By Bharadwaj Rachakonda – rbharadwaj022@gmail.com LinkedIn

Database Management System Overview

What is a Database?

A database is a collection of related data.
By data, we mean known facts that can be recorded and that have meaning.

Database Management System (DBMS)

A database management system is the storage of data in an organized manner, enabling queries to retrieve the data.

Characteristics of a Database

- **Self-describing Nature:** The database system contains a definition or description of the database structure and constraints, stored in the DBMS catalog as meta-data.
- **Insulation between Programs and Data (Data Abstraction):** The structure of data files is stored in the DBMS catalog separately from the access programs. We call this property Program-data independence. User application programs can operate on the data by invoking these operations through their names and arguments, regardless of how the operations are implemented. This may be termed program-operation independence.
- **Support for Multiple Views:** Different users can have different views of the data.
- **Data Sharing and Multiuser Transaction Processing:** Multiple users can access and modify data at the same time, supported by transaction management, concurrency control mechanisms. For example, when several reservation agents try to assign a seat on an airline flight, the DBMS should ensure that each seat can be accessed by only one agent at a time for assignment to a passenger. These types of applications are generally called online transaction processing (OLTP) applications.

Advantages of DBMS

- Redundancy control
- Normalization
- Indexing

- Flexibility
- Transaction management
- Authorization control
- Backup and recovery
- Data integrity
- Data abstraction

Three-Tier Architecture

Goal: Separate user applications from the physical database.

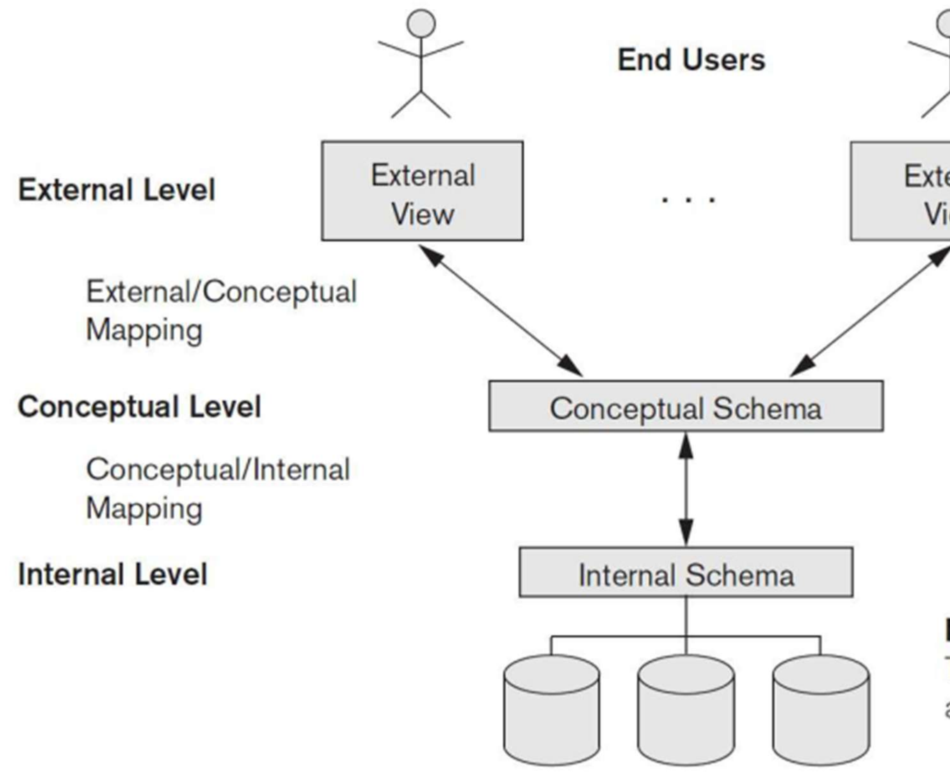


Figure 2.2 The three-schema architecture.

