Unit -1

- **Data** is a raw and unorganized fact that is required to be processed to make it meaningful.
- When data is processed, organized, structured, or presented in a given context, so as to make it useful is called **information**

Database

- A database is a collection of related **data representing some aspect of the real world**, also called the mini-world or the universe of discourse (UoD).
- **It is a logically coherent collection**, meaning it is not just a random assortment of data but is organized with inherent meaning and structure.
- Databases are **designed, built, and populated for a specific purpose,** catering to the needs of applications or systems that interact with them.
- They **can vary in size and complexity**, ranging from small databases used by individual applications to large-scale enterprise databases handling vast amounts of data

Database Management System

- A modern database system is a complex software system whose **task is to manage a large, complex collection of data.**
- **DBMS contains information about a particular enterprise**
  - Collection of interrelated data
  - Set of programs to access the data
  - An environment that is both convenient and efficient to use
- Database systems are **used to manage collections of data** that are:
  - **Highly valuable**
  - **Relatively large**
  - **Accessed by multiple users and applications, often at the same time.**

Student Database

1. Information on Enterprise
2. Convenient and Efficient Environment
3. Collection of Interrelated Data
4. Set of programs to access the data
5. Managing large , complex collection of data
6. Accessed by multiple users
7. Highly valuable data

Creation of Database

1. Define Database : Data types
2. Construct Database : Create tables
3. Manipulate Database : Queries
4. Share Database : Access control

Importance of Database

Databases help us to:

- **Store** large amounts of data (file structure, disk management)
- **Understand** the data (data models)
- Keeps data **secure** (security, recovery)
- Find required data and use/**manipulate** it (query languages, concurrency control, and data analysis tools)
- Get **accurate information** (as databases have built-in constraints and checks help in this)
- Maintain **Data integrity** (ensures data is accurate and consistent)

Database Application

- Enterprise
- Manufacturing
- Banking
- Universities
- Airlines
- Telecom
- Web services
- Navigation

Database Modes

Online Transaction Processing (OLTP):

- Used by a large number of users for small data retrieval and updates
- common in most database applications like banking, universities, and airlines.

Data Analytics:

- Involves processing data to draw conclusions and create predictive models for business decisions.
- Examples include loan approval, targeted advertisements, and manufacturing decisions.
- Data mining combines AI and statistical techniques for efficient analysis of large databases.

File Processing System disadvantages

- Data Redundancy and Inconsistency
  - Store information in centralized location thus Minimizes redundancy
- Difficulty in accessing the data
  - Use a powerful query language for answering new queries
- Data isolation
  - Centralized system
- Integrity problems
  - Specify integrity constraints at the database level
- Atomicity problems
  - Automatic rollback in case of failure
- Concurrent access anomalies
  - Transaction and Locking to be implemented
- Security problems

Database Characterstics

1. Self-describing nature of a database system
2. Insulation between programs and data, and data abstraction

3. Support multiple views of the data
4. Sharing of data and multiuser transaction processing

Advantage of DBMS

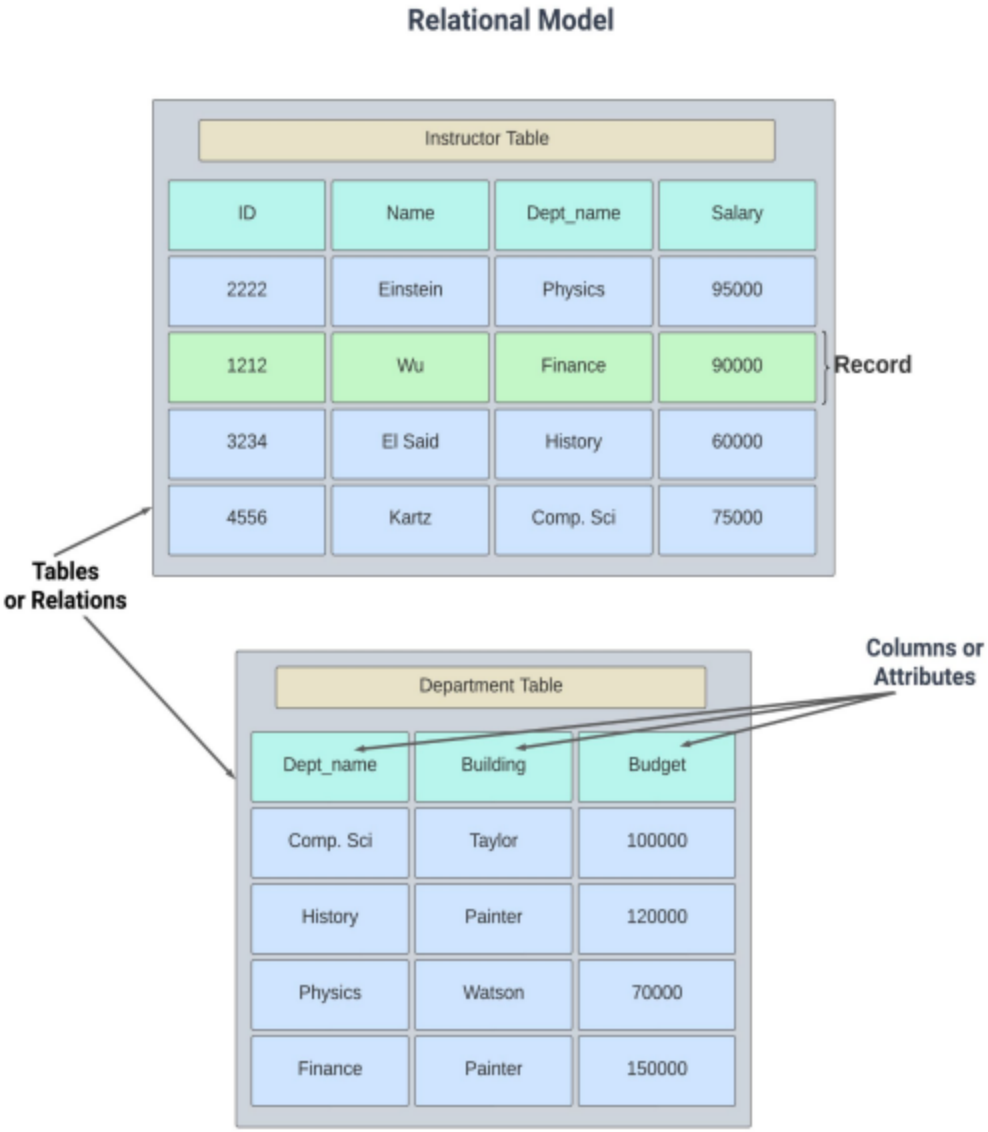
- Controlling redundancy in data storage.
- Sharing of data among multiple users.
- Restricting unauthorized access to data
- Providing persistent storage for program Objects
- Providing Storage Structures and search techniques for efficient Query Processing
- Providing backup and recovery
- Providing multiple interfaces to different classes of users.
- Representing complex relationships among data.
- Enforcing integrity constraints on the database.
- Drawing inferences and actions from the stored data using deductive and active rules and triggers
- Potential for enforcing standards
- Reduced application development time
- Flexibility to change data structures
- Availability of current information
- Economies of scale

Data View

1. Data Model

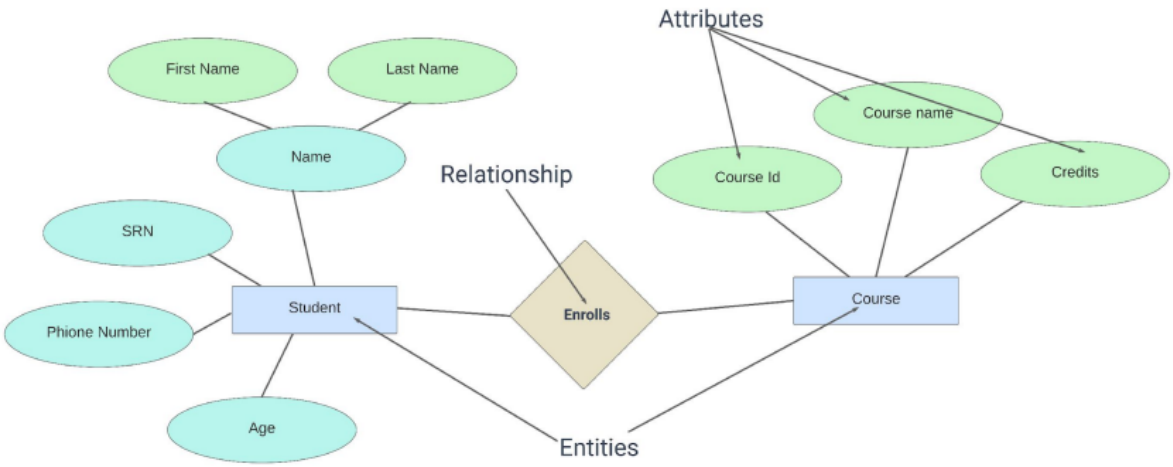
A collection of conceptual tools for describing data, data relationships, data semantics, and consistency constraints.

1. **Relational Model**
    - Uses a **collection of tables** to represent both **data and the relationships among those data**.
    - Each table has multiple columns, and each column has a unique name.
    - **Tables are also known as relations.**
    - The relational model is an example of a record-based model. Record-based models are so named because the database is structured in fixed-format records of several types.
    - **Each table contains records of a particular type.** Each record type defines a fixed number of fields or attributes.
    - **Each row of the table represents one piece of information**
    - The **columns of the table correspond to the attributes of the record type.**
    - The relational data model is the **most widely used data model**, and a vast majority of current database systems are based on the relational model.



2. **Entity-Relationship Model**

- The entity-relationship (E-R) data model uses a collection of basic **objects, called entities,** and relationships among these objects.
- An entity is a “thing” or “object” in the real world that is distinguishable from other objects.
- The entity-relationship model is widely used in database design.



