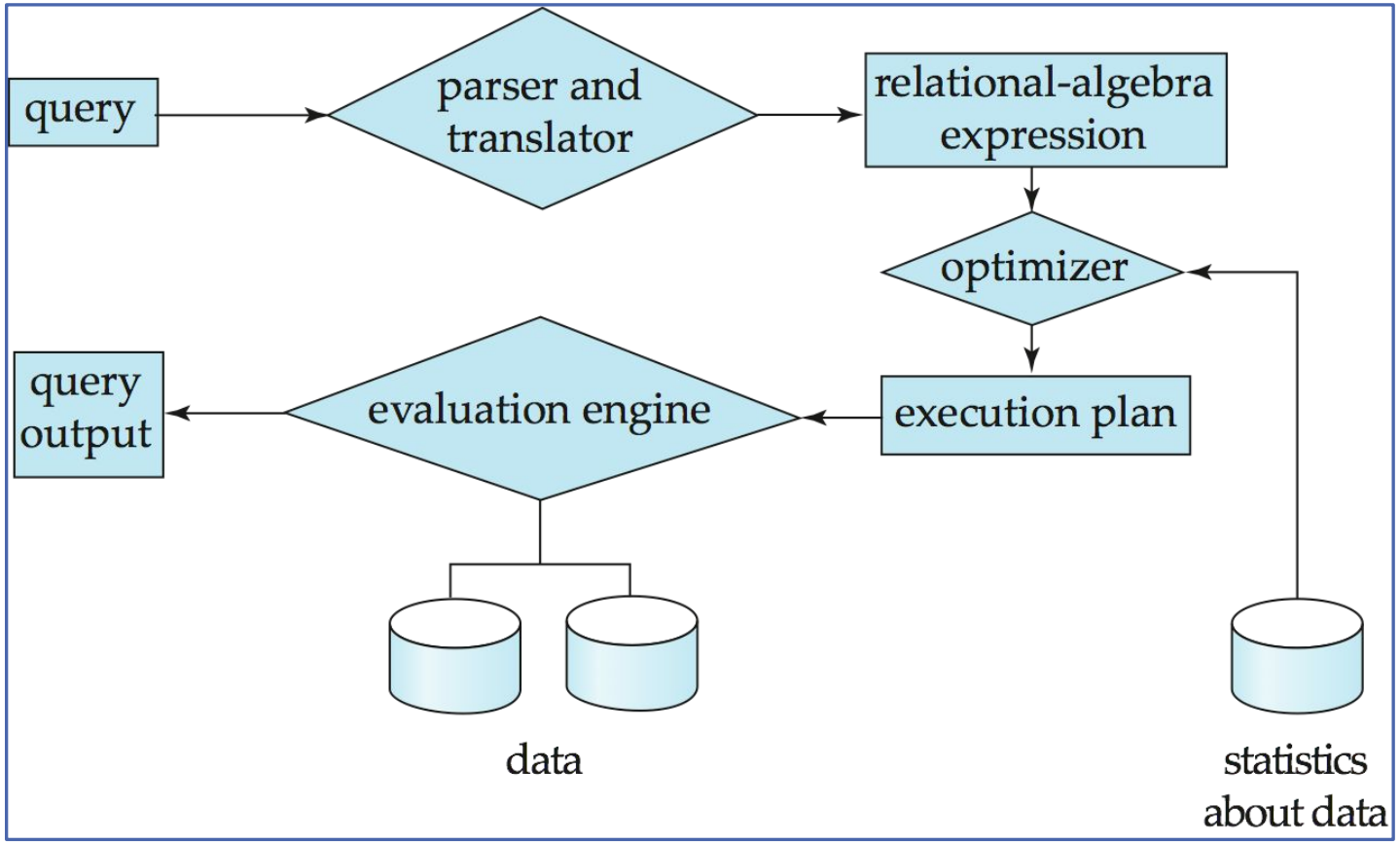
Database System Components : Database Users



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:

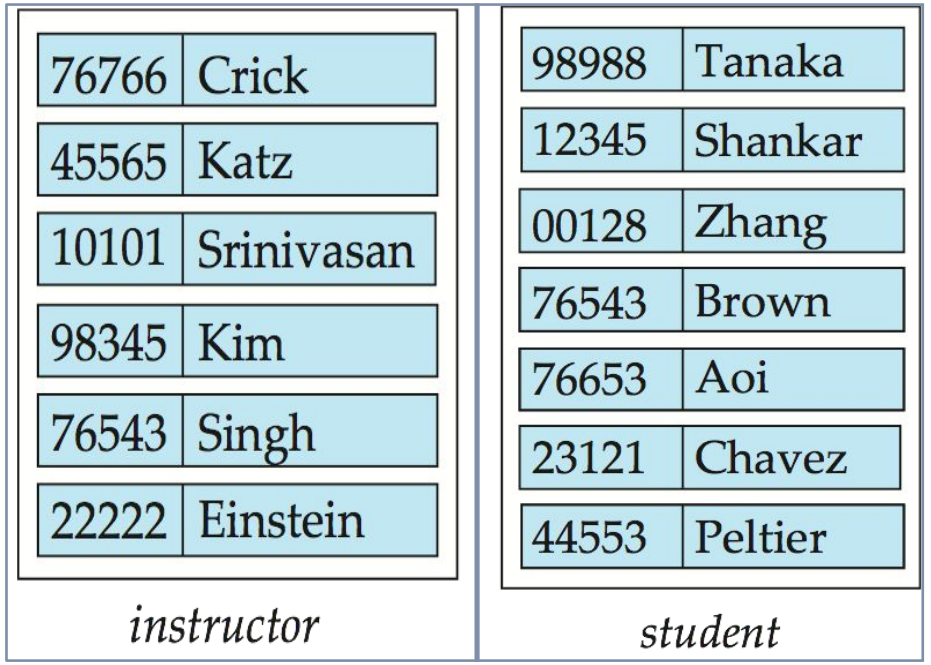
- **Transaction Manager**
- **Concurrency Control Manager**

- **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



76766	Crick
45565	Katz
10101	Srinivasan
98345	Kim
76543	Singh
22222	Einstein

instructor

98988	Tanaka
12345	Shankar
00128	Zhang
76543	Brown
76653	Aoi
23121	Chavez
44553	Peltier

student

Relationship Sets

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

Example:

44553 (Peltier) *advisor* 22222 (Einstein)

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

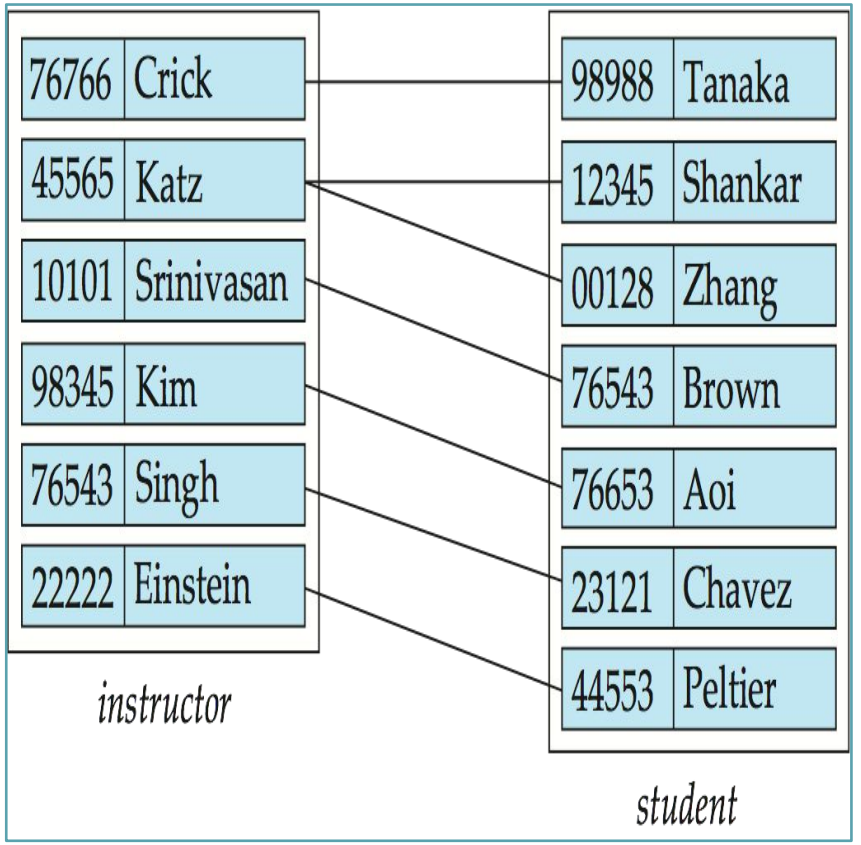
A **relationship set** is a mathematical relation among $n \geq 2$ entities, each taken from entity sets

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

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

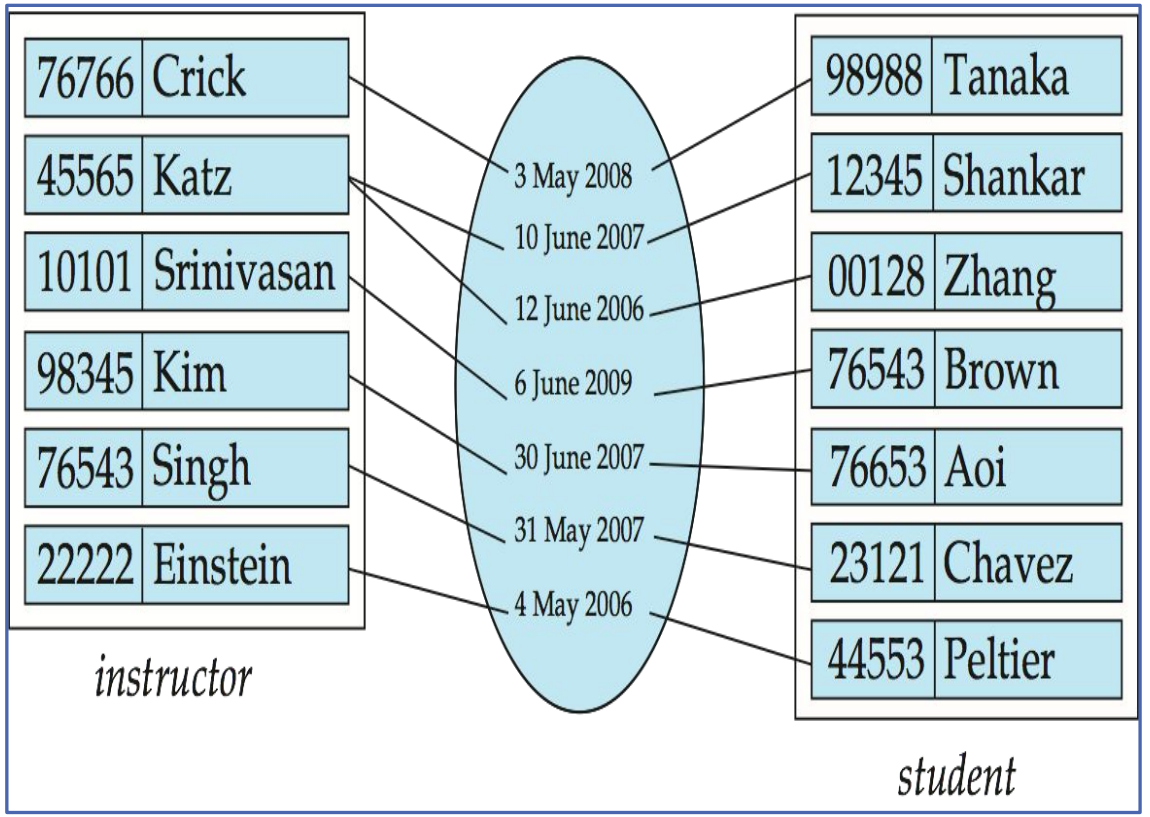
- Example:

$(44553,22222) \in advisor$



Relationship Sets

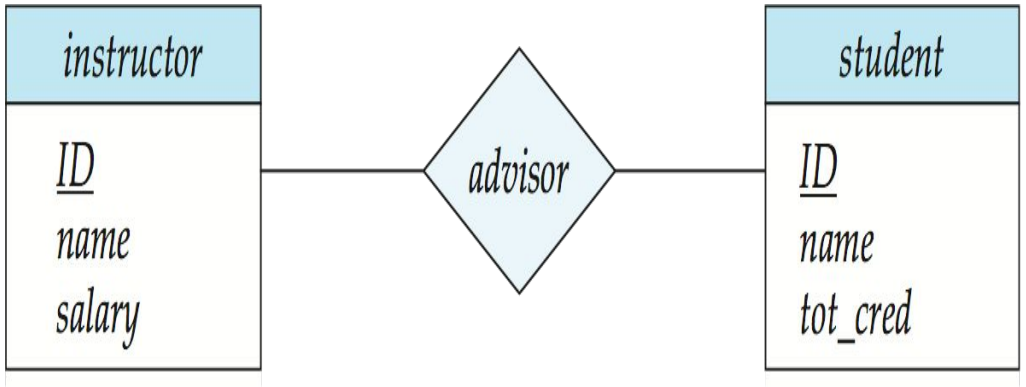
- An **attribute** can also be property of a relationship set.
- For instance, the *advisor* relationship set between entity sets *instructor* and *student* may have the attribute *date* which tracks when the student started being associated with the advisor



