ACDS Lecture Series

Lecture - 2

CSIR

Database Development

G. N. Sastry & Team

ADVANCED COMPUTATION AND DATA SCIENCES (ACDS) DIVISION

CSIR-North East Institute of Science and Technology, Jorhat, Assam, India

2. Database Development

2.1 Overview

Database: A collection of related data.

Management: Define the structures for storage of data and provide mechanisms for manipulation.

Database Management System (DBMS): The software that allow users to define, create, maintain, manage and control access to the databases ensuring the safety and security of the data stored despite system crashes or attempts at unauthorized access.

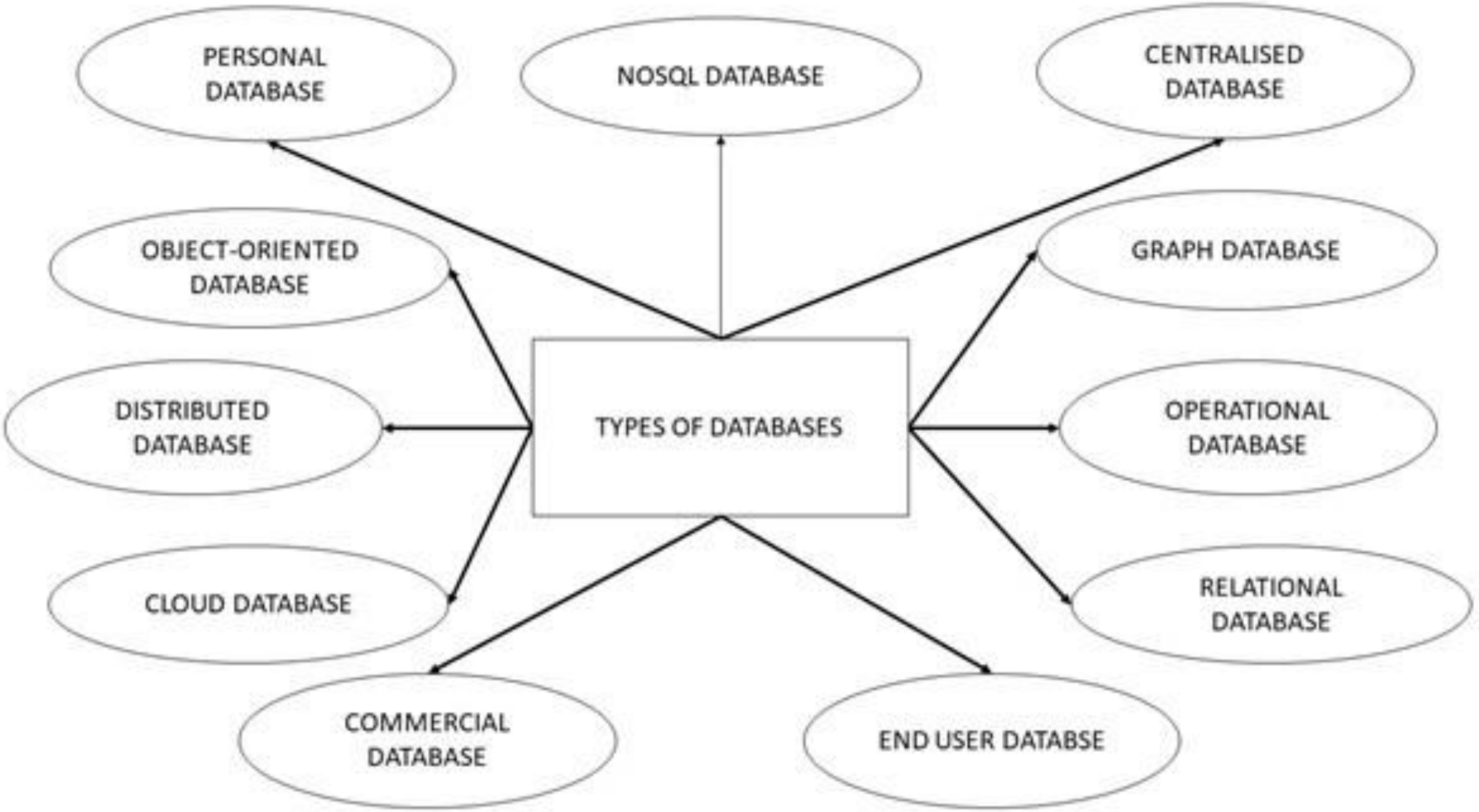
(Database) Application Programs

- A computer program that interacts with database by issuing request (like SQL statement) to DBMS.
- Users interact with the database through a number of application programs that are used to create and maintain the database and to generate information.
- These programs can be batch applications or more typically nowadays, they will be online applications.
- The application programs may be written in some programming language.

Applications

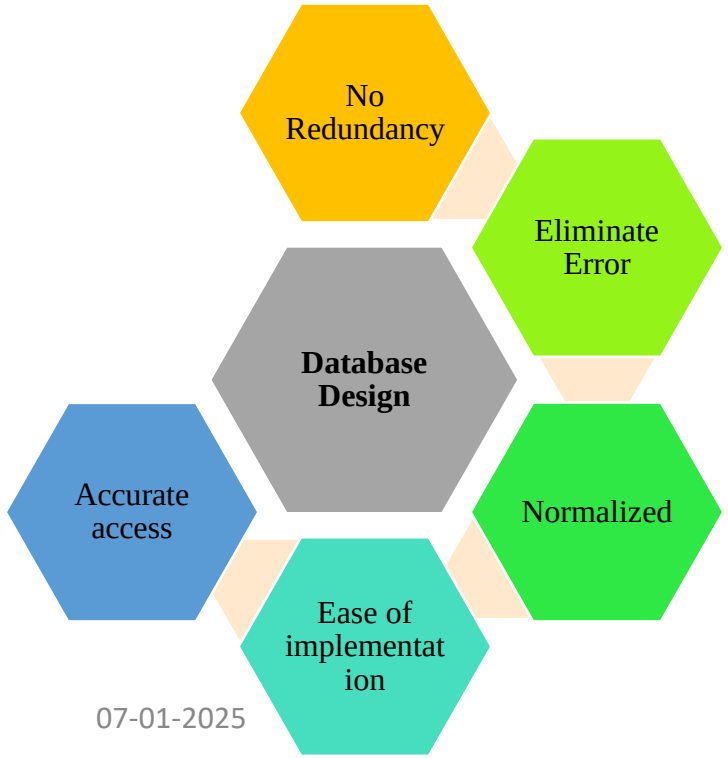
- Purchases using your credit card
- Booking a ticket at the travel agents
- Using the local library
- Purchases from the supermarket

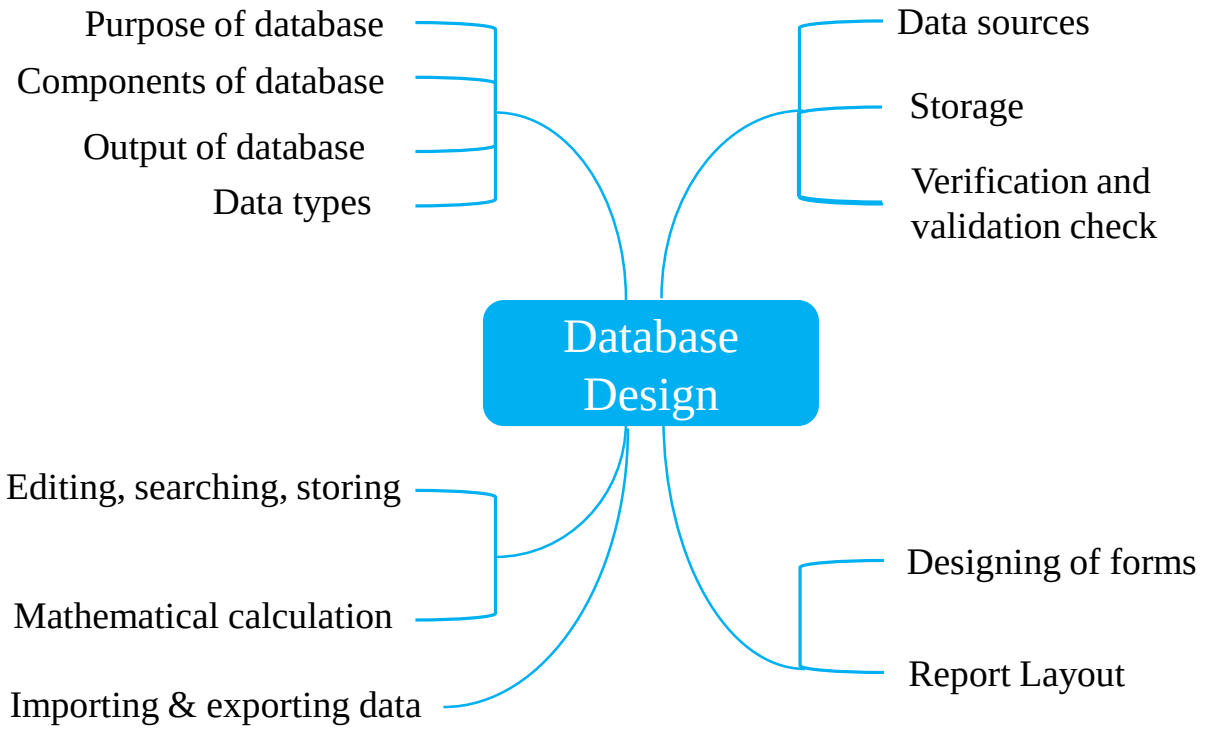
2.1 Database Types



2.2 Why Database Design is Important

- **Designing a database refers to designing the structure of database to store and manage data**
- **The success of a database is determined by the effectiveness of the database design**
- **A well designed database facilitates data management and generates accurate and valuable information**
- **Importance of a good design**
  - **Eliminates data redundancy**
  - **Eliminates errors leading to bad decision**

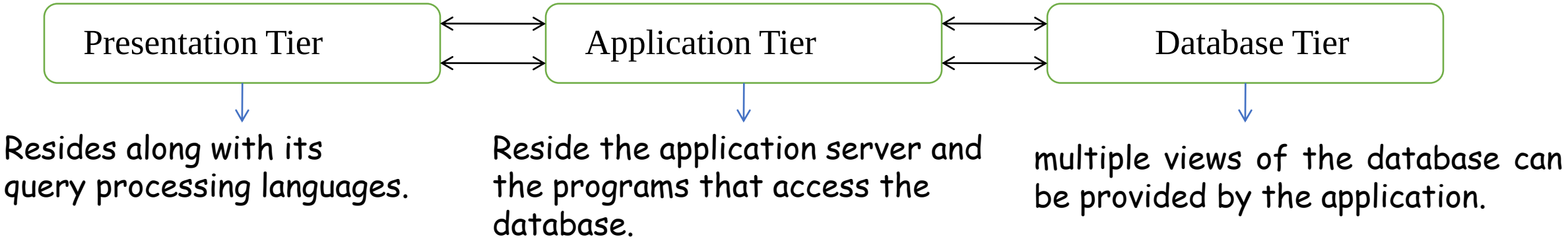




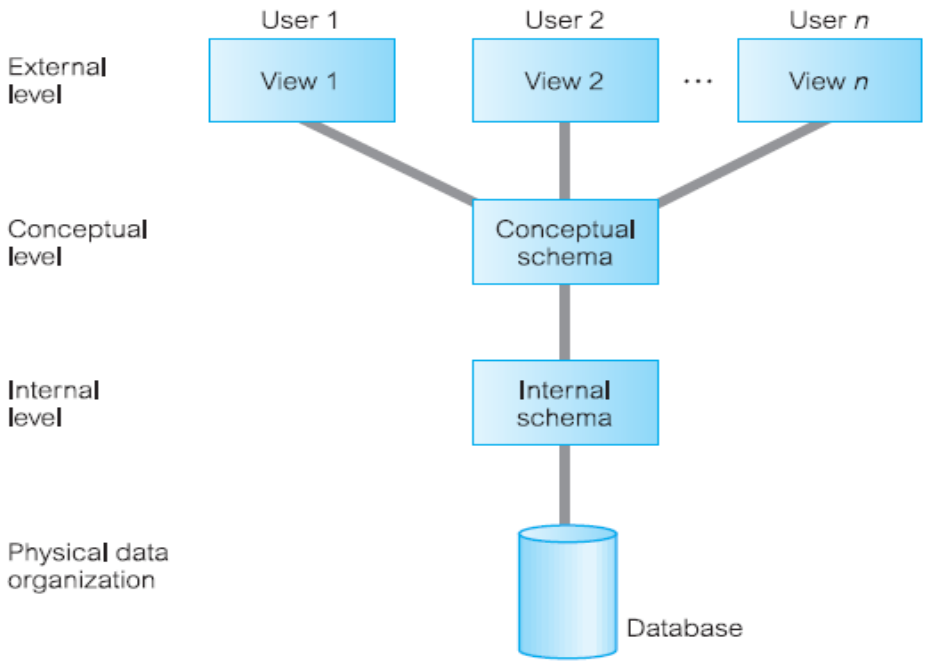
2.3 Database Architecture

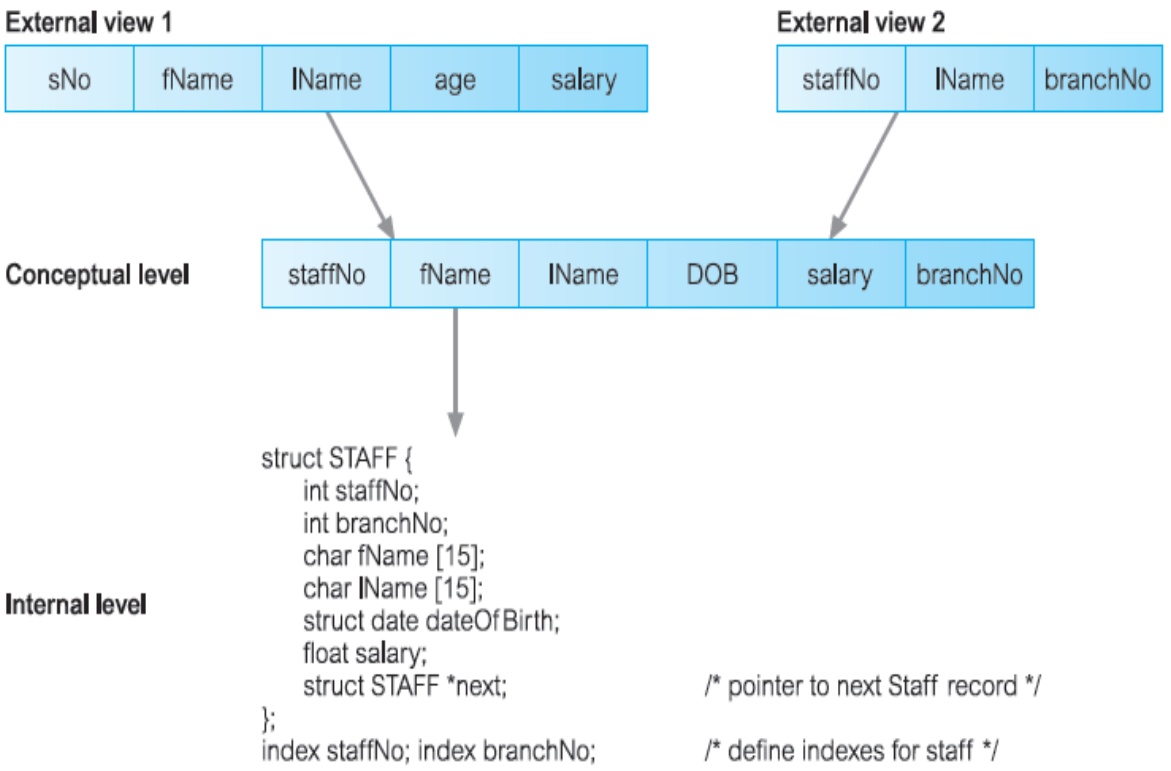
Database Management System:- stores data in such a way that it becomes easier to retrieve, manipulate and produce information.

- The design of a DBMS depends on its architecture. The architecture of a DBMS can be seen as either single tier or multi-tier.
- In 1-tier architecture, the DBMS is the only entity where the user directly sits on the DBMS and uses it.
- If the architecture of DBMS is 2-tier, then it must have an application through which the DBMS can be accessed. Programmers use 2-tier architecture where they access the DBMS by means of an application.
- A 3-tier architecture separates its tiers from each other based on the complexity of the users and how they use the data present in the database.



2.3 ANSI SPARC Architecture





- **Conceptual Level**
  - Complete view the logical structure of the entire database.
  - Describes *what* data is stored in the database that is independent of any storage considerations and the relationships among the data.
  - The conceptual level represents:
    - ✓ all entities, their attributes, and their relationships;
    - ✓ the constraints on the data;
    - ✓ semantic information about the data;
    - ✓ security and integrity information.
  - The conceptual level supports each external view, in that any data available to a user must be contained in, or derivable from, the conceptual level.

- **Internal Level**
  - The physical representation of the database on the computer.
  - Describes *how* the data is stored in the database.
  - Describe the physical implementation of the database i.e. to store data on storage devices using data structures and file organizations methods (indexing, hashing) to achieve optimal runtime performance and storage space utilization.
- **External Level**
  - The way each user perceive or view the data.
  - Consists of number of different external views of the database for different users.
  - Different views may have different representations of same data. For example, one user may view dates in the form (day, month, year) while another may view dates as (year, month, day).
  - Some views might include derived or calculated data that are not actually stored in the database as such but created when needed. For example, it is unlikely that ages would be stored as this data would have to be updated daily. But DOB would be stored and age will be calculated by the DBMS when it is referenced.
  - All in all, the external view includes only those entities, attributes, and relationships in the 'real world' that the user is interested in. Other entities, attributes, or relationships that are not of interest may be represented in the database, but the user will be unaware of them.

Database Schema (Intension): The overall description of the database**.**

- There are three different types of schema in the database defined according to the levels of abstraction of the three-level architecture .
  - At highest level, it is multiple external schemas. Each external schema describes the part of the database that a particular user group is interested in and hides the rest of the database from that user group.
  - At the conceptual level, we have conceptual schema, which describes all the entities, attributes, and relationships together with integrity constraints that describes the structure of the whole database.
  - At the lowest level of abstraction we have the internal schema, which is a complete description of the physical storage structure of the database.

Mapping between these three types of schema must for consistency

- Each external schema is related to the conceptual schema by the external/conceptual mapping. The conceptual schema is related to the internal schema through a conceptual/internal mapping. There is only one conceptual schema and one internal schema per database.

Database Instance (extension or state): The actual data in the database at any particular point in time. Many database instances can correspond to the same database schema as actual data in the database may change frequently; it changes every time whenever we insert or delete a new data.

STUDENT

Name	StudentNumber	Class	Major

COURSE

CourseName	CourseNumber	CreditHours	Department

PREREQUISITE

CourseNumber	PrerequisiteNumber

SECTION

SectionIdentifier	CourseNumber	Semester	Year	Instructor

GRADE_REPORT

StudentNumber	SectionIdentifier	Grade

STUDENT

Name	Student_number	Class	Major
Smith	17	1	CS
Brown	8	2	CS

COURSE

Course_name	Course_number	Credit_hours	Department
Intro to Computer Science	CS1310	4	CS
Data Structures	CS3320	4	CS
Discrete Mathematics	MATH2410	3	MATH
Database	CS3380	3	CS

SECTION

Section_identifier	Course_number	Semester	Year	Instructor
85	MATH2410	Fall	07	King
92	CS1310	Fall	07	Anderson
102	CS3320	Spring	08	Knuth
112	MATH2410	Fall	08	Chang
119	CS1310	Fall	08	Anderson
135	CS3380	Fall	08	Stone

GRADE_REPORT

Student_number	Section_identifier	Grade
17	112	B
17	119	C
8	85	A
8	92	A
8	102	B
8	135	A

PREREQUISITE

Course_number	Prerequisite_number
CS3380	CS3320
CS3380	MATH2410
CS3320	CS1310

2.4 Data Models

- A collection of concepts that can be used to describe the structure of a database. Structure of a database mean the data types, relationships and constraints apply to the data.
- There are many data models proposed which fall into three broad categories:
    1. **Record-based or Representational-based**
    2. **Object-based**
    3. **Physical data models**

Record-Based Data Models:

- In a record-based model, the database consists of a number of fixed-format records.
- Each record type defines a fixed number of fields, each typically of a fixed length.
- There are three principal types of record-based logical data model
    - **Network data model:** Data is represented as collections of records, and relationships are represented by sets. The records are organized as graph structures with records appearing as nodes and edges as sets.
    - **Hierarchical data model:** Represented as a tree graph, allows a node to have only one parent
    - **Relational data model:** Based on the concept of mathematical relations. Data and relationships are represented as tables, each of which has a number of columns with unique name.

Object-Based Data Models:

- Object-based data models use concepts such as entities, attributes, and relationships.
- An entity is a distinct object (a person, place, thing, concept, event) in the organization that is to be represented in the database.
- An attribute is a property that describes some aspect of the object that we wish to record, and a relationship is an association between entities.

Some of the more common types of object-based data model are:

- Entity–Relationship
- Semantic
- Functional
- Object-Oriented

Physical Data Models

- Physical data models describe how data is stored in the computer, representing information such as record structures, record orderings, and access paths.
- There are not as many physical data models as logical data models, the most common ones being the *unifying model* and the *frame memory*.

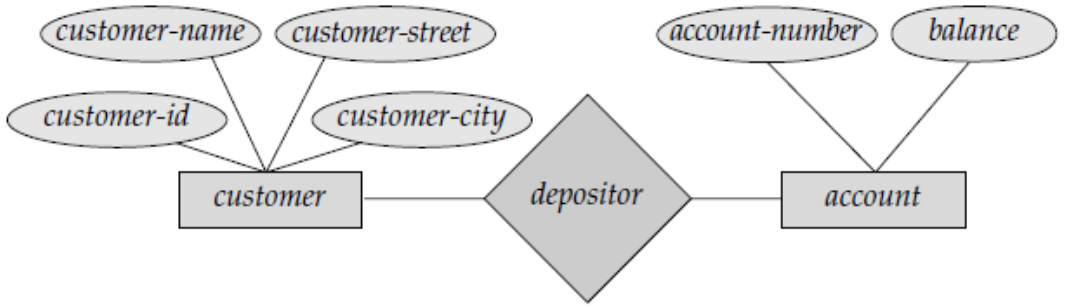
- The Record and Object models are used to describe data at the conceptual and external levels.
  - Record-based data models provide overall logical structure of the database but do not provide allowing the user to specify constraints on the data.
  - Object-based data models provide allowing the user to specify constraints on the data but do not provide overall logical structure specification.
- The physical data model is used to describe data at the internal level i.e. provide concepts that describe the details of how data is stored on the computer storage media, typically.