3. **Semi Structured Data Model**

- The semi-structured data model permits the specification of data where **individual data items of the same type may have different sets of attributes.**
- This is in contrast to the data models mentioned earlier, where every data item of a particular type must have the same set of attributes.
- **JSON and Extensible Markup Language (XML) are widely used semi-structured data** representations
- A database model where there is **no separation between the data and the schema**, and the amount of structure used depends on the purpose.
- The advantages of this model are the following: **It can represent the information of some data sources that cannot be constrained by schema.**

```
<purchase_order>
    <identifier> P-101 </identifier>
    <purchaser>
        <name> Cray Z. Coyote </name>
        <address> Route 66, Mesa Flats, Arizona 86047, USA </address>
    </purchaser>
    <supplier>
        <name> Acme Supplies </name>
        <address> 1 Broadway, New York, NY, USA </address>
    </supplier>
    <itemlist>
        <item>
            <identifier> RS1 </identifier>
            <description> Atom powered rocket sled </description>
            <quantity> 2 </quantity>
            <price> 199.95 </price>
        </item>
        <item>
            <identifier> SG2 </identifier>
            <description> Superb glue </description>
            <quantity> 1 </quantity>
            <unit-of-measure> liter </unit-of-measure>
            <price> 29.95 </price>
        </item>
    </itemlist>
    <total_cost> 429.85 </total_cost>
    <payment_terms> Cash-on-delivery </payment_terms>
    <shipping_mode> 1-second-delivery </shipping_mode>
</purchase_order>
```

4. **Object-Based Data Model**

- Object-oriented programming (especially in Java, C++, or C#) has become the dominant software-development methodology.
- This led initially to the development of a **distinct object-oriented data model,** but today the concept of objects is well integrated into relational databases.
- Standards exist to **store objects in relational tables**. Database systems allow,procedures to be stored in the database system and executed by the database system.

- This can be seen as extending the relational model with notions of **encapsulation, methods, and object identity**

2. Data Abstraction

Developer hide all these complexity from the users through several layers of data abstraction, in order to simplify the users' interaction with the system.

Levels of Data Abstraction:

1. **Physical Level:**
   - Lowest abstraction level, **detailing actual data storage mechanisms.**
   - Example: Describing how student records, course information, and department data are physically stored on the hard drive or in memory.
2. **Logical Level:**
   - Describes **data types and their relationships.**
   - Presents a more organized and simplified view.
   - Example: Defining entities like "Department," "Student," and "Course" along with their attributes and relationships.
3. **View Level:**
   - Highest abstraction level for user interaction.
   - Displays **specific portions of the database, customized for users.**
   - Example: Faculty members having a view showing only the courses they teach and the students enrolled, while students have a view displaying only their registered courses and respective grades.

Achieving User-Friendly Interaction:

- Data abstraction layers shield users from complexities.
- **Physical Level:** Users unaware of storage intricacies.
- **Logical Level:** Users interact with simplified structures.

- **View Level:** Customized views based on user needs.

Data Independence:

- **Physical Data Independence:**
  - **Modifying physical schema without impacting logical or conceptual schema.**
  - Example: Changing storage devices or data structures without affecting the conceptual structure.
- **Logical Data Independence:**
  - **Modifying logical schema without affecting external schema or applications.**
  - Example: Inserting or deleting attributes without altering user views.

Examples with University Database:

- **Physical Schema Example:**
  - Specification of how student records, course information, and department data are stored on the hard drive, e.g., using B-trees or hash indexes for efficient retrieval.
- **Logical Schema Example:**
  - Definition of entities like "Department," "Student," and "Course," their attributes (e.g., department name, student ID), and relationships (e.g., "Students belong to Departments," "Courses are taken by Students").
- **View Schema Example:**
  - Faculty members see a view showing only the courses they teach and the students enrolled. Students have a view displaying only their registered courses and respective grades.

Instances and Schema

Database Evolution:

- Databases evolve with information changes (insertions, deletions, modifications).
- **Instance:** Snapshot of the database at a specific moment.

Types of Schemas:

1. **Physical Schema:**
   - **Describes physical database design details.**
   - **Physical schema changes do not impact application programs.**
   - Example: Specification of data storage format, file organization, and indexing methods like B-trees.
2. **Logical Schema:**
   - **Defines higher-level database structure.**
   - Example: Defining relationships between tables, such as the one-to-many relationship between students and courses.
3. **View Schema (Subschema):**
   - **Describes customized views for specific users or applications.**
   - Example: Creating a view for faculty members to see only courses they teach, **hiding irrelevant details.**

Significance of Logical Schema:

- Influences application programs, providing the foundation for development.
- **Physical Data Independence:** Allows modifications without application rewrite.

Consideration of University Database:

- **Physical Schema Example:**
  - Detailing how data is stored on the hard disk using specific indexing methods for optimized retrieval.
- **Logical Schema Example:**

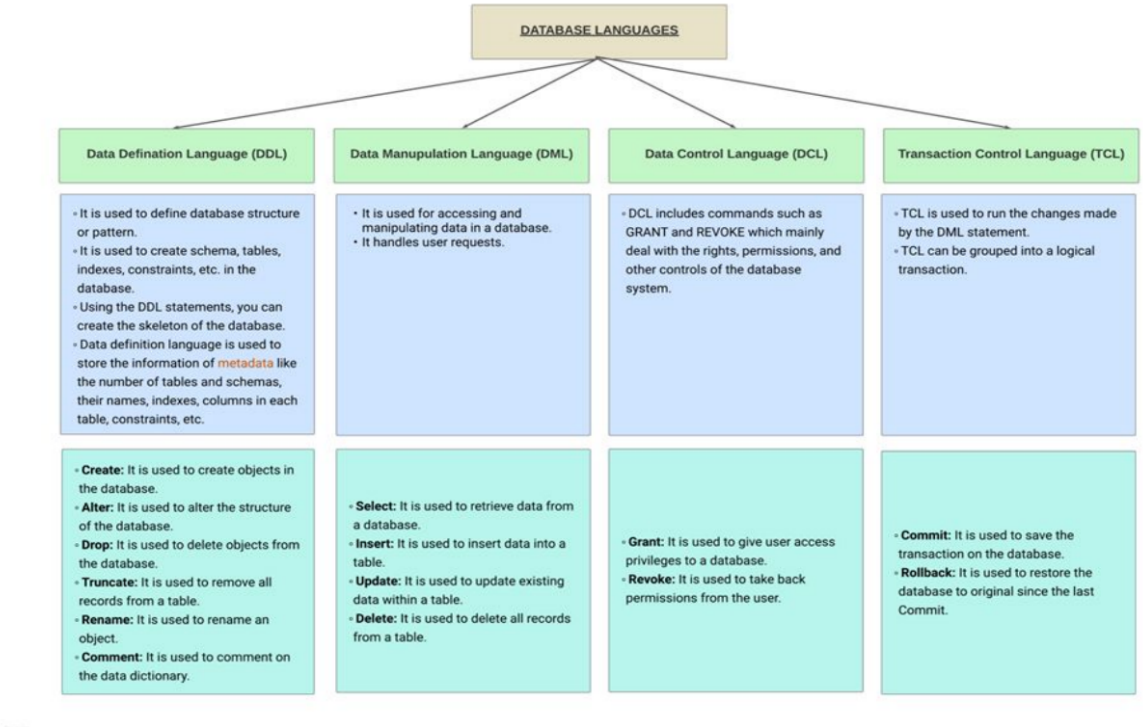
- Defining relationships like one-to-many between students and courses, shaping the overall database structure.

- **View Schema Example:**
  - Creating views tailored for faculty and students, offering customized perspectives on the database.

Ensuring Efficient Design:

- Logical schema's impact on application development.
- Physical schema hidden beneath logical schema, enabling changes without application rewrite.
- Strive for schemas without redundant information to optimize storage and retrieval.

Database Languages



Data Definition Language (DDL)

CREATE , ALTER , DROP , TRUNCATE (CAD-T)

Storage Structure Specification:

- **DDL specifies storage structure** and access methods through specialized statements.
- Data storage and definition language includes creating templates stored in a data dictionary.
- **Data Dictionary Content:**
  - Contains metadata (data about data).
  - Includes:
    - **Database schema.**
    - **Integrity constraints (e.g., Primary keys).**
    - **Authorization details (permissions on data access).**

Metadata in Data Dictionary:

- **Database Schema:**
  - Structural details defining entities, attributes, and relationships.
- **Integrity Constraints:**
  - Rules ensuring data consistency.
  - Example: Ensuring a department's account balance never becomes negative.
- **Authorization:**
  - Defines who can access what in the database.
  - Specifies read, insert, update, delete authorizations.

Importance of Data Dictionary:

- **Special table accessed and updated only by the database system.**
- **Consulted before reading or modifying actual data.**
- **Key for maintaining data integrity and controlling access.**

Constraint Handling in DDL

Consistency Constraints:

- Ensures data values meet specified constraints.
- Example: University database ensuring departmental funding doesn't go negative.
- **Domain Constraints:**
  - Associates a domain of possible values with each attribute.
  - Acts as a constraint on attribute values.
  - Example: Specifying integer or character types for attributes.
- **Referential Integrity:**
  - Ensures a value in one relation for specific attributes appears in another relation.
  - Example: Ensuring a department listed for a course exists in the university's department records.
- **Authorization Constraints:**
  - Controls user operations on the database.
  - Specifies read, insert, update, delete authorizations.
  - Example: Restricting certain users to read-only access.

Implementation Challenges:

- Constraints checked during database updates.
- Only implement constraints with minimal testing overhead.

Data Manipulation Language (DML)

SELECT , INSERT , DELETE , UPDATE (SIDe-U)

Overview:

- **DML enables users to access or manipulate data following the data model.**
- Types of access: retrieval, insertion, deletion, modification.

Procedural vs. Declarative DML:

- **Procedural DML:**
  - Requires users to specify what data is needed and how to obtain it.
- **Declarative DML:**
  - Requires users to specify what data is needed without specifying how to obtain it.
  - Generally easier to learn but demands efficient implementation by the database system.

Query Language and Processor:

- **Query Language:**
  - Statement requesting data retrieval.
  - Forms a part of DML, emphasizing ease of use.
- **Query Processor:**
  - Translates DML queries into actions at the physical level.
  - Balances human interaction efficiency with system optimization.

SQL Query Language

Nonprocedural Nature:

- SQL is a nonprocedural query language.
- Example: Retrieving names of instructors in the Computer Science department.

```
SELECT name
FROM instructor
WHERE dept_name = 'Comp. Sci.';
```

- **Not a Turing machine equivalent language, often embedded in higher-level languages.**

SQL in Application Programs:

- Access databases through language extensions or application program interfaces (ODBC, JDBC).

SQL Data Definition Language

Rich DDL in SQL:

- Defines tables with data types and integrity constraints.
- Example: Defining the department table.

```
CREATE TABLE department (
  dept_name CHAR(20),
  building CHAR(15),
  budget NUMERIC(12,2)
);
```

- Supports integrity constraints like primary keys.