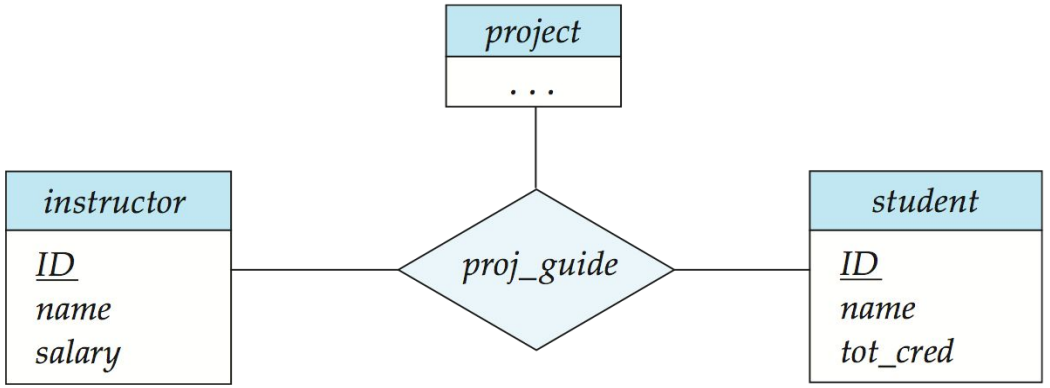
Degree of a Relationship Set

- **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

- 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*)

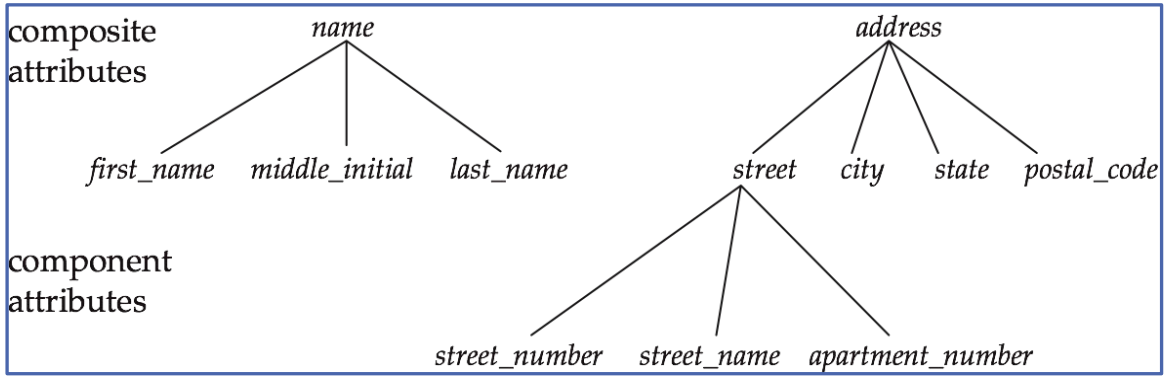
    course= (*course_id, title, credits*)
- **Domain** – the set of permitted values for each attribute

- **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



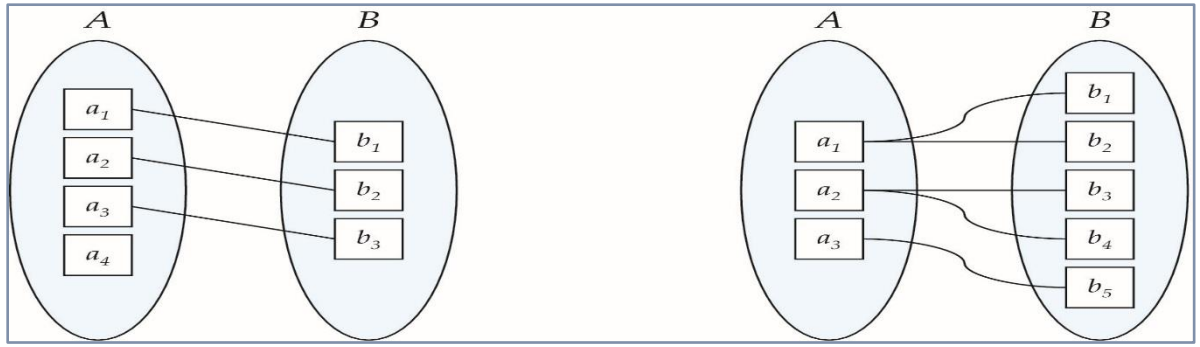
Mapping Cardinality Constraints

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

Most useful in describing binary relationship sets.

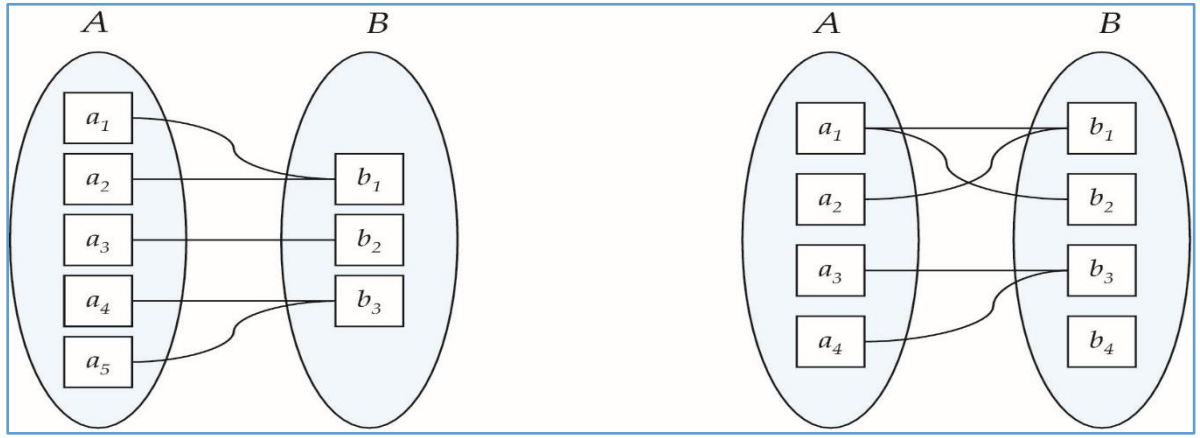
For a binary relationship set the mapping cardinality must be one of the following types:

- ***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

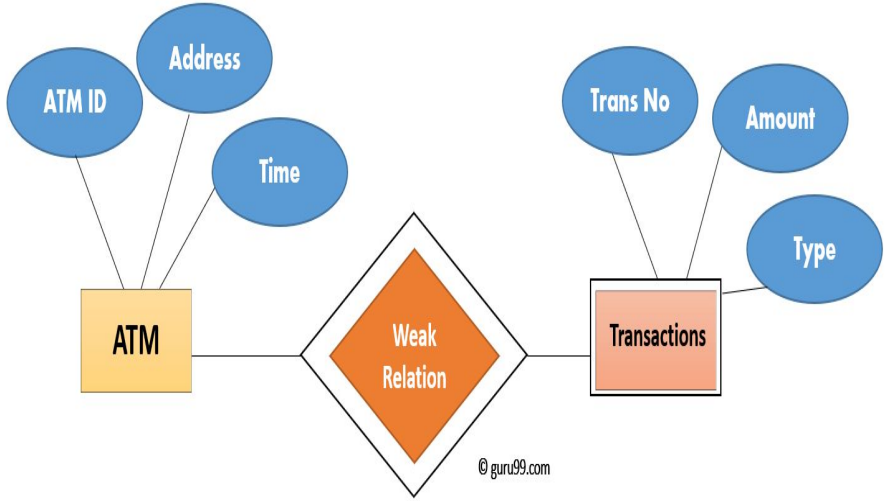
The combination of primary keys of the participating entity sets forms a super key of a relationship set.

- **(*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

MIT-WPU
॥ विश्वशान्तिर्ध्रुवं ध्रुवा ॥

Entity Relationship Diagram

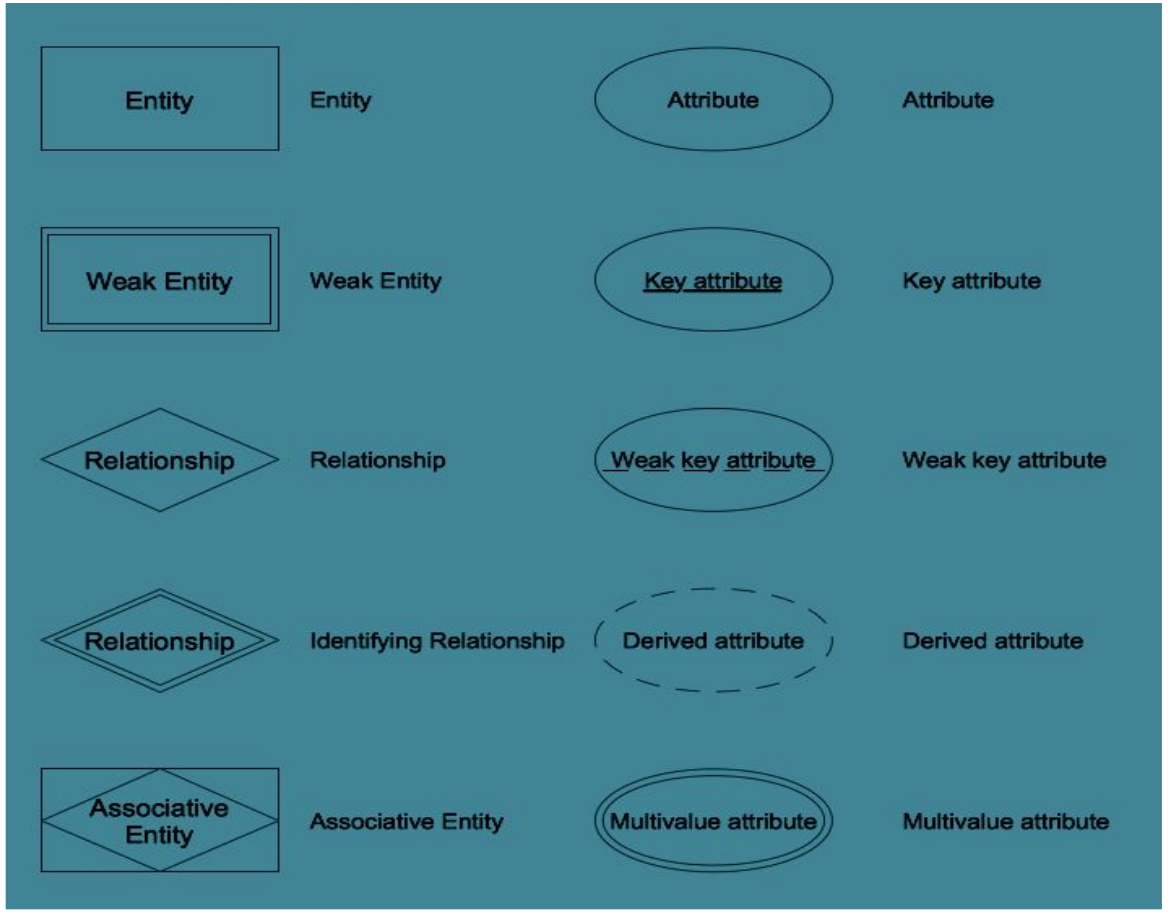


Entity-Relationship Diagram

It is a graphical Representation of ER Model

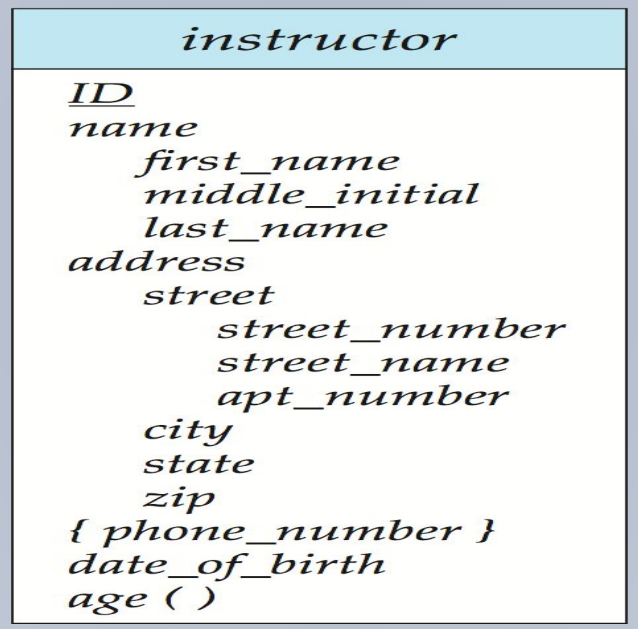
Basic Notations :