1. **Internal Level:**
   Has an internal schema describing the physical storage structure of the database, including record formats and access paths.

2. **Conceptual Level:**
   Has a conceptual schema describing the structure of the whole database for a community of users, focusing on entities, data types, relationships, operations, and constraints.
3. **External (View) Level:**
   Includes several external schemas, each showing the database portion relevant to a specific user group, hiding the rest.

Request Transformation:
The DBMS transforms a request on an external schema into a conceptual schema request, then into an internal schema request. Data is reformatted to the user's view as needed. These transformations are called **mappings**.

Architecture of DBMS:

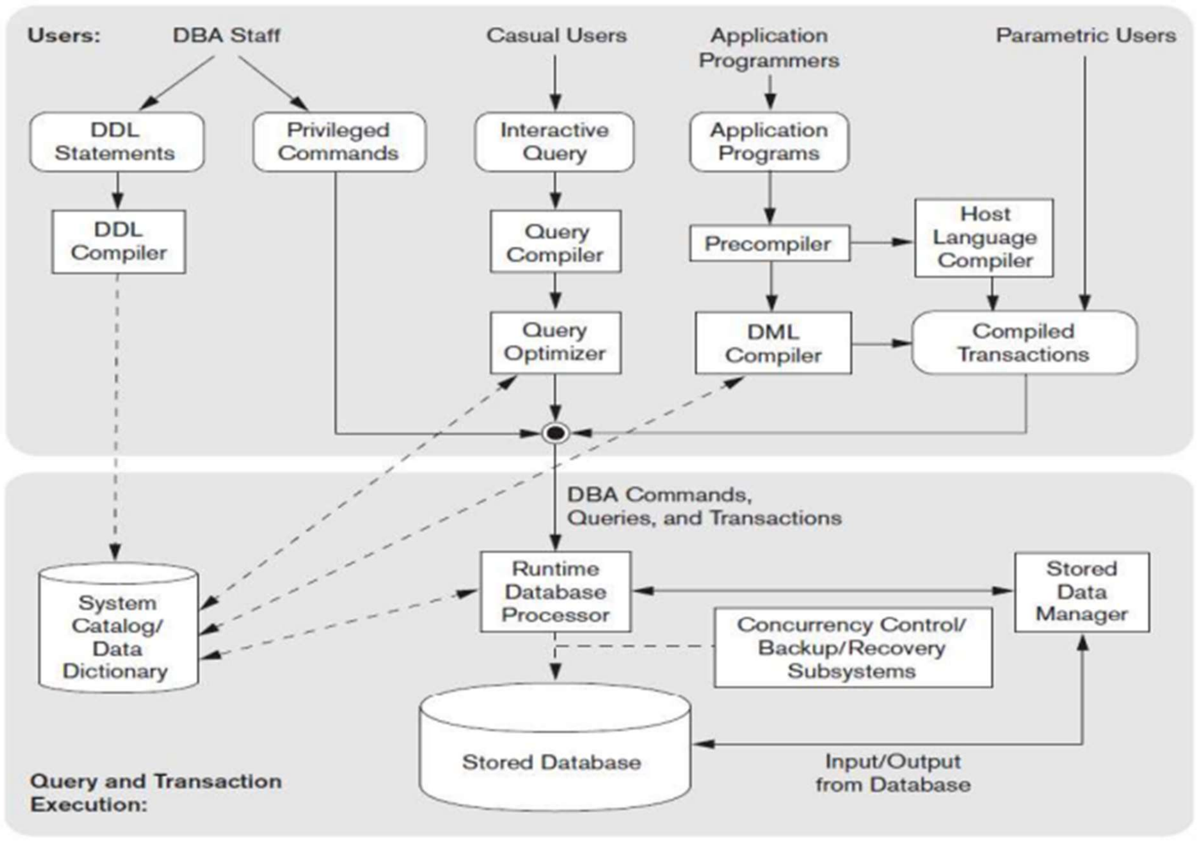


Figure 2.3 Component modules of a DBMS and their interaction

Data Independence

The ability to change the schema at one level without changing the schema at the next higher level is called **Data Independence**.