Branch

branchNo	street	city	postCode
B005	22 Deer Rd	London	SW1 4EH
B007	16 Argyll St	Aberdeen	AB2 3SU
B003	163 Main St	Glasgow	G11 9QX
B004	32 Manse Rd	Bristol	BS99 1NZ
B002	56 Clover Dr	London	NW10 6EU

Staff

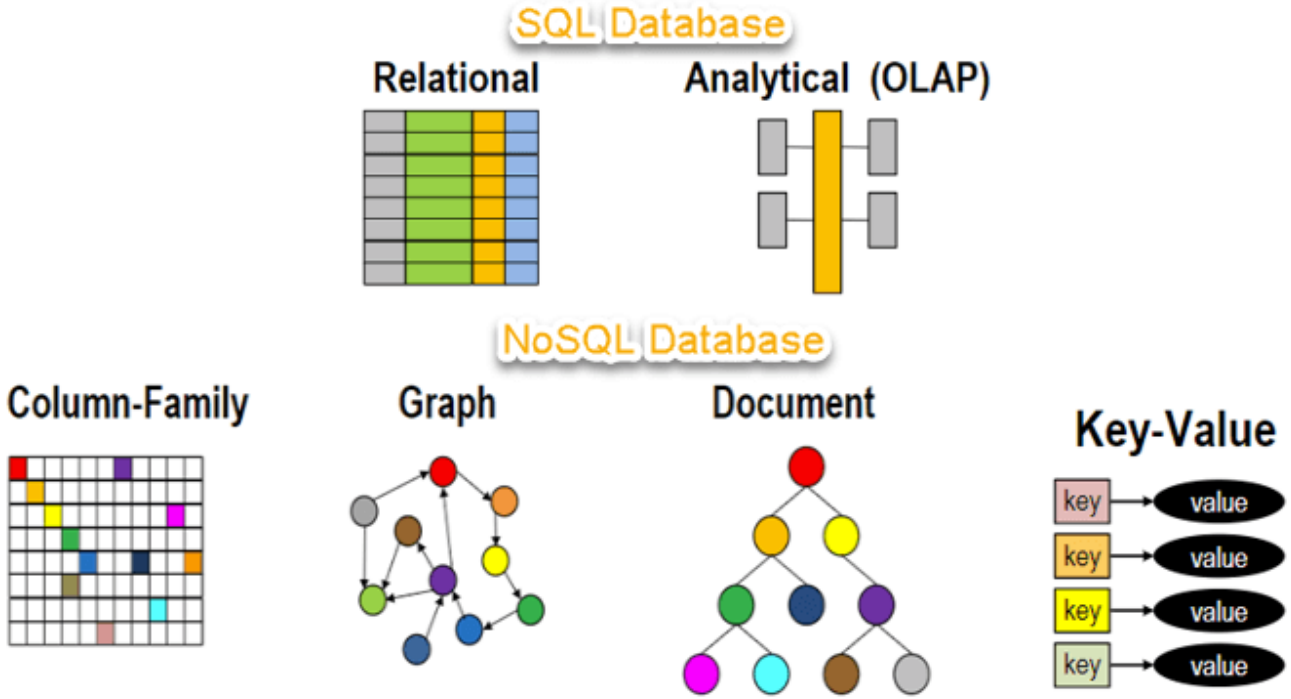
staffNo	fName	lName	position	sex	DOB	salary	branchNo
SL21	John	White	Manager	M	1-Oct-45	30000	B005
SG37	Ann	Beech	Assistant	F	10-Nov-60	12000	B003
SG14	David	Ford	Supervisor	M	24-Mar-58	18000	B003
SA9	Mary	Howe	Assistant	F	19-Feb-70	9000	B007
SG5	Susan	Brand	Manager	F	3-Jun-40	24000	B003
SL41	Julie	Lee	Assistant	F	13-Jun-65	9000	B005



2.4 Relational vs. Non Relational

Relational

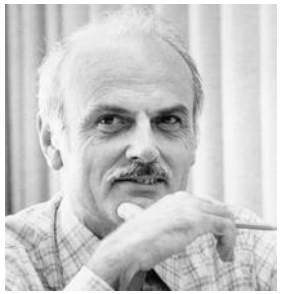
- MySQL
- PostgreSQL
- SQLite3
- Oracle



Non-Relational

- Key Value – Amazon DynamoDB, Redis
- Graph – Sones, InfiniteGraph
- Column – HBASE, Cassandra
- Document – MongoDB, CouchDB

2.4 Relational Data Structure



(Proposed by Edgar. F. Codd "70)

Relation: A two-dimensional table with columns and rows.

Attribute: A named column of a relation. Attributes can appear in any order.

Domain: A domain is the set of allowable values for one or more attributes.

Every attribute in a relation is defined on a domain.

Tuple: A tuple is a row of a relation. There are no duplicate tuples within a relation.

Degree: The degree of a relation is the number of attributes it contains.

A relation with only one attribute would have degree one and be called a unary relation.

Similarly a relation with two or 3 or n attributes are called binary, ternary and n-ary respectively.

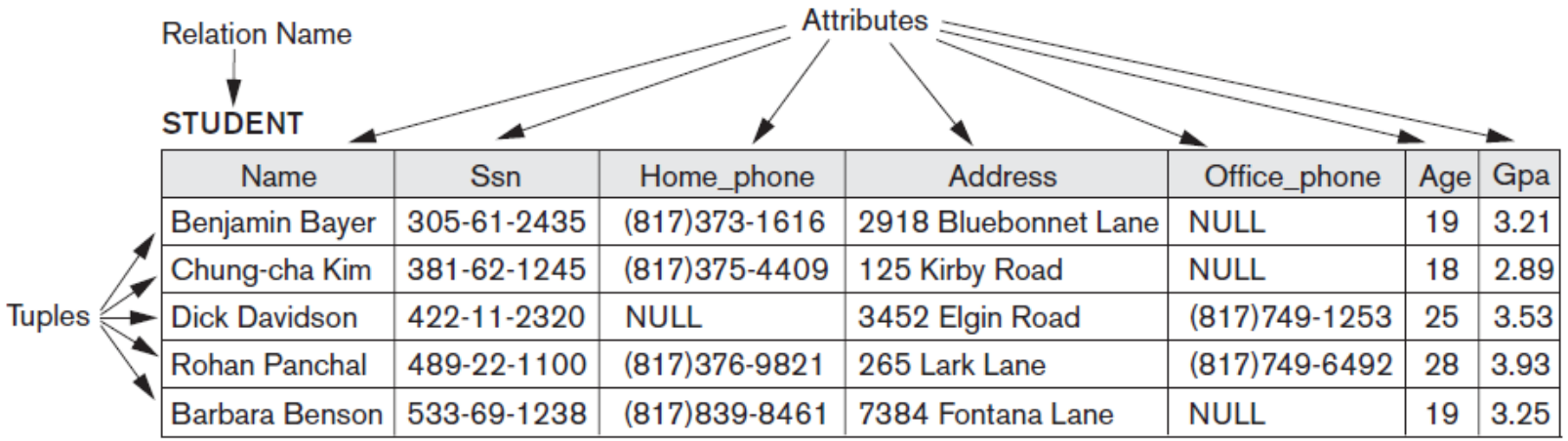
Cardinality: The cardinality of a relation is the number of tuples it contains.

Null: Represents a value of an attribute that is unknown for a tuple.

A null is not the same as a zero or spaces which are values, but a null represents absence of a value.

Example

- All data is logically structured within relation (A two dimensional table).
- Each relation has a name and is made up of named attributes (columns).
- Each tuple (row) contains one value per attribute.



Name	Ssn	Home_phone	Address	Office_phone	Age	Gpa
Benjamin Bayer	305-61-2435	(817)373-1616	2918 Bluebonnet Lane	NULL	19	3.21
Chung-cha Kim	381-62-1245	(817)375-4409	125 Kirby Road	NULL	18	2.89
Dick Davidson	422-11-2320	NULL	3452 Elgin Road	(817)749-1253	25	3.53
Rohan Panchal	489-22-1100	(817)376-9821	265 Lark Lane	(817)749-6492	28	3.93
Barbara Benson	533-69-1238	(817)839-8461	7384 Fontana Lane	NULL	19	3.25

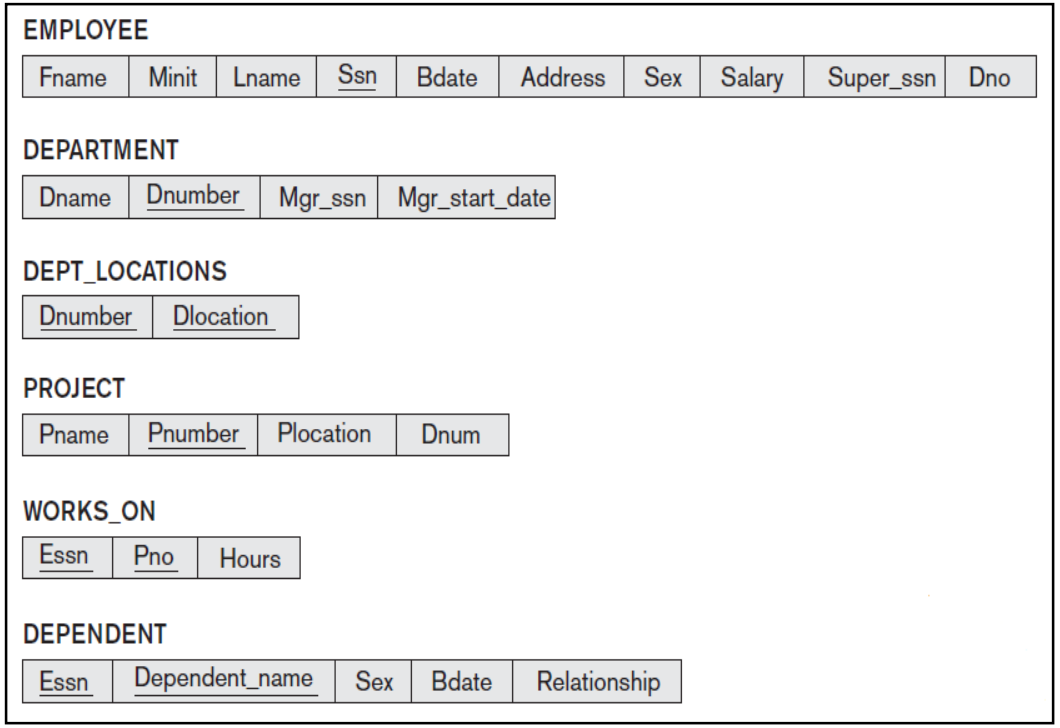
Alternate names:

Formal terms	Alternative 1	Alternative 2
Relation	Table	File
Tuple	Row	Record
Attribute	Column	Field

2.4 Relational Schema

Relational database schema: A set of relation schemas, each with a distinct name.

- If *R*1, *R*2, . . . , *Rn* are a set of relation schemas, then we can write the *relational database schema*, or simply *relational schema*, *R*, as: *R* = {*R*1, *R*2, . . . , *Rn*}



EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000	333445555	5
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
Alicia	J	Zelaya	999887777	1968-01-19	3321 Castle, Spring, TX	F	25000	987654321	4
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5
Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston, TX	M	25000	987654321	4
James	E	Borg	888665555	1937-11-10	450 Stone, Houston, TX	M	55000	NULL	1

DEPARTMENT

Dname	Dnumber	Mgr_ssn	Mgr_start_date
Research	5	333445555	1988-05-22
Administration	4	987654321	1995-01-01
Headquarters	1	888665555	1981-06-19

DEPT_LOCATIONS

Dnumber	Dlocation
1	Houston
4	Stafford
5	Bellaire
5	Sugarland
5	Houston

Super key: An attribute, or set of attributes, that uniquely identifies a tuple in a relation.

- However, a super key may contain additional attributes that are not necessary for unique identification
- we are interested in identifying super keys that contain only the minimum number of attributes necessary for unique identification.

Candidate key: A super key such that has no proper subset within the relation.

- A candidate key or simply key *K*, for a relation *R* has two properties:
  - **Uniqueness** – In each tuple of *R*, the values of *K* uniquely identify that tuple
  - **Irreducibility** – no proper subset of *K* has the uniqueness property
- There may be several CK for a relation. When a key consists of more than one attribute, we call it a composite key.

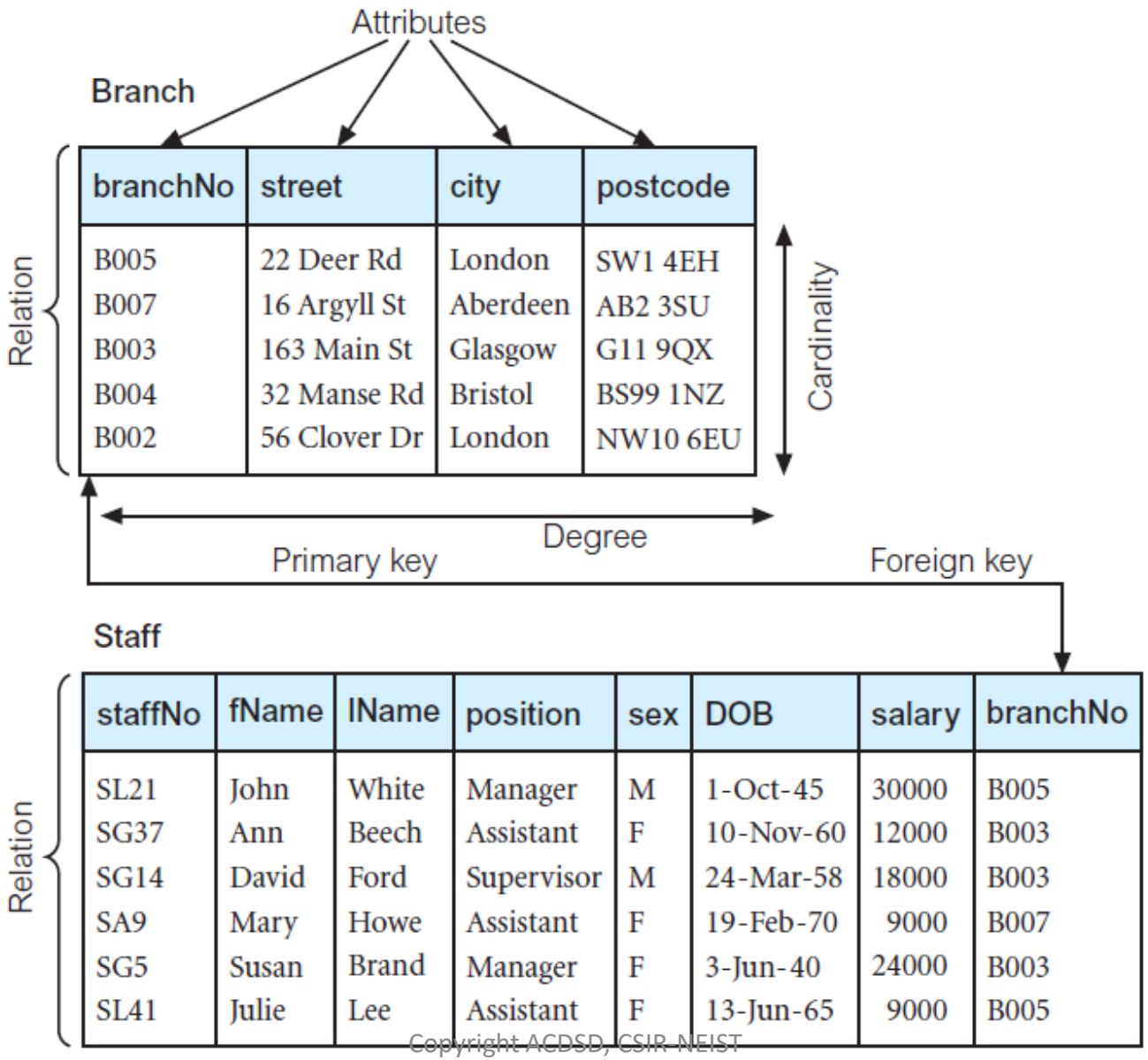
Primary key : The candidate key that is selected to identify tuples uniquely in a relation.

- The candidate keys that are not selected to be the primary key are called *alternate keys.*
- Since a relation has no duplicate tuples, it is always possible to identify each row uniquely i.e. a relation always has a primary key. In the worst case, the entire set of attributes could serve as the primary key

Foreign key: An attribute or set of attributes within one relation that matches the CK of some (possibly the same) relation.

- When an attribute appears in more than one relation, its appearance usually represents a relationship between tuples of the two relations.

2.4 Relational Keys



Branch

branchNo	street	city	postcode
B005	22 Deer Rd	London	SW1 4EH
B007	16 Argyll St	Aberdeen	AB2 3SU
B003	163 Main St	Glasgow	G11 9QX
B004	32 Manse Rd	Bristol	BS99 1NZ
B002	56 Clover Dr	London	NW10 6EU

Staff

staffNo	fName	lName	position	sex	DOB	salary	branchNo
SL21	John	White	Manager	M	1-Oct-45	30000	B005
SG37	Ann	Beech	Assistant	F	10-Nov-60	12000	B003
SG14	David	Ford	Supervisor	M	24-Mar-58	18000	B003
SA9	Mary	Howe	Assistant	F	19-Feb-70	9000	B007
SG5	Susan	Brand	Manager	F	3-Jun-40	24000	B003
SL41	Julie	Lee	Assistant	F	13-Jun-65	9000	B005

2.4 Integrity Constraints

- There are two relational integrity rules-
  1. Domain integrity constraint
  2. Key integrity constraint
  3. Entity integrity constraint

  applies to the primary keys of base relations *(relation that corresponds to an entity in the conceptual schema)*

  4. Referential integrity constraint → applies to foreign keys

Domain integrity: Every tuple value should be in in the domain of the attribute.

Key integrity: No two tuples can have same value on all attributes.

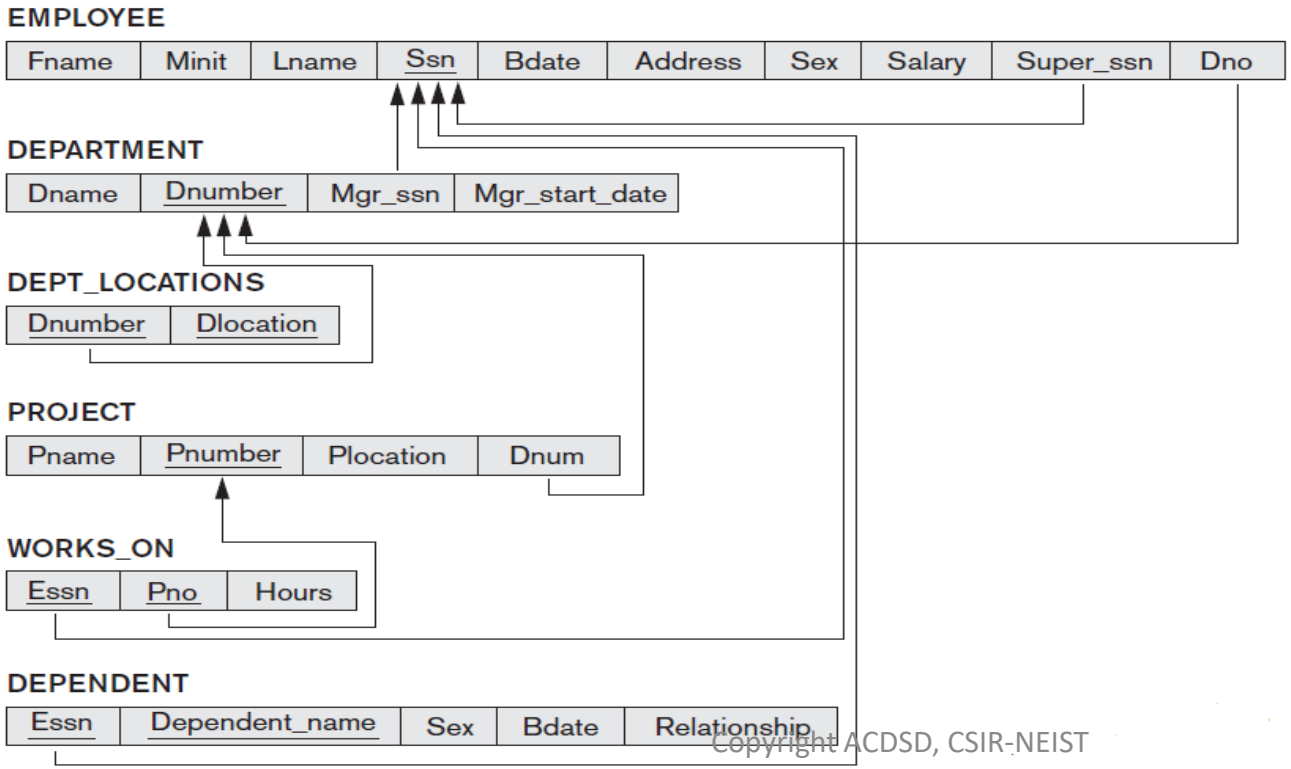
Entity integrity: In a base relation, no attribute of a primary key can be null.

By definition, a primary key is a minimal identifier that is used to identify tuples uniquely. Allowing a null for a primary key, contradicts the definition.

If we examine this rule in detail, we would find some anomalies. First, why does the rule apply only to primary keys and not more generally to candidate keys, which also identify tuples uniquely? Secondly, why is the rule restricted to base relations?

- **Referential integrity:** If a foreign key exists in a relation, either the FK key value must match a candidate key value of some tuple in its home relation or the FK value must be wholly null.
- **General Constraints:** Additional rules specified by the users or database administrators of a database that define or constrain some aspect of the enterprise.

  Example, if an upper limit of 20 has been placed upon the number of staff that may work at a branch office, then the user must be able to specify this general constraint.



Actions upon violating Integrity constraints

- Whenever we are trying to insert, delete and modify any tuple value to any attribute the above integrity constraints can be violated.

- On insert there may be violation of-
  - Domain constraint
  - Key constraint
  - Entity constraint
  - Referential entity constraint
- On modify there may be violation of-
  - Domain constraint
  - Key constraint
  - Entity constraint
  - Referential entity constraint
- **Actions to be taken in such cases:**
  1. **Reject (ignore):** reject the action.
  2. **Cascade:** Delete the reference.
  3. **Set null:** Set null in reference.

- On delete there may be violation of-
  - Referential entity constraint

- In the three-level ANSI-SPARC architecture, we described an external view as the structure of the database as it appears to a particular user.
- In the relational model, the word ‘view’ has a slightly different meaning.
- A view is a virtual or derived relation: a relation that does not necessarily exist in its own right, but may be dynamically derived from one or more base relations.

- **Base relation:** A named relation corresponding to an entity in the conceptual schema, whose tuples are physically stored in the database.
- **View:** The dynamic result of one or more relational operations operating on the base relations to produce another relation. A view is a virtual relation that does not necessarily exist in the database but can be produced upon request by a particular user.
- The contents of a view are defined as a query on one or more base relations. Any operations on the view are automatically translated into operations on the relations from which it is derived.
- Views are **dynamic**, meaning that changes made to the base relations that affect the view are immediately reflected in the view. When users make permitted changes to the view, these changes are made to the underlying relations.

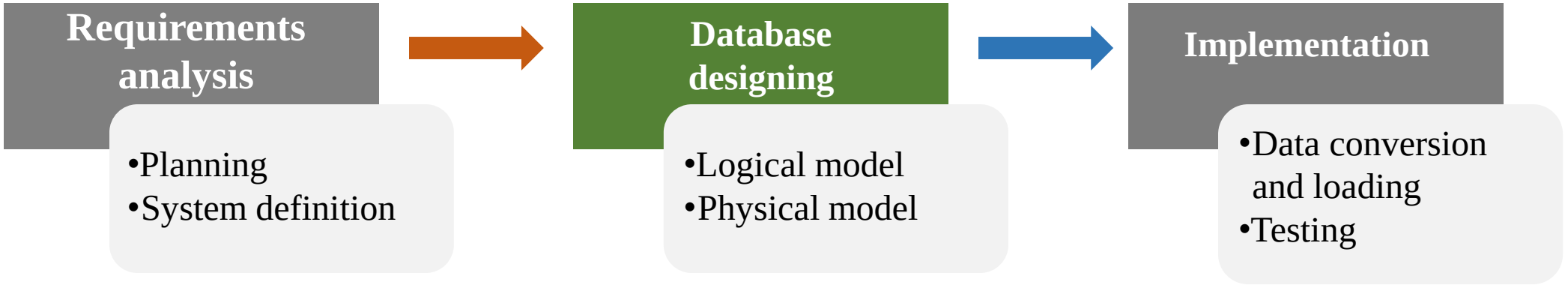
Purpose of Views

- The view mechanism is desirable for several reasons:
  - It provides a powerful and flexible security mechanism by hiding parts of the database from certain users.
  - Users are not aware of the existence of any attributes or tuples that are missing from the view.
  - It permits users to access data in a way that is customized to their needs, so that the same data can be seen by different users in different ways, at the same time.