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

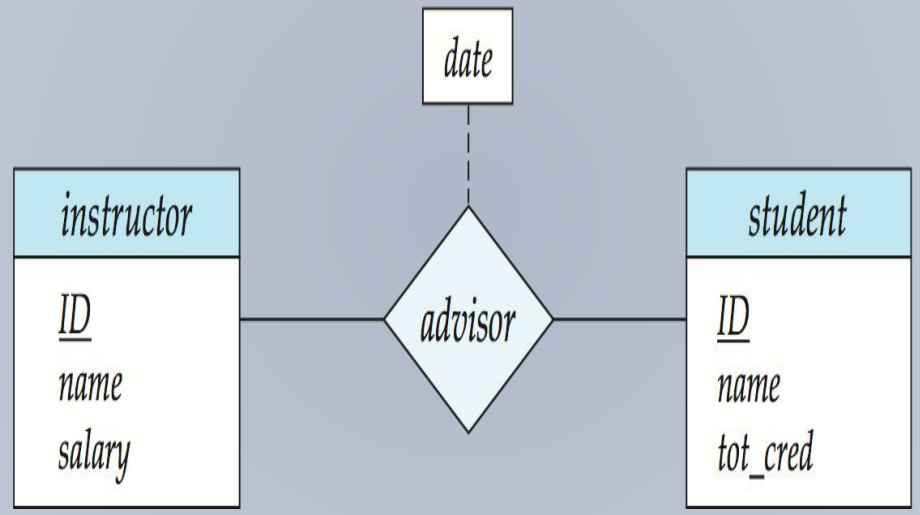


Entity Relationship Diagram Continued

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



- **Relationship Sets with Attributes**

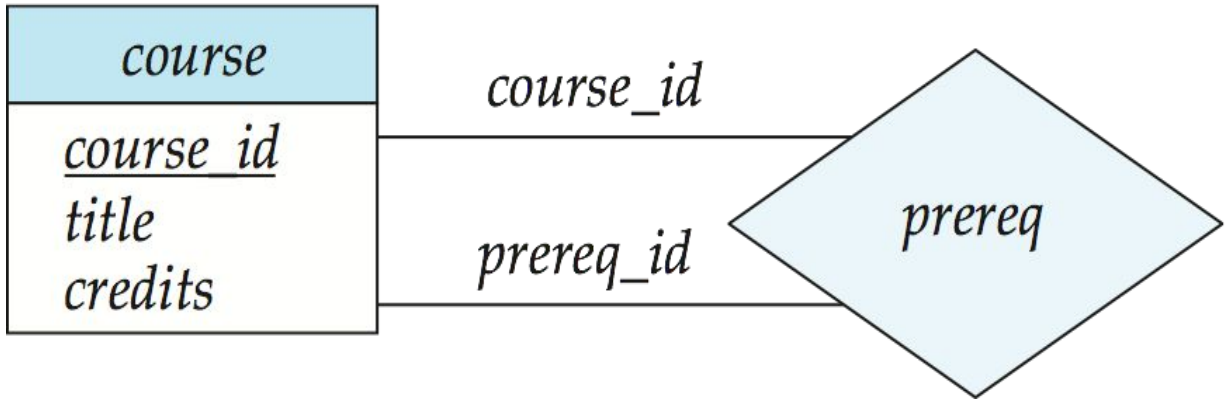


Entity Relationship Diagram Continued

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

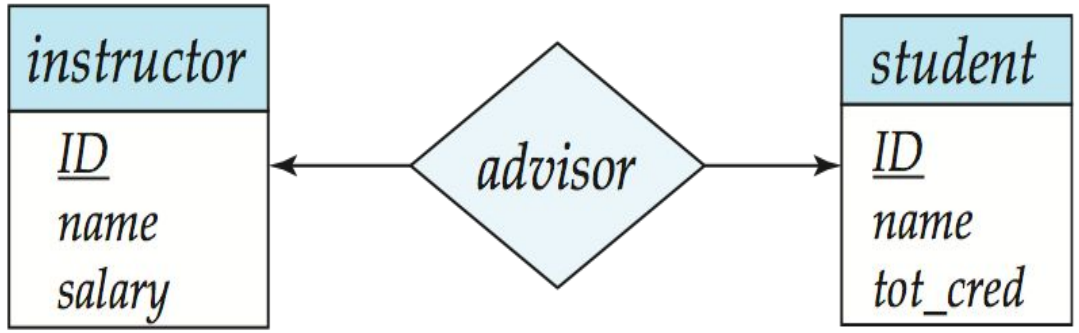
Cardinality Constraints

- 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*

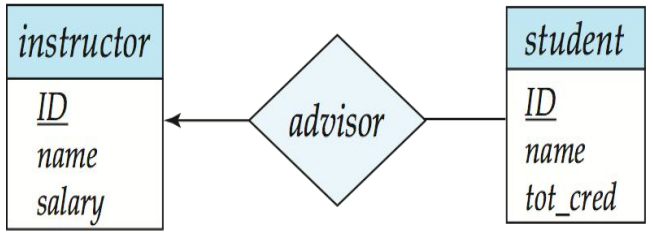


Entity Relationship Diagram Continued

b. One-to-Many Relationship

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

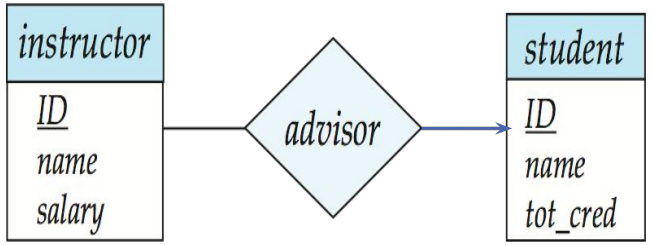
- a an instructor is associated with several (including 0) students via *advisor*
- student is associated with at most one instructor via advisor



c. Many-to-One Relationship

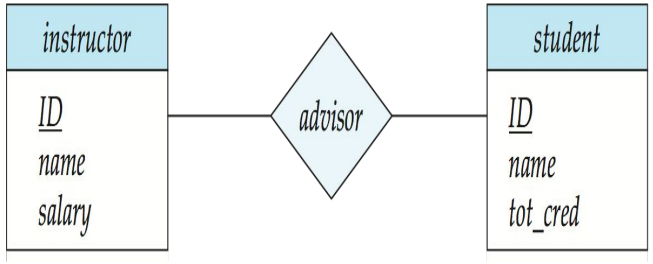
In a many-to-one relationship between an *instructor* and a *student,*

- 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

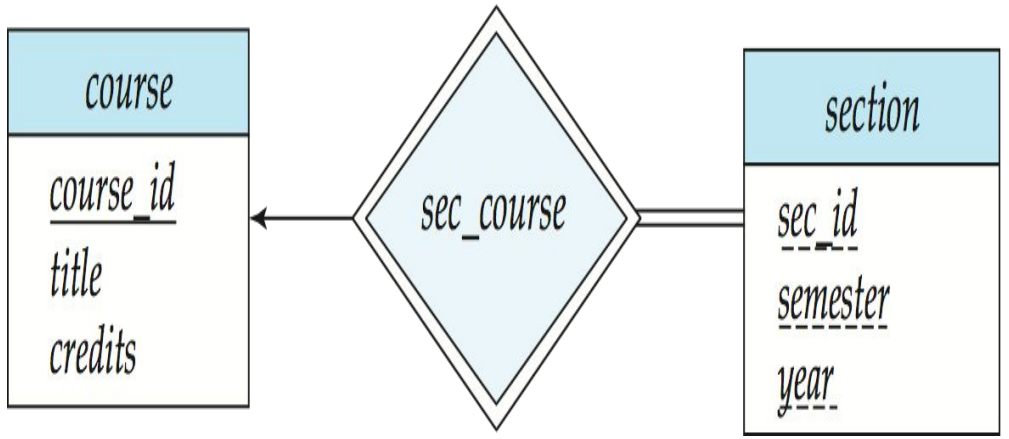
- An instructor is associated with several (possibly 0) students via *advisor*
- A student is associated with several (possibly 0) instructors via *advisor*



Entity Relationship Diagram Continued

Participation of an Entity Set in a Relationship Set

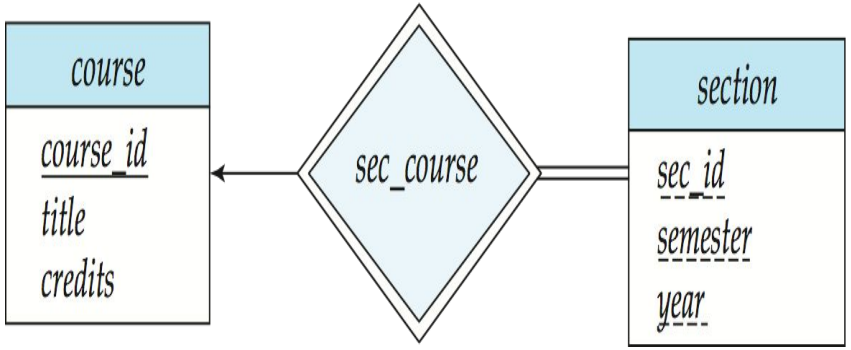
- **Total participation (indicated by double line)**: every entity in the entity set participates in at least one relationship in the relationship set
  - E.g., participation of *section* in *sec_course* is total
    - every *section* must have an associated course
- **Partial participation**: some entities 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*)

E-R Diagram Example

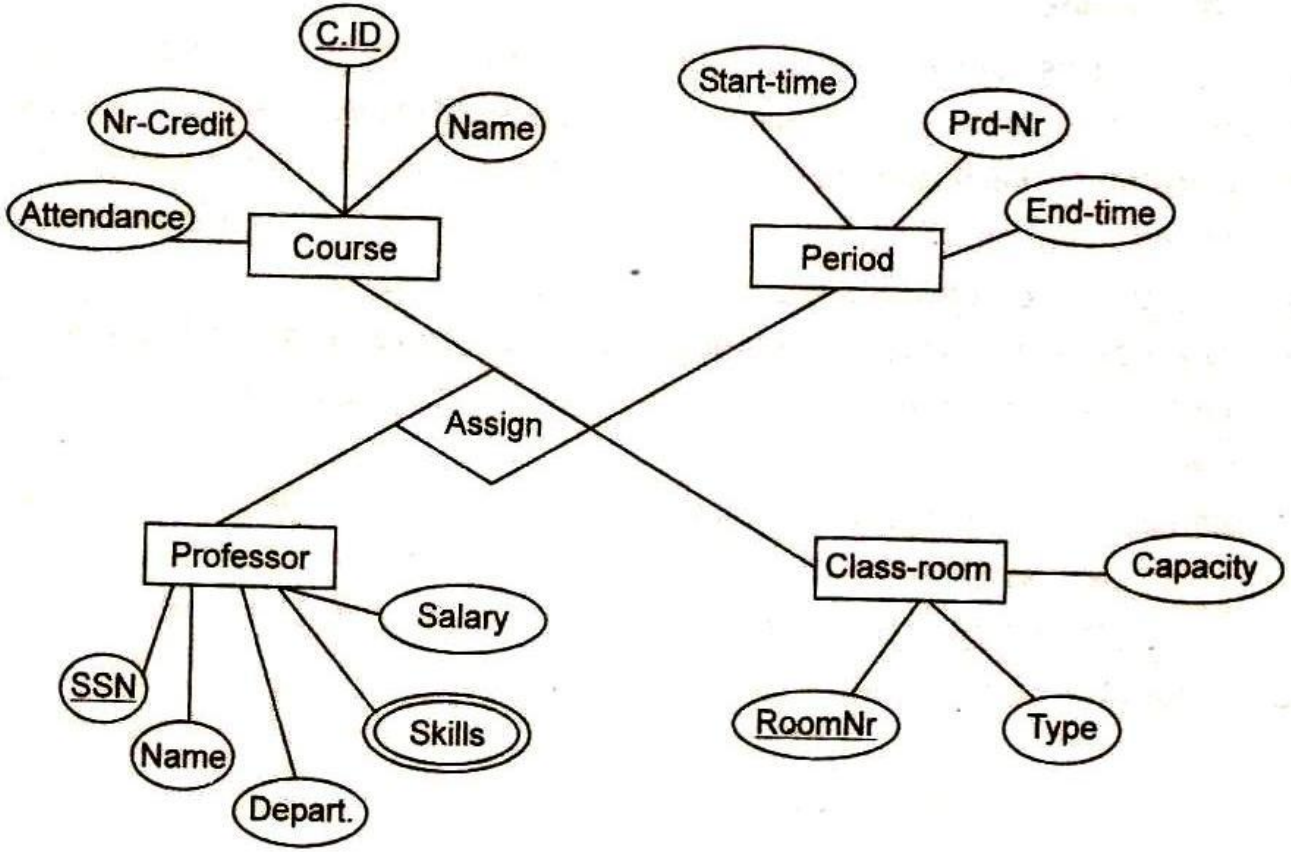
Question : **Design a ER Diagram for University Database based on scenario given below :**

A university includes information about the course timetable of an academic institution. For each Course the following information is recorded an identification number, name of the course, name of the teacher assigned to teach the course, the number of periods each week it will be taught, the number of

For each teacher the following information is recorded: SSN, name, name of the department he/she works with, skills, the yearly salary. For each class period the following information is recorded: period number, starting time, ending time. For each room the following information is recorded: room number, room type (classroom, office, auditorium, and computer lab), capacity.

