**View of Data**

A database system is a collection of interrelated data and a set of programs that allow users to access and modify these data. A major purpose of a database system is to provide users with an abstract view of the data. That is, the system hides certain details of how the data are stored and maintained.

**Data Abstraction**

For the system to be usable, it must retrieve data efficiently. The need for efficiency has led designers to use complex data structures to represent data in the database. Since many database system users are not computer trained, developers hide the complexity from users through several levels of abstraction, to simplify users’ interactions with the system:

***• Physical level.*** The lowest level of abstraction describes how the data are actually stored. The physical level describes complex low level data structures

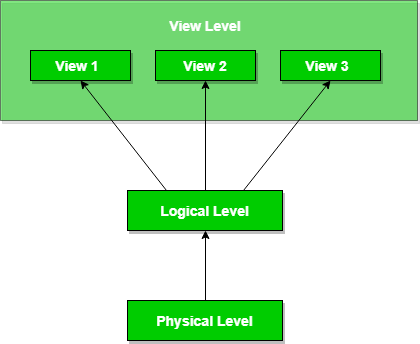
in detail.

***• Logical level.*** The next higher level of abstraction describes what data are stored in the database, and what relationships exist among those data. The logical level thus describes the entire database in terms of a small number of relatively simple structures. Although implementation of the simple structures at the logical level may involve complex physical level structures, the user of the logical level does not need to be aware of this complexity. This

is referred to as physical data independence. Database administrators, who must decide what information to keep in the database, use the logical level of abstraction.

***• View level***. The highest level of abstraction describes only part of the entire database. Even though the logical level uses simpler structures, complexity remains because of the variety of information stored in a large database.Many users of the database system do not need all this information; instead, they need to access only a part of the database. The view level of abstraction exists to simplify their interaction with the system. The system may provide many views for the same database.

Figure shows the relationship among the three levels of abstraction



**Instances and Schemas**

Databases change over time as information is inserted and deleted. The collection of information stored in the database at a particular moment is called an *instance* of the database. The overall design of the database is called the *database schema.*

Database systems have several schemas, partitioned according to the levels of abstraction. The physical schema describes the database design at the physical level, while the logical schema describes the database design at the logical level. A database may also have several schemas at the view level, sometimes called subschemas, that describe different views of the database.

**Data Models**

Underlying the structure of a database is the data model: a collection of conceptual tools for describing data, data relationships, data semantics, and consistency constraints. A data model provides a way to describe the design of a database at the physical, logical, and view levels.

There are a number of different data models that we shall cover in the text. The data models can be classified into four different categories:

***• Relational Model.*** The 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. 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.

***• 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

***• Object Based Data Model***. Object oriented programming (especially in Java, C++, or C#) has become the dominant software development methodology. This led to the development of an object oriented data model that can be seen as extending the E R model with notions of encapsulation, methods (functions), and object identity. The object relational data model combines features of the object oriented data model and relational data model.

***• Semistructured Data Model.*** The semistructured 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. The Extensible Markup Language (XML) is widely used torepresent semistructured data

**Database Languages**

A database system provides a data definition language to specify the database schema and a data manipulation language to express database queries and updates. In practice, the data definition and data manipulation languages are not two separate languages; instead they simply form parts of a single database language, such as the widely used SQL language.

**Data Manipulation Language**

A data manipulation language (DML) is a language that enables users to accessor manipulate data as organized by the appropriate data model. The types of access are:

• Retrieval of information stored in the database

• Insertion of new information into the database

• Deletion of information from the database

• Modification of information stored in the database

There are basically two types:

***• Procedural DMLs*** require a user to specify what data are needed and how to get those data.

***• Declarative DMLs*** (also referred to as nonprocedural DMLs) require a user to specify what data are needed without specifying how to get those data. Declarative DMLs are usually easier to learn and use than are procedural DMLs.

A *query* is a statement requesting the retrieval of information. The portion of a DML that involves information retrieval is called a query language. Although technically incorrect, it is common practice to use the terms query language and data manipulation language synonymously.