Updating Views

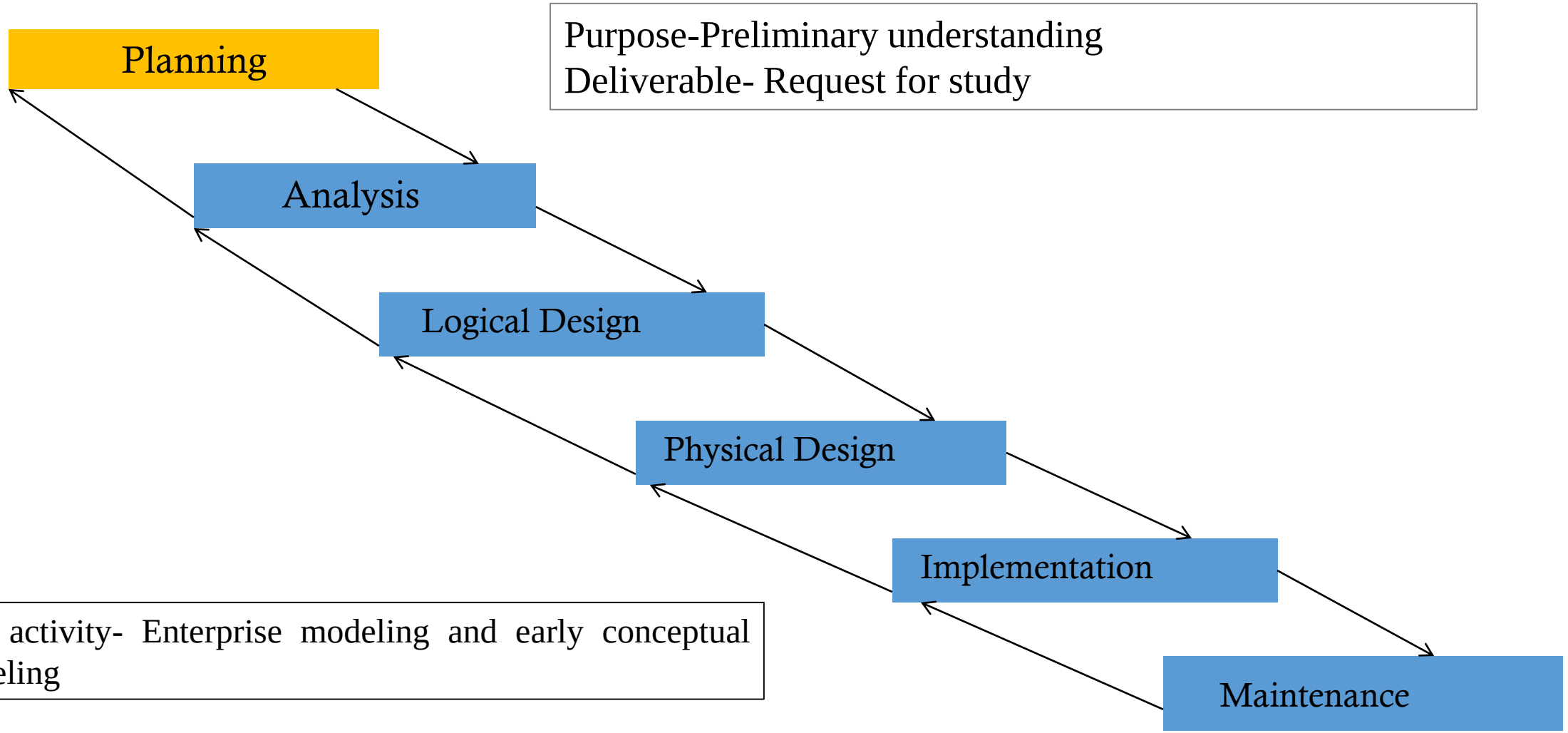
- All updates to a base relation should be immediately reflected in all views that reference that base relation. Similarly, if a view is updated, then the underlying base relation should reflect the change.
- However, there are restrictions on the types of modification that can be made through views. The conditions under which most systems determine whether an update is allowed through a view:
  - Updates are allowed through a view defined over a single base relation and containing either the primary key or a candidate key of the base relation.
  - Updates are not allowed through views involving multiple base relations.
  - Updates are not allowed through views involving aggregation or grouping operations.

2.5 Database Development life cycle

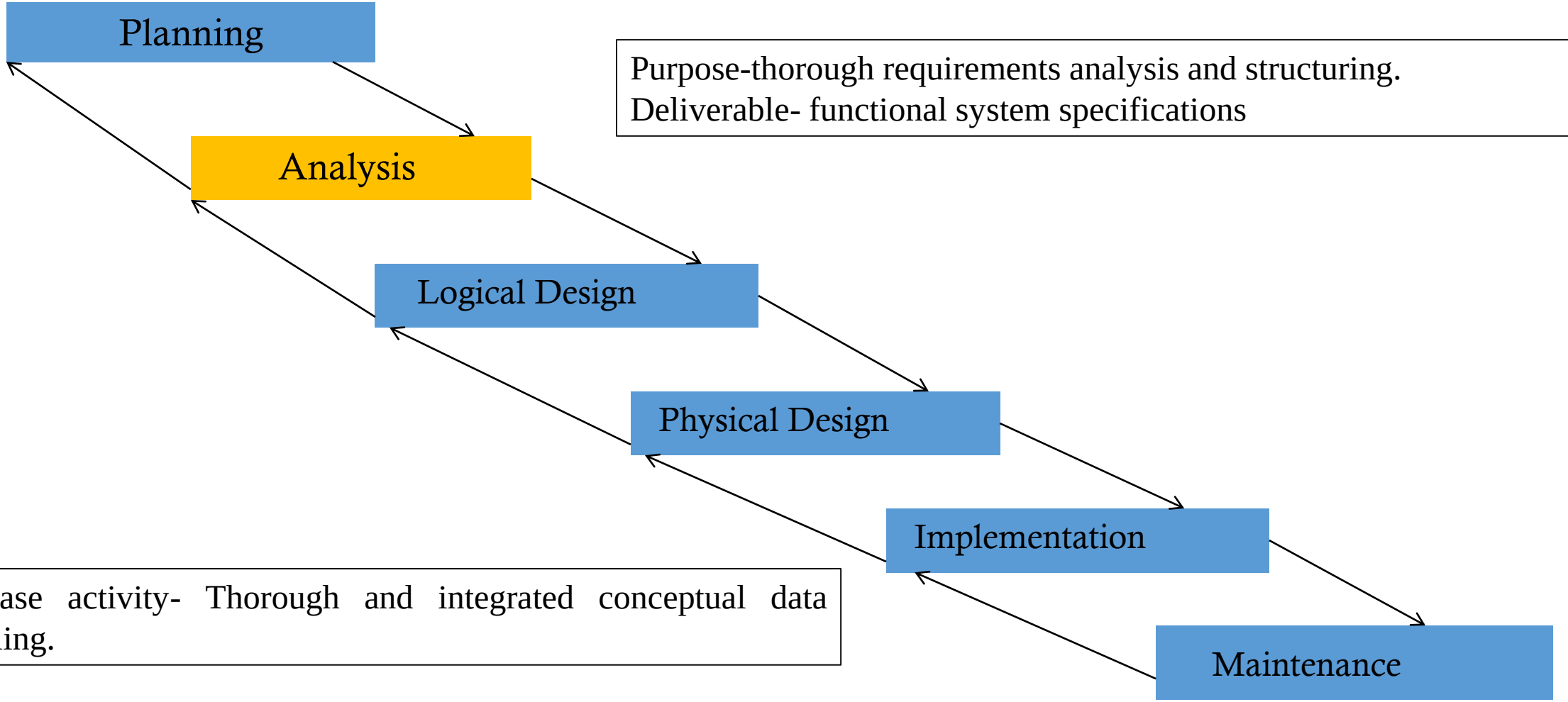


- The database development life cycle has a number of stages that are followed when developing database systems.
- The steps in the development life cycle do not necessary have to be followed religiously in a sequential manner.
- On small database systems, the database system development life cycle is usually very simple and does not involve a lot of steps.
- In order to fully appreciate the above diagram, let's look at the individual components listed in each step.

2.5 Systems Development Life Cycle



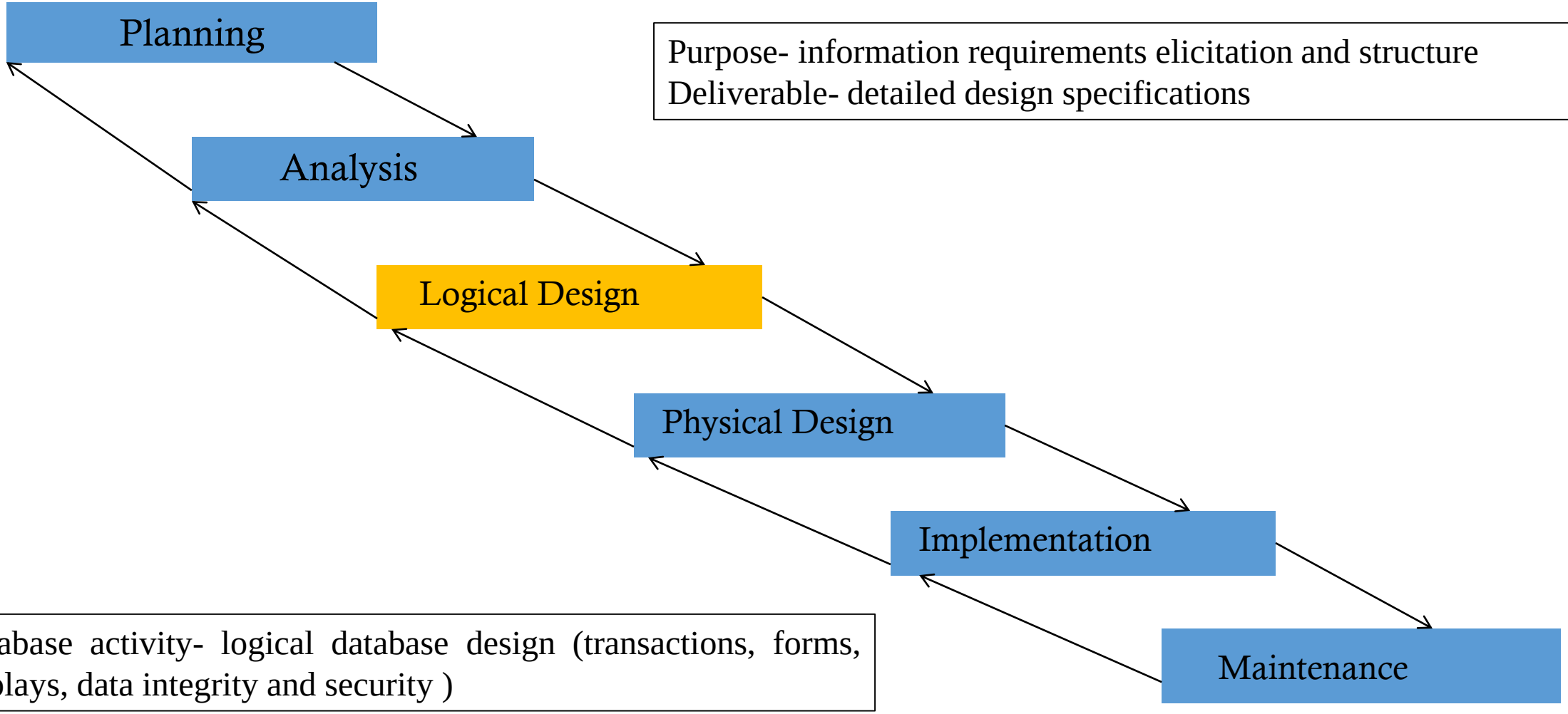
2.5 Systems Development Life Cycle



Purpose-thorough requirements analysis and structuring.
Deliverable- functional system specifications

Database activity- Thorough and integrated conceptual data modeling.

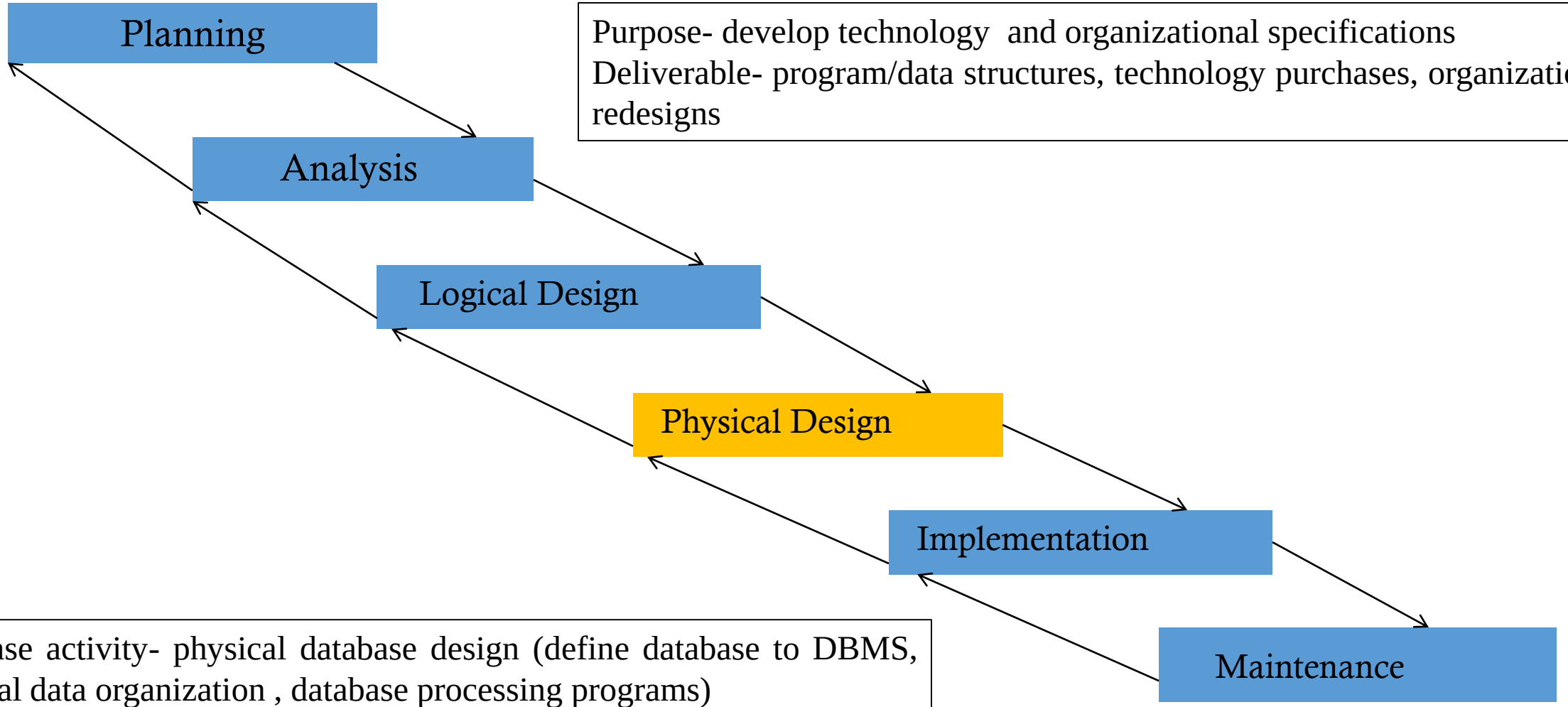
2.5 Systems Development Life Cycle



Purpose- information requirements elicitation and structure
Deliverable- detailed design specifications

Database activity- logical database design (transactions, forms, displays, data integrity and security)

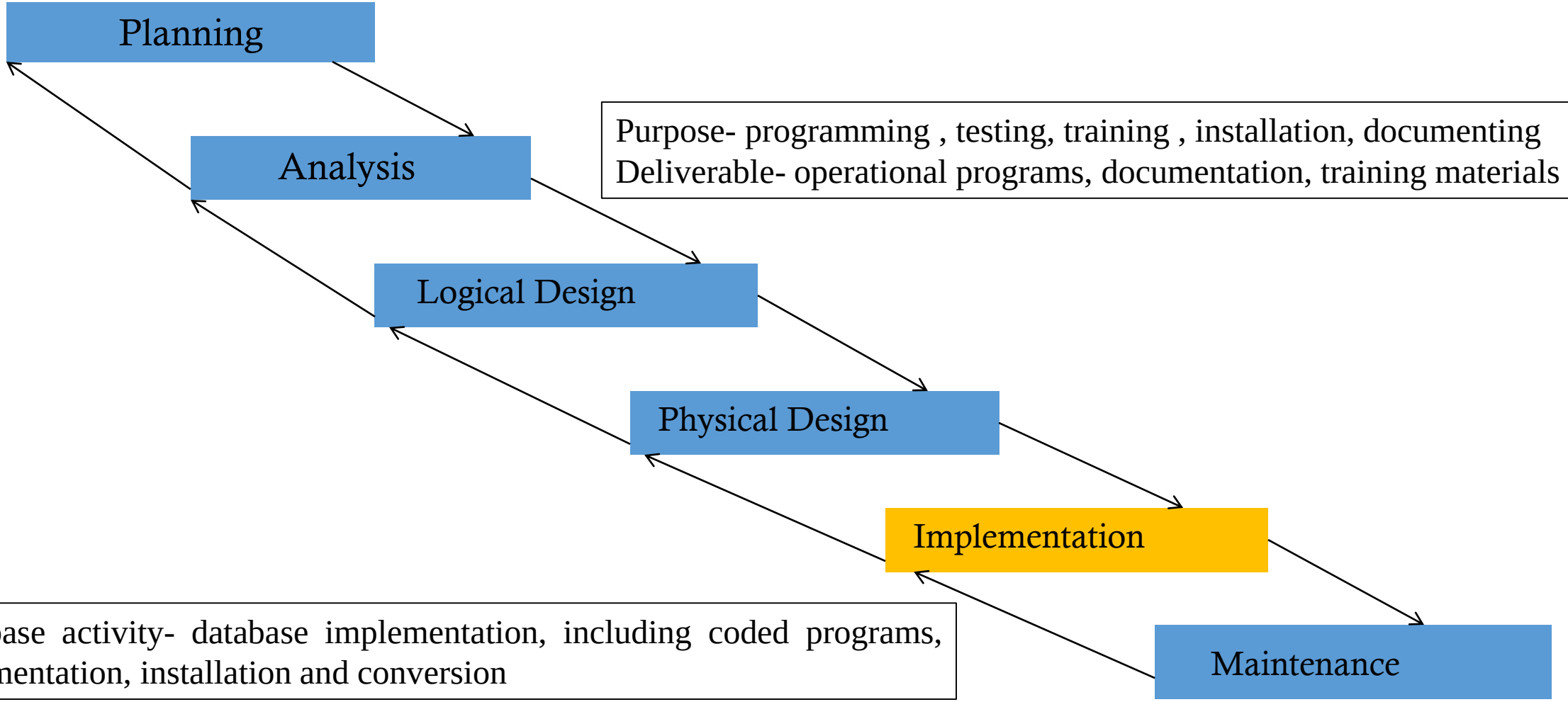
2.5 Systems Development Life Cycle



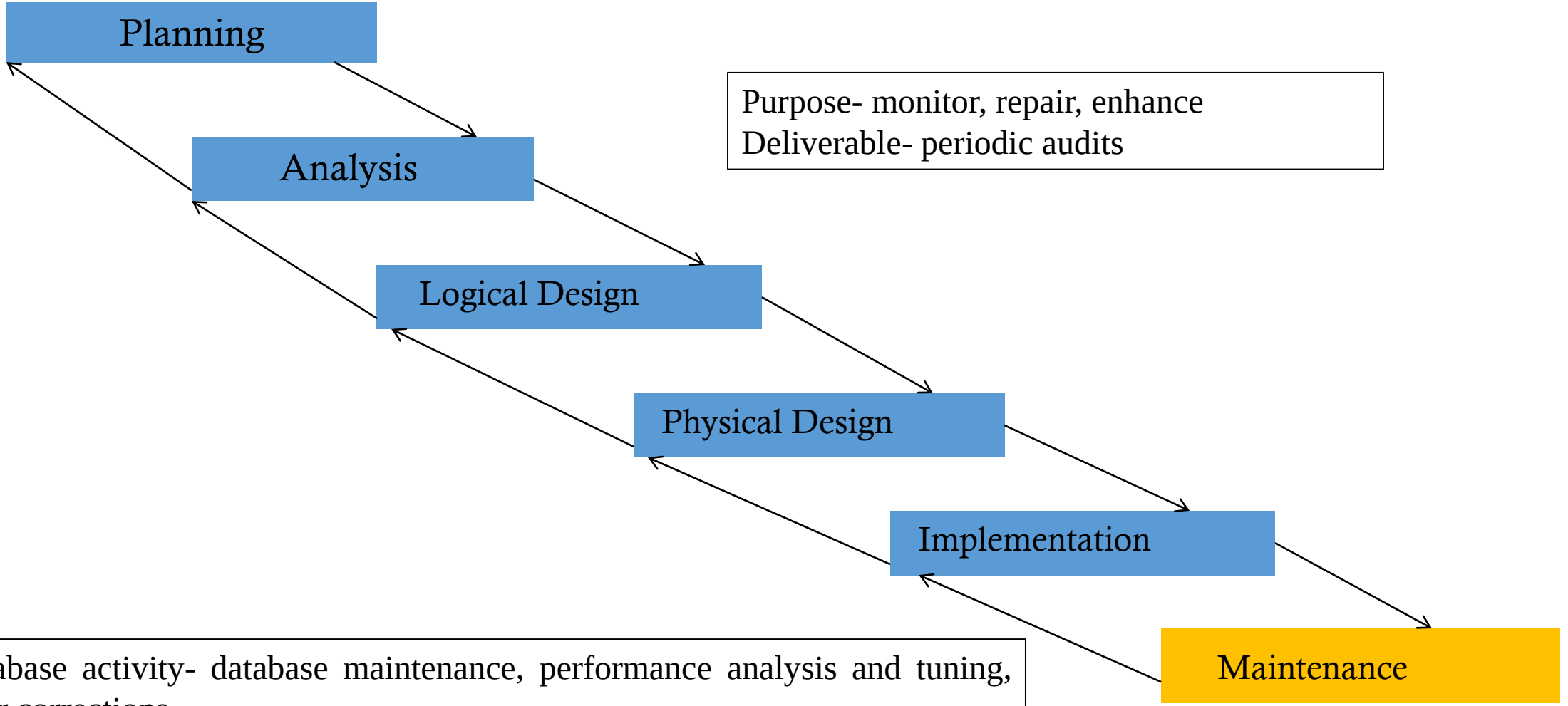
Purpose- develop technology and organizational specifications
Deliverable- program/data structures, technology purchases, organization redesigns

Database activity- physical database design (define database to DBMS, physical data organization , database processing programs)

2.5 Systems Development Life Cycle



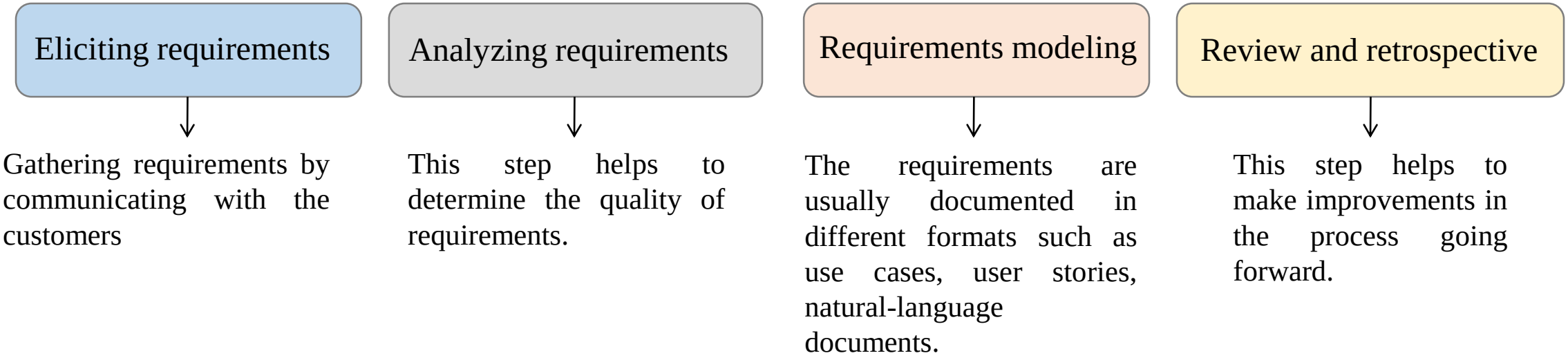
2.5 Systems Development Life Cycle



2.5 Systems Development Life Cycle

- Requirements analysis means to analyze, document, validate and manage software or system requirements. High-quality requirements are documented, actionable, measurable, testable, traceable, helps to identify business opportunities, and are defined to a facilitate system design.
- **Planning:** This stages concerns with planning of entire Database development Life Cycle.
- **System definition:** This stage defines the scope and boundaries of the proposed database system.