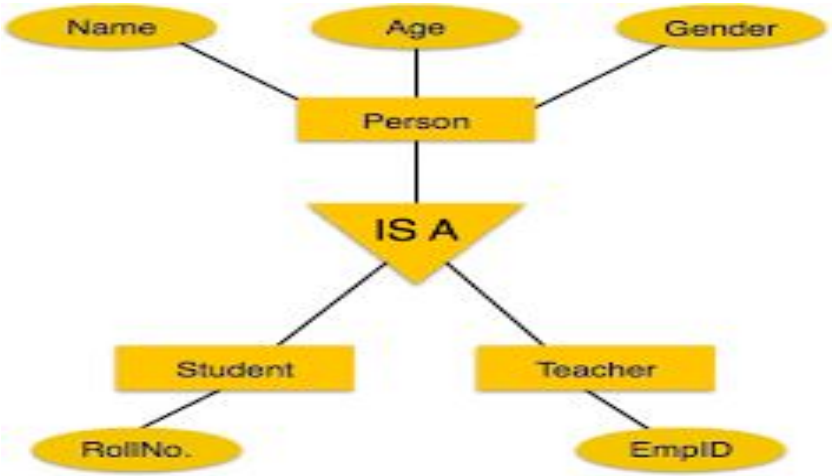
The above information is used in order to make the right assignment of a teacher that teaches a particular course to a time period and a classroom.

E-R Diagram for University Database



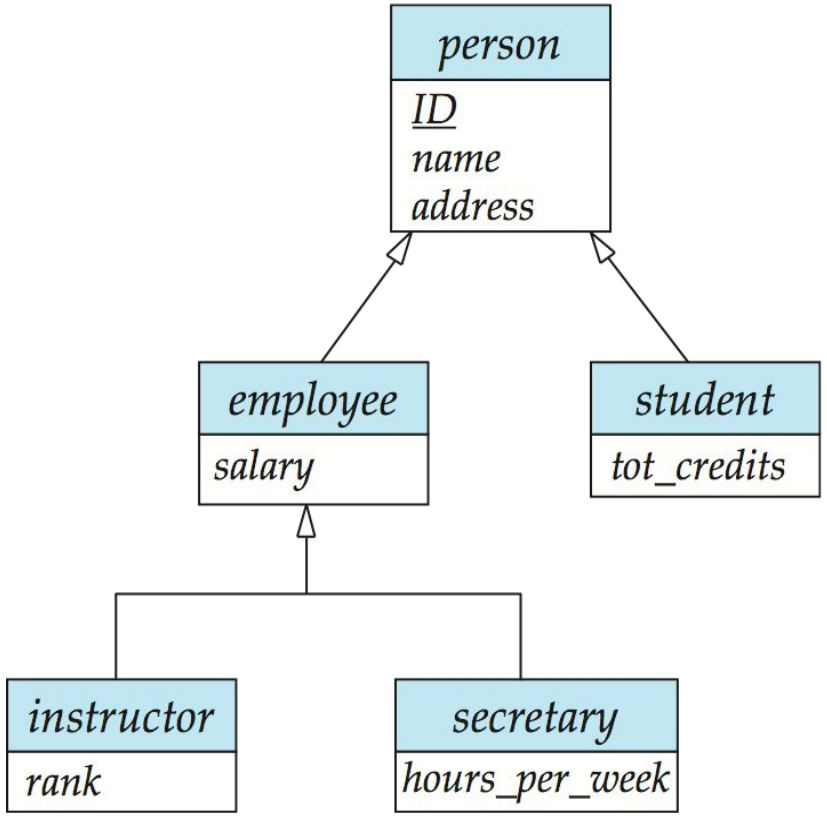
Extended ER Features :

- Specialization
- Generalization
- Aggregation



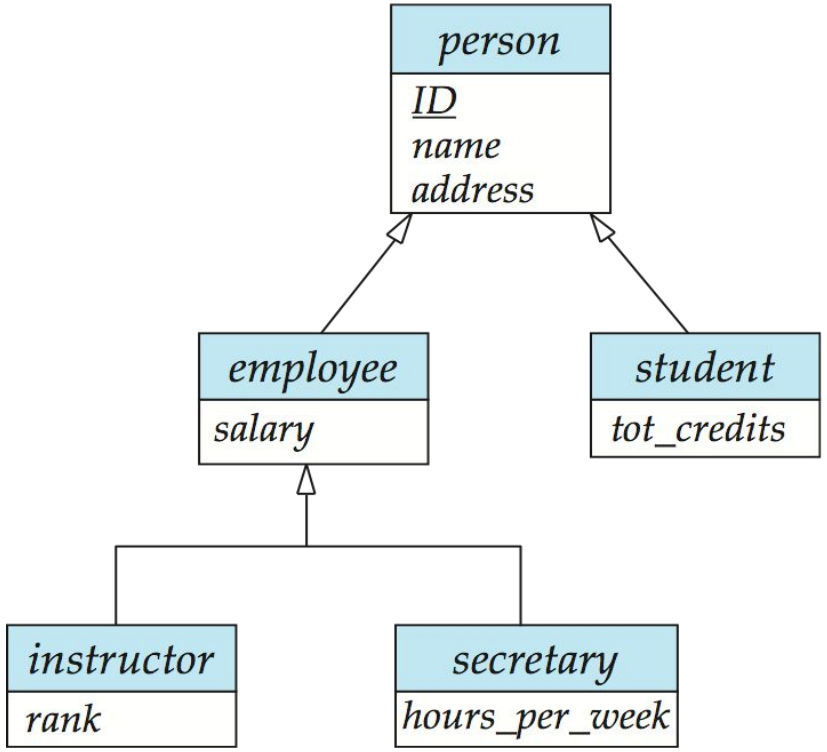
Extended ER Features : Specialization

- 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 ISA (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.



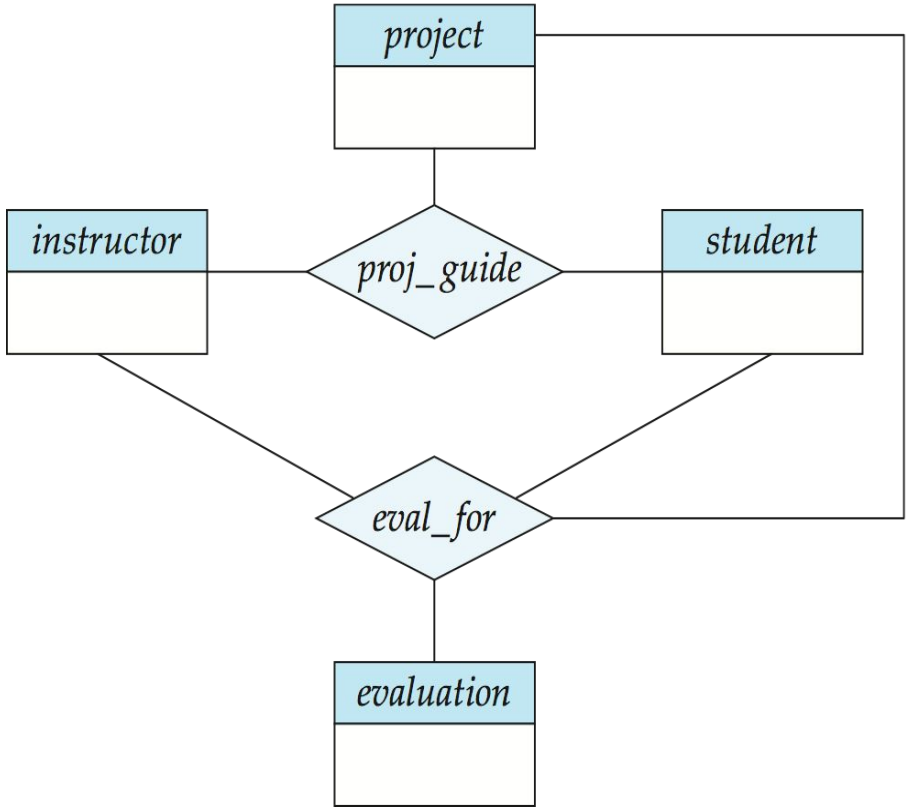
Extended ER Features : Generalization

- **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.



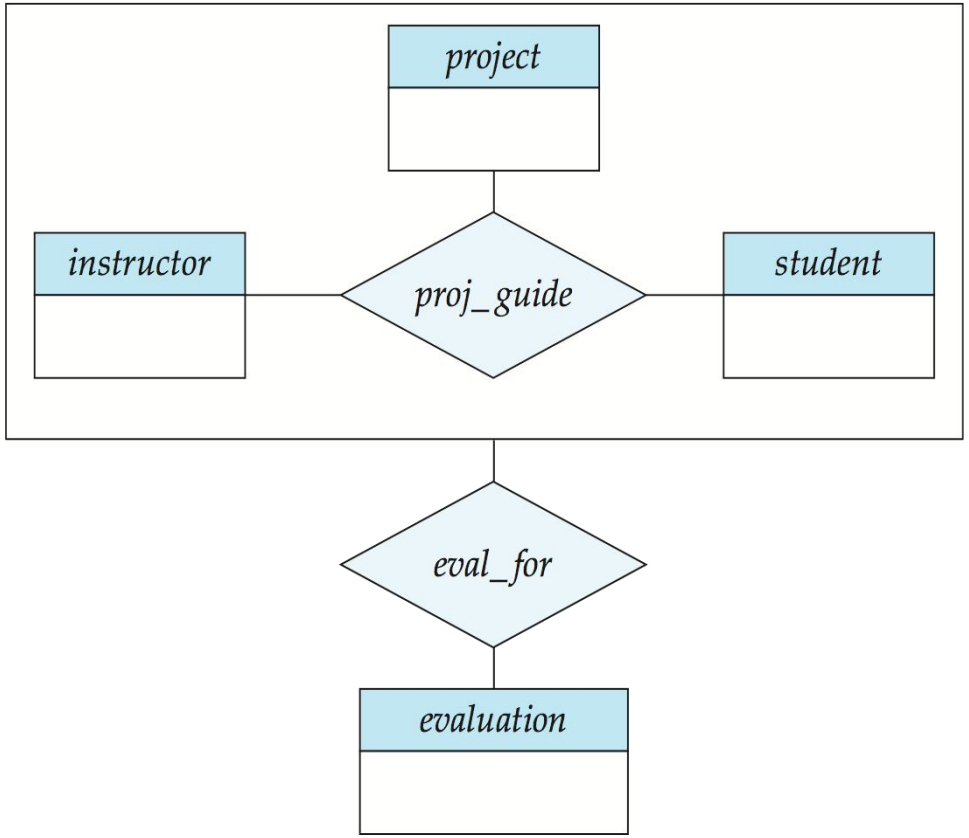
Extended ER Features : Aggregation

- 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

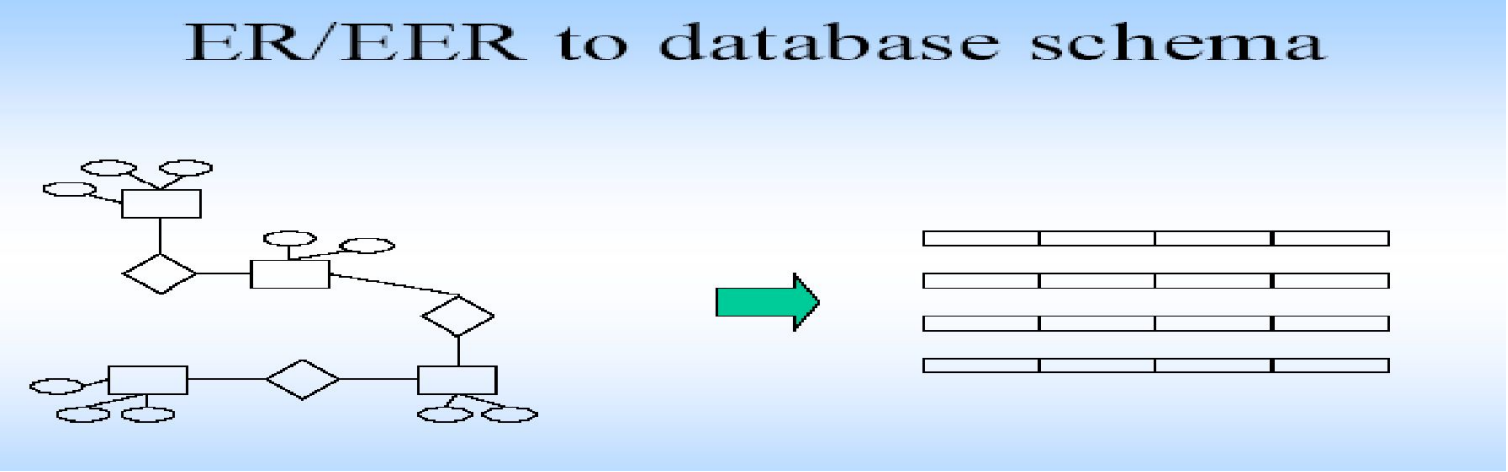


Extended ER Features : Aggregation

- 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



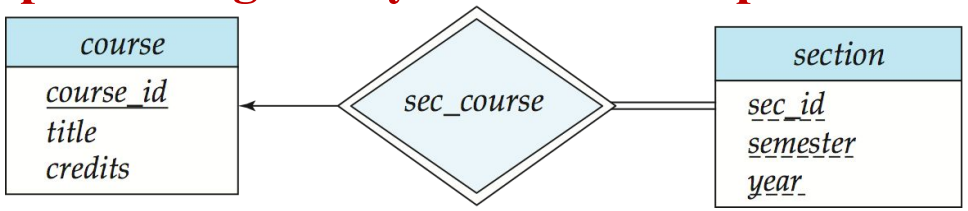
MIT-WPU
॥ विश्वशान्तिर्ध्रुवं ध्रुवा ॥

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)