**Data Definition Language**

We specify a database schema by a set of definitions expressed by a special language called a data definition language (DDL). The DDL is also used to specify additional properties of the data.

We specify the storage structure and access methods used by the database system by a set of statements in a special type of DDL called a data storage and definition language. These statements define the implementation details of the database schemas, which are usually hidden from the users. The data values stored in the database must satisfy certain consistency constraints.

Database systems enforce integrity constraints with minimal overhead:

***Domain Constraints***: Every attribute has a defined domain (e.g., integer, date). These constraints ensure data validity and are checked when new data is entered.

***Referential Integrity***: Ensures relationships between tables remain valid. For example, a course’s department must exist in the department table. Violations result in rejected modifications.

***Assertions:*** General conditions the database must always satisfy, beyond domain and referential constraints. Example: “Each department must offer at least five courses per semester.” Modifications are only allowed if they uphold assertions.

***Authorization***: Controls user access levels, such as read, insert, update, or delete permissions, to ensure data security.

The output of the DDL is placed in the data dictionary, which contains metadata— that is, data about data. The data dictionary is considered to be a special type of table that can only be

accessed and updated by the database system itself.

**Relational Databases**

A relational database is based on the relational model and uses a collection of

*tables* to represent both data and the relationships among those data. It also includes a DML and DDL.

**Tables**

Each table has multiple columns and each column has a unique name. 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. The columns of the table correspond to the attributes of the record type.

**Data Manipulation Language**

The SQL query language is nonprocedural. A query takes as input several tables

(possibly only one) and always returns a single table. Here is an example of an SQL query that finds the names of all instructors in the History department:

select instructor.name

from instructor

where instructor.dept name = ’History’;

Queries may involve information from more than one table. For instance, the following query finds the instructor ID and department name of all instructors associated with a department with budget of greater than $95,000.

select instructor.ID, department.dept name

from instructor, department

where instructor.dept name= department.dept name and

department.budget > 95000;

Application programs are programs that are used to interact with the database

To access the database, DML statements need to be executed from the host language.

There are two ways to do this:

• By providing an application program interface (set of procedures) that can be used to send DML and DDL statements to the database and retrieve the results.

The Open Database Connectivity (ODBC) standard for use with the C language is a commonly used application program interface standard. The Java Database Connectivity (JDBC) standard provides corresponding features to the Java language.

• By extending the host language syntax to embed DML calls within the host language program. Usually, a special character prefaces DML calls, and a preprocessor, called the DML precompiler, converts the DML statements to normal procedure calls in the host language.

**Database Architecture**

The architecture of a database system depends on the underlying computer system.

**Centralized & Client Server:** A single server handles requests from multiple clients.

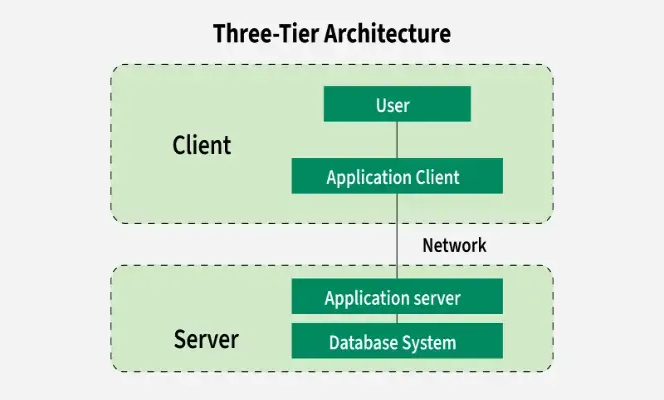
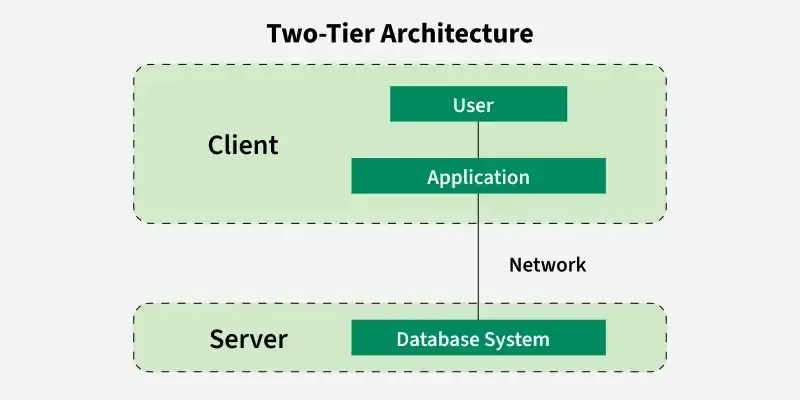
Parallel Databases: Designed to leverage multiple processors for faster performance.

