Database Management

Mr. Sandeep L Dhende

Assistant Professor, Dept. of E&TC, PICT, Pune

Outline

- Introduction to Database Management Systems
- Purpose of Database Systems
- Database-System Applications
- Data Abstraction
- Database System Structure

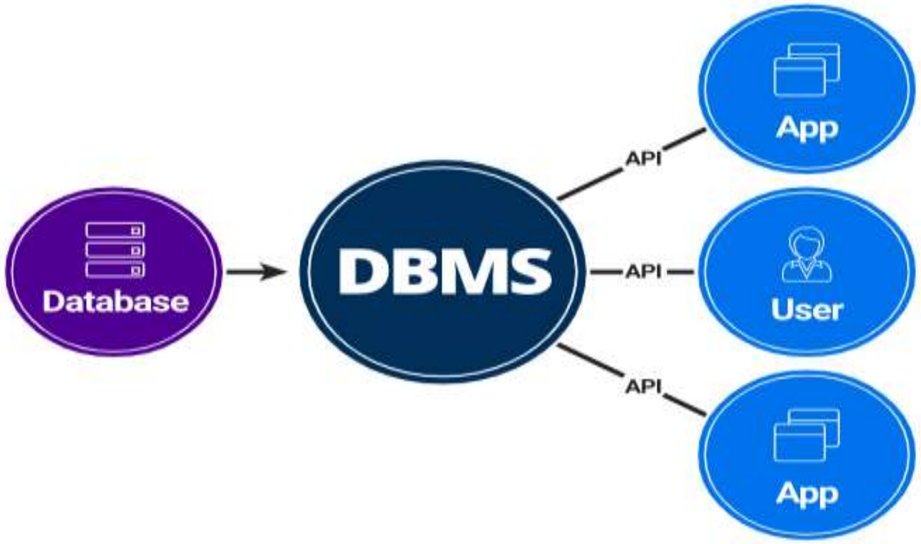
Database Management System

- 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.

Database Management System Cont.

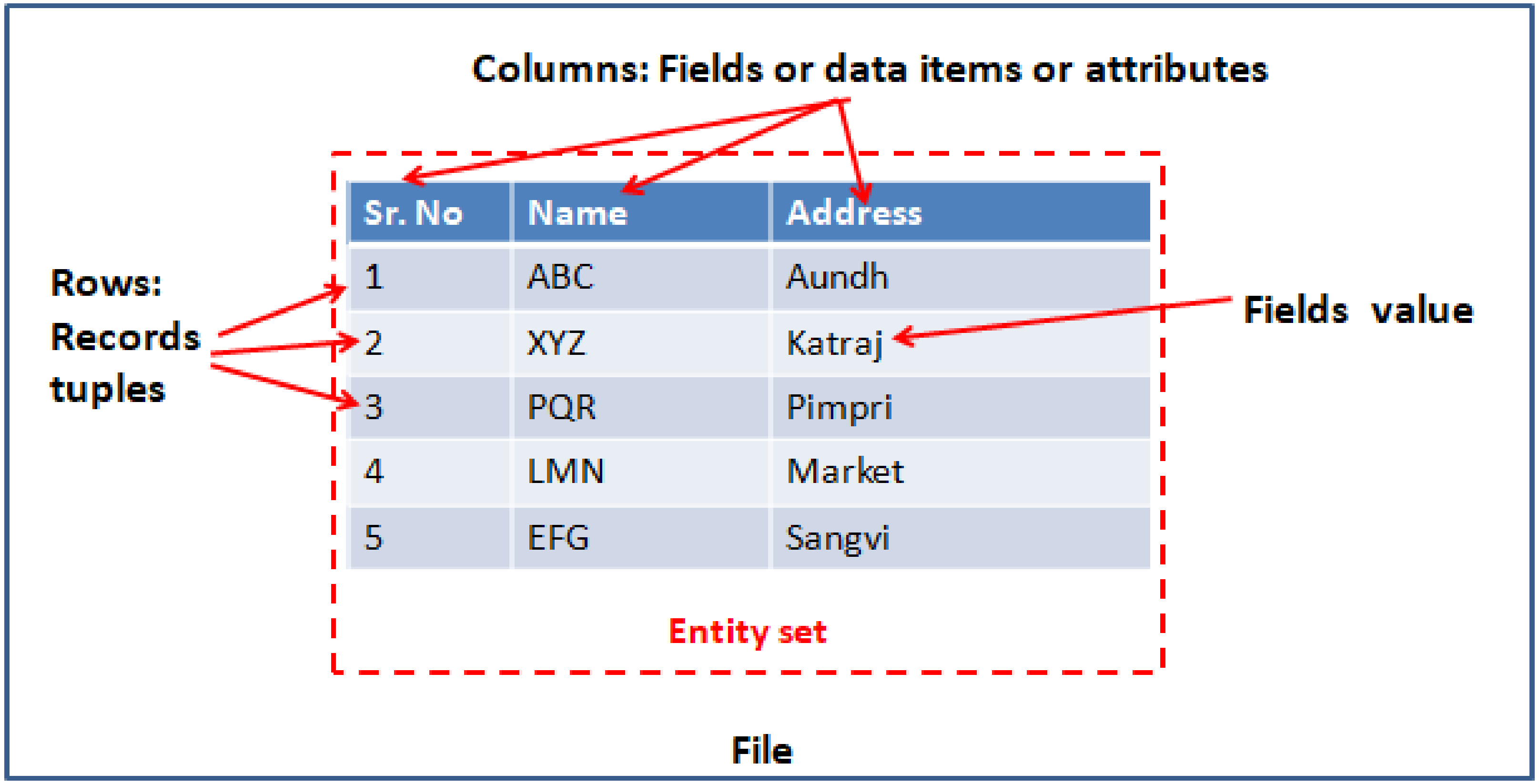
- A modern database system is a complex software system whose task is to manage a large and complex collection of data.
- Databases can be very large.
- Databases touch all aspects of our lives.