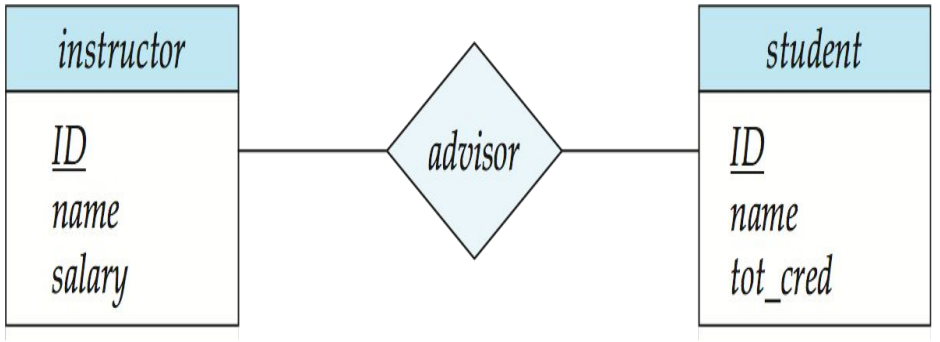
course_id	title	credits

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

section (course_id, sec_id, sem, year)

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:

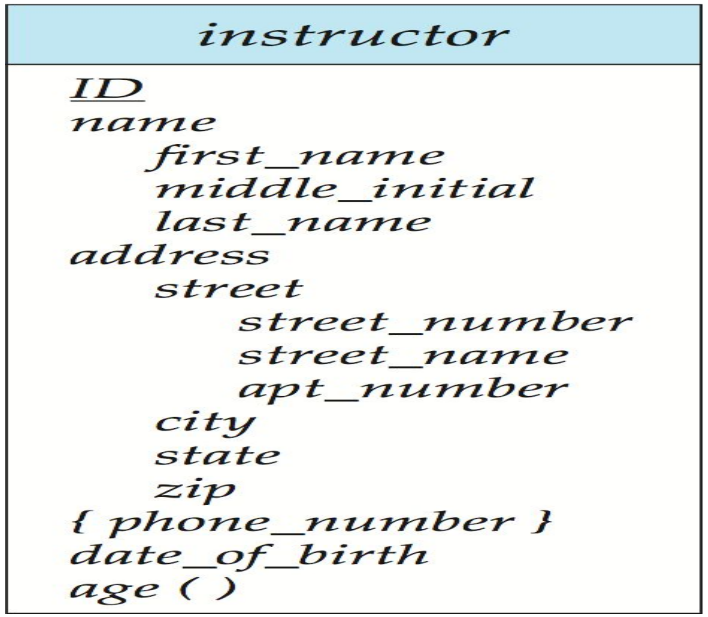
- Many-to-one and one-to-many relationship sets that are total on the many-side can be represented by adding an extra attribute to the “many” side, containing the primary key of the “one” side.

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)

instructor

ID
name
first_name
middle_initial
last_name
address
street
street_number
street_name
apt_number
city
state
zip
{ phone_number }
date_of_birth
age ( )

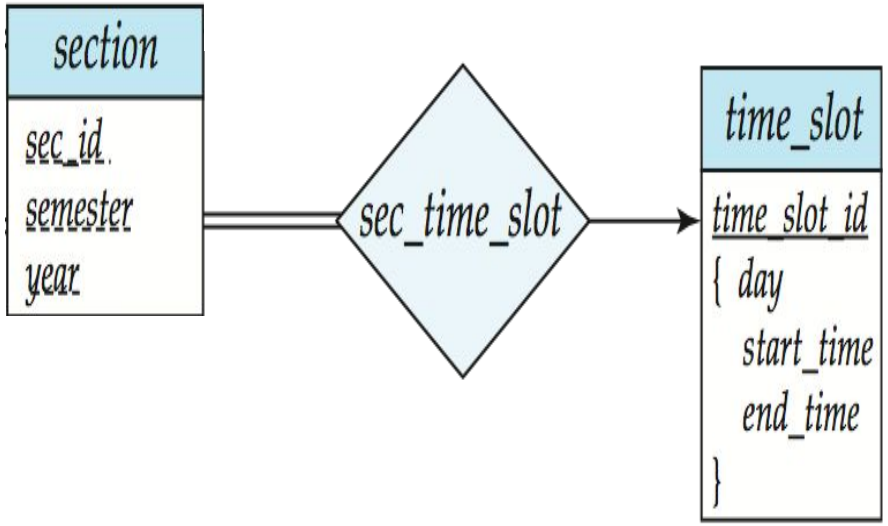
ID	Phone_number
22222	456-7890
22222	123-4567

Reduction to Relation Schema

Representing Composite and Multi-valued Attributes

Special case : entity *time_slot* has only one attribute other than the primary-key attribute, and that attribute is multivalued

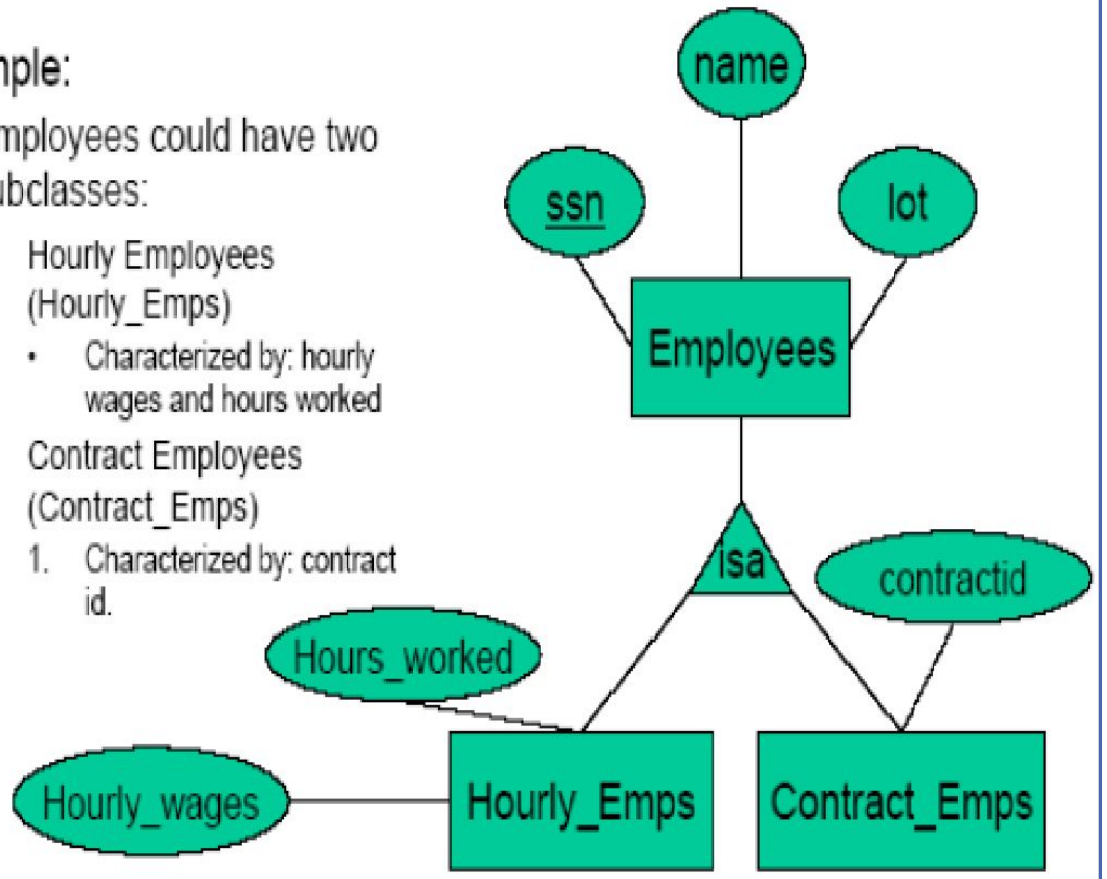
- Optimization: Don't create the relation corresponding to the entity, just create the one corresponding to the multivalued attribute
- *time_slot*(*time_slot_id, day, start_time, end_time*)
- *time_slot* attribute of *section (*from *sec_time_slot*) cannot be a foreign key due to this optimization



Extended ER Features Reduction to Relation Schemas

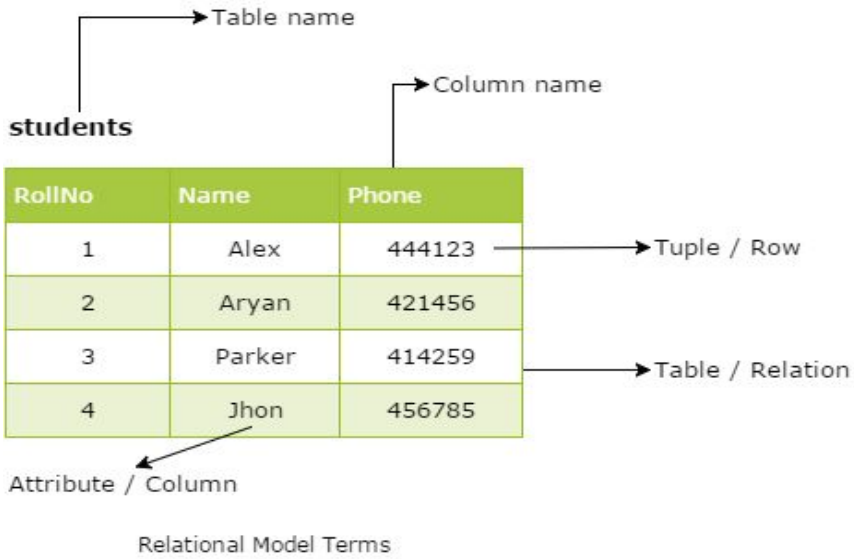
Example:

- 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.