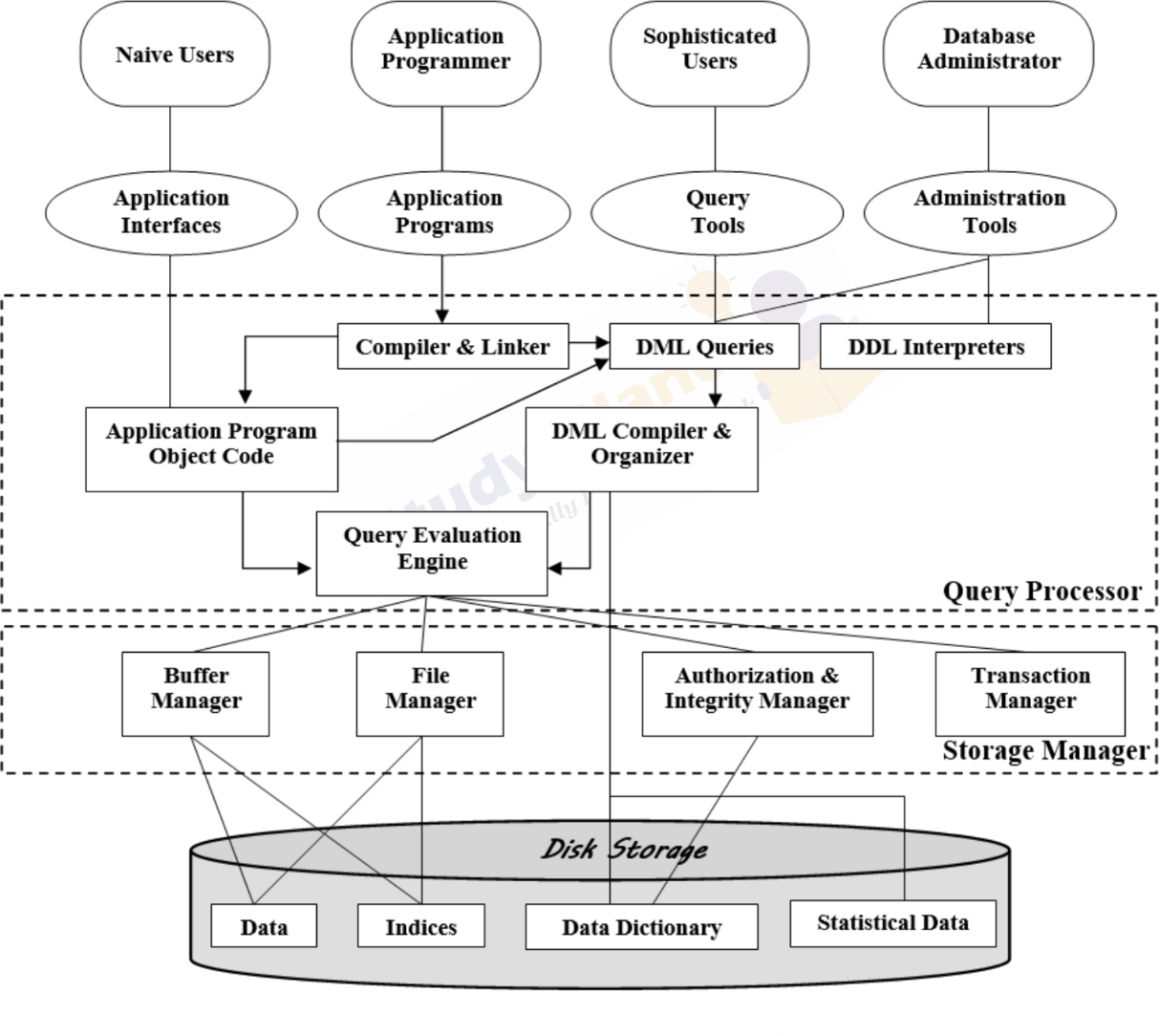
Distributed Databases: Spread across multiple geographically separated machines for scalability and redundancy.