SQL Data Manipulation Language

Variety of DML Statements:

- SQL provides a range of DML statements for querying databases.
- Example: SQL query to find names of instructors in the History department.

```
SELECT instructor.name
FROM instructor
```

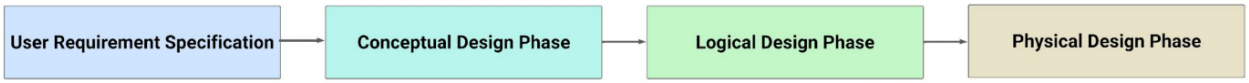
```
WHERE instructor.dept_name = 'History';
```

DML in Application Programs:

- Embedded in host languages (C/C++, Java, Python) for powerful computations.
- Application programs interact with databases through an API (e.g., ODBC/JDBC).

Database Design

Database design mainly involves the **design of the database schema**. The design of a complete database application environment that meets the needs of the enterprise being modeled requires attention to a broader set of issues



User Requirement

- Interact with domain experts to understand all data needs and functional Requirement
- Document the requirements for clarity

Conceptual Design

- Choose High Level Data Model
- Develop Conceptual Schema
- Remove redundant features
- Check all functional requirements are met
- Review the schema checking all requirements are satisfied

Logical Design

- Use normalization
- Map high level conceptual schema to the data model

Physical Design

- File organization and storage structure are specified
- performace optimization

Design Pitfalls to Avoid:

1. **Redundancy:**
    - **Repeating the same information unnecessarily.**
    - Example: Storing department names along with department IDs for every course.
    - **Drawbacks:**
        - Inconsistency in redundant copies if updated without care.
        - Difficulty in managing changes consistently.
2. **Incompleteness:**
    - **Not representing new information effectively.**
    - Example: Difficulty in adding details about a new course if a flawed design is present.
    - **Drawbacks:**
        - Challenges in representing new data.
        - Workarounds using null values may lead to less desirable solutions.

Challenges in Design Choices

1. **Avoiding Poor Designs is Insufficient:**

- Multiple good design options might be available, requiring careful selection.

2. **The Complexity of Design Choices:**
   - Different design approaches can influence the effective modeling of business operations.
3. **Impact of Design Choices:**
   - Choices can significantly affect how various entities and relationships are represented.
4. **Combination of Science and Aesthetic Judgment:**
   - Database design requires a mix of scientific principles and intuitive decision-making.
5. **Scale of Decision-Making:**
   - Numerous entities and relationships demand thoughtful consideration.

Database Engine

Modular Structure of a Database System:

- A database system comprises modules handling various responsibilities.
- Functional components include:
  - **Storage Manager**
  - **Query Processor**
  - **Transaction Management**

Storage Manager

Role and Tasks:

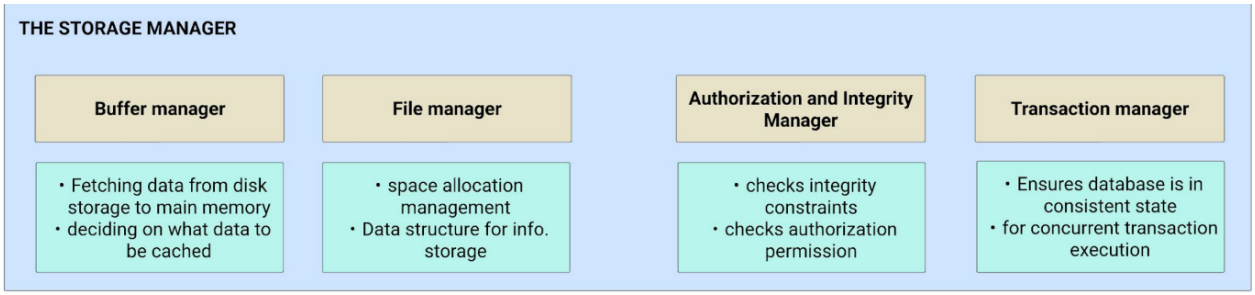
- **Interface Function:**
  - **Mediates between low-level database data and application programs/queries.**

- **Responsibilities:**
  - **Interaction with OS file manager.**
  - **Efficient storage, retrieval, and updating of data.**

Implemented Data Structures:

- **Data Files:**
  - Store the database itself.
- **Data Dictionary:**
  - Stores metadata about the database structure, including the schema.
- **Indices:**
  - Provide fast access to data items.
  - Pointers to items holding specific values.

Storage Manager Components:



Query Processor

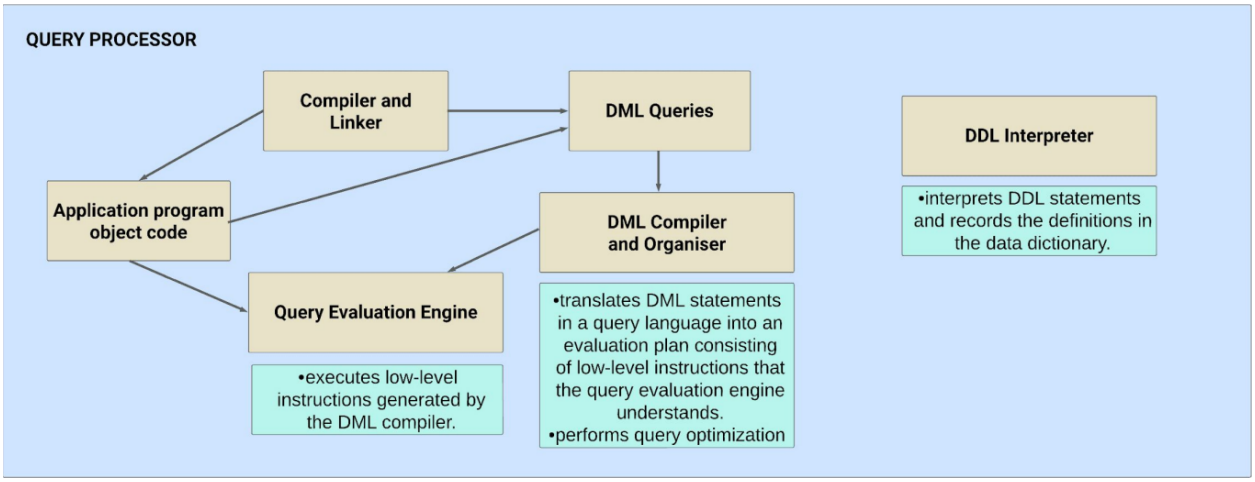
Importance and Components:

- **Simplifying Data Access:**
  - Assists the database system in simplifying and facilitating data access.
- **Components:**
  - **DDL Interpreter**
  - **DML Compiler:**

- Performs query optimization, selecting the lowest-cost evaluation plan.

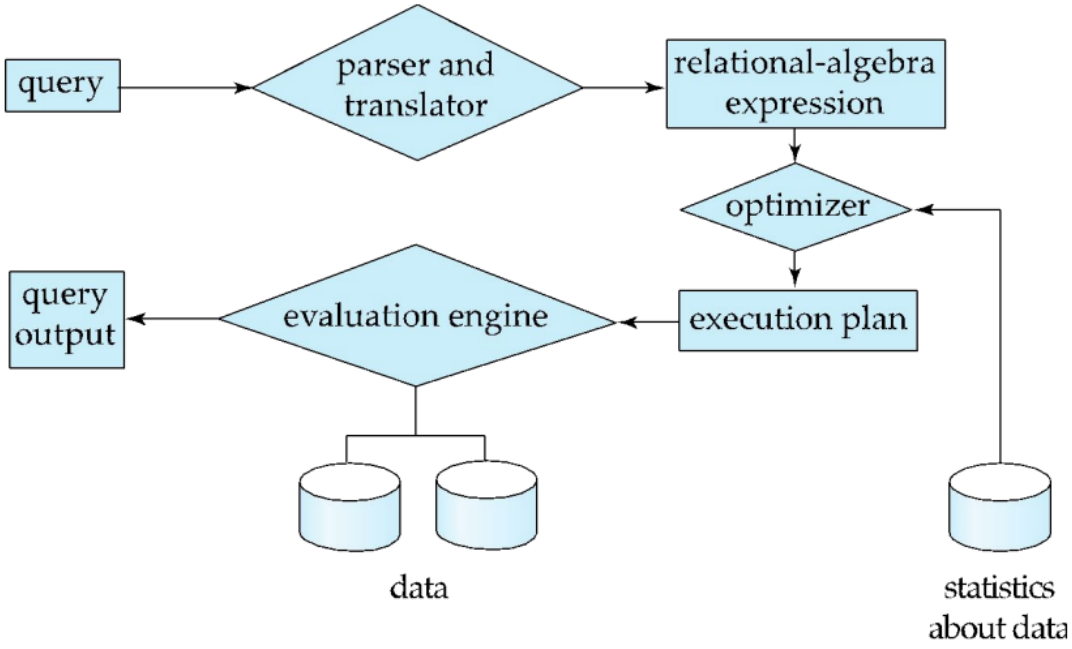
- **Query Evaluation Engine**

Internal Structure of Query Optimizer:



Query Processing Phases:

1. **Parsing and Translation**
2. **Optimization**
3. **Evaluation**



Transaction Management

Transaction Definition:

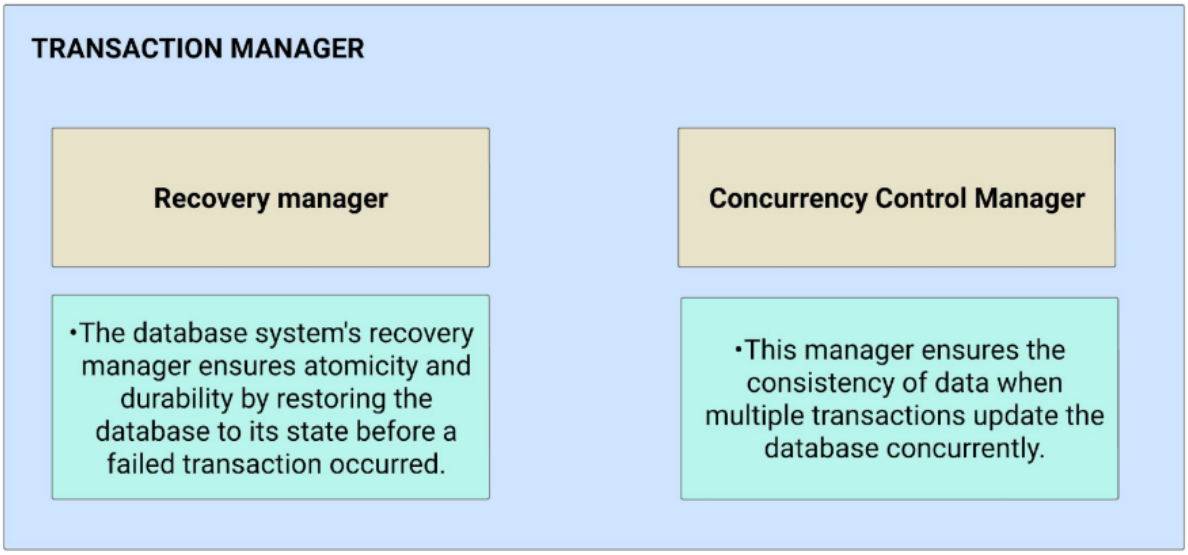
- A collection of operations performing a single logical function in a database application.
- Ensures database consistency despite system and transaction failures.