Relational Model



Relational Model Terms

Relational Model: Basic Structure

Formally, given sets $D_1, D_2, \ldots D_n$ a **relation** r is a subset of

$$D_1 \times D_2 \times \ldots \times D_n$$

Thus, a relation is a set of n-tuples $(a_1, a_2, \ldots, a_n)$ where each $a_i \in D_i$

Example: *customer_name* = {Jones, Smith, Curry, Lindsay}

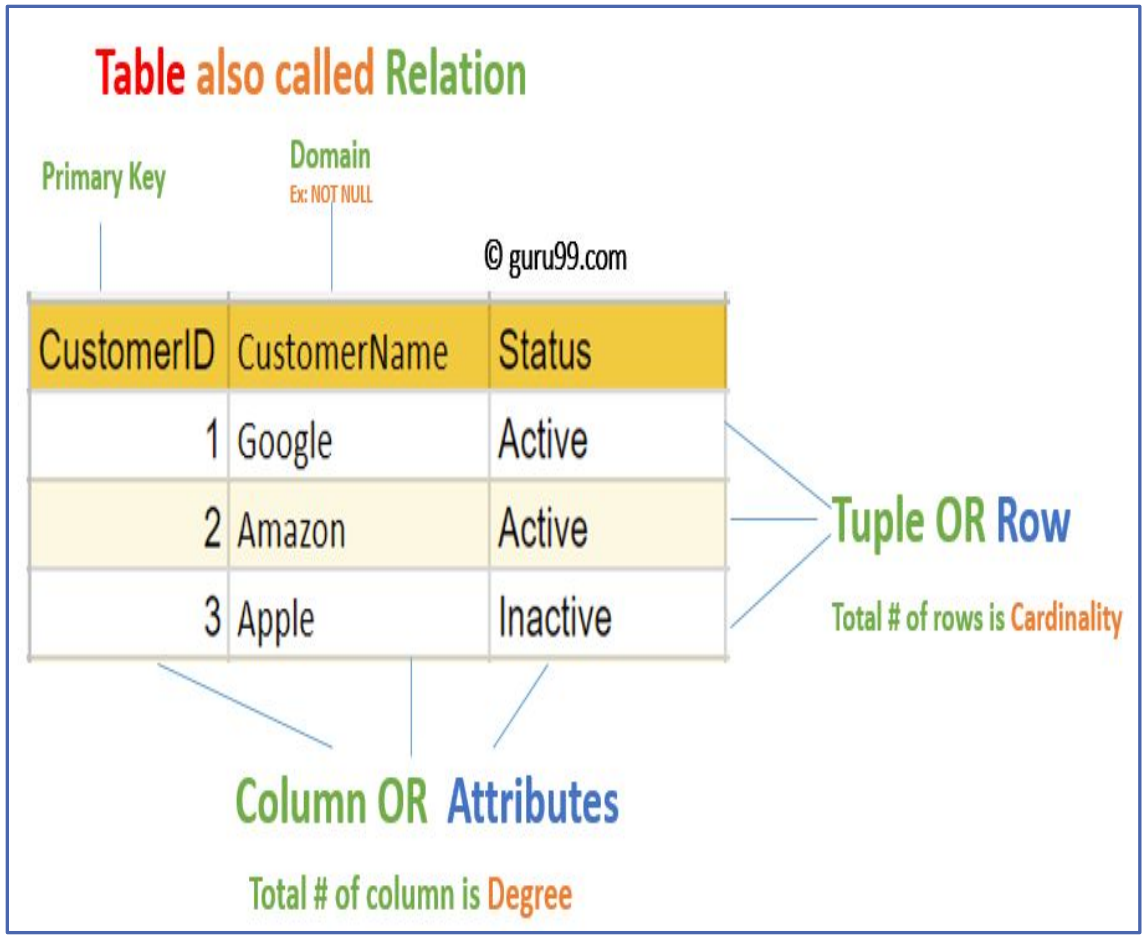
customer_street = {Main, North, Park}

customer_city = {Harrison, Rye, Pittsfield}

Then r = { (Jones, Main, Harrison),
(Smith, North, Rye),
(Curry, North, Rye),
(Lindsay, Park, Pittsfield) }

is a relation over

customer_name , customer_street, customer_city



Relational Model: Attribute types

- Each attribute of a relation has a name
- The set of allowed values for each attribute is called the **domain** of the attribute
- Attribute values are (normally) required to be **atomic**; that is, indivisible
  - Note: multivalued attribute values are not atomic ({secretary. clerk}) is example of multivalued attribute *position*
  - Note: composite attribute values are not atomic
- The special value *null* is a member of every domain
- The null value causes complications in the definition of many operations

Relational Model: Relation Schema

- $A_1, A_2, \ldots, A_n$ are *attributes*
- $R = (A_1, A_2, \ldots, A_n)$ is a *relation schema*

Example:

Customer_schema = (*customer_name, customer_street, customer_city*)

r(*R*) is a *relation* on the *relation schema R*

Example:

customer (Customer_schema)

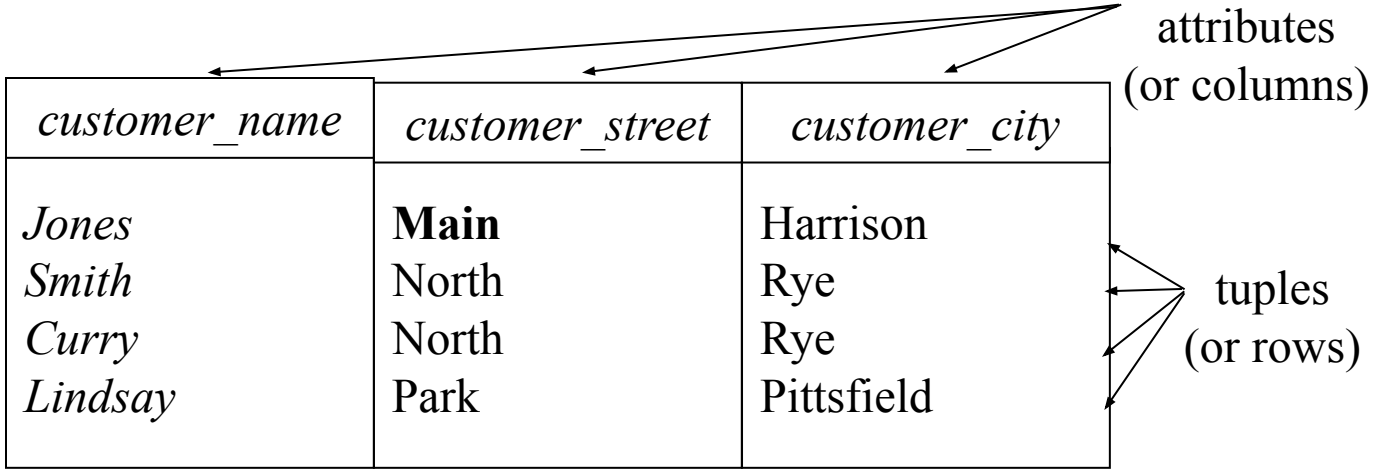
Customer

Customer_name	Customer_street	Customer_city

Relational Model: Relation Instance

The current values (*relation instance*) of a relation are specified by a table

An element *t* of *r* is a *tuple*, represented by a *row* in a table



Customer

Relational Model: Database

- A database consists of multiple relations
- Information about an enterprise is broken up into parts, with each relation storing one part of the information

  account : stores information about accounts

  depositor : stores information about which customer owns which account

  customer : stores information about customers

- Storing all information as a single relation such as *bank* (*account_number, balance, customer_name*, ..)

results in repetition of information (e.g., two customers own an account) and the need for null values (e.g., represent a customer without an account)

customer-id	customer-name	customer-street	customer-city
192-83-7465	Johnson	12 Alma St.	Palo Alto
019-28-3746	Smith	4 North St.	Rye
677-89-9011	Hayes	3 Main St.	Harrison
182-73-6091	Turner	123 Putnam Ave.	Stamford
321-12-3123	Jones	100 Main St.	Harrison
336-66-9999	Lindsay	175 Park Ave.	Pittsfield
019-28-3746	Smith	72 North St.	Rye

(a) The *customer* table

account-number	balance
A-101	500
A-215	700
A-102	400
A-305	350
A-201	900
A-217	750
A-222	700

(b) The *account* table

customer-id	account-number
192-83-7465	A-101
192-83-7465	A-201
019-28-3746	A-215
677-89-9011	A-102
182-73-6091	A-305
321-12-3123	A-217
336-66-9999	A-222
019-28-3746	A-201

(c) The *depositor* table

Relational Model: Keys

Let K ⊆ R

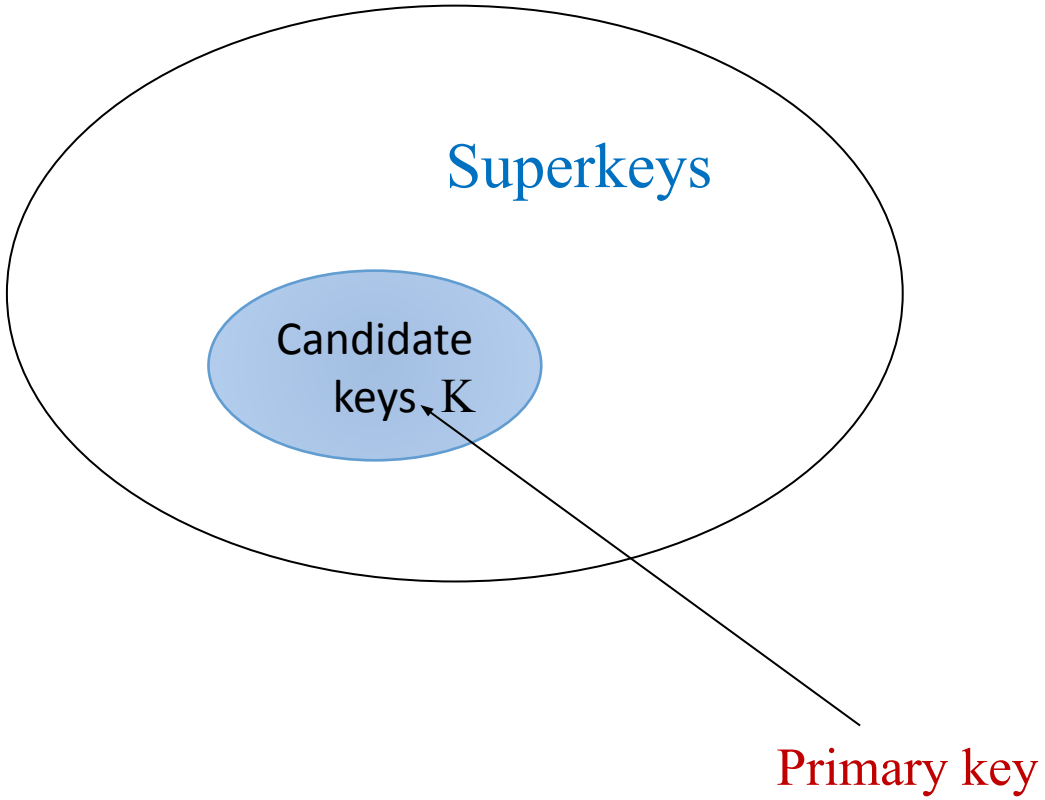
- *K* is a **superkey** of *R* if values for *K* are sufficient to identify a unique tuple of each possible relation *r(R)*
  - by "possible *r* " we mean a relation *r* that could exist in the enterprise we are modeling.
  - Example: {*customer_name, customer_street*} and {*customer_name*}

    are both superkeys of *Customer*, if no two customers can possibly have the same name.

- *K* is a **candidate key** if *K* is minimal

  Example: {*customer_name*} is a candidate key for.

- **Primary Key : Any candidate key**



MIT-WPU
॥ विश्वशान्तिर्ध्रुवं ध्रुवा ॥

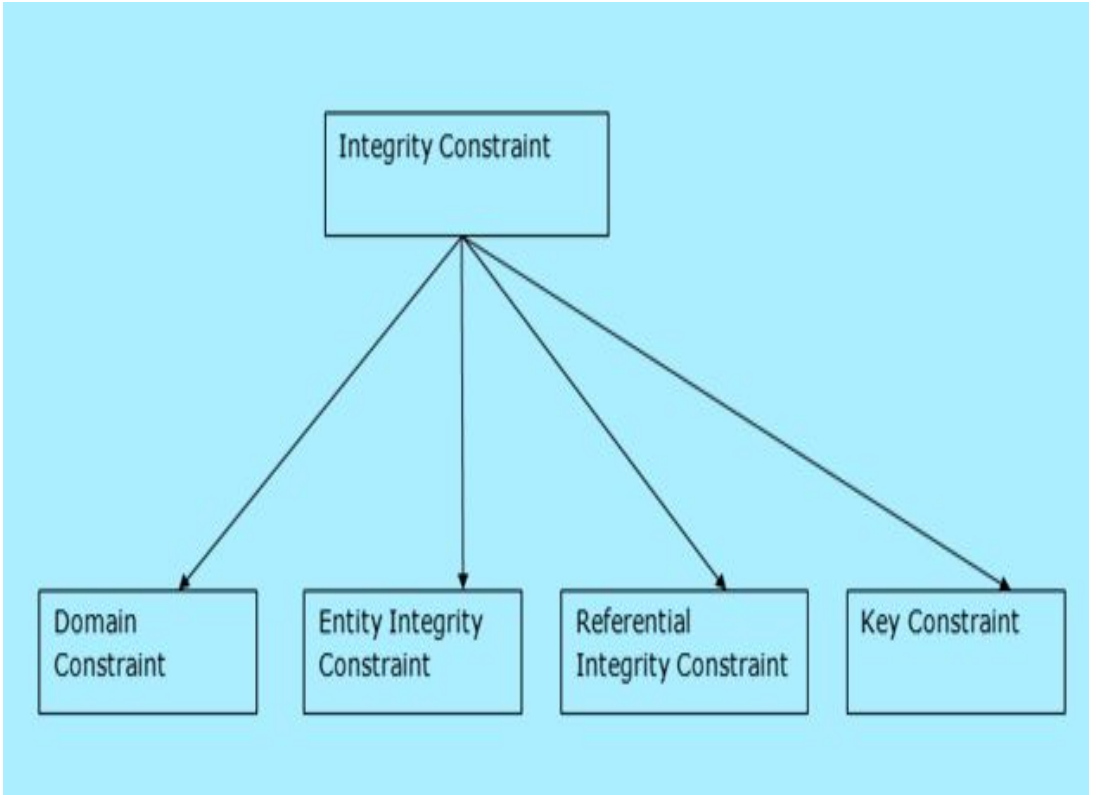
Relational Integrity Constraints

Integrity Constraints

Integrity constraints guard against accidental damage to the database, by ensuring that authorized changes to the database do not result in a loss of data consistency.