Process



2.6 Requirement Analysis

- Requirements analysis consists of two main activities:
  - Requirements gathering
  - Analysis of the gathered requirements
- Analyst gathers requirements through:
  - observation of existing systems,
  - studying existing procedures,
  - discussion with the customer and end-users,
  - analysis of what needs to be done, etc.
- ***Requirement Gathering:***
  - If the project is to automate some existing procedures, requirements can be gathered from
    - input and output formats
    - accurate details of the operational procedures
  - In the absence of a working system,
    - lot of imagination and creativity are required

- Interacting with the customer to gather relevant data:
  - requires a lot of experience.
- Some desirable attributes of a good system analyst:
  - Good interaction skills,
  - imagination and creativity,
  - experience.

- ***Analysis of gathering:***
  - After gathering all the requirements analyse it:
    - Clearly understand the user requirements,
    - Detect inconsistencies, ambiguities, and incompleteness.
  - Incompleteness and inconsistencies:
    - Resolved through further discussions with the end-users and the customers.
  - Inconsistent requirement:
    - Some part of the requirement: contradicts with some other part.
  - Incomplete requirement:
    - Some requirements have been omitted: due to oversight.

- **Logical Model:** This is concerned with developing a database model based on requirements.
- **Physical Model:** This stage implements the logical model of the database and physical implementation factors.
- Designing an efficient, useful database is a matter of following the proper process, including different phases:-

  1) Requirements analysis, or identifying the purpose of your database.

  2) Organizing data into tables.

  3) Specifying primary keys and analyzing relationships.

  4) Normalizing to standardize the tables.

2.7 Introduction

- After gathering and capturing information about what the users require of a database system and we have documented the requirements for the database system, we are ready to begin the database design stage.
- One of the most difficult aspects of database design is the fact that designers, programmers, and end-users tend to view data and its use in different ways. Unfortunately, unless we gain a common understanding that reflects how the enterprise operates, the design we produce will fail to meet the users' requirements.

- To ensure that we get a precise understanding of the nature of the data and how it is used by the enterprise, we need to have a model for communication that is non-technical and free of ambiguities.
- Entity–Relationship (ER) model is one such approach to design conceptual schema.

- ER modelling is a top-down approach to database design that begins by identifying the important data called *entities* and *relationships* between the data that must be represented in the model.
- We then add more details such as the information we want to hold about the entities and relationships called *attributes* and any *constraints* on the entities, relationships, and attributes.

2.7 Entity and Attributes

In the *entity-relationship model* (or *E /R model)*, the structure of data is represented graphically, as an "entity-relationship diagram," using three principal element types:

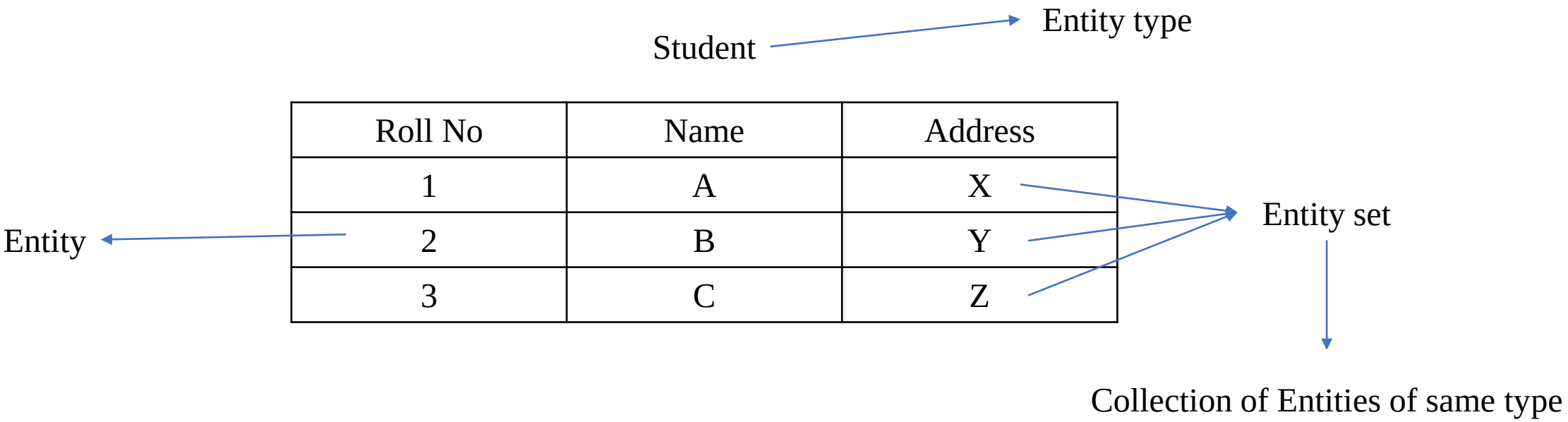
Entity: An entity is an abstract object of some sort and a collection of similar entities forms an entity set represented by rectangles.

Attributes: Entity sets have associated *attributes*, which are properties of the entities in that set represented by ovals. For instance, the entity set *Movies* might be given attributes such as *title* and *length.*

Relationships: *Relationships* are meaningful connections among two or more entity sets represented by diamonds. For instance, if *Movies* and *Stars* are two entity sets, we could have a relationship *Stars-in* that connects movies and stars.

- **Attribute domain:** The set of allowable values for one or more attributes.
- Edges connect entity set to its attributes and also connect a relationship to its entity sets.

Entity type vs. Entity set vs. Entity



Entity type and Entity are used interchangebly

Types of Entity and Attributes

- **Strong entity:** An entity type that is *not* dependent on some other entity type.
- **Weak entity:** An entity type that is dependent on some other entity type.
- **Simple attribute:** Composed of a single component with an independent existence.
- **Composite attribute:** Composed of multiple components, each with an independent existence. Composite attributes can be divided into subparts (that is, other attributes).
- **Single-valued attribute:** Holds a single value for each occurrence of an entity type.
- **Multi-valued attribute :** Holds multiple values for each occurrence of an entity type.
- **Derived attribute:** Represents a value that is derivable from the value of a related attribute or set of attributes, not necessarily in the same entity type.

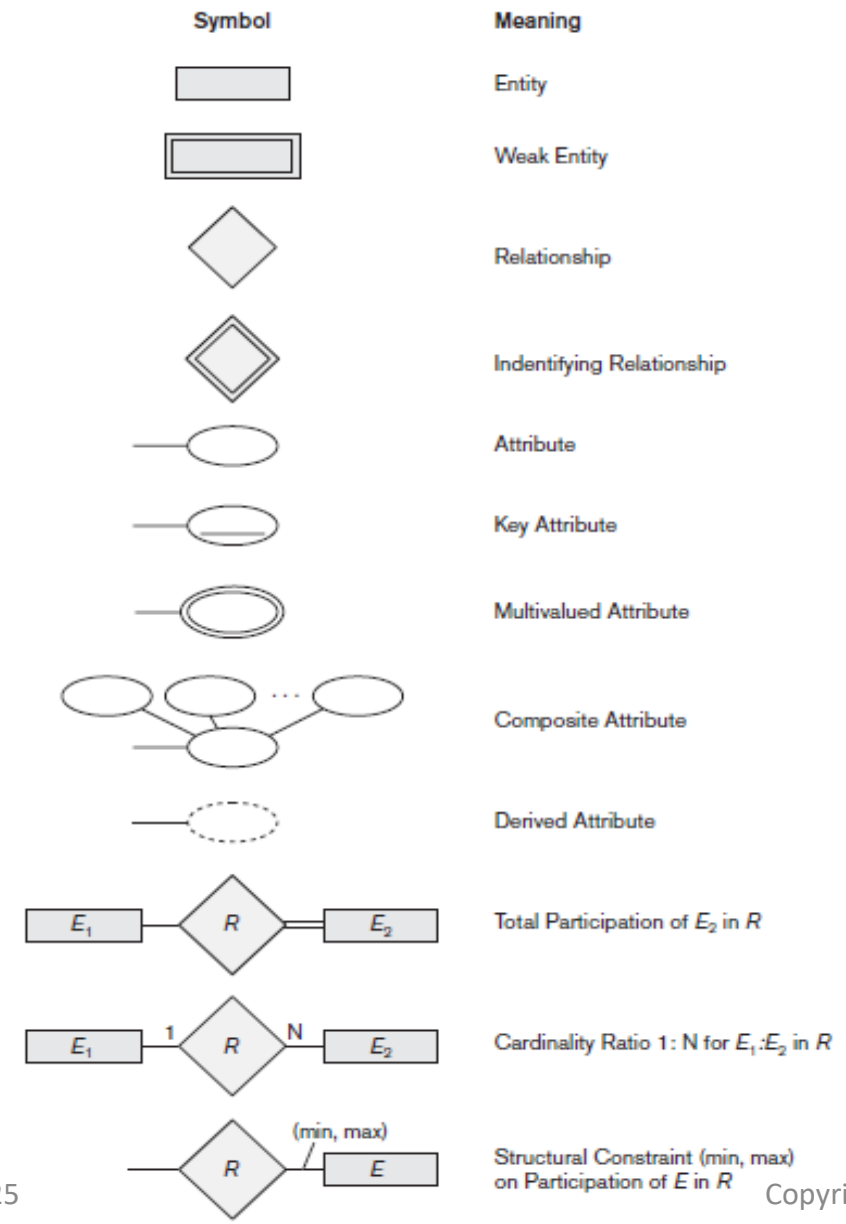
Note: Weak entity types are sometimes referred to as *child*, *dependent*, or *subordinate* entities and strong entity types as *parent*, *owner*, or *dominant* entities.

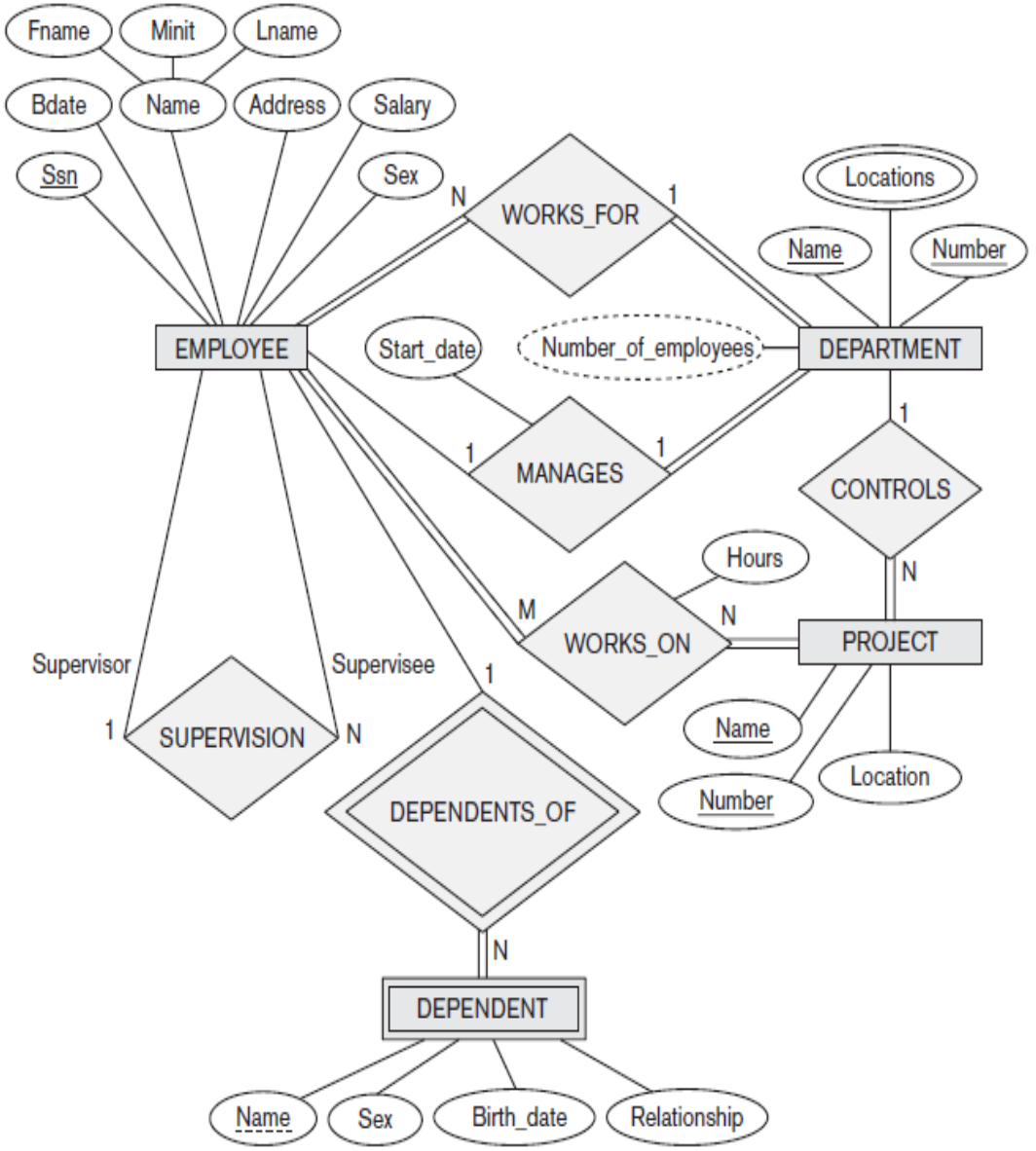
Types of Relationship

- **Degree of a relationship:** The number of participating entity types in a relationship.
- The entities involved in a particular relationship type are referred to as **participants** in that relationship. The number of participants in a relationship type is called the **degree** of that relationship.
    - Therefore, the degree of a relationship indicates the number of entity types involved in a relationship. A relationship of degree two is called **binary**, A relationship of degree three is called **ternary,** A relationship of degree four is called **quaternary** and so on.

- The term 'complex relationship' is used to describe relationships higher than binary.
- A relationship type where the *same* entity type participates more than once in *different roles known as* **Recursive relationship**. Recursive relationships are sometimes called *unary* relationships but degree is two.
- **Attributes on Relationships:** Relationship can also have attributes, similar to those of entity types. For example, to record the number of hours per week that an employee works on a particular project, we can include an attribute Hours for the WORKS_ON relationship.
- **Identifying relationship:** The relationship type that relates a weak entity type to its owner the of the strong entity type.

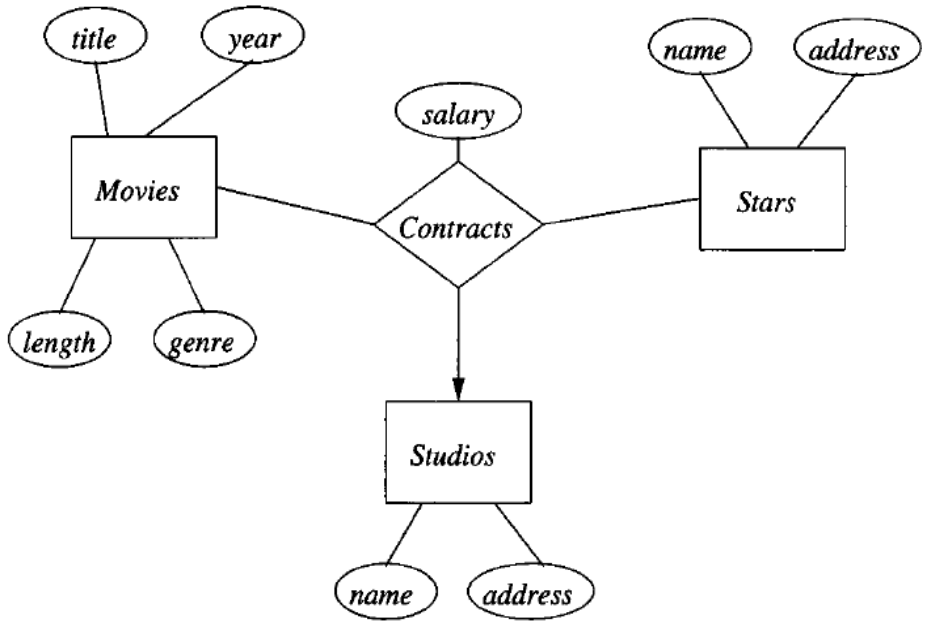
2.7 ER Diagram





Attributes Assigned to Relationships

When?



Sometimes it is convenient, or even essential, to associate attributes with a relationship, rather than with any one of the entity sets that the relationship connects.

Example, consider the relationship in fig., which represents contracts between a star and studio for a movie.

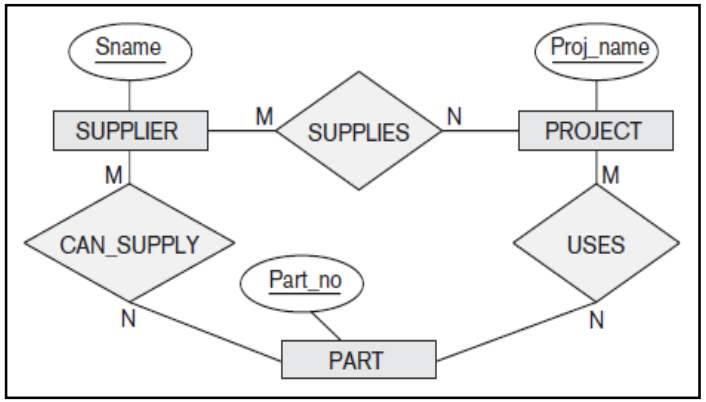
- We might wish to record the salary associated with relation contract.
- However, we cannot associate it with the star; a star might get different salaries for different movies.

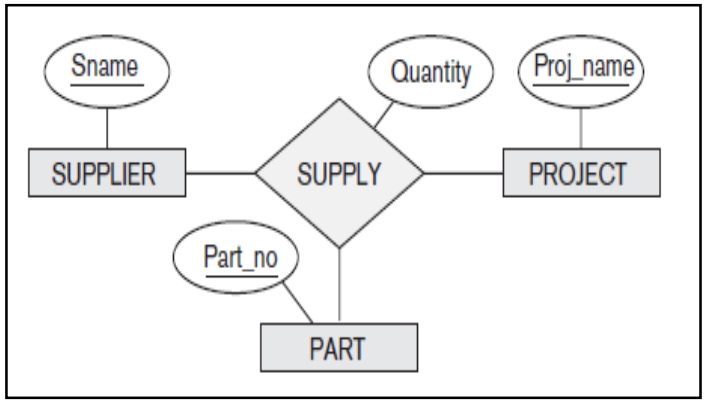
  Similarly, it does not make sense to associate the salary with a studio (they may pay different salaries to different stars) or with a movie (different stars in a movie may receive different salaries).

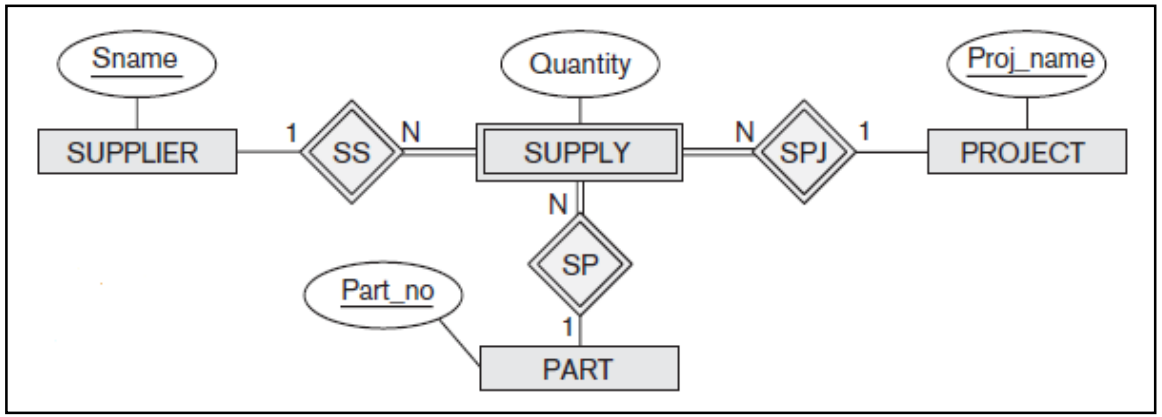
Note: It is never necessary to place attributes on relationships. We can instead invent a new entity which can have the attributes ascribed to the relationship. If we then include this entity set in the relationship, we can omit the attributes on the relationship itself.

Complex Relationship

- A ternary relationship type represents different information than the three binary relationship types.







Cardinality ratio

- Describes the maximum number of possible relationship occurrences for an entity participating in a given relationship type.
  - For example, in the WORKS_FOR binary relationship type, EPARTMENT:EMPLOYEE is of cardinality ratio 1:N, meaning that each department can be related to (that is, employs) any number of employees but an employee can be related to (work for) only one department.

- This constraint specifies the *maximum* number of relationship instances that each entity can participate in.
  - Meaning in relationship WORKS_FOR, a particular department entity can be related to any number of employees (N indicates there is no maximum number). On the other hand, an employee can be related to a maximum of one department.
- The possible cardinality ratios for a binary relationship are 1:1, 1:N, N:1, and M:N.