- **Logical Data Independence:** Ability to change the conceptual schema without affecting external schemas or application programs.
- **Physical Data Independence:** Ability to change the internal schema without affecting the conceptual schema.

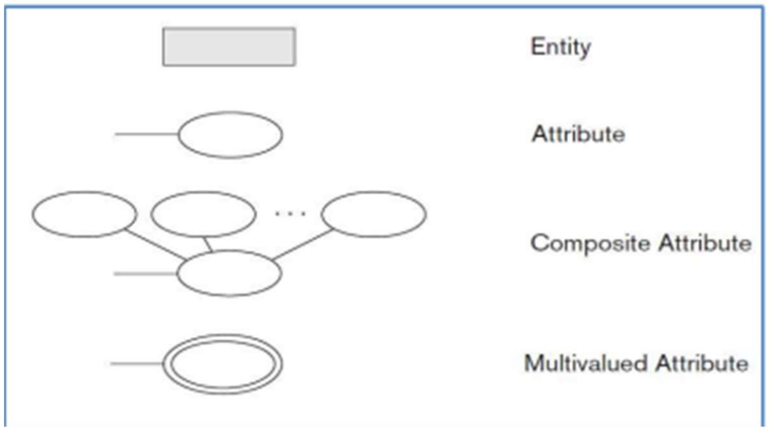
Data Models

- **Conceptual Data Models:** Use concepts such as entities, attributes, and relationships.
  - *Entity:* A real-world object or concept (e.g., employee, project).
  - *Attribute:* A property that describes an entity (e.g., employee name, salary).
  - *Relationship:* An association among two or more entities.
- **Representational Data Models:** Use record structures; also known as record-based data models.
- **Physical Data Models:** Describe how data is stored as files on the computer (record formats, orderings, access paths).

Schema and State

- **Database Schema:** The description of the database structure.
- **Database State (Snapshot):** The data in the database at a specific moment in time.

ER (Entity-Relationship) Diagrams



- **Entity:** A thing or object in the real world with an independent existence.

- **Attributes:** Properties that describe an entity.
- **Entity Type:** A set of entities with the same attributes.
- **Entity Set:** The collection of all entities of a particular type at any time.
- **Degree of Relationship Type:** The number of participating entity types.
- **Total Participation:** Also called existence dependency; a feature of *weak entities*.
- **Structural Constraints:** Defined by cardinality ratio and participation constraints.

Definition of Relational Algebra:

Relational algebra is a **formal procedural query language** used in relational databases for querying and manipulating data. These operations form the theoretical foundation for SQL and other database query languages and help in understanding query processing and optimization.

Fundamental Operations in Relational Algebra (with Examples):

Suppose you have a table Student:

ID	Name	Age	Dept
1	Alex	22	CSE
2	Riya	21	ECE
3	Mohit	23	CSE

1. Selection (σ)

Selects rows that satisfy a specific condition, like WHERE in SQL.

Example: Students in the CSE department

$$\sigma_{Dept='CSE'}(Student)$$

Result:

ID	Name	Age	Dept
1	Alex	22	CSE
3	Mohit	23	CSE

2. Projection (π)

Selects specific columns from a relation.

Example: List of names and ages

$$\pi_{Name,Age}(Student)$$

Result:

Name	Age
Alex	22
Riya	21
Mohit	23

3. Union (∪)

Combines rows from two relations with the same attributes.