Database applications follow a two tier or three tier architecture

***Two Tier Architecture:*** The client machine hosts the application and directly interacts with the database server using query languages (via ODBC/JDBC).

***Three Tier Architecture:*** The client acts as a front end, communicating with an application server , which handles business logic and interacts with the database. This model is better suited for large scale and web applications .





**Database Users & Administrators**

A database system facilitates data storage and retrieval , with users categorized as database users or database administrators (DBAs).

**Types of Database Users**

***1. Naïve Users***

Unsophisticated users who interact with the database through predefined applications.

Use forms interfaces to input data or retrieve reports.

Example: A university clerk uses a form to add a new instructor, or a student registers for classes via a web application.

***2. Application Programmers***

Software developers who write application programs to interact with the database.

Use Rapid Application Development (RAD) tools to create user interfaces with minimal programming.

Example: A developer creates a student registration system that connects to the university database.

***3. Sophisticated Users***

Directly interact with the database using query languages (e.g., SQL) or data analysis tools.

Typically include data analysts and researchers who explore and extract insights from data.

Example: A business analyst retrieves sales trends using complex SQL queries.

***4. Specialized Users***

Develop non traditional database applications outside of standard data processing.

Work with complex data types like graphics, multimedia, or artificial intelligence systems.

Example: Engineers working on computer aided design (CAD) or scientists managing environmental modelling databases.

**Database Administrator (DBA) Responsibilities**

A Database Administrator (DBA) manages and maintains the database system to ensure efficiency, security, and data integrity.

***1. Schema Definition & Modification***

Defines the structure of the database using Data Definition Language (DDL).

Modifies schemas and physical organization as needed (e.g., adding new tables or updating relationships).

***2. Storage Structure & Access Method Definition***

Determines how data is stored and accessed efficiently.

Optimizes database indexing, partitioning, and caching for better performance.

***3. Authorization & Security Control***

Manages user access and permissions to protect data from unauthorized access.

Defines different levels of access (e.g., read, write, update, delete).

***4. Routine Maintenance***

Backups data regularly to prevent loss due to disasters.

Monitors disk space and upgrades storage when necessary.

Optimizes database performance by managing expensive queries and preventing system slowdowns.

The DBA plays a crucial role in database management by ensuring *data integrity, security, performance, and availability*. while different types of users interact with the database in various ways, depending on their needs and expertise.

**Structure of Relational Databases**

A relational database consists of multiple tables (relations), each uniquely named. Each table contains rows (tuples) representing relationships between columns (attributes) .

***Tables & Attributes***

* Each table has a unique name and consists of multiple columns (attributes).
* Example: An instructor table has columns: `ID`, `name`, `dept\_name`, `salary`.

***Tuples & Relations***

* Each row (tuple) in a table represents a relationship among values.
* Example: A row in the prereq table links two courses where one is a prerequisite for another.

***Relation Terminology:***

* Relation = Table
* Tuple = Row
* Attribute = Column
* Relation Instance = A specific set of rows in a table.

***Order of Tuples***

The order of rows does not matter in a relation; it is considered a set of tuples .