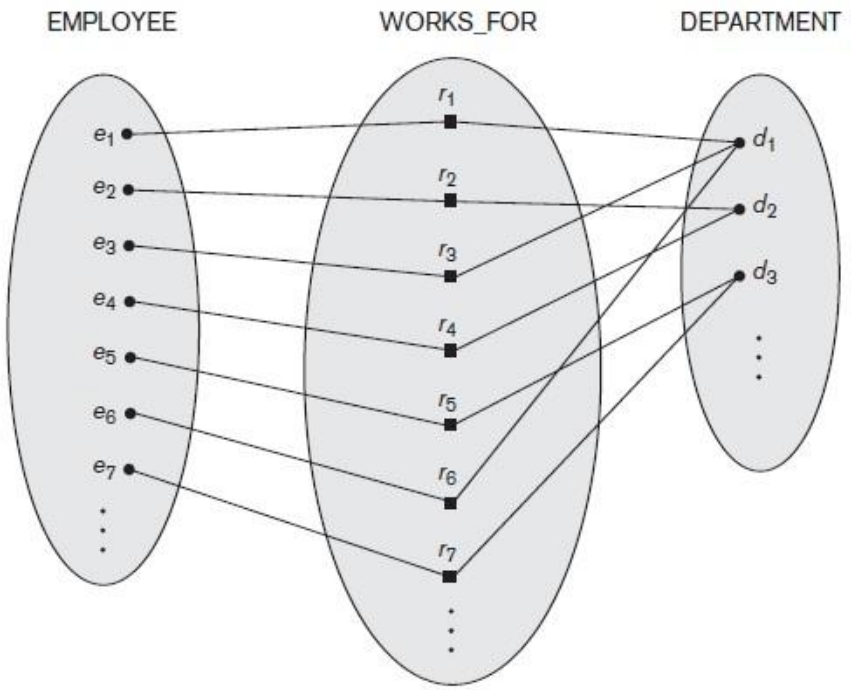
Participation constraints

- Determines whether all or only some entity occurrences participate in a relationship.
- This constraint specifies the *minimum* number of relationship instances that each entity can participate in and is sometimes called the **minimum cardinality constraint**.
- There are two types of participation constraints—total and partial.
  - *Every* employee must work for a department, then an employee entity can exist only if it participates in at least one WORKS_FOR relationship instance. Thus, the participation of EMPLOYEE in WORKS_FOR is called **total (mandatory or existence) participation**, meaning that every entity in *the total set* of employee entities must be related to a department entity via WORKS_FOR.
  - Every employee need not to manage a department, so the participation of EMPLOYEE in the MANAGES relationship type is **partial (optional)**, meaning that *some* or *part of the set of* employee entities are related to some department entity via MANAGES but not necessarily all.

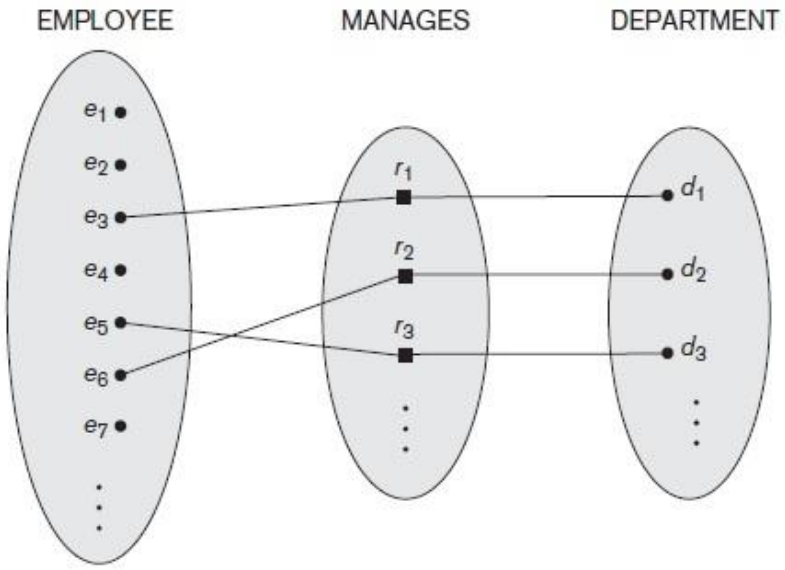
Note:

- A weak entity type always has a *total participation constraint* (existence dependency) with respect to its identifying relationship because a weak entity cannot be identified without an owner entity.
- However, not every existence dependency results in a weak entity type.

2.7 Structural Constraints

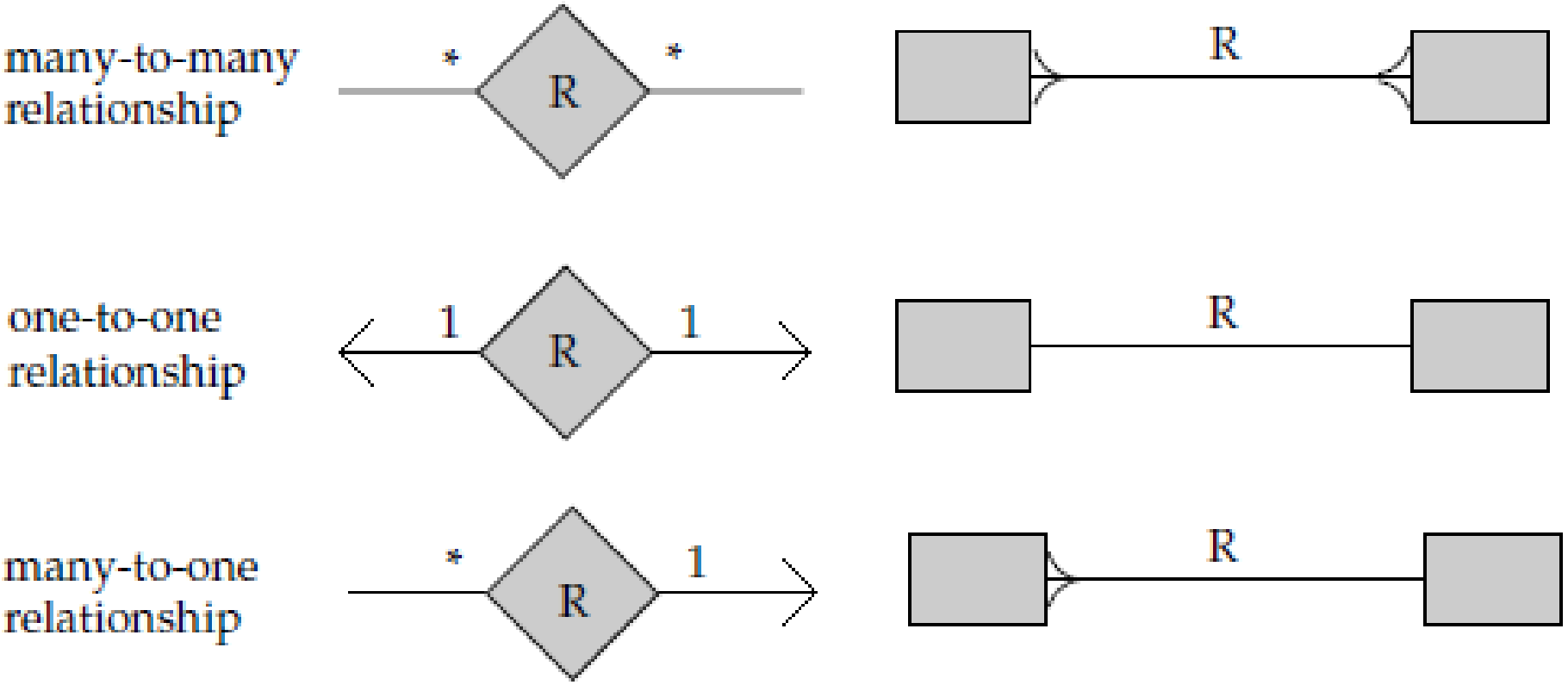


N :1



Partial ***1:1*** *Total*

2.7 Alternate Representation

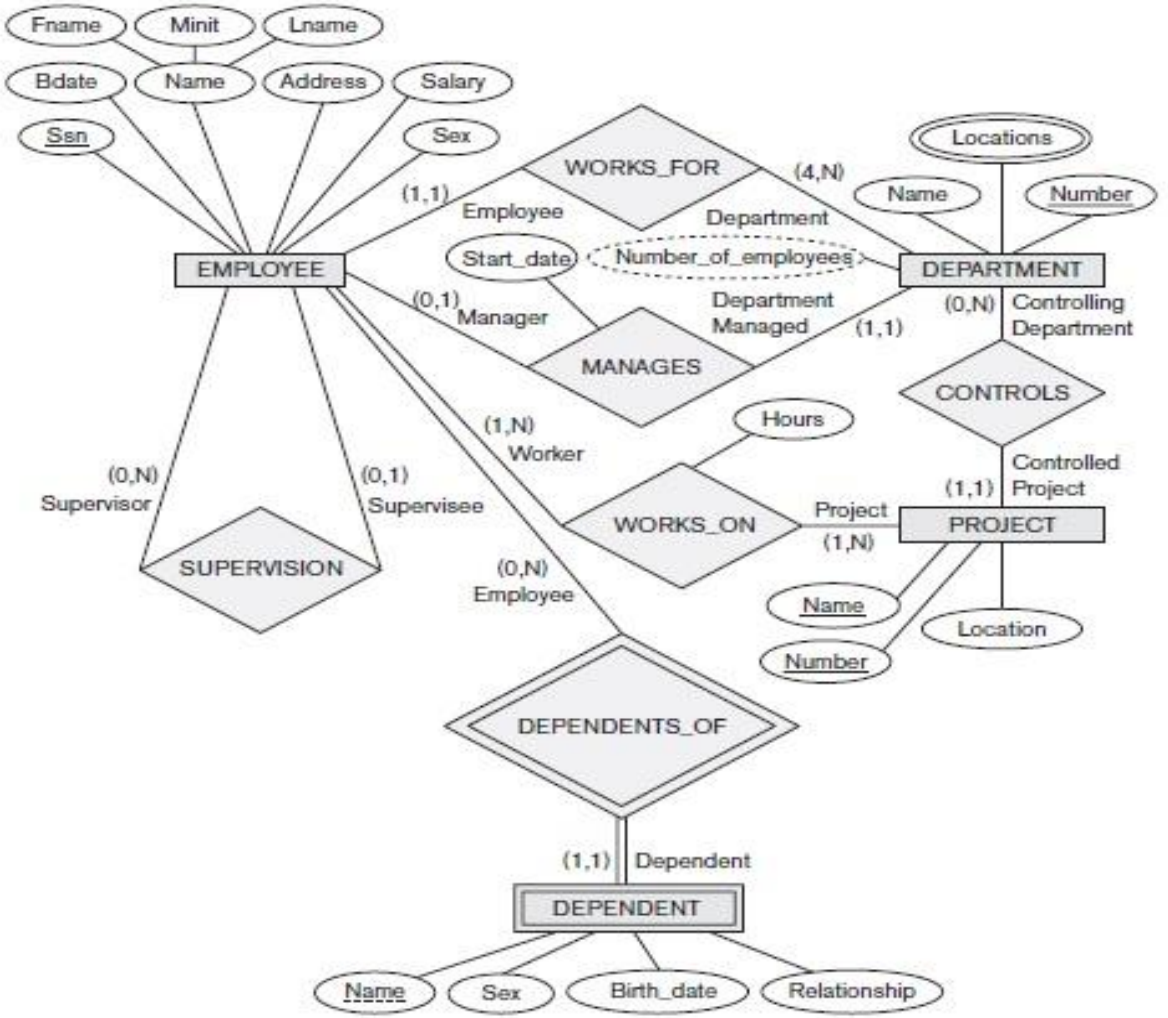


Min-Max notation

- Specify structural constraints on relationships, which replaces the cardinality ratio (1:1, 1:N, M:N) and participation constraints (sing-line/double-line notation).
- A pair of integer numbers (min, max) with each *participation* of an entity type *E* in a relationship type *R*, where $0 \leq \min \leq \max$ and $\max \geq 1$.

- Max: Cardinality ratio
- Min: Participation constraint
  - min = 0 implies partial participation.
  - min > 0 implies total participation.

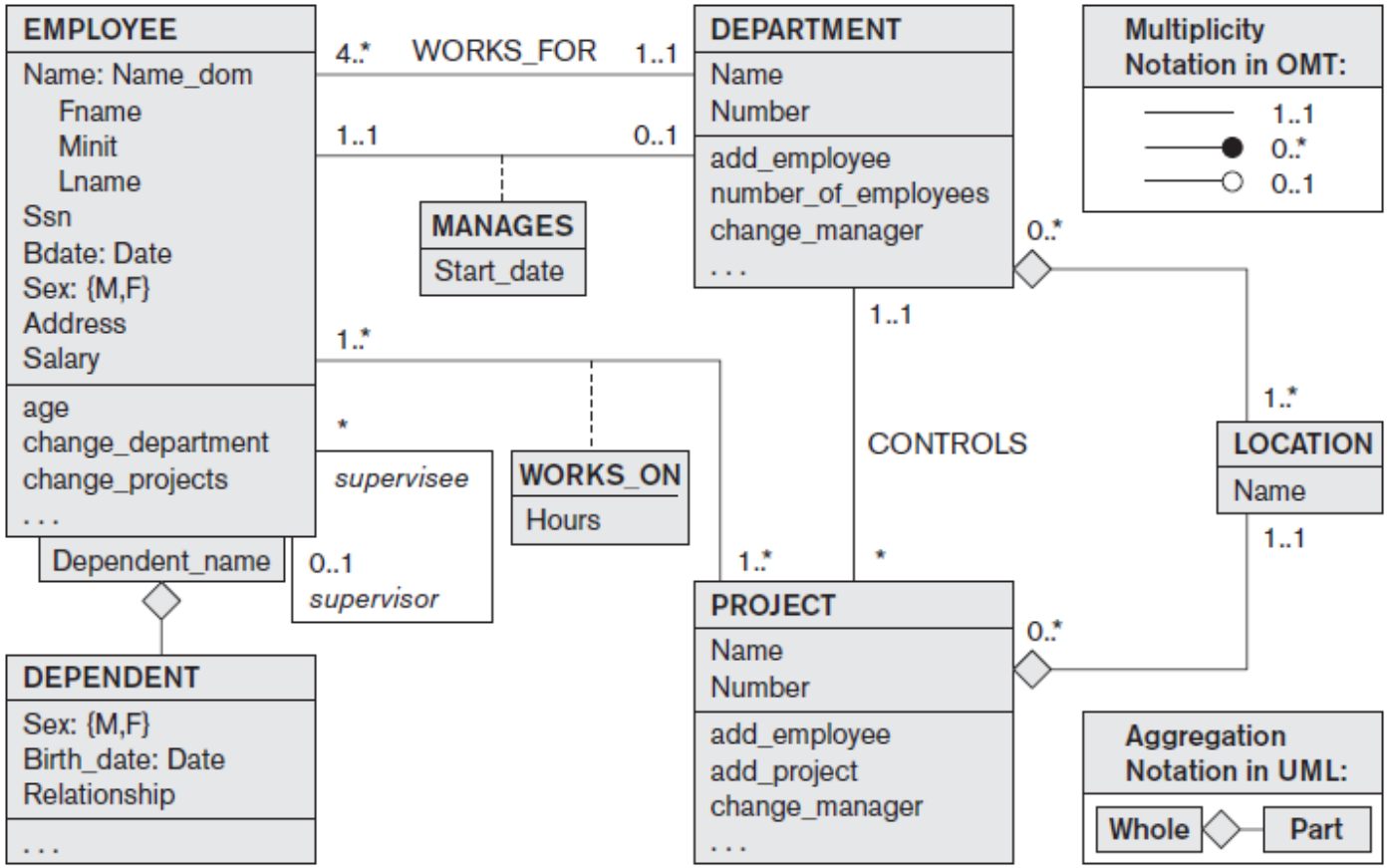
2.7 Alternate Representation



- ✓ Each employee works for minimum 1 department and maximum 1 department.
- ✓ Each department should have min 4 employee and maximum n employee

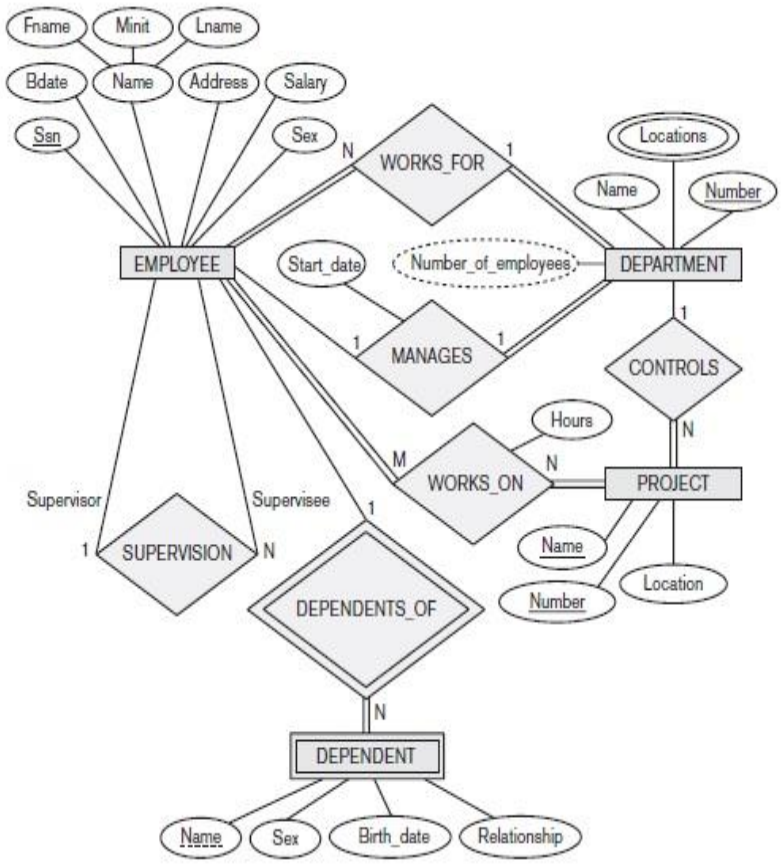
2.7 Alternate Representation

UML Notation



In some notations, like in object modeling methodologies such as UML, the (min, max) is placed on the *opposite sides* to the ones we have shown.

For example, for the WORKS_FOR relationship in the (1,1) would be on the DEPARTMENT side and the (4,N) would be on the EMPLOYEE side.



EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno

DEPARTMENT

Dname	Dnumber	Mgr_ssn	Mgr_start_date

DEPT_LOCATIONS

Dnumber	Dlocation

PROJECT

Pname	Pnumber	Plocation	Dnum

WORKS_ON

Essn	Pno	Hours

DEPENDENT

Essn	Dependent_name	Sex	Bdate	Relationship

For entity

- For each strong entity make a relation(table) including all simple attributes (component of composite attribute).
- For each weak entity, make a relation including all simple attributes.
  - Include the PK attribute of the owner entity as foreign key attributes. The primary key of the relation is the combination of the PK of the owner(s) and the partial key of the weak entity type if any.

For relation

For 1:1 relation

- **Merged relation approach:** Merge the two entity and the relationship to a single relation.
  - This is possible when *both participations are total*, as this would indicate that the two tables will have the exact same number of tuples at all times.
  - And if relational attribute presents then add that attribute in the merged table.

- **Foreign key approach:** Two tables for two entity are sufficient.
  - Add the PK of one entity as FK in the other entity.
    - It is better to add the FK in the total participation (if not present then to any side) side
    - Include all the simple attributes (simple components of composite attributes) into that side
    - And if relational attribute presents then add that attribute in the total participation side.

- **Cross-reference or relationship relation approach:** Make a new third relation *R* for the purpose of cross-referencing the PK of the two entity.
  - The relation will include the PK of each table as foreign keys.
  - The primary key of *R* will be one of the two FK, and the other foreign key will be a unique key of *R*.
  - And if relational attribute presents then add that attribute in the new table.

For 1:N relation

- **Foreign key approach:** Two tables for two entity are sufficient.
  - ➢ Add the PK of one entity as FK in the other entity.
    - Make the FK in the many side in 1:M relation.
    - If relational attribute presents add all that simple components in the many side.

- **Cross-reference or relationship relation approach:** Make a new third relation *R* for the purpose of cross-referencing the PK of the two entity.
  - ➢ The relation will include the PK of each table as foreign keys.
  - ➢ The primary key will be same as the many side.
  - ➢ And if relational attribute presents then add that attribute in the new table.

  This option can be used if few tuples in *S* participate in the relationship to avoid excessive NULL values in the foreign key.

For M:N relation

- **Cross-reference approach:** For M:N relation make a new table adding PK's of both entity.
  - ➢ And if relational attribute presents then add that attribute in the new table.

For N-ary relation

- For n-ary relation make a new relation including PK of all the participating entities in the relationship and the combination of all the PK will be the PK for the relation.

Multivalued and Complex attribute

- A new table is required for multivalued attribute and add the PK of the entity to which the multivalued attribute belongs as foreign key.
- For complex attribute also make a new table and break the composite part of the complex attribute.

Complex attribute is a combination of multivalued and composite attributes

- **Data conversion and loading** - This stage is concerned with importing and converting data from the old system into the new database.
- **Testing** - This stage is concerned with the identification of errors in the newly implemented system .It checks the database against requirement specifications.
- Cycle of coding, testing and debugging continues until database is ready for delivery.
- Then database will be create and system is customize.
  - Creation of tables and views
  - user authorizations

Maintenance–

Three types of maintenance activity:-

1) Corrective maintenance
2) Adaptive maintenance
3) Perfective maintenance

- The main objective of designing a database is to create an accurate representation of the data and relationships between the data that allows us to store information without unnecessary redundancy and to retrieve information easily.

To achieve this objective, we can use one or more database design techniques
- Entity–Relationship (ER) model
- Normalization.

- Normalization is a database design technique, which begins by examining the relationships (called functional dependencies) between attributes.

- It is the process of decomposing relations with anomalies to produce well-structured relations that contains minimal redundancy and allows insertion, modification, and deletion without errors or inconsistencies.

2.8 Redundancy and Data Anomaly

Redundant data: Data stored repeatedly in more than once, i.e. redundant data could be removed without the loss of information.

- Redundancy can lead to anomalies. The different anomalies are insertion, deletion, and update anomalies.

Staff no.	Job	Dept. no.	Dept. name	City
100	sales man	10	sales	Trichy
101	manager	20	accounts	Coimbatore
102	clerk	30	accounts	Chennai
103	clerk	30	operations	Chennai

- *Insertion Anomaly:* Cannot insert a department without inserting a staff that works in that department.
- *Update Anomaly:* We could not change the name of the department that "100" works in without simultaneously changing the department that "102" works.
- *Deletion Anomaly:* By removing, employee 100, we have removed all information pertaining to the sales department.

2.8 Decomposing Relation

- The accepted way to eliminate these anomalies is to decompose relations.
- Decomposition of a relation R involves splitting the attributes of R to make the schemas of two new relations.

The decomposition should always satisfy the properties -

- **Lossless decomposition:** No spurious rows should be generated when relations are reunited through a natural join operation.
- **Dependency preservation:** The decomposed relations have the same capacity to represent the integrity constraints as the original relations and thus to reveal illegal updates. In other words, each FD is represented in some individual relations resulting after decomposition

2.8 Decomposing Relation

R(X, Y, Z)

X	Y	Z
1	2	3
3	2	6
5	4	2

R1(X, Y)

X	Y
1	2
3	2
5	4

R2(Y, Z)

Y	Z
2	3
2	6
4	2

X	Y	Z
1	2	3
1	2	6
3	2	3
3	2	6
5	4	2

Extra tuples

As natural join is associative and commutative.
It does not matter in what order we join the projections; we shall get the same result.

Definition: Describes the relationship between attributes in a relation.