Assume another table, Student2:

ID	Name	Age	Dept
2	Riya	21	ECE
4	Priya	22	IT

$$Student \cup Student2$$

Result:

ID	Name	Age	Dept
1	Alex	22	CSE
2	Riya	21	ECE
3	Mohit	23	CSE
4	Priya	22	IT

4. Set Difference (−)

Tuples in one relation but not in another.

$$Student - Student2$$

Result:

ID	Name	Age	Dept
1	Alex	22	CSE
3	Mohit	23	CSE

5. Cartesian Product (×)

Combines each row from one relation with every row of another.

6. Rename (ρ)

Renames a relation or its attributes.

$$\rho_S(Student)$$

Renames the table Student to S.

7. Join (⋈)

Combines related tuples from two relations based on a condition.

Suppose StudentDept = { (CSE, Computer Science), (ECE, Electronics) }

$$Student \bowtie_{Student.Dept=StudentD \quad .Code} StudentDept$$

Result:

ID	Name	Age	Dept	Code	DeptName
1	Alex	22	CSE	CSE	Computer Science
2	Riya	21	ECE	ECE	Electronics
3	Mohit	23	CSE	CSE	Computer Science

Joins (Detailed):

Suppose Table A:

ID	Name
1	Alice
2	Bob
3	Carol

Table B:

ID	Score
1	92
2	85
4	80

Inner Join (⋈ or ▷◁):

Returns only matching rows from both tables.

$$A \bowtie_{A.ID=B.ID} B$$

Result:

ID	Name	Score
1	Alice	92
2	Bob	85

Left Outer Join (⟕):

Returns all rows from the left table; unmatched rows from the right table have NULLs.

A

Result:

ID	Name	Score
1	Alice	92
2	Bob	85
3	Carol	NULL

Right Outer Join (⟖):

Returns all rows from the right table; unmatched rows from the left table have NULLs.

A

Result:

ID	Name	Score
1	Alice	92
2	Bob	85
4	NULL	80

Summary Table for Joins:

Operation	Included Rows	Nulls in...
Inner Join	Matches only	None
Left Outer Join	All left, matches from right	Right columns
Right Outer Join	All right, matches from left	Left columns

Types of Keys

- **Super Key:**
  Any set of attributes that uniquely identifies a tuple (row) in a relation (table). A table may have many super keys.

- **Candidate Key:**
  A minimal super key. It uniquely identifies tuples and cannot have any unnecessary attributes.

- **Primary Key:**
  The candidate key chosen to uniquely identify tuples in a relation. There can be only one primary key per table.

- **Compound Key:**
  A key that consists of two or more attributes that together uniquely identify a record, where each attribute is also a key by itself.

- **Composite Key:**
  A combination of two or more attributes that together uniquely identify a row, where none of the attributes is a key individually.

- **Foreign Key:**
  An attribute in one table that refers to the primary key of another table, used to establish a relationship between the two tables.

Normalization

Normalization is a systematic approach to organizing data within a database to reduce redundancy and eliminate undesirable characteristics such as insertion, update, and deletion anomalies.

Types of Anomalies

- **Insertion Anomalies:**
  Occur when it is not possible to insert data because required fields are missing or data is incomplete.

- **Deletion Anomalies:**
  Occur when deleting a record leads to unintentional loss of additional data.

- **Update Anomalies:**
  Occur when modifying data leads to inconsistencies or errors.

Normal Forms

- **First Normal Form (1NF):**
  All columns contain atomic (indivisible) values.

- **Second Normal Form (2NF):**
  No partial dependency exists—every non-key attribute depends on the whole primary key, not just part of it.

- **Third Normal Form (3NF):**
  No transitive dependencies—non-key attributes do not depend on other non-key attributes.

- **Boyce-Codd Normal Form (BCNF):**
  A stricter version of 3NF. For every non-trivial functional dependency (X → Y), X must be a super key.

- **Fourth Normal Form (4NF):**
  No multi-valued dependencies. A multi-valued dependency occurs when one attribute determines another, and both attributes are independent of all other attributes in the table.

- **Fifth Normal Form (5NF):**
  All join dependencies are removed, ensuring tables are decomposed such that no information is lost or duplicated through recombination.