***Domains & Atomicity***

* Each attribute has a domain (set of permitted values).
* Atomic Domains : Values must be indivisible units (e.g., `phone\_number` should not store multiple numbers in a single field).

***Null Values***

* Used when a value is unknown or missing (e.g., an instructor without a phone number).
* Can cause complications in queries and updates, so they should be avoided if possible.

Relational databases provide structured data storage , ensuring relationships between entities are well defined.

**Database Schema**

A database schema defines the logical design of a database, while a database instance is a snapshot of the data at a particular moment.

***Relation Schema vs. Relation Instance:***

* Schema : Defines the structure (attributes and domains).
* Instance : The actual data at a given time.
* Example: `department(dept\_name, building, budget)` defines the structure, while an actual table with values is an instance.

***Relations and Their Purpose:***

* Relations share attributes to establish links between tables.
* Example: `dept\_name` appears in both `instructor` and `department` to connect related data.

**University Database Example:**

Courses & Sections: Courses may have multiple offerings in different semesters.

`section(course\_id, sec\_id, semester, year, building, room\_number, time\_slot\_id)`

Instructor Assignments:

`teaches(ID, course\_id, sec\_id, semester, year)`

Other Relations:

`student(ID, name, dept\_name, tot\_cred)` → Student details

`advisor(s\_id, i\_id)` → Student advisor relationships

`takes(ID, course\_id, sec\_id, semester, year, grade)` → Course enrollments

`classroom(building, room\_number, capacity)` → Room details

`time\_slot(time\_slot\_id, day, start\_time, end\_time)` → Class timing

The schema remains fixed but instances change over time as data is added, modified, or deleted.

**Keys in a Relational Database**

To ensure that each tuple (row) in a relation (table) is unique databases use keys. Keys are attributes or sets of attributes that identify tuples uniquely.

***1. Superkey***

* A superkey is a set of attributes that uniquely identifies a tuple in a relation.
* Example: `ID` in the `instructor` table is a superkey because no two instructors share the same `ID`.

***2. Candidate Key***

* A candidate key is a minimal superkey, meaning it contains no unnecessary attributes.
* Example: In `instructor(ID, name, dept\_name, salary)`, both `{ID}` and `{name, dept\_name}` could be candidate keys if names are unique within departments.

***3. Primary Key***

* A primary key is a candidate key chosen to uniquely identify tuples in a relation
* Should be stable (values rarely change) and simple .
* Example: In `department(dept\_name, building, budget)`, `dept\_name` is the primary key.

***4. Foreign Key***

* A foreign key in one relation references the primary key of another relation.
* Ensures referential integrity (values in the referencing relation must exist in the referenced relation).
* Example: `dept\_name` in `instructor` is a foreign key referencing `department(dept\_name)`.

***5. Referential Integrity Constraint***

* Ensures that referenced data exists.
* Example: `teaches(ID, course\_id, sec\_id, semester, year)` references `section(course\_id, sec\_id, semester, year)`, ensuring sections have instructors.

Keys maintain uniqueness, integrity, and relationships within a database.

**Schema Diagrams in Databases**

A schema diagram visually represents a database schema, showing relations (tables) along with primary keys and foreign key dependencies .

***1. Structure of Schema Diagrams***

* Each relation (table) is depicted as a box with its name at the top.
* Attributes are listed inside the box.
* Primary key attributes are underlined .
* Foreign keys are represented by arrows pointing to the referenced table’s primary key.
* Referential integrity constraints (besides foreign keys) are not explicitly shown .