For example, if A and B are attributes of relation R, B is functionally dependent on A (denoted A → B), if each value of A is associated with exactly one value of B. However, for a given value of B there may be several different values of A.

Note: (A and B may each consist of one or more attributes.)

Notation of Functional Dependency

- The notation of functional dependency is A → B.
- The meaning of this notation is:
    1. "A" determines "B"
    2. "B" is functionally dependent on "A"
    3. "A" is called determinant and "B" is called dependent

2.8 Functional Dependencies (FD)

Types of Functional dependencies

- ***Trivial:*** A → B If B is a proper subset of A.
- ***Full:*** A → B If removal of any attribute from A results in the dependency no longer exist
- ***Partial:*** A → B If there is some attribute that removed from A and yet the dependency still holds.
- ***Transitive:*** A → CIf A → B and B → C exist and B is non prime attribute.

Functional Dependency Inference Rules (Armstrong's Axioms - how new FDs can be inferred from given ones**)**

1. **Reflexivity:** If B is a subset of A, then A → B
2. **Augmentation:** If A → B, then AC → BC where *C is any set of attributes in R*
3. **Transitivity:** If A → B and B → C, then A → C

Several further rules can be derived from the three given above:

1. **Self-determination:** A → A
2. **Pseudotransitivity:** If X→Y and YW→Z then XW→Z ***Transitivity is a special case of pseudotransitivity when W is null***
3. **Decomposition:** If A → BC then A → B and A → C
4. **Union:** If A → B and A → C, then A → BC
5. **Composition:** If A → B and C → D then AC → BD

❑ Computing the Closure of Attributes

- If LHS of a FD is a subset of the closure to be found only then update the closure by adding RHS of that FD to the closure.
- Do repeat the process for all FD.
- Do repeat the loop again and again until the final closure is found.

```
closure = X;
        repeat until there is no change: {
                if there is an FD A → B in F such that U ⊆ closure,
                        then set closure == closure AB
                                                }
```

❑ To find Keys

- If closure of a attribute or set of a attributes contains all the attributes in that relation then that attribute or set of attributes is called a key(Super key)
- Minimal of all those keys will be the candidate key

❑ Possible FD's and CK of sub relation

- Let, R(ABCD) and set of FD's {A→B, C→D}
- Find the CK of sub relation R1(AB) and R2(ACD)

For R1	
	$A+=B$
Possible FD's	A→B
CK is	A

For R2		
	$A+ = B$	$AC+= ABCD$
	$C+=CD$	$AD+=ABD$
		$CD+=CD$
Possible FD's	C→D	AC→D
CK is	AC	

❑ To find redundant FDs

- FD1 is redundant FD if closure of LHS of FD1 w.r.t. ***(All FD – FD1)*** contain all the attributes of RHS of FD1

❑ Minimal cover of FDs

- Decompose all the FDs with a single attribute on the right side (using the decomposition axiom)
- Minimize the Left Side of Each FD if possible
- Eliminate redundant FD

R(ABCDE) and FDs {A → D, BC → *AD, C* → B, E → A, E → D}

1. A → D, BC → *A*, BC → *D*, *C* → B, E → A, E → D
2. BC → *A* *Closure of B=B , Closure of C= BCAD contain B* so we can eliminate B from LHS so BC → *A* *becomes* C → *A* *Similarly* BC → *D becomes* C → *D*
3. From 2 we get,
   A → D, C → *A*, C → *D*, *C* → B, E → A, E → D where C → D is redundant so we can remove that.

Note: Minimal cover is not unique and no trivial FD can be in a minimal cover.

❑ Equivalence between sets of Functional Dependencies

- Two sets of FD's F and G are equivalent, if all FDs in F can be inferred from the FDs in G and vice versa.

Example: Consider two sets of FDs, F and G,

F = {A -> B, B -> C, AC -> D} and G = {A -> B, B -> C, A -> D}

- Take the attributes from the LHS of FDs in F and compute attribute closure for each using FDs in G:

  A+ using G = ABCD; A -> A; **A -> B**; A -> C; A -> D;

  B+ using G = BC; B -> B; **B -> C**;

  AC+ using G = ABCD; AC -> A; AC -> B; AC -> C; **AC -> D**;

- Notice that all FDs in F (highlighted) can be inferred using FDs in G.
- To see if all FDs in G are inferred by F, compute attribute closure for attributes on the LHS of FDs in G using FDs in F:

  A+ using F = ABCD; A -> A; **A -> B**; A -> C; **A-> D**;

  B+ using F = BC; B -> B; **B -> C**;

- Since all FDs in F can be obtained from G and vice versa, we conclude that F and G are equivalent.

❑ **To check Lossless join/Losy decomposition**

- Check closure of the common attributes of any two decomposed relations.
- If it contain the attributes of at least any one of the chosen decomposed relation then Lossless decomposition.

If there are more than one decomposed relation

- Combine the two relations from step 2.
- Check closure of the common attributes of combined relation got in step 3 and any of the remaining relation.
- If it contain the attributes of at least any one of the chosen relation then combin the attributes of two relations.
- Repeat this for further decomposed relation (if there).
- Finally if it satisfies for last two relations then Lossless decomposition.

❑ **Dependency preservation**

- Find minimal cover of the set of FDs.
- LHS and RHS of each FD in original relation must contain collectively in any one of the decomposed relation

Example, R(A->B, B->C, C->D and D->E) and decomposed relations R1(AB), R2(BCE), R3(CDE)

Here as LHS and RHS of the FD D->E in original relation is not collectively contained in any one of the decomposed relation hence not dependency preserved.

❑ Check Dependency preserved or not

- Example, R(ABCDE) and set of FD's AB→C, B→C
- Decomposed Relations R1 (AB) and R2(BC)

- It seems like AB→C is not preserved in any one of the decomposed relations.
- But After finding minimal cover it seems AB→C is redundant
- Hence, minimal cover is B→C
- So given decomposition is dependency preserved.

- In other way,

For R1

$A+ = A$ $B+ = B$

$AB+ = ABC$

Hence AB→C is preserved.

First Normal Form (1NF): Relation should not have any multivalued attributes.

Second Normal Form (2NF): A relation schema R is in 2NF if every nonprime attribute A in R is fully functionally dependent on the candidate key of R.

Prime attribute: part of any candidate key

Third Normal Form (3NF): A relation schema *R* is in 3NF if it satisfies 2NF and no nonprime attribute of *R* is transitively dependent on the candidate key.

Boyce Code Normal Form (BCNF): A relation schema R is in BCNF if whenever a nontrivial functional dependency X→A holds in R, then X is a super key of R.

Fourth Normal Form (4NF): A relation schema R that is in Boyce–Codd normal form and does not contain nontrivial multi-valued dependencies.

Generalisation of the definitions:

- **1NF:** Every attribute value should be atomic.
- **2NF:** Non prime attributes can not depend upon on proper subset of any Candidate key.
- **3NF:** For all FD A → B A should be candidate key or B should be prime attribute
- **BCNF:** For all FDs A → B A should be candidate key
- **4NF:** In BCNF and at least one of its keys consists of a single attribute.

1NF:

- Form new relations for each multivalued attribute.

Relation Department is in unnormalised form

Department number	Department name	Location
1	Nilgiris	{Coimbatore, Chennai}
2	Subiksha	{Chennai, Tirunelveli}
3	Krishna	Trichy
4	Kannan	Coimbatore

Decomposing relation R into R1 and R2 which are in 1NF

Department number	Department name
1	Nilgiris
2	Subiksha
3	Krishna
4	Kannan

Department number	Department name
1	Coimbatore
1	Chennai
2	Chennai
2	Tirunelveli
3	Trichy
4	Coimbatore

2NF: Say FDs $A \rightarrow B$ and $X \rightarrow Y$ violating 2NF among set of FDs of relation R(ABCDE)

Then find $A+ = \alpha$ $X+= \beta$

Then decompose relation R as R1 (α) R2(β) R3(R – (α-A) – (β-X))

Check all the possible FDs in decomposed relations R1 and R2 and check if both are in 2NF or not.

3NF: Decompose into 2NF and check all the possible FDs in decomposed relations.

Say FDs $A \rightarrow B$ and $X \rightarrow Y$ violating 3NF among set of FDs in any of the decomposed relation R_d

Then find $A+ = \alpha$ $X+= \beta$

Then decompose relation R as R1 (α) R2(β) R3(R – (α-A) – (β-X))

Check all the possible FDs in decomposed relations R1 and R2 and check if both are in 3NF or not.

BCNF: Decompose into 2NF and then to 3NF and check all the possible FDs in decomposed relations.

Say FDs $A \rightarrow B$ and $X \rightarrow Y$ violating BCNF among set of FDs in any of the decomposed relation R_d

Then find $A+ = \alpha$ $X+= \beta$

Then decompose relation R as R1 (α) R2(β) R3(R – (α-A) – (β-X))

Check all the possible FDs in decomposed relations R1 and R2 and check if both are in BCNF or not.

Example: R(ABCD) and FD's {AB→C, C→B}

CK→ {AB, AC}

C→B violate 2NF

Taking C→B C+={CB}

we get, R1(CB) R2(AC) Possible FD's for R1 is C→B and for R2 is nothing

Hence, R1, and R2 are in 2NF and also in 3NF and BCNF.

Decomposition is losseless and not dependency preserved. (AB→C is not preserved)

Example: R(ABCD) and FD's {AB→C, B→C}

CK→ {AB}

B→C violate 2NF

Taking B→C B+={BC}

we get, R1(BC) R2(AB) Possible FD's for R1 is C→B and for R2 is nothing

Hence, R1, and R2 are in 2NF and also in 3NF and BCNF.

Decomposition is losseless and also dependency preserved (AB→C is actually preserved).

4NF:

- Although BCNF removes any anomalies due to functional dependencies, but not Multi-Valued Dependency (MVD), which can also cause data redundancy (Fagin, 1977).

- Multi-Valued Dependency (MVD): Represents a dependency between attributes (for example, A, B, and C) in a relation, such that for each value of A there is a set of values for B and a set of values for C. However, the set of values for B and C are independent of each other.

  We represent a MVD between attributes A, B, and C in a relation using the following notation:

  A –>> B

  A –>> C

  Example

branchNo	sName	oName
B003	Ann Beech	Carol Farrel
B003	David Ford	Carol Farrel
B003	Ann Beech	Tina Murphy
B003	David Ford	Tina Murphy

- For branch B003 there are staff Ann Beech and David Ford work at branch B003 and property owners Carol Farrel and Tina Murphy registered at branch B003

  i.e. MVDs are branchNo –>> sName branchNo –>> oName

- **Types of MVDs:**

  Trivial: A MVD A –>> B in relation R is defined as being trivial if

  (a) B is a subset of A or

  (b) A ∪ B = R.

  Nontrivial: if neither (a) nor (b) is satisfied.

Note

- Every FD is MVD but every MVD is not FD
- If A->> BC then A->B and A->C
- If A->>B and A->>C then A-> BC

- **Decomposition into 4 NF:** Say FDs A – >> B and X –>> Y are MVDs presents in Relation R, Then decompose relation R as

  R1 (AB)

  R2(XY)

  R3(R-B-Y) which are in 4NF

Comparison among Normal Form

Factor	3NF	BCNF	4NF
Lossless dependency	Yes	Yes	Yes
Preserves FD's	Yes	No	No
Eliminates Redundancy due to FD's	No	Yes	Yes
Preserves MVD's	No	No	No
Eliminates Redundancy due to MVD's	No	No	Yes

3NF is considered as adequate among all.

Disadvantages of Normalization

Normalization produces many tables. A query might require retrieval of data from multiple normalized tables. This can result in complicated table joins.

All the joins required to merge data will slow down the process and so Performance.

We should be very careful in decomposing – up to what extent?

2.9 Database Management Systems

- A DBMS is a software program that enables the creation and management of databases. Generally, these databases will be more complex than the text file/ spreadsheet.
- Some database management systems include a graphical user interface that allows you to create and maintain databases by "pointing and clicking".
- Other database systems use a command line interface that requires that you use certain programming commands to create and maintain your databases.

Examples-

- Microsoft Access
- Microsoft SQL Server
- MySQL
- Oracle

2.10 Creating a Database

- With database management systems, many tasks can be done either via programmatically or a user interface.
- **Programmatically:** Many database administrators(DBAs) use Structured Query Language(SQL) to perform many database tasks.
- To enter SQL ,open an interface that allows you to enter the code.
- **User Interface:** Most database systems make it very easy to create a database via a user interface. Generally, it selects an option from a menu, then provides a name for the database.

Steps-

1) Go to the file menu

2) Name the Database

3) The Database

Syntax: *create database datebasename*;

2.11 Create tables in a Database

- In relational database terms, a **table** is responsible for storing data in the database. Database tables consist of rows and columns.
- With Database Management system, need to create tables before enter the data.
- Programmatically:

```
CREATE TABLE Student (
    Id int,
    Name Varchar(50),
    DOB date,
    Class Varchar,
    Gender  Varchar
)
```

- Specify the name of the table, the name of each column and the data type of each column.

- User Interface:

  Design View – Depends on which Database system you use.

  Modifying a Table in Datasheet View – database system that allow you to make changes to the table definition while in dataset view.

  Datasheet View - If you run a query against your table to see what data is in it, you will see the grid like structure that represents the table.

- A database language consists of two parts:
  - Data Definition Language (DDL) and
  - Data Manipulation Language (DML)

The Data Definition Language (DDL)

- Allows to define entities, attributes, and relationships required for the application, together with any associated integrity and security constraints. In other words, to define a schema or to modify an existing one. It cannot be used to manipulate data.
- The result of the compilation of the DDL statements is a set of tables stored in special files collectively called the **system catalog** (data directory or dictionary).
- The system catalog integrates the **metadata**, that is data that describes objects in the database and makes it easier for those objects to be accessed or manipulated.
- The metadata contains definitions of records, data items, and other objects that are of interest to users or are required by the DBMS.

The Data Manipulation Language (DML)

- Also called *data sublanguages* because they do not include constructs for all computing needs such as conditional or iterative statements, which are provided by the high-level languages.
- A language that provides a set of operations to support the basic data manipulation
- operations on the data held in the database.
  - Data manipulation operations usually include the following:
    - ✓ insertion of new data into the database;
    - ✓ modification of data stored in the database;
    - ✓ retrieval of data contained in the database;
    - ✓ deletion of data from the database.
- There are two types of DML: *procedural and non-procedural*.

Procedural DMLs

- Allows the user to tell the system what data is needed and exactly how to retrieve that.

Non-procedural DMLs

- Allows the user to state only what data is needed rather than how it is to be retrieved.

- Donald Chamberlin in 1974 from the IBM San José Laboratory defined SEQUEL.
- In 1976, a revised version SEQUEL/2 was defined and name changed to SQL.
- Still many pronounce SQL as 'See-Quel', though official pronunciation is 'S-Q-L'.
- In 1982 ANSI began work on a Relational Database Language based on a IBM concept paper.
- ISO joined in this work in 1983, and together they defined a standard for SQL.
- In 1986, a standard for SQL was defined by ANSI which later adopted by ISO in 1987.

Criticism

- Date claimed that important features such as referential integrity constraints and certain relational operators had been omitted.
- He also pointed out that the language was extremely redundant; in other words, there was more than one way to write the same query.
- Further, SQL does not adhere strictly to the definition of the relational model. For example, SQL allows the table produced as the result of the SELECT statement to contain duplicate rows, it imposes an ordering on the columns and it allows the user to order the rows of a result table.

- It is ***non-procedural*** language: you specify *what* information you require, rather than *how* to get it.
- SQL statement are ***case insensitive.*** [One important exception to this rule is that literal character data must be typed *exactly* as it appears in the database].
- SQL is ***free-format*** i.e. parts of statements not needed to typed at particular locations on the screen.
- Although SQL is free-format, in a set of SQL statements for more readable indentation are used.

ISO SQL standard has two major components of statements

- Data Definition Language (DDL) for defining the database schema structure and controlling access to the data;
- Data Manipulation Language (DML) for retrieving and updating, deletion, insertion data

Terminology

- The ISO SQL standard does not use the formal terms of relations, attributes, and tuples,
- Instead using the terms tables, columns, and rows.

2.12.1 Data Types

Data type	Declarations			
boolean	BOOLEAN			
character	CHAR	VARCHAR		
bit†	BIT	BIT VARYING		
exact numeric	NUMERIC	DECIMAL	INTEGER	SMALLINT
approximate numeric	FLOAT	REAL	DOUBLE PRECISION	
datetime	DATE	TIME	TIMESTAMP	
interval	INTERVAL			
large objects	CHARACTER LARGE OBJECT		BINARY LARGE OBJECT	

† BIT and BIT VARYING have been removed from the SQL:2003 standard.

Date and time values can be specified like this:
date ’2001-04-25’
time ’09:30:00’
timestamp ’2001-04-25 10:29:01.45’

- Integrity constraints imposed in order to protect the database from becoming inconsistent.
- We consider five types of integrity constraint-

1. **Data constraints:** columns must contain a valid value; they are not allowed to contain nulls.
2. **Domain Constraints:** Every column has a domain, a set of legal values.
3. **Entity integrity:** The primary key of a table must contain a unique, non-null value for each row.
4. **Referential integrity:** If the foreign key contains a value, that value must refer to an existing, valid row in the parent table.

   A foreign key is a column, or set of columns, that links each row in the child table containing the foreign key to the row of the parent table containing the matching candidate key value.

5. **General Constraints:** Updates to tables may be constrained by enterprise rules governing the real-world transactions that are represented by the updates.

   For example, We may have a rule that prevents a member of staff from managing more than 100 properties at the same time.

- When the user attempts to delete a row from a parent table and there are one or more matching rows in the child table, SQL supports four options for action to be taken:

1. **SET CASCADE Delete:** The row from the parent table and automatically delete the matching rows in the child table.
2. **SET NULL Delete:** The row from the parent table and set the foreign key value(s) in the child table to NULL. This is valid only if the foreign key columns do not have the NOT NULL qualifier specified.
3. **SET DEFAULT Delete:** The row from the parent table and set each component of the foreign key in the child table to the specified default value. This is valid only if the foreign key columns have a DEFAULT value specified.
4. **NO ACTION:** Reject the delete operation from the parent table. This is the default setting if the ON DELETE rule is omitted.

CREATE SCHEMA		DROP SCHEMA
CREATE TABLE	ALTER TABLE	DROP TABLE
CREATE VIEW		DROP VIEW

CREATE SCHEMA and DROP SCHEMA STATEMENT

CREATE SCHEMA [Name] **AUTHORIZATION [**CreatorIdentifier]
DROP SCHEMA Name [**RESTRICT** | **CASCADE**]

```
CREATE TABLE EMPLOYEE
        ( Name          VARCHAR(15)     NOT NULL,
        Ssn             CHAR(9)         NOT NULL,
        Bdate           DATE,
        Address         VARCHAR(30),
        Salary          DECIMAL(10,2),  DEFAULT 60000,
        Super_ssn  CHAR(9),
        Dno             INT             NOT NULL,
        BANK_BAL        INT             NOT NULL        CHECK (balance > 0),
        PRIMARY KEY (Ssn),
        FOREIGN KEY (Super_ssn) REFERENCES EMPLOYEE(Ssn)
                ON DELETE SET NULL  ON UPDATE CASCADE,
        FOREIGN KEY (Dno) REFERENCES DEPARTMENT(Dnumber)
                ON DELETE NO ACTION  ON UPDATE CASCADE );
CREATE TABLE DEPARTMENT
        ( Dname         VARCHAR(15)     NOT NULL        UNIQUE,
        Dnumber   INT                   NOT NULL,
        Mgr_ssn         CHAR(9)         NOT NULL,
        Mgr_start_date  DATE,
        PRIMARY KEY (Dnumber),
        FOREIGN KEY (Mgr_ssn) REFERENCES EMPLOYEE(Ssn)
                ON DELETE CASCADE  ON UPDATE CASCADE );
```

ALTER TABLE TableName
[**ADD** [**COLUMN**] columnName dataType [**NOT NULL**] [**UNIQUE**]
[**DROP** [**COLUMN**] columnName [**RESTRICT** | **CASCADE**]]

DROP TABLE TableName [**RESTRICT** | **CASCADE**]

RESTRICT: The DROP operation is rejected if the column is referenced by another database object (for example, by a view definition). This is the default setting.
CASCADE: The DROP operation proceeds and automatically drops the column from any database objects it is referenced by. This operation cascades, so that if a column is dropped from a referencing object, SQL checks whether *that* column is referenced by any other object and drops it from there if it is, and so on.

CREATE VIEW view name
AS SELECT Attributes
FROM Relation
WHERE condition;

DROP VIEW ViewName [**RESTRICT** |**CASCADE**]

2.12.1 SQL Data Definition Commands

COMMAND OR OPTION	DESCRIPTION
Create schema authorization	Creates a database schema
Create table	Creates a new table in the user's database schema
Not null	Ensures that a column will not have null values
Unique	Ensures that a column will not have duplicate values
Primary key	Defines a primary key for a table
Foreign key	Defines a foreign key for a table
Default	Defines a default value for a column (when no value is given)
Check	Validates data in an attribute
Create index	Creates an index for a table
Create view	Creates a dynamic subset of rows/columns from one or more tables
Alter table	Modifies a tables definition (adds, modifies, or deletes attributes or constraints)
Create table as	Creates a new table based on a query in the user's database schema
Drop table	Permanently deletes a table (and its data)
Drop index	Permanently deletes an index
Drop view	Permanently deletes a view

The SQL DML statements are namely

- SELECT – to query data in the database.
- INSERT – to insert data into a table.
- UPDATE – to update data in a table.
- DELETE – to delete data from a table.

SELECT Statement

- The purpose of the SELECT statement is to retrieve and display data from one or more
- Database tables. It is an extremely powerful command capable of performing the equivalent of the relational algebra's *Selection*, *Projection*, and *Join* operations in a single statement.
- SELECT is the most frequently used SQL command and has the following general form:

```
SELECT [Attributes]                  // specifies columns  to appear in the output
FROM   [Relations]                   // specifies the table or tables to be used
[WHERE condition]                    // filters the rows subject to some condition
[GROUP BY column List ]              // groups rows with same column value
        [HAVING condition]           // filters the groups subject to some condition
[ORDER BY column List]               // specifies the order of the output
```

- The order of the clauses in the SELECT statement *cannot* be changed.
- The only two mandatory clauses are the first two: SELECT and FROM; the remainder are optional.

Select, From and Where clauses

- Select clause specifies the attributes to be selected.
- From clause specifies the relation or relations from where attributes to be selected.
- Where clause specifies the rows to be selected based on the condition mentioned in the clause.

The five basic search conditions (ISO terminology) are as follows:

1. ***Comparison***: Compare the value
   - SQL uses six common comparison operators: =, <>, <, >, <=, and >=
2. ***Range:*** Test whether the value of an expression falls within a specified range of values.
   - Search condition (BETWEEN/NOT BETWEEN)
3. ***Set membership:*** Test whether the value of an expression equals one of a set of values.
   - Search condition (IN/NOT IN)
4. ***Pattern match:*** Test whether a string matches a specified pattern.
   - Search condition (LIKE/NOT LIKE)
   - SQL has two special pattern-matching symbols: % represents any sequence of zero or more characters (*wildcard*) and _ represents any single character.
5. ***Null:*** Test whether a column has a null (unknown) value.
   - Search condition (IS NULL/IS NOT NULL)

Aggregate Functions

- Besides retrieving rows and columns from the database, we also want some form of summation or **aggregation** of data.