Transaction Properties:

- **Atomicity:**
  - All-or-none execution.
- **Consistency:**
  - Database remains valid throughout the transaction.
- **Durability:**
  - Changes are permanent even after system failure.

Transaction Management Component:

- Ensures consistency and durability.
- Manages transactions widely used in financial, telecommunication, and other applications.
- **Transaction Manager:**
    - Application of transactions in various domains, maintaining consistency and atomicity.



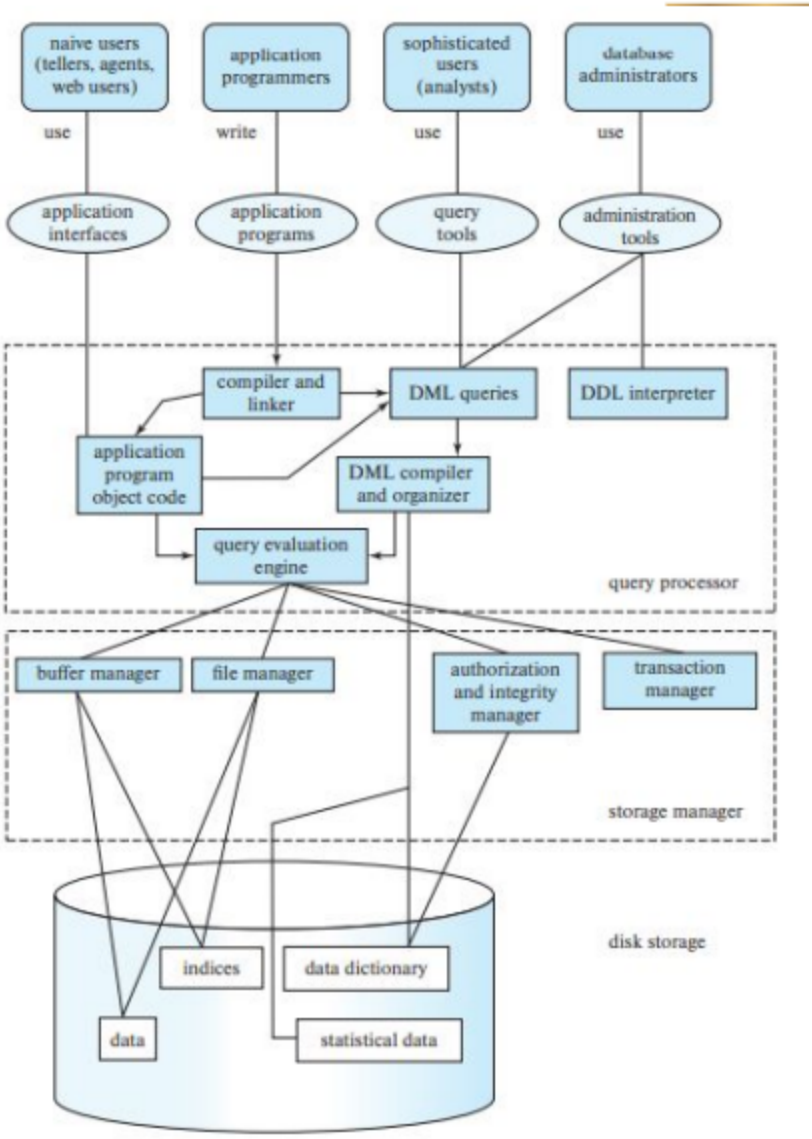
- **Storage Manager:**
    - **Facilitates interaction between the database and applications.**
    - **Implements data structures like data files, data dictionary, and indices.**
    - **Comprises components like Authorization and Integrity Manager, Transaction Manager, File Manager, and Buffer Manager.**
- **Query Processor:**
    - **Aids in simplified data access for users.**
    - **Components include DDL Interpreter, DML Compiler, and Query Evaluation Engine.**
    - **Features a query optimizer with parsing, translation, optimization, and evaluation phases.**

- **Transaction Management:**
  - **Manages transactions to ensure consistency and durability.**
  - **Properties include atomicity, consistency, and durability.**
  - **Transaction Manager oversees the application of transactions in various domains.**

Database Architecture

Centralized Databases (Centralized Architecture)

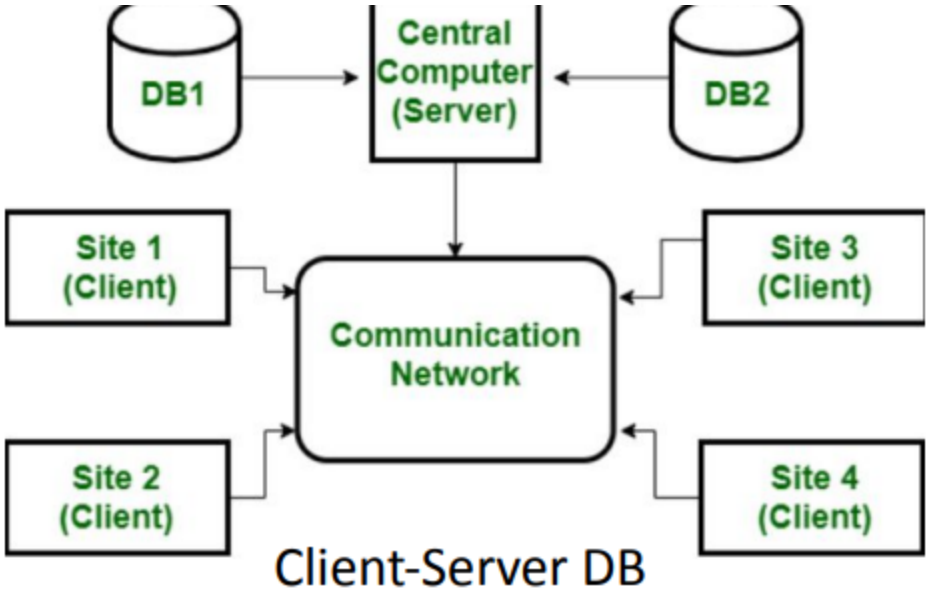
- **Characteristics:**
  - Suitable for shared-memory server architectures with multiple CPUs.
  - Limited scalability for larger data volumes and higher processing speeds.
  - Typically one to a few cores with shared memory.
- **Architecture Type:**
  - Centralized/Shared Memory Database



Centralized/Shared Memory DB

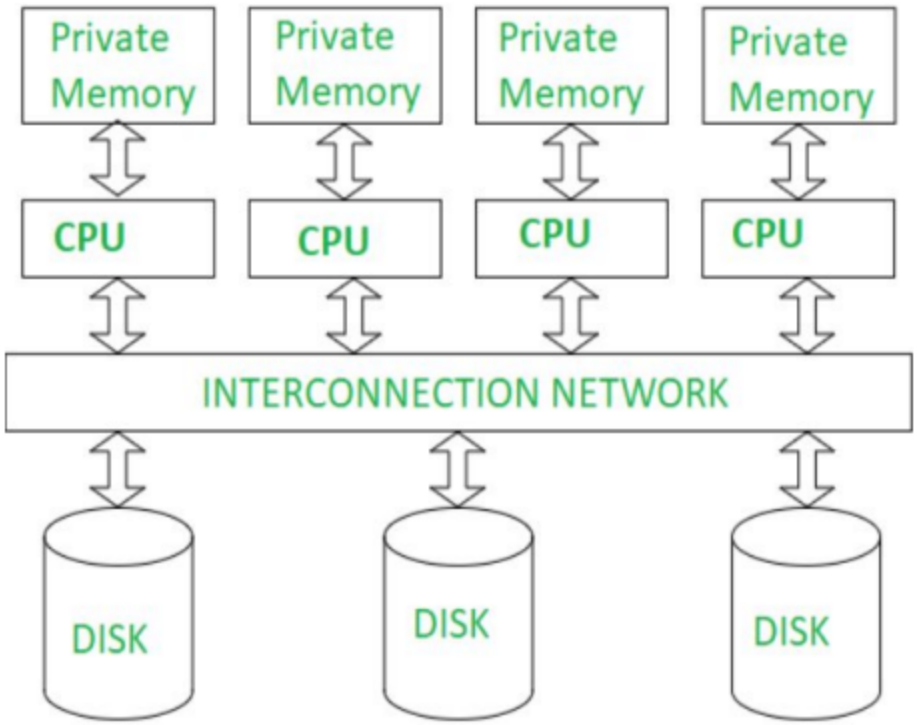
Client-Server Architecture

- **Characteristics:**
  - One server machine executes work for multiple client machines.
- **Architecture Types:**
  - Client-Server Database
  - Shared Memory Database



Parallel Databases

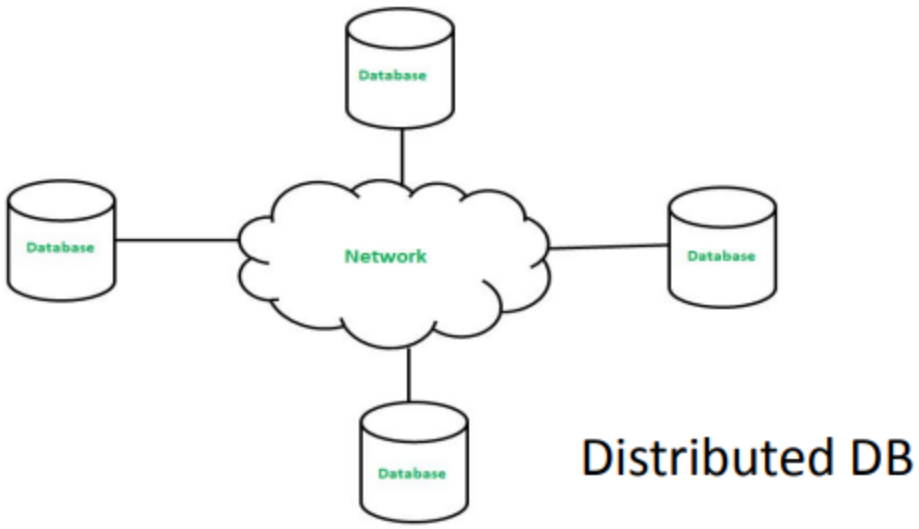
- **Characteristics:**
  - Designed for clusters of multiple machines.
  - Enables better scalability and higher processing capabilities.
  - Involves many cores with shared memory and shared disk.
- **Architecture Type:**
  - Parallel Database