Lossless Decomposition

In DBMS, a **lossless decomposition** is the process of splitting a relation schema into two or more schemas so that the original relation can be reconstructed perfectly by joining the decomposed tables—no data is lost or added.

Denormalization

Denormalization is a technique applied after normalization to optimize performance by adding redundant data to tables. This helps avoid costly joins but is not the same as reversing or skipping normalization—it is a deliberate optimization strategy.

Transactions and Concurrency Control

Concurrency Control

Concurrency control ensures that multiple transactions can simultaneously access or modify data without causing errors or inconsistencies.

Common Anomalies

- **Dirty Reads:**
  Occur when one transaction reads data written by another transaction that has not yet been committed. If the other transaction rolls back, this may cause inconsistencies.

- **Lost Updates:**
  When two or more transactions update the same data item at the same time, one update may overwrite the other, leading to data loss.

- **Inconsistent Reads:**
  If a transaction reads the same data multiple times during its execution, and the data is changed by another transaction between reads, inconsistent results may occur.

Schedules in Concurrency Control

- **Recoverable Schedule:**
  A schedule where a transaction commits only if all transactions it depends on have committed first. This ensures the system can recover correctly from failures.

- **Cascadeless Schedule:**
  A schedule that avoids cascading rollbacks. If one transaction fails, dependent transactions are not forced to roll back, increasing system stability.

Lock-Based Concurrency Control

Types of Locks

- **Shared Lock (S):**
  Also known as a read-only lock. Multiple transactions can hold shared locks on the same data

item simultaneously, but cannot modify the data while holding this lock.
(Requested using the lock-S instruction.)

- **Exclusive Lock (X):**
  Allows both reading and writing of a data item. Only one transaction can hold an exclusive lock on a data item at any time.
  (Requested using the lock-X instruction.)

Two-Phase Locking (2PL)

A transaction follows the Two-Phase Locking protocol when:

- **Growing Phase:**
  The transaction may acquire new locks but cannot release any.

- **Shrinking Phase:**
  The transaction can release existing locks but cannot acquire new ones.

Timestamp-Based Concurrency Control

This method uses timestamps to order transactions to ensure serializability and prevent conflicts.

ACID Properties

- **Atomicity:**
  Each transaction is all-or-nothing; either all operations succeed, or none do.

- **Consistency:**
  Transactions transform the database from one valid state to another.

- **Isolation:**
  Transactions execute independently, and intermediate results are not visible to other transactions.

- **Durability:**
  Once a transaction is committed, its changes are permanent, even in the event of a system failure.

Serializability

Guarantees that the outcome of executing transactions in parallel is the same as if they were executed serially.

- **Conflict Serializability:**
  A schedule is conflict serializable if it can be transformed into a serial schedule by swapping non-conflicting operations.

- **View Serializability:**
  A schedule is view serializable if, for each read/write operation, transactions see the same data as they would in some serial order.

Schedule Types in Concurrency Control

- **Recoverable Schedule:**
  A schedule in which a transaction commits only after all transactions from which it has read data have committed as well. This ensures that no transaction commits based on uncommitted (and possibly incorrect) data, allowing for proper recovery in case of failures.

- **Cascading Schedule:**
  A schedule where a rollback (failure) of one transaction may cause other dependent transactions to roll back as well. This happens if a transaction reads data written by another uncommitted transaction.

- **Cascadeless Schedule:**
  A schedule that prevents cascading rollbacks by ensuring that a transaction only reads data from transactions that have already committed. As a result, the failure of one transaction does not cause others to roll back.

- **Non-Recoverable Schedule:**
  A schedule where a transaction commits data that it has read from another uncommitted transaction. If the first transaction fails and is rolled back, the database may be left in an inconsistent state, as the second transaction has already committed based on potentially invalid data.

Note: Non-serializable schedules may lead to database inconsistency if not managed properly.