COUNT – returns the number of values in a specified column SUM – returns the sum of the values in a specified column AVG – returns the average of the values in a specified column MIN – returns the smallest value in a specified column MAX – returns the largest value in a specified column

- ✓ COUNT, MIN, and MAX apply to both numeric and non-numeric fields
- ✓ But SUM and AVG may be used on numeric fields only.
- ✓ **Apart from COUNT(*), each function eliminates nulls first and operates only on the remaining non-null values.**
  - COUNT(*) is a special use of COUNT, which counts all the rows of a table, regardless of whether nulls or duplicate values occur.
- ✓ Keyword DISTINCT is used before the column name to eliminate duplicates, although ALL is assumed if nothing is specified.
  - DISTINCT has no effect with the MIN and MAX functions but may have an effect on SUM or AVG.

Group by , Having and order by clause

- **GROUP BY clause**
  - Groups the data from SELECT table(s) and produces a single summary row for each group.
  - Not all attributes used in GROUP BY clause need to appear in the SELECT clause.
- **HAVING clause**
  - Designed for use with the GROUP BY clause to restrict the groups that appear in the final result table. However, SQL query can contain HAVING clause even if it does not have a GROUP BY clause.
  - The ISO standard requires that column names used in the HAVING clause must also appear in the GROUP BY list or be contained within an aggregate function.
  - Although similar in syntax, HAVING and WHERE serve different purposes The WHERE clause filters individual rows going into the final result table, whereas HAVING filters **groups** going into the final result table.
  - The condition in the HAVING clause always includes at least one aggregate function, otherwise the search condition could be moved to the WHERE clause and applied to individual rows. (Remember that aggregate functions cannot be used in the WHERE clause)
  - The HAVING clause is not a necessary part of SQL – any query expressed using a HAVING clause can always be rewritten without the HAVING clause.

- **Order by clause (Sorting results)**
  - In general, the rows of SQL query result table are not arranged in any particular order.
  - However, we can ensure the results of a query are sorted using the ORDER BY clause in the SELECT statement.

ORDER BY <list of attributes>

  - The ORDER BY clause allows the retrieved rows to be ordered in ascending (ASC) or descending (DESC) or in alphabetical order.

INSERT Statement

INSERT INTO TableName [(columnList)]
VALUES (dataValueList)

Insert a new row into the* Staff *table supplying data for all columns

INSERT INTO Staff
VALUES ('SG16', 'Alan', 'Brown', 'Assistant', 'M', **DATE** '1957-05-25', 8300, 'B003');

As we are inserting data into each column in the order the table was created.

Insert a new row into the* Staff *table supplying data for all mandatory columns

staffNo, fName, lName, position, salary, *and* branchNo.

INSERT INTO Staff (staffNo, fName, lName, position, salary, branchNo)
VALUES ('SG44', 'Anne', 'Jones', 'Assistant', NULL, 'B003');

As we are inserting data only into certain columns, we must specify the names of the columns that we are inserting data into.

UPDATE Statement

```
UPDATE TableName
SET columnName1 = dataValue1 [, columnName2 = dataValue2 . . . ]
[WHERE searchCondition]
```

Give all staff a 3% pay increase

```
UPDATE Staff
SET salary = salary*1.03;
```

No WHERE clause has been specified, so the update operation applies to all rows in the table.

Promote David Ford (staffNo = 'SG14') to Manager and increase his salary by 5%

```
UPDATE Staff
SET position = 'Manager', salary = salary*1.05
WHERE staffNo = 'SG14';
```

WHERE clause finds the rows for staffNo SG14 and update salary accordingly

DELETE Statement

DELETE FROM TableName
[**WHERE** searchCondition]

Delete all rows from the* Viewing *table

DELETE FROM Viewing;

No WHERE clause has been specified, so the delete operation applies to all rows in the table.

This removes all rows from the table leaving only the table definition, so we can to insert data into the table at a later stage.

Delete all viewings that relate to property PG4

DELETE FROM Viewing **WHERE** propertyNo = 'PG4';

The WHERE clause finds the rows for property PG4 and the delete these particular rows.

2.12.1 SQL Data Manipulation Command

COMMAND OR OPTION	DESCRIPTION
Insert	Inserts row(s) into a table
Select	Selects attributes from rows in one or more tables or views
Where	Restricts the selection of rows based on a conditional expression
Group by	Groups the selected rows based on one or more attributes
Having	Restricts the selection of grouped rows based on a condition
Order by	Orders the selected rows based on one or more attributes
Update	Modifies an attribute’s values in one or more table’s rows
Delete	Deletes one or more rows from a table
Commit	Permanently saves data changes
Rollback	Restores data to their original values

2.12.1 SQL Data Manipulation Command

COMMAND OR OPTION	DESCRIPTION
COMPARISON OPERATORS	
=, <, >, <=, >=, <>	Used in conditional expressions
LOGICAL OPERATORS	
AND/OR/NOT	Used in conditional expressions
SPECIAL OPERATORS	
BETWEEN	Checks whether an attribute value is within a range
IS NULL	Checks whether an attribute value is null
LIKE	Checks whether an attribute value matches a given string pattern
AGGREGATE FUNCTIONS	
COUNT	Returns the number of rows with non-null values for a given column
MIN	Returns the minimum attribute value found in a given column
MAX	Returns the maximum attribute value found in a given column
SUM	Returns the sum of all values for a given column
AVG	Returns the average of all values for a given column

SQL Includes commands to commit, rollback , etc. with data tables.

- **COMMIT :** It permanently saves data changes

  Example:

```
Delete from table_name Where condition;
COMMIT;
```

- **ROLLBACK :** It restores data to their original values.

  Example:

```
Delete from table_name Where condition;
ROLLBACK;
```

Disambiguating Attributes

- A query involving several relations and among these relations two or more attributes may have same name.
- If so, we need a way to indicate which of these attributes is meant by a use of their shared name.
- SQL solves this problem by allowing us to place a relation name and a dot in front of an attribute.
- Thus *R.A* refers to the attribute *A* of relation *R*.

Example: The two relations

MovieStar(name, address, gender, birthdate)

MovieExec(name, address, cert#, netWorth)

Each have attributes name and address. Suppose we wish to find pairs consisting of a star and an executive with the same address.

```
SELECT MovieStar.name, MovieExec.name
FROM MovieStar, MovieExec
WHERE MovieStar.address = MovieExec.address;
```

2.12.2 Tuple Variables

- Sometimes we need a query that involves two or more tuples from the same relation.
- A relation *R can be listed* as many times in FROM clause, but need a way to refer to each occurrence of *R*.
- Define for each occurrence of *R* in the FROM clause, an "alias" refereed as a *tuple variable.*
- Each use of *R* in the FROM clause is followed by the (optional) AS and name of the tuple variable;
- In the SELECT and WHERE clauses, we can disambiguate attributes of *R* by preceding them by the appropriate tuple variable and a dot.
- Thus, the tuple variable serves as another name for relation *R.*

```
SELECT Starl.name, Star2.name
        FROM MovieStar AS Starl, MovieStar Star2
        WHERE Starl.address = Star2.address AND Starl.name < Star2.name;
```

- In FROM clause 2 tuple variables declared, Starl and Star2; each is an alias for relation MovieStar.
- The tuple variables are used in SELECT clause to refer to the name components of the two tuples.
- In the WHERE clause, Star l .name < Star2 . name, says that the name of the first star precedes the name of the second star alphabetically.

2.12.2 Subselect and Subqueries

Subselect: SELECT statement embedded within another SELECT statement.

- The results of this **inner** SELECT statement (or **subselect**) are used in the **outer** statement to help determine the contents of the final result.
- **subquery** or **nested query:** A subselect can be used in the WHERE and HAVING clauses of an outer SELECT statement.
- Subselects may also appear in INSERT, UPDATE, and DELETE statements.

List the properties that are handled by staff who work in the branch at '163 Main St'

```
SELECT propertyNo, street, city, postcode, type, rooms, rent
FROM PropertyForRent
WHERE staffNo IN (SELECT staffNo
                  FROM Staff
                  WHERE branchNo = (SELECT branchNo
                                    FROM Branch
                                    WHERE street = '163 Main St'));
```

EXISTS and NOT EXISTS

- The keywords EXISTS and NOT EXISTS are designed for use only with subqueries

Find all staff who work in a London branch office.

```
SELECT staffNo, fName, lName, position
FROM Staff s
WHERE EXISTS (SELECT *
              FROM Branch b
              WHERE s.branchNo = b.branchNo AND city = 'London');
```

ALL and SOME (ANY)

Find all staff whose salary is larger than the salary of at least one member of staff at branch B003.

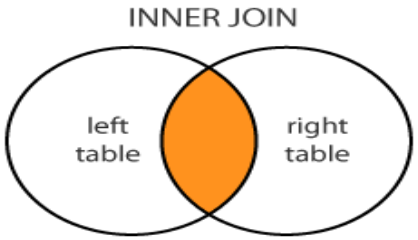
```
SELECT staffNo, fName, lName, position, salary
FROM Staff
WHERE salary > SOME (SELECT salary
FROM Staff
WHERE branchNo = 'B003');
```

2.12.2 Multi Table Queries

CROSS JOIN

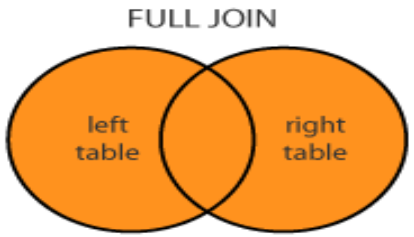
```
SELECT Fname, Lname, Address
FROM (EMPLOYEE CROSSJOIN DEPARTMENT);
```

JOIN




```
SELECT Fname, Lname, Address
FROM (EMPLOYEE JOIN DEPARTMENT)
WHERE Dno < Dnumber;
```

EQUIJOIN




```
SELECT Fname, Lname, Address
FROM (EMPLOYEE JOIN DEPARTMENT)
WHERE Dno = Dnumber;
```

NATURAL JOIN // implicit equijoin on common attributes of Employee Department

```
SELECT Fname, Lname, Address
FROM (EMPLOYEE NATURALJOIN DEPARTMENT)
```

OUTERJOIN

```
SELECT Fname, Lname, Address
FROM (EMPLOYEE OUTERJOIN DEPARTMENT ON Dno = Dnumber)
```

- **Outer joins:** Keeps all the tuples in R, or all those in S, or all those in both relations in the result of the JOIN, regardless of whether or not they have matching tuples in the other relation i.e. it preserves tuples that would have been lost by other types of join.

R ⟕ S	The (left) Outer join is a join in which tuples from R that do not have matching values in the common attributes of S are also included in the result relation. Missing values in the second relation are set to null.

T

A	B
a	1
b	2

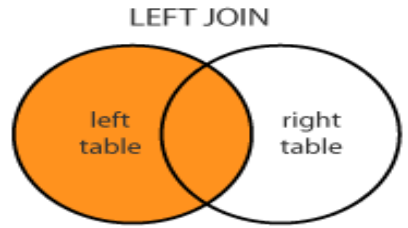
U

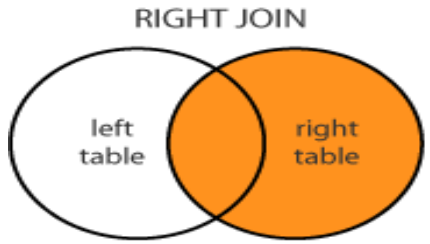
B	C
1	x
1	y
3	z

$T ⟕_C U$

A	B	C
a	1	x
a	1	y
b	2	

(i) Left Outer join





Similarly, **Right Outer join** keeps every tuple in the right-hand relation in the result. There is also a **Full Outer join** that keeps all tuples in both relations, padding tuples with nulls when no matching tuples are found.

UNION, INTERSECT, EXCEPT

- Combine the results of two or more queries into a single result table:
- Most important one being that the two tables have to be **union-compatible.**

Construct a list of all cities where there is either a branch office or a property.

```
(SELECT city                    or      (SELECT *
FROM Branch                             FROM Branch
WHERE city IS NOT NULL)                         WHERE city IS NOT NULL)
UNION                                   UNION CORRESPONDING BY city
(SELECT city                                    (SELECT *
FROM PropertyForRent                            FROM PropertyForRent
WHERE city IS NOT NULL);                        WHERE city IS NOT NULL);
```

Construct a list of all cities where there is both a branch office and a property.

```
(SELECT city                    or      (SELECT *
FROM Branch)                            FROM Branch)
INTERSECT                               INTERSECT CORRESPONDING BY city
(SELECT city                                    (SELECT *
FROM PropertyForRent);                          FROM PropertyForRent);
```

We could rewrite this query without the INTERSECT operator, for example:

```
SELECT DISTINCT b.city
FROM Branch b, PropertyForRent p
WHERE b.city = p.city;
```

or

```
SELECT DISTINCT city
FROM Branch b
WHERE EXISTS (SELECT *
              FROM PropertyForRent p
              WHERE b.city = p.city);
```

Construct a list of all cities where there is a branch office but no properties.

```
(SELECT city
FROM Branch)
EXCEPT
(SELECT city
FROM PropertyForRent);
```

or

```
(SELECT *
FROM Branch)
EXCEPT CORRESPONDING BY city
              (SELECT *
              FROM PropertyForRent);
```

We could rewrite this query without the EXCEPT operator, for example:

```
SELECT DISTINCT city
FROM Branch
WHERE city NOT IN (SELECT city
FROM PropertyForRent);
```

or

```
SELECT DISTINCT city
FROM Branch b
WHERE NOT EXISTS
              (SELECT *
              FROM PropertyForRent p
              WHERE b.city = p.city);
```

2.12.2 Primary Key

- A primary key is a field in a table which uniquely identifies each row/record in a database table.
- Primary keys must contain unique values. A primary key column cannot have NULL values.
- A table can have only one primary key, which may consist of single or multiple fields. When multiple fields are used as a primary key, they are called a composite key.

Create Primary Key :-

- Syntax to define the ID attribute as a primary key in a CUSTOMERS table.

```
CREATE TABLE CUSTOMER(
   ID   INT                    NOT NULL,
   NAME VARCHAR (20)           NOT NULL,
   AGE  INT                    NOT NULL,
   ADDRESS   CHAR (25),
   SALARY   DECIMAL (18,2),
   PRIMARY KEY (ID)
) ;
```

2.12.2 Primary Key

- To create a PRIMARY KEY constraint on the "ID" column when the CUSTOMERS table already exists, use the following SQL syntax −

```
ALTER TABLE CUSTOMER ADD PRIMARY KEY (ID) ;
```

- For defining a PRIMARY KEY constraint on multiple columns, use the SQL syntax given below.

```
CREATE TABLE CUSTOMER(
   ID   INT                    NOT NULL,
   NAME VARCHAR (20)           NOT NULL,
   AGE  INT                    NOT NULL,
   ADDRESS   CHAR (25),
   SALARY   DECIMAL (18,2),
   PRIMARY KEY (ID, NAME)
) ;
```

- To create a PRIMARY KEY constraint on the "ID" and "NAMES" columns when CUSTOMERS table already exists

```
ALTER TABLE CUSTOMERs
  ADD CONSTRAINTS PK_CUSTID PRIMARY KEY (ID, NAME) ;
```

2.12.2 Foreign Key

- A foreign key is a key used to link two tables together. This is sometimes also called as a referencing key.
- A Foreign Key is a column or a combination of columns whose values match a Primary Key in a different table.
- The relationship between 2 tables matches the Primary Key in one of the tables with a Foreign Key in the second table.

Example :

CUSTOMERS table

```
CREATE TABLE CUSTOMERS(
   ID   INT              NOT NULL,
   NAME VARCHAR (20)     NOT NULL,
   AGE  INT              NOT NULL,
   ADDRESS   CHAR (25) ,
   SALARY   DECIMAL  (18, 2),
   PRIMARY KEY (ID)
);
```

2.12.2 ORDERS Table

```
CREATE TABLE ORDERS (
   ID              INT                    NOT NULL,
   DATE            DATETIME ,
   CUSTOMER_ID INT references CUSTOMERS(ID)
   AMOUNT      double,
   PRIMARY KEY (ID)
);
```

- If the ORDERS table has already been created and the foreign key has not yet been set, the use the syntax for specifying a foreign key by altering a table.

```
ALTER TABLE ORDERS
   ADD FOREIGN KEY (Customer_ID) REFERENCES CUSTOMERS (ID);
```

2.13 Database and the Internet

Databases are NOT the Internet. We access databases with Internet browsers, but we are not searching the Internet.

Internet

- A collection of computer networks that share various types of information worldwide
- The World Wide Web (www) is a small part of the Internet that is made up of 'sites' (or documents) that allows sharing of the Internet information
- Internet information is accessed through URL addresses and/or search engines

Database

- An organized collection of information whose contents can be easily accessed, updated and managed
- Databases can be found in different formats and accessed via the Internet using browsers
- Library databases include thousands of magazine articles, newspapers, and scholarly journals

Accessing databases over the Internet

- **Server-Side Extensions** Serve as middleware between the web server and the data access methods
- **Web Server Interfaces** Two commonly used methods that allow the web server to communicate through the extensions
  - **Common Gateway Interface (CGI)** - Script files that communicate with the middleware
  - **Application Programming Interface (API)** - Share its connection with the database with multiple requests
- **Client-Side Extensions** -
  - **Plug-in extensions** - Additional software added to the browser that run when they are needed
  - **JavaScript** - An embedded script file (interpreted) that is part of a web page
  - **Java applets** - Programs that can be run by a client if the Java runtime environment has been installed;
  - **ActiveX controls** - Like Java applets, but they use code invented by Microsoft, and only run in Microsoft browsers
  - **VBScript** - a Microsoft script language, and its scripts can be run in web pages or separately
- **Web Application Servers** - Specialized server programs that can **perform the middleware duties** for the primary web server
- **Web Database Development** - Making web pages that are meant to interface with a database

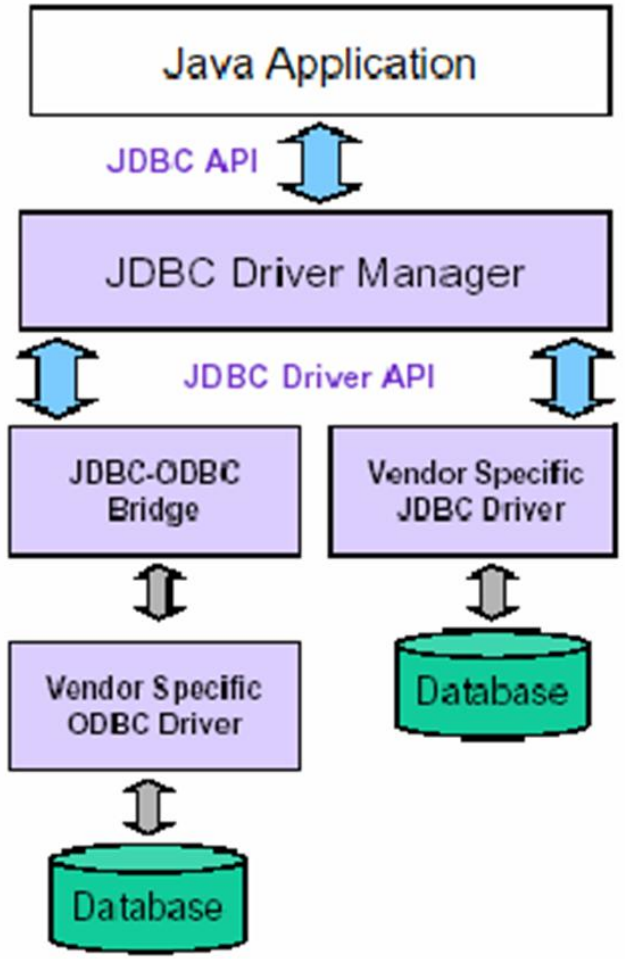
2.13.1 Database Connectivity and Web Technology

- The database connectivity refers to the mechanisms through which application programs connect and communicate with data repositories.
- Database connectivity software is also known as **database middleware** because it interfaces between the application program and the database.
- The data repository, also known as the *data source*, represents the data management application (that is, an Oracle RDBMS, SQL Server DBMS, or IBM DBMS) that will be used to store the data generated by the application program.
- There are many different ways to achieve database connectivity.

1. Native SQL connectivity (vendor provided).
2. Microsoft's Open Database Connectivity (ODBC).
3. Data Access Objects (DAO) and Remote Data Objects (RDO).
4. Microsoft's Object Linking and Embedding for Database (OLE-DB).
5. Microsoft's ActiveX Data Objects (ADO.NET).
6. Sun's Java Database Connectivity (JDBC).

2.13.1 Java Database Connectivity (JDBC)

- An API that lets you access virtually any tabular data source from the Java programming language.
- Basic steps to connect database in Java
  - Establish a **connection**
  - Create JDBC **Statements**
  - Execute **SQL** Statements
  - GET **ResultSet**
  - **Close** connections



Establishing the Connection

- **import java.sql.*;**
- **Load the vendor specific driver**
  - Class.forName("oracle.jdbc.driver.OracleDriver");
    - What do you think this statement does, and how?
    - Dynamically loads a driver class, for Oracle database
- **Make the connection**
  - Connection con = DriverManager.getConnection("jdbc:oracle:thin:@oracle-prod:1521:OPROD", username, passwd);
    - What do you think this statement does?
    - Establishes connection to database by obtaining a *Connection* object

Create JDBC statement (s)

- Statement stmt = con.createStatement() ;
- Creates a Statement object for sending SQL statements to the database

Executing SQL statements

- String createLehigh = "Create table Lehigh " +
  "(SSN Integer not null, Name VARCHAR(32), " + "Marks Integer)";
  stmt.**executeUpdate**(createLehigh);
  //What does this statement do?
- String insertLehigh = "Insert into Lehigh values“ + "(123456789,abc,100)";
  stmt.**executeUpdate**(insertLehigh);

Getting ResultSet

```
String queryLehigh = "select * from Lehigh";
ResultSet rs = Stmt.executeQuery(queryLehigh);
//What does this statement do?
while (rs.next()) {
    int ssn = rs.getInt("SSN");
    String name = rs.getString("NAME");
    int marks = rs.getInt("MARKS");
}
```

Terminating Connections

- stmt.close( );
- con.close( );

2.13.2 Internet Databases

There are millions of people all over the world use computers and Web browser software to access the Internet, connecting to databases over the Web. Web database connectivity opens the door to new innovative services that:

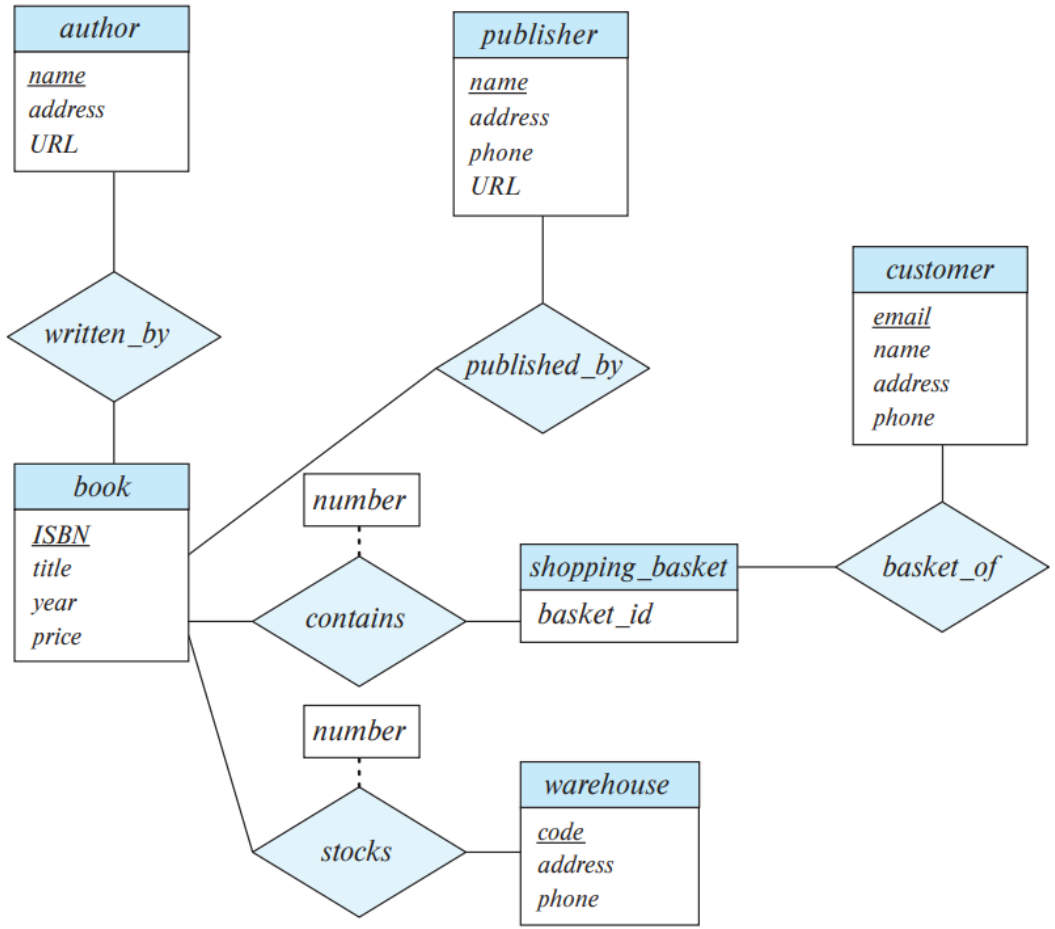
- Permit rapid responses to competitive pressures by bringing new services and products to market quickly.
- Increase customer satisfaction through the creation of Web-based support services.
- Yield fast and effective information dissemination through universal access from across the street or across the globe.