Parallel DB

Distributed Databases

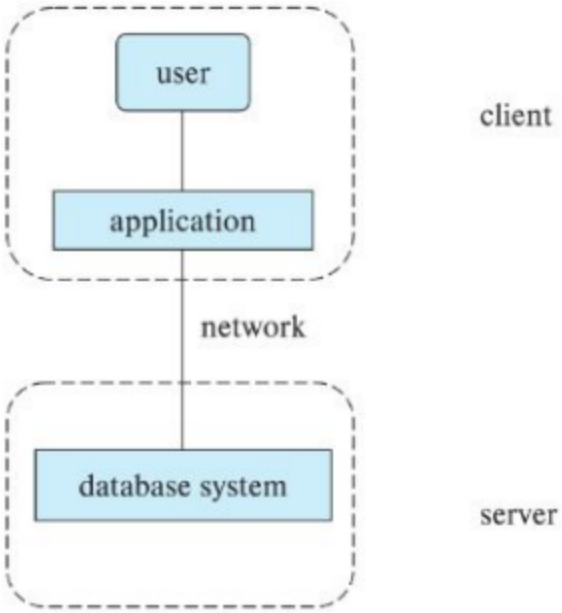
- **Characteristics:**
  - Allows data storage and query processing across geographically separated machines.
  - Facilitates large-scale data management with schema and data heterogeneity.
- **Architecture Type:**
  - Distributed Database



Application Architectures

Two-Tier Architecture

- **Overview:**
  - Earlier-generation database applications used a two-tier architecture.
  - Application resides on the client machine and invokes database system functionality on the server machine through query language statements.
  - Direct communication between the client and the database server.

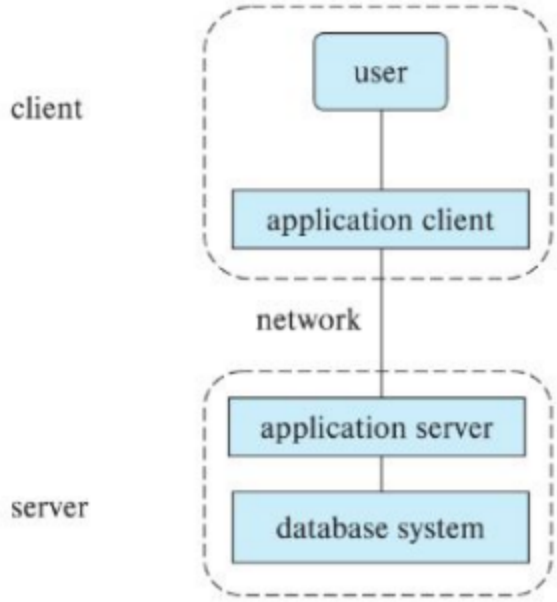


Two Tier Architecture

Three-Tier Architecture

- **Overview:**
  - Modern database applications use a three-tier architecture.
  - Client machine acts as a front end without direct database calls; common with web browsers and mobile applications.
  - **Components:**
    - **Front End:**
      - Communicates with the application server.
    - **Application Server:**
      - Communicates with the database system to access data.
      - Embeds the business logic of the application.
  - **Advantages:**

- Enhanced security and performance compared to two-tier applications.



Three Tier Architecture

Conclusion

- **Database Architecture:**
  - Centralized, client-server, parallel, and distributed architectures cater to different scalability and processing needs.
  - Each architecture type has unique characteristics and use cases.
- **Application Architectures:**
  - Two-tier architecture involves direct client-database communication.
  - Three-tier architecture separates the front end, application server, and database system, providing improved security and performance.

Database Users

Types of Database Users

1. **Naïve Users:**
    - **Description:**
        - Inexperienced users who **interact with the system** through predefined interfaces like **web or mobile applications.**
    - Often use forms to input information.
    - **Example:**
        - A student registering for a class through a web application by filling out a form.
2. **Application Programmers:**
    - **Description:**
        - Skilled **computer professionals who write application programs.**
        - Have tools to develop custom user interfaces.
    - **Example:**
        - A programmer creating a new mobile app with a custom interface for user interaction.
3. **Sophisticated Users:**
    - **Description:**
        - Users who interact with the system without writing code.
        - **Utilize database query languages or data analysis software for requests and data exploration.**
    - **Example:**
        - An analyst querying a database to retrieve specific information for a report without coding.

Database Administrators(DBA)

- **Description:**

- Individuals with **central control over both data and programs accessing the data.**

- **Functions:**

1. **Schema Definition:**
    - Creation of the original database schema using Data Definition Language (DDL).
2. **Storage Structure and Access-Method Definition:**
    - Specification of parameters for data physical organization and index creation.
3. **Schema and Physical-Organization Modification:**
    - Implementation of changes to meet organization needs or enhance performance.
4. **Granting Authorization for Data Access:**
    - Regulation of user access by granting various types of authorization.
5. **Routine Maintenance:**
    - Periodic backups, ensuring disk space availability, and monitoring database performance.

Relational Model

Thus, in the relational model:
- the term **relation** is used to refer to a **table**
- The term **tuple** is used to refer to a **row**.
- The term **attribute** refers to a **column** of a table.

ID	*name*	*dept_name*	*salary*
10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	60000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	62000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

Record or Tuple

Instructor table or Relation

- A specific instance of a relation, which includes a particular set of rows, is referred to as a **Relational Instance.**
- Therefore, For each attribute of a relation, there is a set of permitted values, called the **Domain of that attribute.**
- To maintain simplicity and consistency in the database, we require that all domains for attributes in any **relation be Atomic**
- **Null values** can cause challenges when accessing or updating the database. They may lead to issues in calculations, comparisons, and querying operations.
- It's thus best to **eliminate null values whenever possible to maintain data consistency and integrity.**

Relational Schema

- The schema is a logical design of the database
- In general, a relation schema consists of a list of attributes and their corresponding domains.
- Whereas the database instance Is the snapshot of the data in the database at a given instance of time

Constraints in Database Management Systems

1. Inherent or Implicit Constraints:

- **Description:**
  - Based on the data model itself.
  - Example: Relational model prohibits using a list as a value for any attribute.

2. Schema-based or Explicit Constraints:

- **Description:**
  - Expressed in the schema using model-specific facilities.

- Example: Maximum cardinality ratio constraint in the Entity-Relationship (ER) model.

3. Application-based or Semantic Constraints:

- **Description:**
  - Beyond the model's expressive power, specified and enforced by application programs.
  - Example: Business rules or logic that cannot be expressed within the data model.

Types of (Explicit/Schema-based) Constraints in the Relational Model:

1. Domain Constraints:

- **Description:**
  - **Specifies that values of each attribute must be atomic and belong to the attribute's domain.**
  - Example: Defining data types such as integers, characters, booleans, etc.
  - Constraints are easily tested during data entry.

2. Key Constraints:

- **Description:**
  - **Ensure tuples within a relation are distinct.**
  - Superkeys, candidate keys, and primary keys are involved.
  - Example: Defining unique identifiers (primary keys) for entities.

3. Entity Integrity Constraints:

- **Description:**
  - **Primary key attributes cannot have null values in any tuple.**
  - **Ensures the uniqueness of primary key values.**
  - Example: Ensuring every student has a unique student ID.

4. Referential Integrity Constraints:

- **Description:**
  - **Specifies relationships between relations using foreign keys.**
  - **Values in foreign key columns must exist in the referenced primary key.**
  - Example: Ensuring department information is consistent in related tables.

Key Constraints:

1. Superkey:

- **Set of attributes allowing the unique identification of a tuple.**
- Example: {ID} is a superkey for the "instructor" relation.

2. Candidate Key:

- **Minimal superkey; no proper subset is a superkey.**
- Example: {ID} is a candidate key for the "instructor" relation.

3. Primary Key:

- **Chosen from candidate keys; uniquely identifies tuples.**
- Example: SerialNo chosen as the primary key for the "CAR" relation.

Entity Integrity Constraint:

- **Ensures primary key attributes cannot contain null values.**
- Example: No student tuple can have a null value in the student ID field.

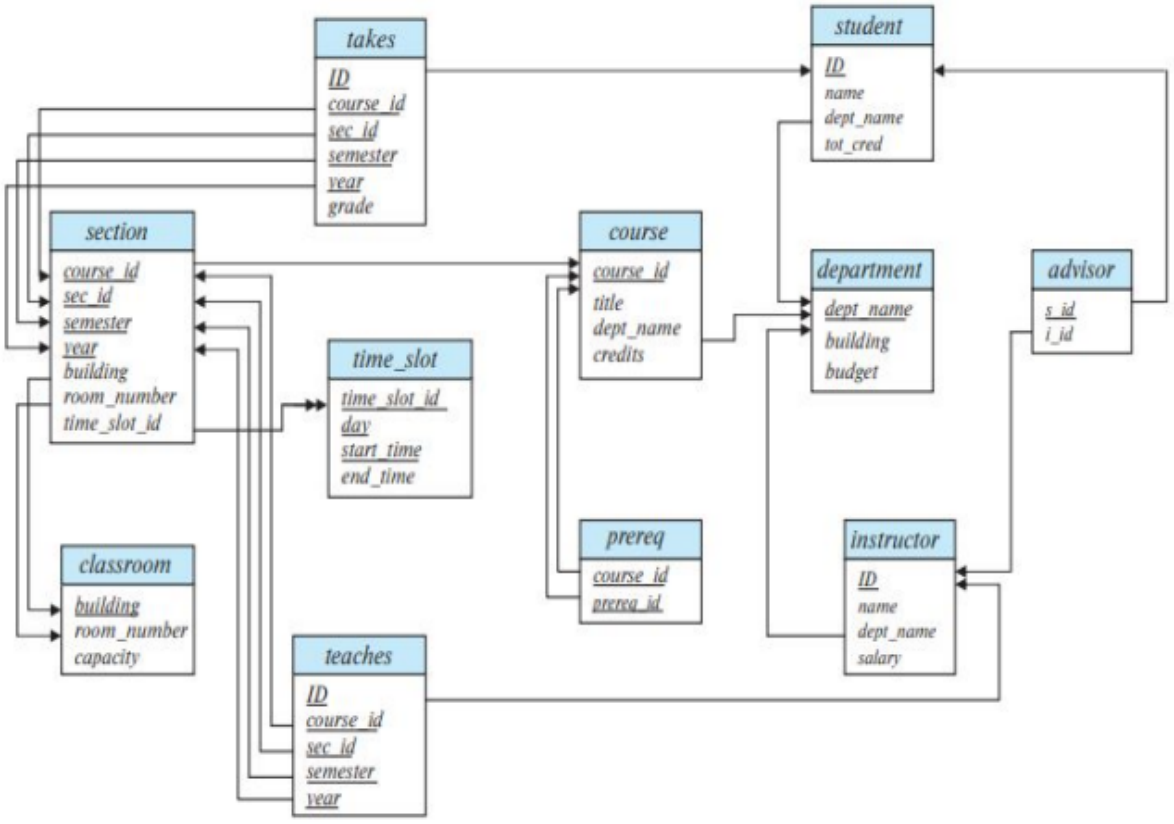
Referential Integrity Constraint:

- **Specifies relationships between relations using foreign keys.**
- Example: Ensuring that every department ID in the "instructor" relation references a valid department ID in the "department" relation.