***2. Schema Diagram for a University Database***

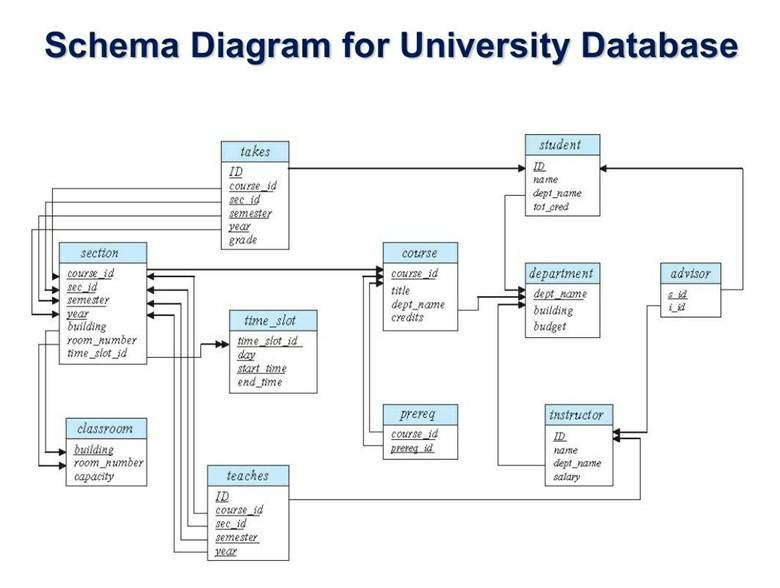
The schema diagram for a university includes tables like:

* `instructor(ID, name, dept\_name, salary)`
* `department(dept\_name, building, budget)`
* `course(course\_id, title, dept\_name, credits)`
* `section(course\_id, sec\_id, semester, year, building, room\_number, time\_slot\_id)`
* `teaches(ID, course\_id, sec\_id, semester, year)`
* ...and more.

***3. Diagramming Tools***

* Database systems often provide graphical tools to design schema diagrams.
* Another method, the Entity Relationship (ER) diagram , represents additional constraints and relationships

Schema diagrams offer a clear visualization of a database structure, showing tables, keys, and relationships .



**Relational Query Languages**

A query language allows users to retrieve information from a database. These languages operate at a higher level than standard programming languages.

**Types of Query Languages**

***1. Procedural Query Languages***

* The user specifies how to retrieve data by providing a sequence of operations.
* Example: Relational Algebra (a formal, procedural query language).

***2. Nonprocedural Query Languages***

* The user specifies what data is needed without detailing the steps to get it.
* Examples: Tuple Relational Calculus (TRC) – Uses logical conditions to specify queries.
* Domain Relational Calculus (DRC) – Defines queries using domain values rather than tuples.

**SQL and Practical Query Languages**

* SQL (Structured Query Language) is a widely used query language that combines elements of both procedural and nonprocedural approaches.
* Relational Algebra consists of operations that take one or two relations as input and return a new relation.
* Relational Calculus defines the result using predicate logic without specifying a retrieval process.

Relational query languages provide various ways to retrieve data , with SQL being the most commonly used in practice.

**Relational Operations**

Relational query languages provide operations that can be applied to one or two relations , always producing a new relation as a result. This allows queries to be combined modularly .

**Basic Operations:**

1***. Selection (σ)*** – Retrieves specific tuples (rows) from a relation that satisfy a condition.

Example: Selecting instructors with a salary > $85,000 .

2. ***Projection (π)*** – Retrieves specific attributes (columns) from a relation.

Example: Listing only ID and salary of instructors.

3***. Join (⨝)*** – Combines two relations by matching tuples based on common attributes.

Natural Join : Merges tuples with the same attribute values (e.g., matching instructors with their departments).

4. ***Cartesian Product (×)*** – Combines all tuples from two relations without matching attributes .

5. ***Set Operations:***

* Union (∪) – Combines tuples from two relations.
* Intersection (∩) – Returns common tuples.
* Set Difference ( ) – Returns tuples in one relation but not in another.

**Query Processing & Optimization:**

* Operations can be nested (e.g., selecting instructors with a high salary, then retrieving only ID and salary ).
* Order of operations can affect results .
* Some databases remove duplicate tuples automatically; others retain them for efficiency.

**Updating Relations:**

* Insertion – Adding new tuples.
* Deletion – Removing tuples.
* Modification – Changing attribute values.
* Relation Management – Creating and deleting entire relations.

Relational operations form the foundation of query processing , allowing users to manipulate and retrieve data efficiently .