Examples :

- Domain Constraints
- Referential Integrity Constraints
- Entity Integrity Constraint
- Key Constraints
- Triggers
- Functional Dependencies



Integrity Constraints : Domain Constraints

- They define valid values for attributes
- They are the most elementary form of integrity constraint.
- They test values inserted in the database, and test queries to ensure that the comparisons make sense.
- The check clause in SQL-92 permits domains to be restricted(Not supported in MySQL)
- use check clause to ensure that an hourly-wage domain allows only values greater than a specified value.

```
create domain hourly-wage numeric(5,2)
constraint value-test check (value>=4.00)
```

The domain hourly-wage is declared to be a decimal number with 5 digits, 2 of which are after the decimal point

- The domain has a constraint that ensures that the hourly-wage is greater than 4.00.
- constraint value-test is optional; useful to indicate which constraint an update violated.

Integrity Constraints : Referential Integrity

Ensures that a value that appears in one relation for a given set of attributes also appears for a certain set of attribute in another relation.

- If an account exists in the database with branch name "Perryridge", then the branch "Perryridge" must actually exist in the database.

account (account-no, branch-name, balance)

A-123	Perryridge	5000

Foreign Key(branch-name)

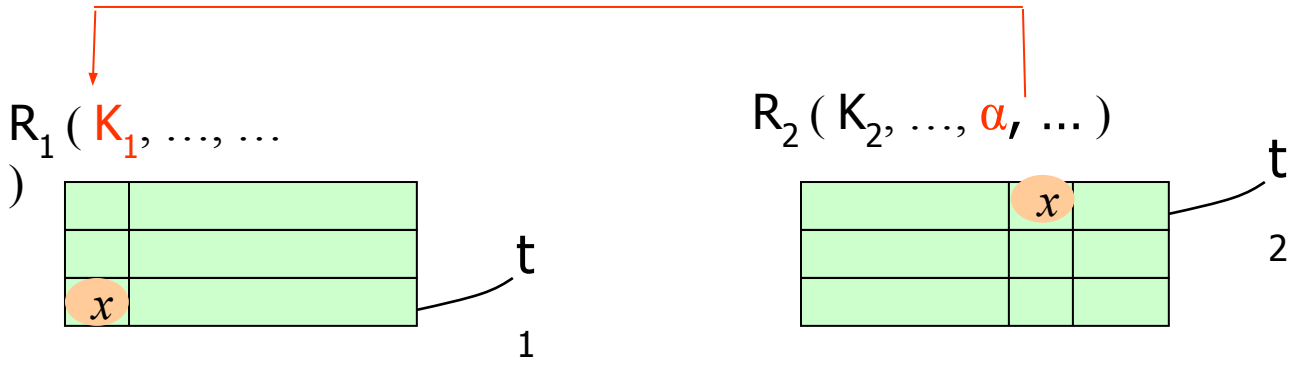
branch (branch-name, branch-city, asset)

Perryridge	Brooklyn	500,000

A set of attributes X in R is a foreign key if it is not a primary key of R but it is a primary key of some relation S.

Referential Integrity : Formal Definition

- Let $r_1(R_1)$ and $r_2(R_2)$ be relations with primary keys K_1 and K_2 respectively.
- The subset α of R_2 is a foreign key referencing K_1 in relation r_1, if for every t_2 in r_2 there must be a tuple t_1 in r_1 such that $t_1[K_1]=t_2[\alpha]$.
- Referential integrity constraint: $\prod_{\alpha}(r_2) \subseteq \prod_{K1} (r_1)$



Referential Integrity for Insertion and Deletion

The following tests must be made in order to preserve the following referential integrity constraint:

- Insert. If a tuple t_2 is inserted into r_2. The system must ensure that there is a tuple t1 in r1 such that $t_1[K] = t_2[\alpha]$. That is

  $t_2[\alpha] \in \prod_K(r_1)$

- Delete. If a tuple t_1 is deleted from r_1, the system must compute the set of tuples in r_2 that reference t_1:

  $\sigma_{\alpha=t1[K]}(r_2)$

- if this set is not empty, either the delete command is rejected as an error, or the tuples that reference t_1 must themselves be deleted (cascading deletions are possible)

Referential Integrity for Update

- if a tuple t_2 is updated in relation r_2 and the update modifies values for the foreign key α, then a test similar to the insert case is made. Let t_2' denote the new value of tuple t_2. The system must ensure that

  $t_2'[\alpha] \in \Pi_K(r_1)$

  new foreign key value must exist

- if a tuple t_1 is updated in r_1, and the update modifies values for primary key(K), then a test similar to the delete case is made. The system must compute

  $\sigma_{\alpha=t1[K]}(r_2)$

  no foreign keys contain the old primary key

- using the old value of t_1 (the value before the update is applied). If this set is not empty, the update may be rejected as an error, or the update may be applied to the tuples in the set (cascade update), or the tuples in the set may be deleted.

Integrity Constraints : Entity Integrity Constraints

Entity Integrity

- **Requirement:**
  - All entries are unique and no part of a primary key may be null.
- **Purpose:**
  - Guarantees that each entity will have a unique identity and ensures that foreign key values can properly refer to primary key values.

Example :

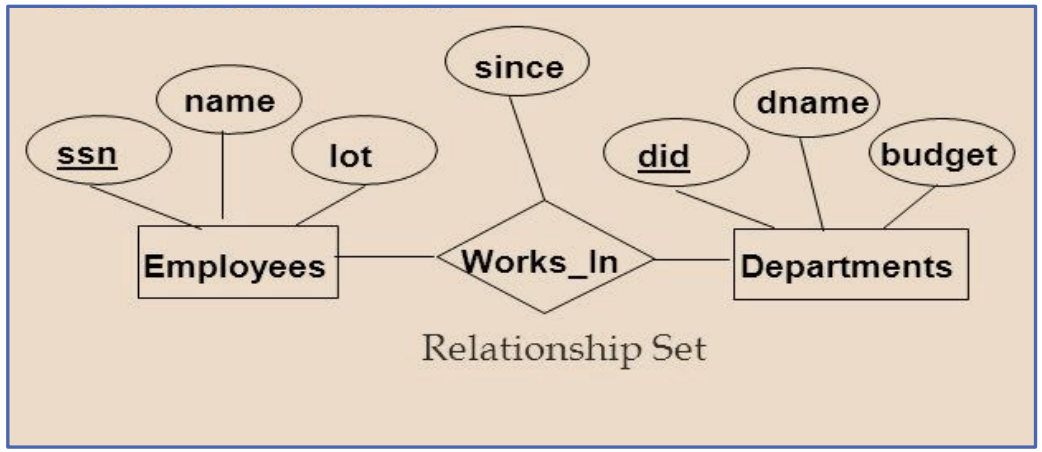
PUBLISHER

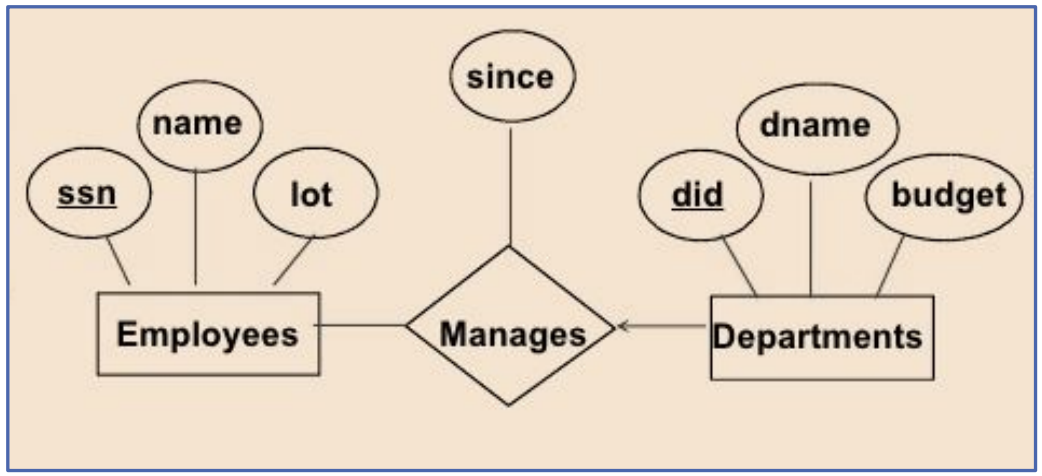
Publisher_Code	Name	City
F-B1	Fajar Bakti	Malaysia
M-G1	McGraw Hill	UK
P-H1	Prentice Hall	UK
T-H1	Thompson	US

Figure 3-38 The Primary Key

Publisher_Code is the PRIMARY KEY in **PUBLISHER** table, so this field cannot have a null value and all entries are unique.

Integrity Constraints :Key Constraints





Key Constraints :

- **Consider Works_in Relationship**: An employee can work in many departments ; a department can have many employees.

In contrast, each department has at most one manager ,according to the key constraint on **Manages Relationship**

Triggers

A trigger is a statement that is executed automatically by the system as a side effect of a modification to the database.

To design a trigger mechanism, we must:

- Specify the conditions under which the trigger is to be executed.
- Specify the actions to be taken when the trigger executes.

Example :

Suppose that instead of allowing negative account balances, the bank deals with overdrafts by

- setting the account balance to zero
- creating a loan in the amount of the overdraft
- giving this loan a loan number which is identical to the account number of the overdrawn account.

The condition for executing the trigger is an update to the account relation that results in a negative balance value.

Example of Translating Conceptual Model into Relation Schemas

Consider the following ‘simple’ conceptual data model:

Staff(Staff-ID, Name, Address, ScalePoint, RateOfPay, DOB,
...)
Student(Enrol-No, Name, Address, ..)
Course(CourseCode, Name, Duration, ...)

Task : To find the number of relations formed from above model

The ‘Translation’ Process

- **Entities** *become* **Relations**
- **Attributes** *become* **Attributes**(?)
- **Key Attribute**(s) *become* **Primary Key**(s)
- **Relationships** *are represented by* additional **Foreign Key Attributes**:
  - for those Relations that are at the **‘M’ end** of each **1:M Relationship**

The ‘Staff’ & ‘Student’ Relations

Staff(Staff-ID, Name, Address, ScalePoint, RateOfPay, DOB, ...)

becomes:

- **Staff**

Staff-ID	**Name**	**Address**	**ScalePoint**	**RateOfPay**	**DOB**

Student(Enrol-No, Name, Address, ...)

becomes:

- **Student**

Enrol-No	**Name**	**Address**	**OLevelPoints**	***Tutor***

NB. Foreign Key Tutor **references** Staff. Staff-ID

The ‘Staff’ & ‘Course’ Relations

Staff

Staff-ID	Name	Address	ScalePoint	RateOfPay	DOB

Course(CourseCode, Name, Duration, ...)

becomes:

Course

CourseCode	Name	Duration

Reflection Spot 2

Q. Based on the relationship between Staff and Course can we add foreign key in any of above relations in the translation process?

‘Staff’, ‘Course’ & ‘Team’ Relations

Staff

Staff-ID	Name	Address	ScalePoint	RateOfPay	DOB

Team

CourseCode	*Staff-ID*

Course

CourseCode	Name	Duration

Answer : We can’t add a Foreign Key; as Both Relations have a ‘M’ end.

Rule : We must create an ‘artificial’ linking Relation.

The ‘artificial’ Relation (i.e. Team):

- The **Primary Key** is a **composite** of CourseCode & Staff-ID
- **Foreign Key** CourseCode **references** Course.CourseCode
- **Foreign Key** Staff-ID **references** Staff.Staff-ID

4 Relations from 3 Entities?

Student

Enrol-No	Name	Address	OLevelPoints	*Tutor*

Staff

Staff-ID	Name	Address	ScalePoint	RateOfPay	DOB

Team

CourseCode	*Staff-ID*

Course

CourseCode	Name	Duration

BUT - are they anomaly free?