Schema Diagrams:

- Visual representation of the database schema, primary keys, and foreign-key constraints.

- Example: A two-headed arrow indicates a referential integrity constraint in schema diagrams.

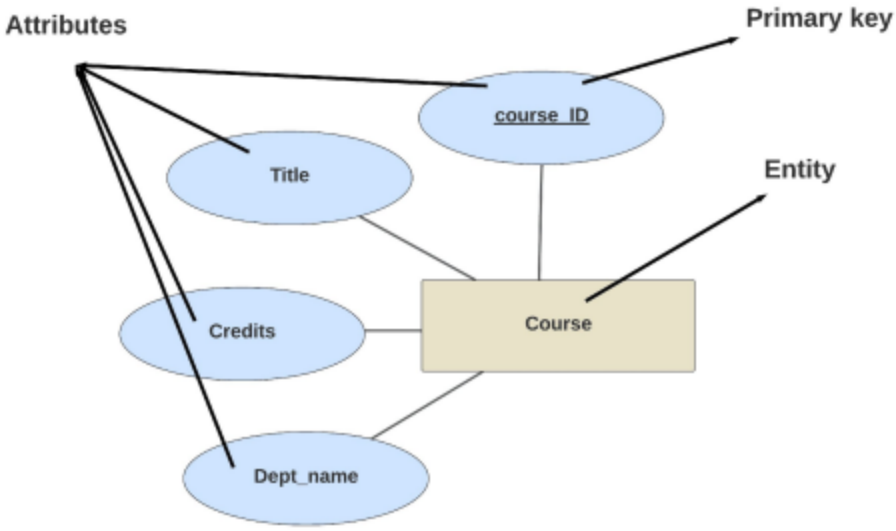


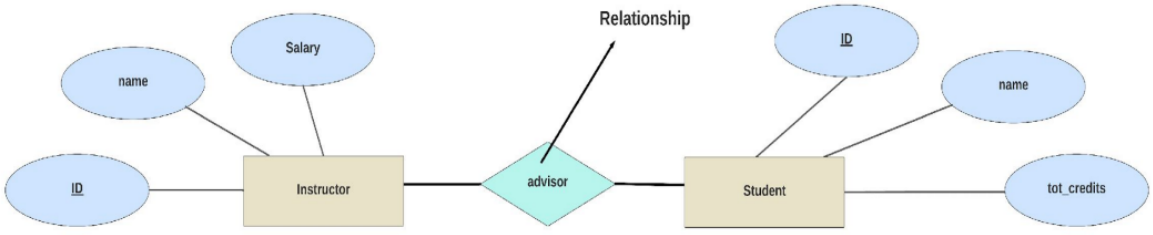
ER Model

- Entity Sets: Represent groups of similar entities from the real world. Each **entity set corresponds to a table in a relational database.**
  - Some **properties must hold unique values** to tell entities apart.
  - An entity set is a group of entities of the same kind **sharing common attributes** or properties.
- Relationship Sets: **Capture associations between different entity sets.** These relationships establish connections between data elements.

- A relationship instance in an E-R schema represents an association between the named entities in the real-world enterprise that is being modeled.
-

- Attributes: Describe characteristics or properties of entities or relationships, helping to define their attributes and properties.
  - Attributes are qualities or characteristics of each entity.
  - They signify the type of information stored for each entity.



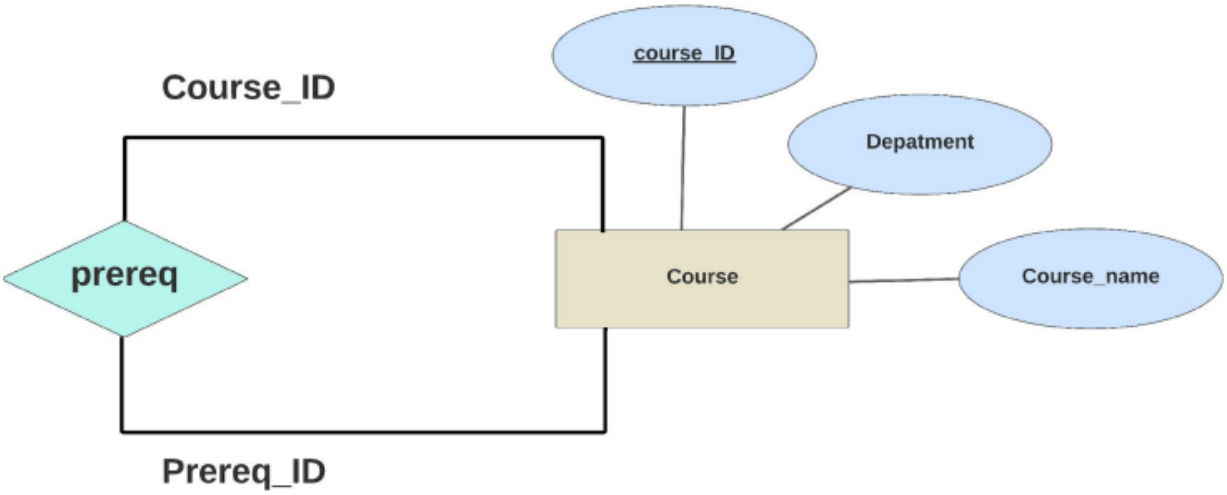


Attribute Types

1. **Recursive relationship**

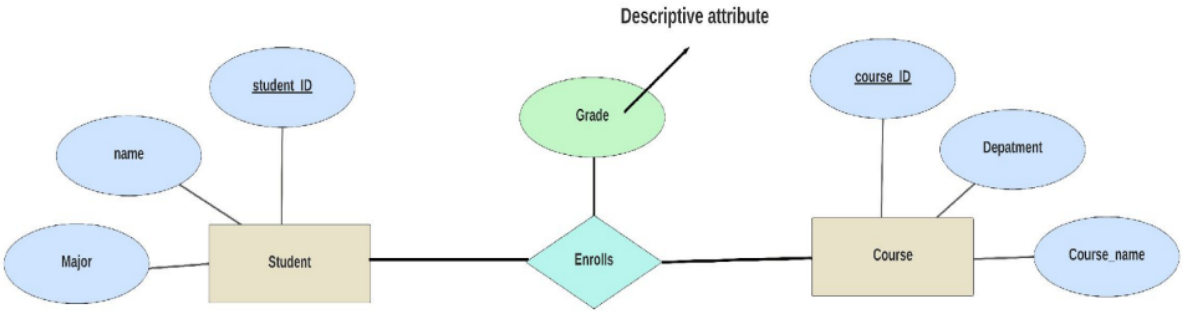
A recursive relationship in the context of the Entity-Relationship (E-R) model in database design refers to a situation where an entity set participates more than once in a relationship set, but in different roles. In other words, an entity is related to other entities of the same set through the same relationship type.

Recursive relationships are common in scenarios where entities of the same type have hierarchical or hierarchical-like relationships within the same set



2. **Descriptive Attribute**

Descriptive attributes are attributes that describe or characterize a relationship between entities, not the entities themselves.

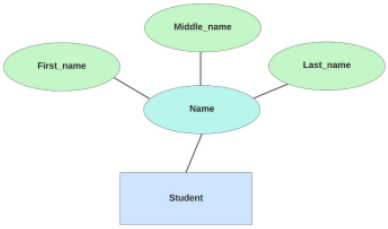


Binary Set ：Two entities and a relationship

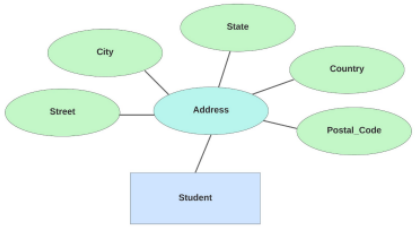
Terenery Set : Three Entities and a relationship

3. **Single Valued Attribute : Cannot be divided more**
4. **Composite Attribute : Can be divided more**

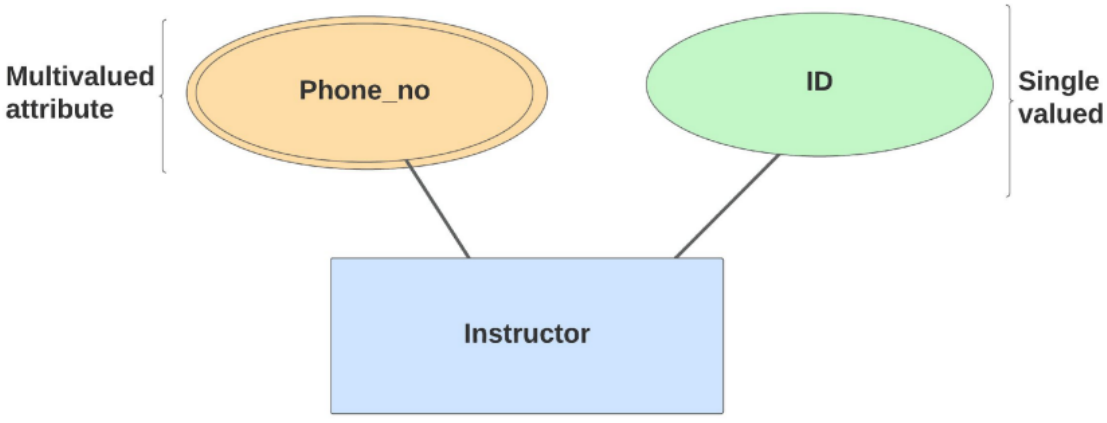
- Name could be subdivided into First Name, Middle Name, Last Name