Comparison of Database and DBMS

Database	DBMS
A database is a collection of logically related data.	Database management system is a set of software or programs that enables storing, modifying, and extracting information from a database.
It is used to search data to answer queries.	It provide users with tools to add, delete, access, modify, and analyze data stored in one location.
A database may be designed for batch processing, real time processing or online processing.	A group can access the data by using query and reporting tools that are part of the DBMS, or by using application programs specifically written to access the data.

Database Table



Sr. No	Name	Address
1	ABC	Aundh
2	XYZ	Katraj
3	PQR	Pimpri
4	LMN	Market
5	EFG	Sangvi

Table of data

University Database Example

- Data consists of information about:
  - Students
  - Instructors
  - Classes
- Application program examples:
  - Add new students, instructors, and courses
  - Register students for courses, and generate class rosters
  - Assign grades to students, compute grade point averages (GPA) and generate transcripts

Database Systems Applications

- Enterprise Information
  - Sales: customers, products, purchases
  - Accounting: payments, receipts, account balances, assets
  - Human Resources: Information about employees, salaries, payroll taxes and benefits.
- Manufacturing: management of production, inventory, orders, supply chain.
- Banking and finance
  - customer information, accounts, loans, and banking transactions.
  - Credit card transactions

Database Systems Applications Cont.

  - Finance: sales and purchases of financial instruments (e.g., stocks and bonds; storing real-time market data
- Universities: student info, registration, grades
- Airlines: reservations, schedules
- Telecommunication: records of calls, texts, and data usage, generating monthly bills, maintaining balances on prepaid calling cards
- Web-based services
  - Online retailers: order tracking, customized recommendations
  - Online advertisements

Database Systems Applications Cont.

- Navigation systems: For maintaining the locations of varies places of interest along with the exact routes of roads, train systems, buses, etc.

Purpose of Database Systems

In the early days, database applications were built directly on top of file systems, which leads to:

- Data redundancy and inconsistency: data is stored in multiple file formats resulting in duplication of information in different files.
- Difficulty in accessing data
  - Need to write a new program to carry out each new task.
- Data isolation
  - Multiple files and formats
- Integrity problems
  - Integrity constraints (e.g., account balance > 0) become "buried" in program code rather than being stated explicitly.
  - Hard to add new constraints or change existing ones.

Purpose of Database Systems Cont.

- Atomicity of updates
  - Failures may leave database in an inconsistent state with partial updates carried out.
  - Example: Transfer of funds from one account to another should either complete or not happen at all.
- Concurrent access by multiple users
  - Concurrent access needed for performance
  - Uncontrolled concurrent accesses can lead to inconsistencies
    - Ex: Two people reading a balance (say 100) and updating it by withdrawing money (say 50 each) at the same time

Purpose of Database Systems Cont.

- Security problems
  - Hard to provide user access to some, but not all, data

Database systems offer solutions to all the above problems.

File System vs DBMS

Sr. No.	File System	DBMS
1.	File system is a software that manages and organizes the files in a storage medium within a computer.	DBMS is a software for managing the database.
2.	Redundant data can be present in a file system.	In DBMS there is no redundant data.
3.	It doesn’t provide backup and recovery of data if it is lost.	It provides backup and recovery of data even if it is lost.