5 Relations from 3 Entities

Student

Enrol-No	Name	Address	OLevelPoints	*Tutor*

Staff

Staff-ID	Name	Address	*ScalePoint*	DOB

Course

CourseCode	Name	Duration

Pay

ScalePoint	RateOfPay

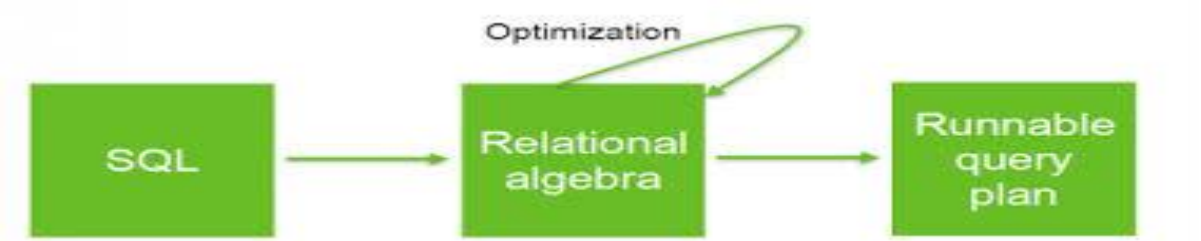
← a ‘split-off’ Relation
- to ‘solve’ a Dependency ‘problem’

Team

CourseCode	*Staff-ID*

← an ‘artificial’ Relation
- to ‘solve’ a M:M ‘problem’

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:

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

Where p is a formula in propositional calculus consisting of terms connected by : $\wedge$ (**and**), $\vee$ (**or**), $\neg$ (**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)

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

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

=

A	C
α	1
β	1
β	2

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

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, $R \cap S = \varnothing$).

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

A	B
α	1
β	2

Relation r

C	D	E
α	10	a
β	10	a
β	20	b
γ	10	b

Relation s

$r \times s$

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

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.

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*)
advisor(*s_ID*, *i_ID*)
time_slot(*time_slot_id*, *day*, *start_time*, *end_time*)
prereq(*course_id*, *prereq_id*)

employee (*person_name*, *street*, *city*)
works (*person_name*, *company_name*, *salary*)
company (*company_name*, *city*)

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*)

Example

What is the result of first performing the cross product of *student* and *advisor*, and then performing a selection operation on the result with the predicate $s\_id = ID$? (Using the symbolic notation of relational algebra, this query can be written as $\sigma_{s\_id=ID}(student \times advisor)$.)

2.7 Consider the relational database of Figure 2.14. 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.

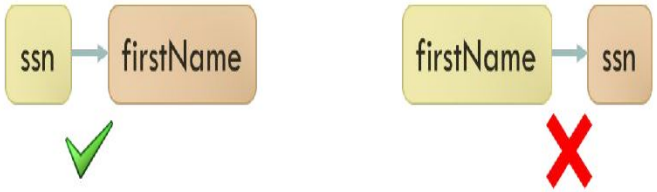
2.8 Consider the bank database of Figure 2.15. Give an expression in the relational algebra for each of the following queries.

a. Find the names of all branches located in "Chicago".

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

Functional Dependency

Patient		
ssn	firstName	lastName
235-14-7854	Sandra	Smith
192-48-0924	John	Moore
821-13-2108	Laura	Turner
874-72-0093	John	Moore



Functional Dependency

- **Redundancy** in relational databases is often caused by a functional dependency
- **A functional dependency (FD)** : a link between two sets of attributes in a relation
- We can **normalize a relation by removing undesirable FD**
- A set of attributes, A, functionally determines another set, B, or: there exists a functional dependency between A and B (A ->B)
- If whenever two rows of the relation have the same values for all the attributes in A, then they also have the same values for all the attributes in B.

Functional Dependencies Continued

Example

ID	First	Last	modCode	modName
11	Bob	Smith	G123	DBMS
21	John	Bloggs	G213	C++
13	Tom	Swift	G435	Java

Set of FDs :

1. {ID} - >{First,Last}
2. {ID,modCode}->{First,Last,modName}
3. {modCode}->{modName}

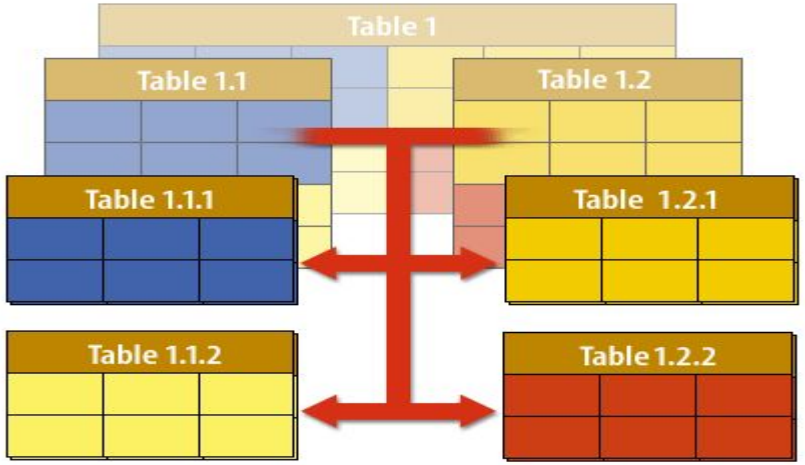
Represented by an arrow sign (→) that is, X→Y, where X functionally determines Y.
The left-hand side attributes determine the values of attributes on the right-hand side.

Armstrong's Axioms

- **Closure** : If **F** is a set of functional dependencies then the closure of F, denoted as F^+, is the set of all functional dependencies logically implied by F.
- Set **of rules, that when applied repeatedly, generates a closure** of functional dependencies.
- **Rules are as follows:**

- **Reflexive rule** : *if* $Y \subseteq X$ *then* $X \rightarrow Y$
- **Augmentation rule** : *if* $X \rightarrow Y$ *then* $XZ \rightarrow YZ$ *for any* Z
- **Transitivity rule** : *if* $X \rightarrow Y$ *and* $Y \rightarrow Z$ *then* $X \rightarrow Z$
- **Union** : *if* $X \rightarrow Y$ *and* $X \rightarrow Z$ *then* $X \rightarrow YZ$
- **Decomposition** : *if* $X \rightarrow YZ$ *then* $X \rightarrow Y$ *and* $X \rightarrow Z$
- **Pseudo Transitivity** : *if* $X \rightarrow Y$ *and* $YZ \rightarrow W$ *then* $XZ \rightarrow W$

Database Normalization



Database Normalization : Need

What is an Anomaly?

Anything we try to do with a database that leads to unexpected and/or unpredictable results.

- Three types of Anomaly to guard against:
  - ***insert***
  - ***delete***
  - ***update***
- Need to check your database design carefully:
  - the only good database is an anomaly free database.

Reflection Spot 1

Question. Consider the database given below. Suppose we have built a new room B112 but it has not been added to the time-table what can be the problem for inserting room information?

CoNo	Tutor	Room	RSize	EnLimit
351	Clark	A401	45	40
354	Turner	C502	100	60
434	Smith	B101	430	320
677	Stella	H607	400	45

1.Insert Anomaly

Answer : The attempt to insert room details will be prevented/not allowed unless and until it has been added in the time table and has associated Course and tutor assigned.

- When we want to enter a value into a data cell but the attempt is prevented, as another value is not known it leads to Insert Anomaly

CoNo	Tutor	Room	RSize	EnLimit
351	Clark	A401	45	40
354	Turner	C502	100	60
434	Smith	B101	430	320
677	Stella	H607	400	45

2. Delete Anomaly

- When a value we want to delete also means we will delete values we wish to keep.

- e.g. If CoNo 351 is deleted, but Room A401will be used elsewhere.

CoNo	Tutor	Room	RSize	EnLimit
351	Clark	A401	45	40
354	Turner	C502	100	60
434	Smith	B101	430	320
677	Stella	H607	400	45

3.Update Anomaly

- When we want to change a single data item value, but must update multiple entries

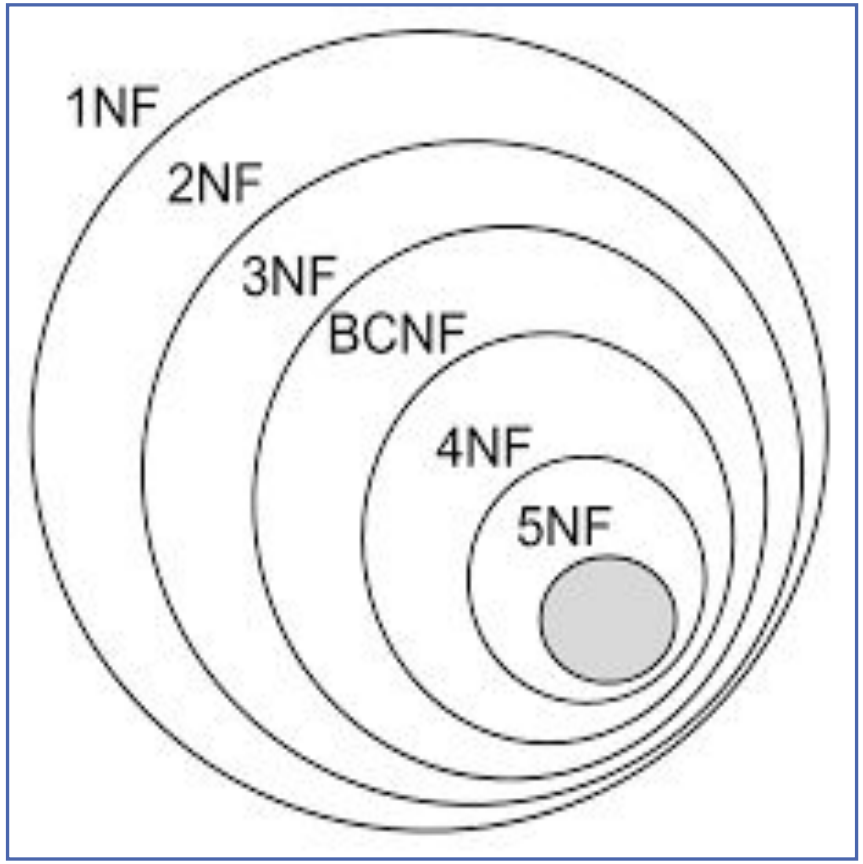
e.g. Room H940 has been improved, it is now of RSize = 500. (Note:- Here in this table is incomplete to see imformation of H940.)

CoNo	Tutor	Room	RSize	EnLimit
351	Clark	A401	45	40
354	Turner	C502	100	60
434	Smith	B101	430	320
677	Stella	H607	400	45

Database Normalization Forms

Normal Forms :

- 1 NF (Atomicity)
- 2 NF (Remove Partial Dependency)
- 3NF (Remove Transitive Dependency)
- Boyce Codd NF (Super key)
- 4 NF (Multi-valued Dependencies)
- 5 NF (Join Dependency)



1 NF : First Normal Form

A method to remove all these anomalies and bring the database to a consistent state.

Consider the relation Course_info

Course	Content
Programming	Java,C
Scripting	JSP,Javascript,PHP

Rules :

- ***All the attributes in a relation must have atomic domains.***
- ***The values in an atomic domain must be indivisible units.***

Conversion to 1NF

Course	Content
Programming	Java
Programming	C
Scripting	JSP,
Scripting	Javascript,
Scripting	PHP

- **Each attribute must contain only a single value from its pre-defined domain.**

2 NF : Second Normal Form

2 NF

Prime attribute − An attribute, which is a part of the prime-key, is known as a prime attribute.

Non-prime attribute − An attribute, which is not a part of the prime-key, is said to be a non-prime attribute.

Consider the relation student_project

{Stud_id, Proj_id, Stud_name, Project_Title}

Rule : Every non-prime attribute should be fully functionally dependent on prime key attribute. That is, if X → A holds, then there should not be any proper subset Y of X, for which Y → A also holds true.

Conversion to 2 NF

From example , we find that Stu_Name can be identified by Stu_ID and Proj_Name can be identified by Proj_ID independently. This is called **partial dependency**, which is not allowed in Second Normal Form.

Therefore, we can convert the relation into as shown below

Student *{Stud_id,Stud-name} and*

Stud_id	Stud_name

Project *{Proj_id,Project_Title}*

Prod_id	Project_Title

3 NF : Third Normal Form

Rules :

- For a relation to be in Third Normal Form, it **must be in Second Normal form** and the following must satisfy −
- No non-prime attribute is transitively dependent on prime key attribute.
- For any non-trivial functional dependency, X → A, then either
  - **X is a super key or,**
  - **A is prime attribute.**

Consider relation Stud_info

Stud_info *{rno,name,marks,zip,city,dob}*

rno	name	marks	zip	city	dob

Reflection Spot 3

Question. Does the above relation satisfy the 3 NF criteria??If no convert it to 3 NF.

3 NF : Third Normal Form

Ans : No. The given relation does not satisfy the 3NF as it contains following transitive dependency:

We have : *Rno->zip but zip->city*

Therefore *Rno->city* ***(Transitive Dependency)***

Conversion to 3NF :

Stud_info *{rno,name,marks,dob,zip}*

rno	name	marks	dob	zip

Zip_city *{zip,city}*

zip	city

References

1.Ramakrishnan, R. and Gherke, J., “Database Management Systems”, 3rd Ed., McGraw-Hill.

2. Connally T, Begg C.,”Database Systems”,Pearson Education

End of Unit 1