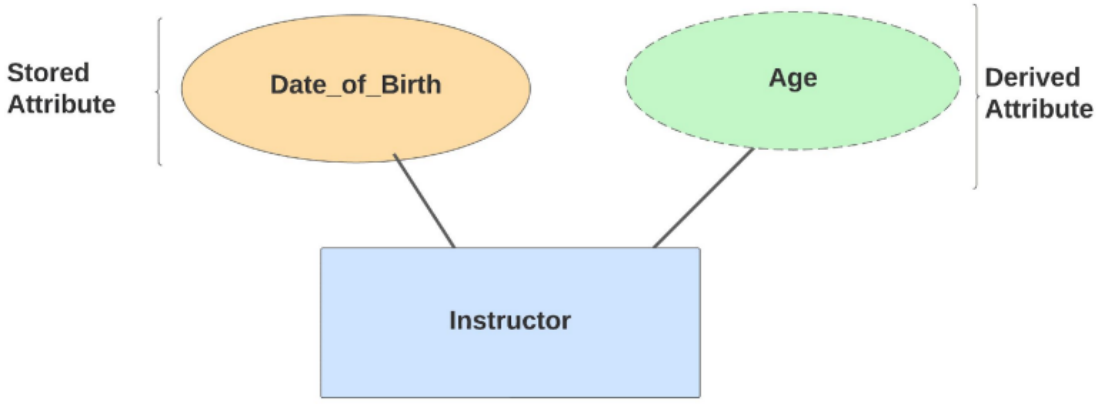
- The address can be further split into street, city, state, country, and pin code



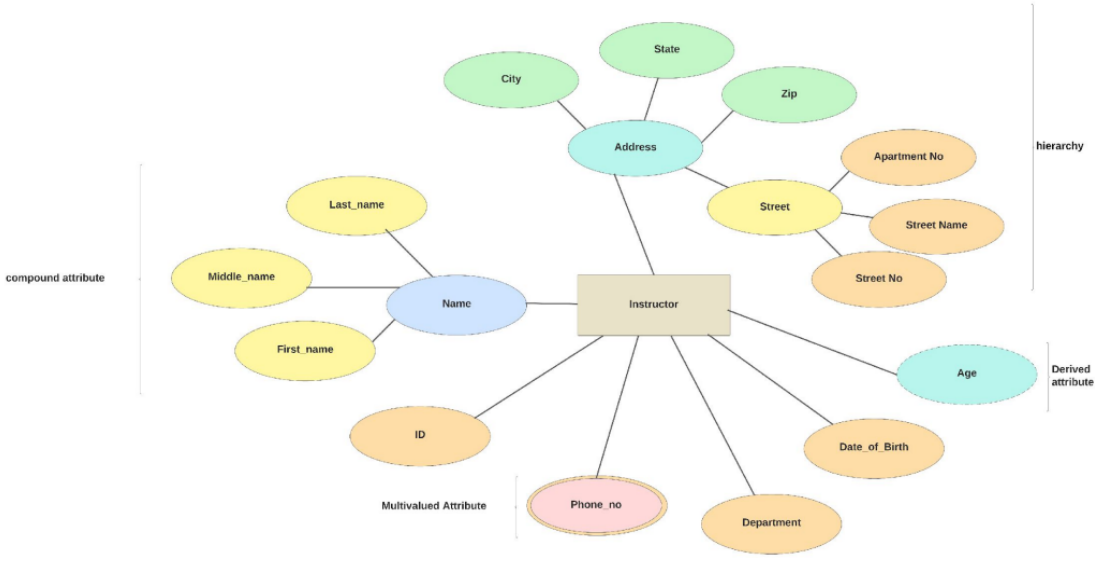
5. **Multivalued Attribute : Can have multiple values for same attribute**



6. **Derived Attribute** : Can be derived from other attributes



7. **Complex Attribute** : Multivalued and Composite
8. **Null Valued** : Left Blank



Binary Relationship Constraints

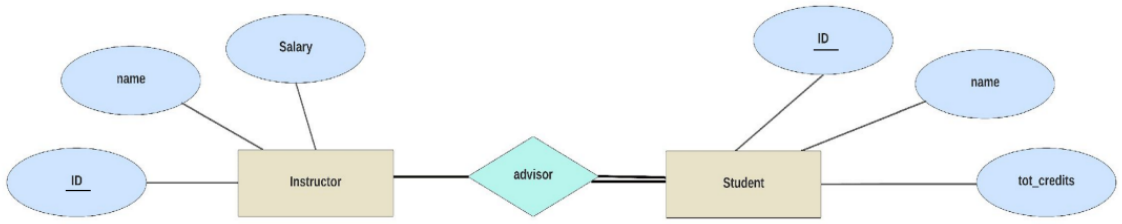
Relationship types usually have certain constraints that limit the possible combinations of entities that may participate in the corresponding relationship set

1. **Cardinality ratio**

Number of entities other entites can be associated with

- **One-to-one**
  - An entity in A is associated with at most one entity in B, and an entity in B is associated with at most one entity in A.
- **One-to-many**
  - An entity in A is associated with any number (zero or more) of entities in B. An entity in B, however, can be associated with at most one entity in A.
- **Many-to-one**
  - An entity in A is associated with at most one entity in B. An entity in B, however, can be associated with any number (zero or more) of entities in A.
- **Many-to-many**
  - An entity in A is associated with any number (zero or more) of entities in B, and an entity in B is associated with any number (zero or more) of entities in A

2. **Participation Constraints and Existence Dependencies.**

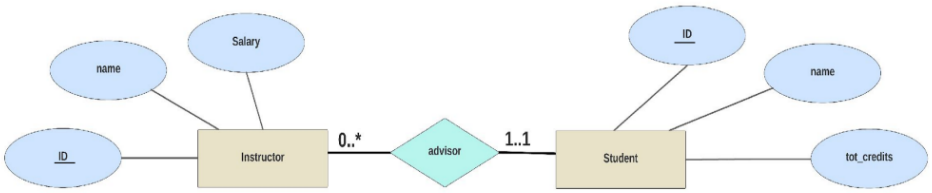


Total Participation : No empty relationship (-)

Partial Participation : Some entites dont participate (=)

Max and Min Cardinality :

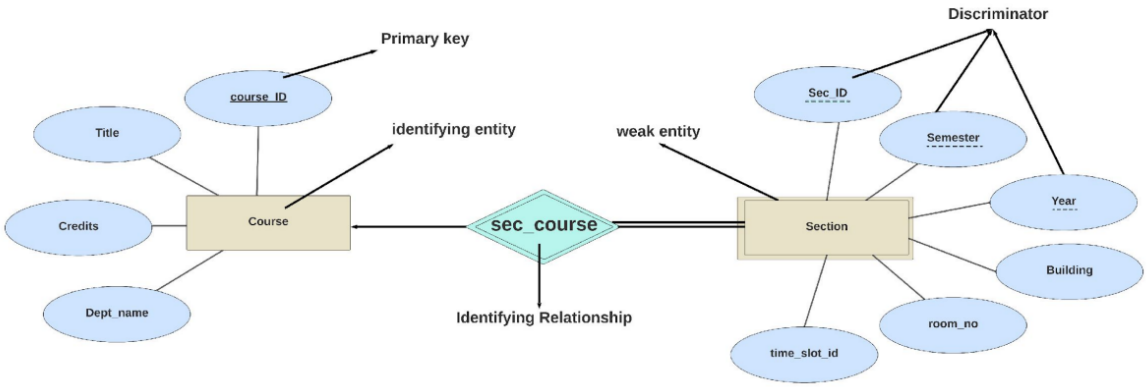
- Could you try interpreting what would this mean

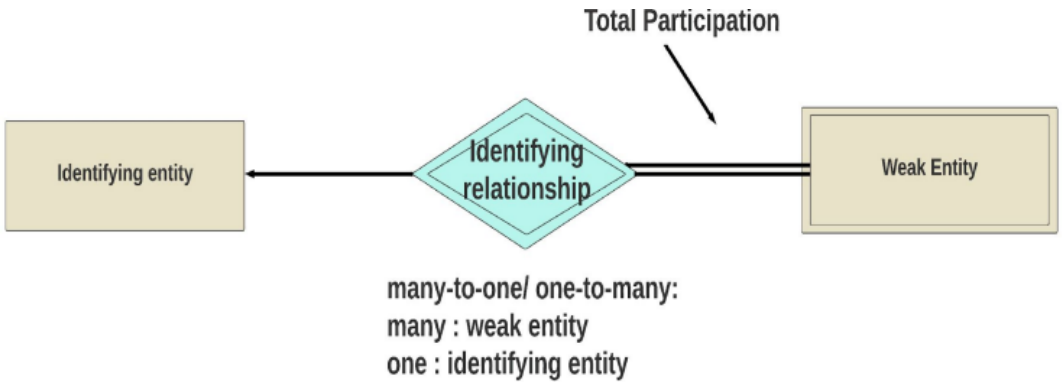


- The interpretation would be:
  - The line between advisor and student has a cardinality constraint of 1..1, meaning the minimum and the maximum cardinality are both 1. That is, **each student must have exactly one advisor.**
  - The limit 0.. * on the line between advisor and instructor indicates that an instructor can have zero or more students. Thus, the **relationship advisor is one-to-many from instructor to student**,
  - further the **participation of student in advisor is total**, implying that a student must have an advisor.

Weak Entity

- **Dependence on Another Entity Set:** Existence dependent on an identifying entity set.
- **Identifying Relationship:** Many-to-One from weak entity set to identifying entity set, with total participation.
- **Discriminator Attributes:** Extra attributes for unique identification.