Characteristics and Benefits of Internet Technologies

Internet characteristic	Benefit
Hardware and software independence	Ability to run on most existing equipment Platform independence and portability
Common and simple user interface	Reduced training time and cost Reduced end-user support cost
Location independence Global	access through Internet infrastructure
Rapid development at manageable costs	Availability of multiple development tools

Worked Examples

Q. 1 Create the tables from the ER diagram given below using SQL:



Solution

create table **author** (name varchar(50) primary key, address varchar(50), url varchar(20));

create table **publisher** (name varchar(50) primary key, address varchar(50), phone integer(12), url varchar (20));

create table **customer** (email varchar(20)) primary key, name varchar(30), address varchar(50), phone integer(12));

create table **book** (ISBN varchar(20)) primary key, title varchar(30), year number(5), price numeric(5,2));

create table **shopping_basket** (basket_id number(10));

create table **warehouse** (code varchar(20)) primary key, address varchar(30), phone number(12));

Worked Examples

Q. 2 Consider a database schema *student(id, name, class, dob, gender)*, create table with *id* as the primary key and insert information into the table. Add a new attribute *guardian* to the schema.

Solution

Creating table:

```
create table student ( id integer(10) primary key, name varchar(50), class integer(3), dob date, gender varchar(10));
```

Inserting information:

```
insert into table student (1011, 'Rahul' , 8, 14-4-1996 , 'male')
insert into table student (1011, 'Rahul' , 8, 14-4-1996 , 'male')
insert into table student (1011, 'Rahul' , 8, 14-4-1996 , 'male')
insert into table student (1011, 'Rahul' , 8, 14-4-1996 , 'male')
```

Adding new attribute:

```
alter student add guardian varchar(50);
```

Worked Examples

Q. 3 Consider the database ‘Student’ having a table test with then information name, roll number and class. Fetch the information from the database using java database connectivity.

Test

Name	Roll_no	Class
Rahul	121	10
Rohan	122	9
Peter	123	8

Solution

```
import java.sql.*;
class MysqlCon
  {
    public static void main(String args[ ])
    {
       try
           {
                Class.forName("com.mysql.jdbc.Driver");
                Connection con =
                DriverManager.getConnection("jdbc:mysql://localhost:3306/student","root","root");
                Statement stmt=con.createStatement();
                ResultSet rs=stmt.executeQuery("select * from test");
                while(rs.next())
                System.out.println(rs.getInt(1)+"  "+rs.getString(2)+"  "+rs.getString(3));
                con.close();
            }
         catch(Exception e)
           {
                System.out.println(e);}
           }
     }
```

Worked Examples

Q. 4 Consider the database ‘Student’ having a table test with then information name, roll number and class. Fetch the information from the database using java database connectivity.

Test

Name	Roll_no	Class
Rahul	121	10
Rohan	122	9
Peter	123	8

Solution

```
import java.sql.*;
class MysqlCon
  {
    public static void main(String args[ ])
    {
       try
           {
                Class.forName("com.mysql.jdbc.Driver");
                Connection con =
                DriverManager.getConnection("jdbc:mysql://localhost:3306/student","root","root");
                Statement stmt=con.createStatement();
                ResultSet rs=stmt.executeQuery("select * from test");
                while(rs.next())
                System.out.println(rs.getInt(1)+"  "+rs.getString(2)+"  "+rs.getString(3));
                con.close();
           }
         catch(Exception e)
           {
                System.out.println(e);}
           }
    }
```

Worked Examples

Q. 5 Consider the database ‘Student’ having a table test with then information name, roll number and class. Fetch the information from the database using java database connectivity.

Test

Name	Roll_no	Class
Rahul	121	10
Rohan	122	9
Peter	123	8

Solution

```
import java.sql.*;
class MysqlCon
  {
    public static void main(String args[ ])
    {
       try
           {
                Class.forName("com.mysql.jdbc.Driver");
                Connection con =
                DriverManager.getConnection("jdbc:mysql://localhost:3306/student","root","root");
                Statement stmt=con.createStatement();
                ResultSet rs=stmt.executeQuery("select * from test");
                while(rs.next())
                System.out.println(rs.getInt(1)+"  "+rs.getString(2)+"  "+rs.getString(3));
                con.close();
            }
         catch(Exception e)
           {
                System.out.println(e);}
           }
     }
```

Worked Examples

Q. 6 Consider the database ‘Student’ having a table test with then information name, roll number and class. Fetch the information from the database using java database connectivity.

Test

Name	Roll_no	Class
Rahul	121	10
Rohan	122	9
Peter	123	8

Solution

```
import java.sql.*;
class MysqlCon
  {
    public static void main(String args[ ])
    {
       try
           {
                Class.forName("com.mysql.jdbc.Driver");
                Connection con =
                DriverManager.getConnection("jdbc:mysql://localhost:3306/student","root","root");
                Statement stmt=con.createStatement();
                ResultSet rs=stmt.executeQuery("select * from test");
                while(rs.next())
                System.out.println(rs.getInt(1)+"  "+rs.getString(2)+"  "+rs.getString(3));
                con.close();
            }
         catch(Exception e)
           {
                System.out.println(e);}
           }
     }
```

Worked Examples

Q. 7 Consider the database ‘Student’ having a table test with then information name, roll number and class. Fetch the information from the database using java database connectivity.

Test

Name	Roll_no	Class
Rahul	121	10
Rohan	122	9
Peter	123	8

Solution

```
import java.sql.*;
class MysqlCon
  {
    public static void main(String args[ ])
    {
       try
           {
                Class.forName("com.mysql.jdbc.Driver");
                Connection con =
                DriverManager.getConnection("jdbc:mysql://localhost:3306/student","root","root");
                Statement stmt=con.createStatement();
                ResultSet rs=stmt.executeQuery("select * from test");
                while(rs.next())
                System.out.println(rs.getInt(1)+"  "+rs.getString(2)+"  "+rs.getString(3));
                con.close();
            }
         catch(Exception e)
           {
                System.out.println(e);}
           }
     }
```

Worked Examples

Q. 8 Consider the database ‘Student’ having a table test with then information name, roll number and class. Fetch the information from the database using java database connectivity.

Test

Name	Roll_no	Class
Rahul	121	10
Rohan	122	9
Peter	123	8

Solution

```
import java.sql.*;
class MysqlCon
  {
    public static void main(String args[ ])
    {
       try
           {
                Class.forName("com.mysql.jdbc.Driver");
                Connection con =
                DriverManager.getConnection("jdbc:mysql://localhost:3306/student","root","root");
                Statement stmt=con.createStatement();
                ResultSet rs=stmt.executeQuery("select * from test");
                while(rs.next())
                System.out.println(rs.getInt(1)+"  "+rs.getString(2)+"  "+rs.getString(3));
                con.close();
            }
         catch(Exception e)
           {
                System.out.println(e);}
           }
     }
```

Worked Examples

Q. 9 Consider the database ‘Student’ having a table test with then information name, roll number and class. Fetch the information from the database using java database connectivity.

Test

Name	Roll_no	Class
Rahul	121	10
Rohan	122	9
Peter	123	8

Solution

```
import java.sql.*;
class MysqlCon
  {
    public static void main(String args[ ])
    {
       try
           {
                Class.forName("com.mysql.jdbc.Driver");
                Connection con =
                DriverManager.getConnection("jdbc:mysql://localhost:3306/student","root","root");
                Statement stmt=con.createStatement();
                ResultSet rs=stmt.executeQuery("select * from test");
                while(rs.next())
                System.out.println(rs.getInt(1)+"  "+rs.getString(2)+"  "+rs.getString(3));
                con.close();
            }
         catch(Exception e)
           {
                System.out.println(e);}
           }
     }
```

Worked Examples

Q. 10 Consider the database ‘Student’ having a table test with then information name, roll number and class. Fetch the information from the database using java database connectivity.

Test

Name	Roll_no	Class
Rahul	121	10
Rohan	122	9
Peter	123	8

Solution

```
import java.sql.*;
class MysqlCon
  {
    public static void main(String args[ ])
    {
       try
           {
                Class.forName("com.mysql.jdbc.Driver");
                Connection con =
                DriverManager.getConnection("jdbc:mysql://localhost:3306/student","root","root");
                Statement stmt=con.createStatement();
                ResultSet rs=stmt.executeQuery("select * from test");
                while(rs.next())
                System.out.println(rs.getInt(1)+"  "+rs.getString(2)+"  "+rs.getString(3));
                con.close();
           }
         catch(Exception e)
           {
                System.out.println(e);}
           }
     }
```

Worked Examples

Q. 11 Consider the database ‘Student’ having a table test with then information name, roll number and class. Fetch the information from the database using java database connectivity.

Test

Name	Roll_no	Class
Rahul	121	10
Rohan	122	9
Peter	123	8

Solution

```
import java.sql.*;
class MysqlCon
  {
    public static void main(String args[ ])
    {
       try
           {
                Class.forName("com.mysql.jdbc.Driver");
                Connection con =
                DriverManager.getConnection("jdbc:mysql://localhost:3306/student","root","root");
                Statement stmt=con.createStatement();
                ResultSet rs=stmt.executeQuery("select * from test");
                while(rs.next())
                System.out.println(rs.getInt(1)+"  "+rs.getString(2)+"  "+rs.getString(3));
                con.close();
            }
         catch(Exception e)
           {
                System.out.println(e);}
           }
     }
```

Worked Examples

Q. 12 Consider the database ‘Student’ having a table test with then information name, roll number and class. Fetch the information from the database using java database connectivity.

Test

Name	Roll_no	Class
Rahul	121	10
Rohan	122	9
Peter	123	8

Solution

```
import java.sql.*;
class MysqlCon
  {
    public static void main(String args[ ])
    {
       try
           {
                Class.forName("com.mysql.jdbc.Driver");
                Connection con =
                DriverManager.getConnection("jdbc:mysql://localhost:3306/student","root","root");
                Statement stmt=con.createStatement();
                ResultSet rs=stmt.executeQuery("select * from test");
                while(rs.next())
                System.out.println(rs.getInt(1)+"  "+rs.getString(2)+"  "+rs.getString(3));
                con.close();
            }
         catch(Exception e)
           {
                System.out.println(e);}
           }
     }
```

Worked Examples

Q. 13 Consider the database ‘Student’ having a table test with then information name, roll number and class. Fetch the information from the database using java database connectivity.

Test

Name	Roll_no	Class
Rahul	121	10
Rohan	122	9
Peter	123	8

Solution

```
import java.sql.*;
class MysqlCon
  {
    public static void main(String args[ ])
    {
       try
           {
                Class.forName("com.mysql.jdbc.Driver");
                Connection con =
                DriverManager.getConnection("jdbc:mysql://localhost:3306/student","root","root");
                Statement stmt=con.createStatement();
                ResultSet rs=stmt.executeQuery("select * from test");
                while(rs.next())
                System.out.println(rs.getInt(1)+"  "+rs.getString(2)+"  "+rs.getString(3));
                con.close();
            }
         catch(Exception e)
           {
                System.out.println(e);}
           }
     }
```

Worked Examples

Q. 14 Consider the database ‘Student’ having a table test with then information name, roll number and class. Fetch the information from the database using java database connectivity.

Test

Name	Roll_no	Class
Rahul	121	10
Rohan	122	9
Peter	123	8

Solution

```
import java.sql.*;
class MysqlCon
  {
    public static void main(String args[ ])
    {
       try
           {
                Class.forName("com.mysql.jdbc.Driver");
                Connection con =
                DriverManager.getConnection("jdbc:mysql://localhost:3306/student","root","root");
                Statement stmt=con.createStatement();
                ResultSet rs=stmt.executeQuery("select * from test");
                while(rs.next())
                System.out.println(rs.getInt(1)+"  "+rs.getString(2)+"  "+rs.getString(3));
                con.close();
            }
         catch(Exception e)
           {
                System.out.println(e);}
           }
     }
```

Worked Examples

Q. 15 Consider the database ‘Student’ having a table test with then information name, roll number and class. Fetch the information from the database using java database connectivity.

Test

Name	Roll_no	Class
Rahul	121	10
Rohan	122	9
Peter	123	8

Solution

```
import java.sql.*;
class MysqlCon
  {
    public static void main(String args[ ])
    {
       try
           {
                Class.forName("com.mysql.jdbc.Driver");
                Connection con =
                DriverManager.getConnection("jdbc:mysql://localhost:3306/student","root","root");
                Statement stmt=con.createStatement();
                ResultSet rs=stmt.executeQuery("select * from test");
                while(rs.next())
                System.out.println(rs.getInt(1)+"  "+rs.getString(2)+"  "+rs.getString(3));
                con.close();
            }
         catch(Exception e)
           {
                System.out.println(e);}
           }
     }
```

Worked Examples

Q. 16 Consider the database ‘Student’ having a table test with then information name, roll number and class. Fetch the information from the database using java database connectivity.

Test

Name	Roll_no	Class
Rahul	121	10
Rohan	122	9
Peter	123	8

Solution

```
import java.sql.*;
class MysqlCon
  {
    public static void main(String args[ ])
    {
       try
           {
                Class.forName("com.mysql.jdbc.Driver");
                Connection con =
                DriverManager.getConnection("jdbc:mysql://localhost:3306/student","root","root");
                Statement stmt=con.createStatement();
                ResultSet rs=stmt.executeQuery("select * from test");
                while(rs.next())
                System.out.println(rs.getInt(1)+"  "+rs.getString(2)+"  "+rs.getString(3));
                con.close();
            }
         catch(Exception e)
           {
                System.out.println(e);}
           }
     }
```

Worked Examples

Q. 17 Consider the database ‘Student’ having a table test with then information name, roll number and class. Fetch the information from the database using java database connectivity.

Test

Name	Roll_no	Class
Rahul	121	10
Rohan	122	9
Peter	123	8

Solution

```
import java.sql.*;
class MysqlCon
  {
    public static void main(String args[ ])
    {
       try
           {
                Class.forName("com.mysql.jdbc.Driver");
                Connection con =
                DriverManager.getConnection("jdbc:mysql://localhost:3306/student","root","root");
                Statement stmt=con.createStatement();
                ResultSet rs=stmt.executeQuery("select * from test");
                while(rs.next())
                System.out.println(rs.getInt(1)+"  "+rs.getString(2)+"  "+rs.getString(3));
                con.close();
            }
         catch(Exception e)
           {
                System.out.println(e);}
           }
     }
```

Worked Examples

Q. 18 Perform natural join between the tables ‘Customer’ and ‘Orders’ with SQL.

Customer

Id	Name	Age	Adress	Salary
1	Ramesh	32	Ahmadabad	2000.00
2	Khilan	25	Delhi	1500.00
3	Kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00

Orders

Oid	Date	Customer_id	Amount
102	2010-10-08	3	3000
100	2010-10-08	3	1500
101	2010-05-22	2	1560
103	2010-08-22	4	2060

Solution

```
select id, Name, Age, Amont
from  Customer,  Orders
where  Customer.Id  =  Orders. Customer_Id;
```

Id	Name	Age	Amount
3	Kaushik	23	3000
3	Kaushik	23	1500
2	Khilan	25	1560
4	Chaitali	25	2060

Worked Examples

Q. 19 Consider the two tables ‘Product’ and ‘Category’ from a database. Identify the product and its category from this information excluding the product ‘Pear’.

Product_id	Product_name	Category_id
1	Pear	50
2	Banana	50
3	Orange	50
4	Apple	50
5	Bread	75
6	Grapes	25
7	Biscuits	NULL

Product

Category_id	Category_name
25	Deli
50	Produce
75	Bakery
100	General Merchandise
125	Technology

Category

Solution

```
SELECT products. Product_name, categories.category_name
FROM products
INNER JOIN categories
ON products. Category_id = categories. Category_id
WHERE Product_name <> ‘Pear’;
```

Product_name	Category_name
Banana	Produce
Orange	Produce
Apple	Produce
Bread	Bakery
Grapes	Deli

Worked Examples

Q. 20 Consider the database ‘Student’ having a table test with then information name, roll number and class. Fetch the information from the database using java database connectivity.

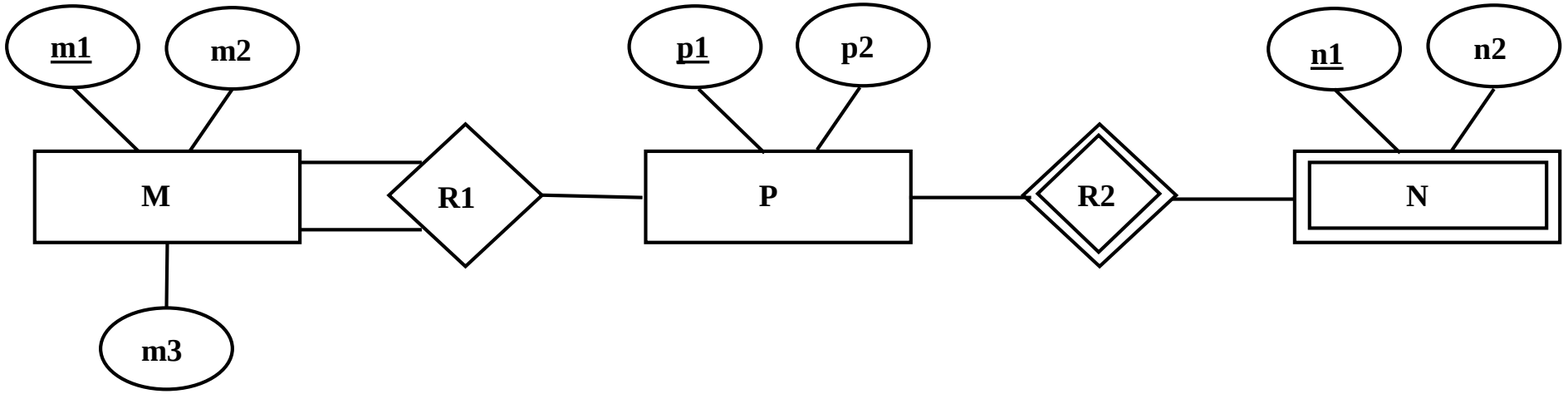
Test

Name	Roll_no	Class
Rahul	121	10
Rohan	122	9
Peter	123	8

Solution

```
import java.sql.*;
class MysqlCon
  {
    public static void main(String args[ ])
    {
       try
           {
                Class.forName("com.mysql.jdbc.Driver");
                Connection con =
                DriverManager.getConnection("jdbc:mysql://localhost:3306/student","root","root");
                Statement stmt=con.createStatement();
                ResultSet rs=stmt.executeQuery("select * from test");
                while(rs.next())
                System.out.println(rs.getInt(1)+"  "+rs.getString(2)+"  "+rs.getString(3));
                con.close();
            }
         catch(Exception e)
           {
                System.out.println(e);}
           }
     }
```

1. With help of a suitable example, explain the various constraints used in a relational model.
2. Draw an ER diagram showing relationship between an employee and a department entity. Show all the possible components.
3. What are Super keys, Candidate keys and Primary keys? How are they related to each other? Explain with an example.
4. Convert the following ER diagram in relational model with minimum tables,



5. Consider the following schemas:

Suppliers(sid, sname, address)

Parts(pid, pname, colour)

Catalog(sid, pid, cost)

a)Write the following queries in relational algebra.

b)Find the *IDs* of the suppliers who supply some red or green part.

c)Find the names of all the parts whose price is more than 4000 but less than 7500.

d)Find the details of all suppliers who supplies “Baskets”.

e)Find the name of all the products supplied by “Agarwal Ltd.” but not by “Sharma Enterprise”

6. Normalize the given relational schemas, with the FDs to BCNF.

Books (accessionno, isbn, title, author, publisher)

Users (userid, name, deptid, deptname)

Accessionno → isbn, isbn → title, isbn → publisher, isbn → author, userid→name,
userid → deptid & deptid → deptname

The Topics Covered in This Section

2.16.1 Books
2.16.2 Video Lectures

- Silberschatz, A., Korth, H. F., & Sudarshan, S. *Database system concepts* . New York: McGraw-Hill. **2019**

➢ Discusses fundamental topics like Database Languages, Database Users and Administrators, Database Design, and Data Storage and Querying ★★★★

- Groff, J., Weinberg, P., and Oppel, A. *SQL The Complete Reference*. McGraw Hill Education. **2017**

➢ Shows how to work with SQL commands and statements, set up relational databases, load and modify database objects, perform powerful queries, tune performance, and implement reliable security policies ★★★

- Date, C.J. *An Introduction to Database System*. Pearson education. **2002**

➢ Provides a solid grounding in the foundations of database technology and shedding some light on how the field is likely to develop in the future. ★★★

- Elmarsi, R., and Navathe, S.B. *Fundamentals of Database Systems*. Pearson education. **2017**

➢ Clear explanations of theory and design, broad coverage of models and real systems, and an up-to-date introduction to modern database technologies result in a leading introduction to database systems. ★★★

- Connolly, T.M., Begg, C.M. *Database Systems, A Practical Approach to Design, Implementation, and Management*. Pearson education. **2019**

➢ Introduces the theory behind databases in a concise yet comprehensive manner, providing database design methodology. ★★

2.16.2 Video Lectures

- Introduction to Databases - Jennifer Widom | Stanford

https://www.youtube.com/watch?v=D-k h0GuFmE&list=PLroEs25KGvwzmvIxYHRhoGTz9w8LeXek0

- Lecture NoSQL databases | Lena Wiese

https://www.youtube.com/watch?v=MUVgXSIqwlQ&list=PLKdGHuzZ3K_oJUPLycaAiKfyhAe6tsrnk

Thank You