Conclusion

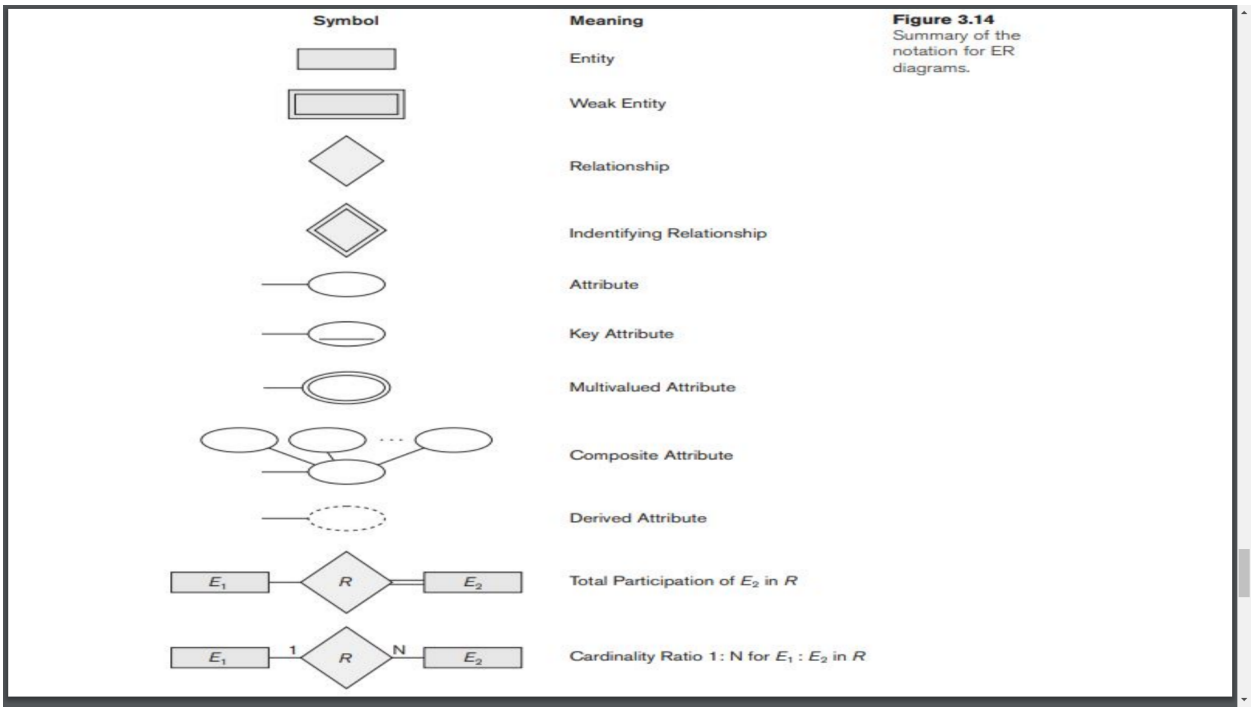
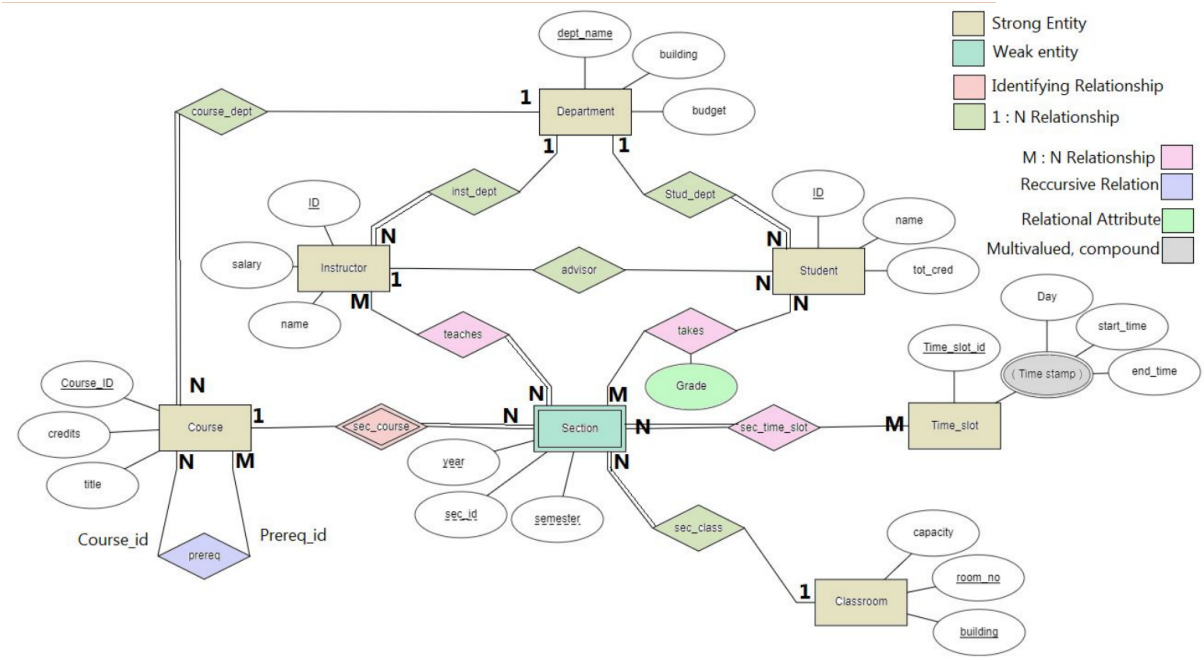
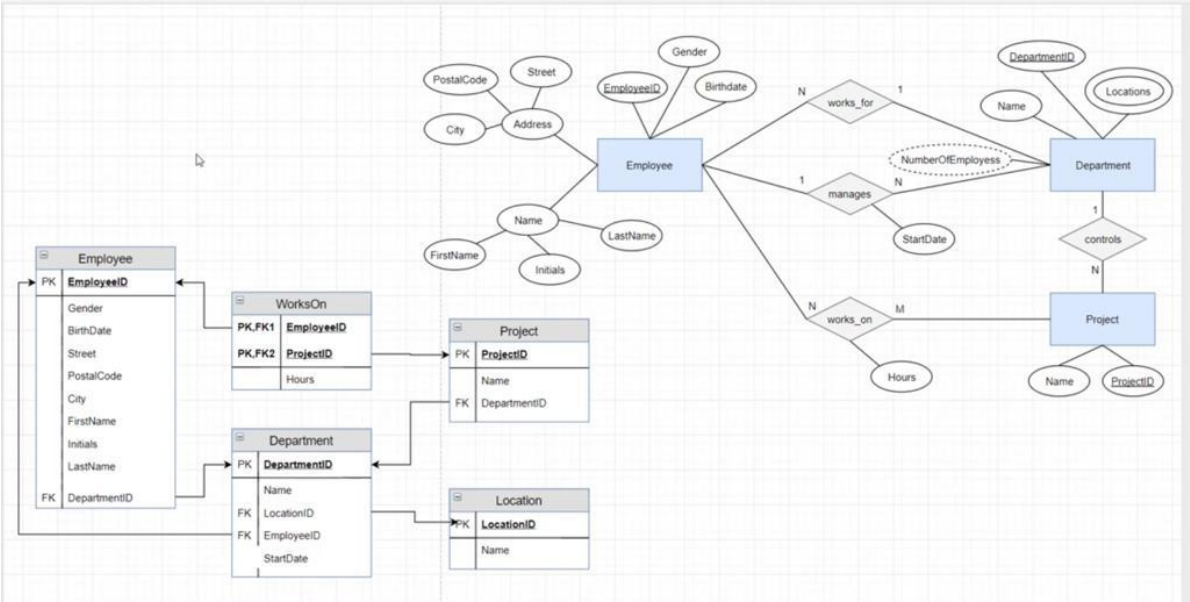


Figure 3.14
Summary of the notation for ER diagrams.

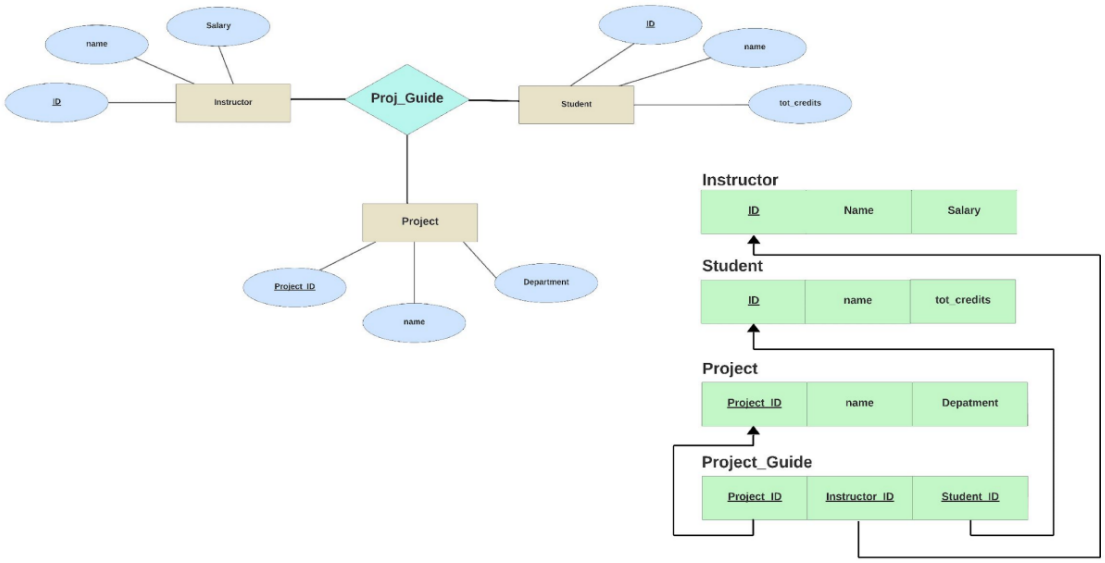


ER Diagram to Relation Schema

- **Multivalued attributes need new table** and should be foreign key in entity it is multivalued in
- **Many to many need new table** with primary keys of both entities as foreign Key plus the attributes if relationship has any
- 1-N primary key of 1 goes to table of N as Foriegn Key
  - Even the relationship attribute goes to the N side not as foriegn Key



for N-ary Relationship



Common Mistakes in ER Diagrams:

- **Incorrect Use of Primary Key:** Don't use the primary key of one entity as an attribute of another. Instead, use relationships to make connections explicit.
- **Unnecessary Inclusion of Primary Key Attributes:** Don't include primary key attributes of entities in the relationship set. They are already implied.
- **Misuse of Single-Valued Attributes:** Be careful when dealing with single-valued attributes. Ensure they represent the correct information.

Managing Large E-R Diagrams:

- **Divide and Conquer:** Large diagrams should be divided into manageable parts. Attributes of an entity should only be shown once to avoid redundancy.

Use of Entity Sets Vs. Attributes:

- **Entity Sets Vs. Attributes:** Decide whether something should be an entity on its own or just an attribute of another entity. For instance, a phone might be an entity if instructors can have multiple phones.

Binary Vs. n-ary Relationship Sets:

- **Binary Vs. n-ary Relationships:** Consider whether a relationship involves two entities or more. Sometimes, complex relationships can be broken down into simpler binary relationships.