File System vs DBMS Cont.

Sr. No.	File System	DBMS
4.	There is no efficient query processing in file system.	Efficient query processing is there in DBMS.
5.	There is less data consistency in file system.	There is more data consistency because of the process of normalization.
6.	It is less complex as compared to DBMS.	It has more complexity in handling as compared to file system.

File System vs DBMS Cont.

Sr. No.	File System	DBMS
7.	File systems provide less security in comparison to DBMS.	DBMS has more security mechanisms as compared to file system.
8.	It is less expensive than DBMS.	It has a comparatively higher cost than a file system.

Database System Environment

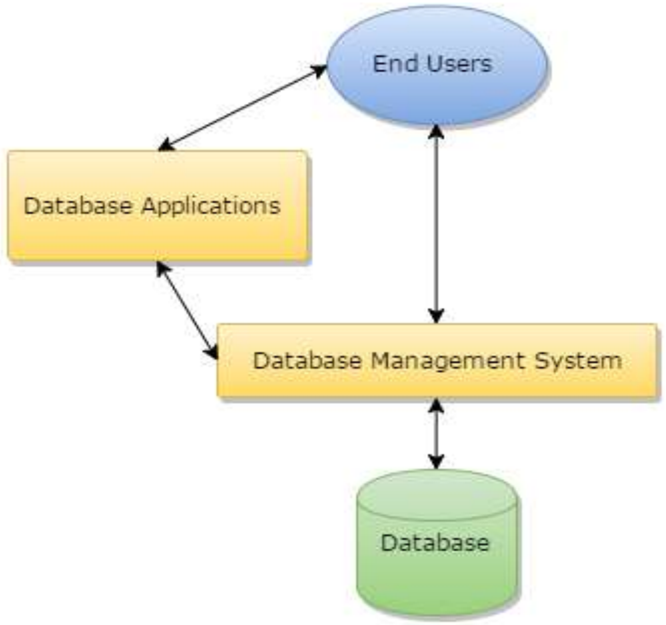


Figure: Database system environment

- The end users or programmers with the help of application program and DBMS software accessing and retrieving the data.

Components of Database System

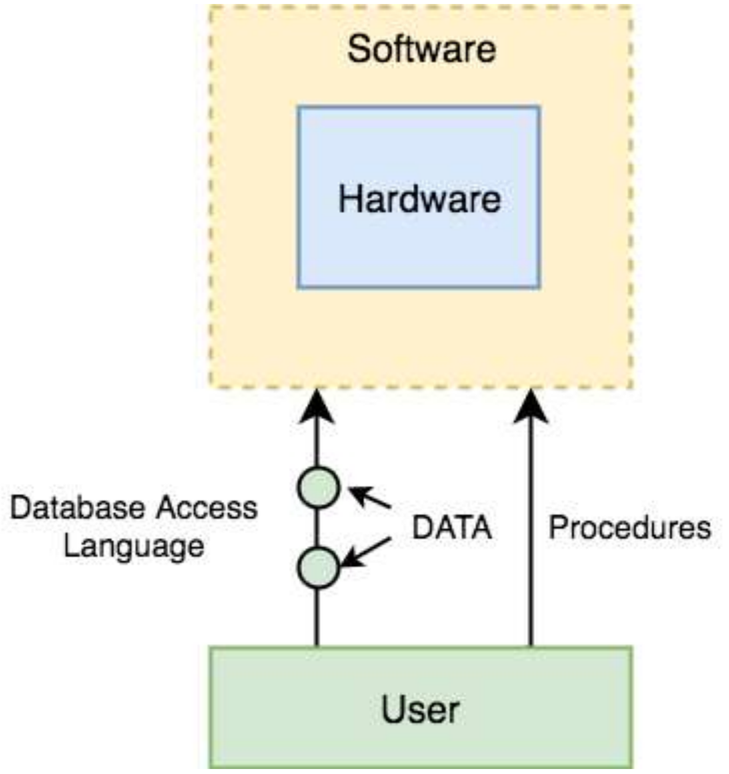


Figure: Components of Database System

Advantages of DBMS

Here, are pros/benefits of DBMS system:

- DBMS offers a variety of techniques to store & retrieve data.
- Uniform administration procedures for data.
- Application programmers never exposed to details of data representation and Storage.
- A DBMS uses various powerful functions to store and retrieve data efficiently.
- Offers Data Integrity and Security.
- The DBMS implies integrity constraints to get a high level of protection against prohibited access to data.
- Reduced Application Development Time.

Advantages of DBMS Cont.

- Consume lesser space.
- Reduction of redundancy.
- Data independence.

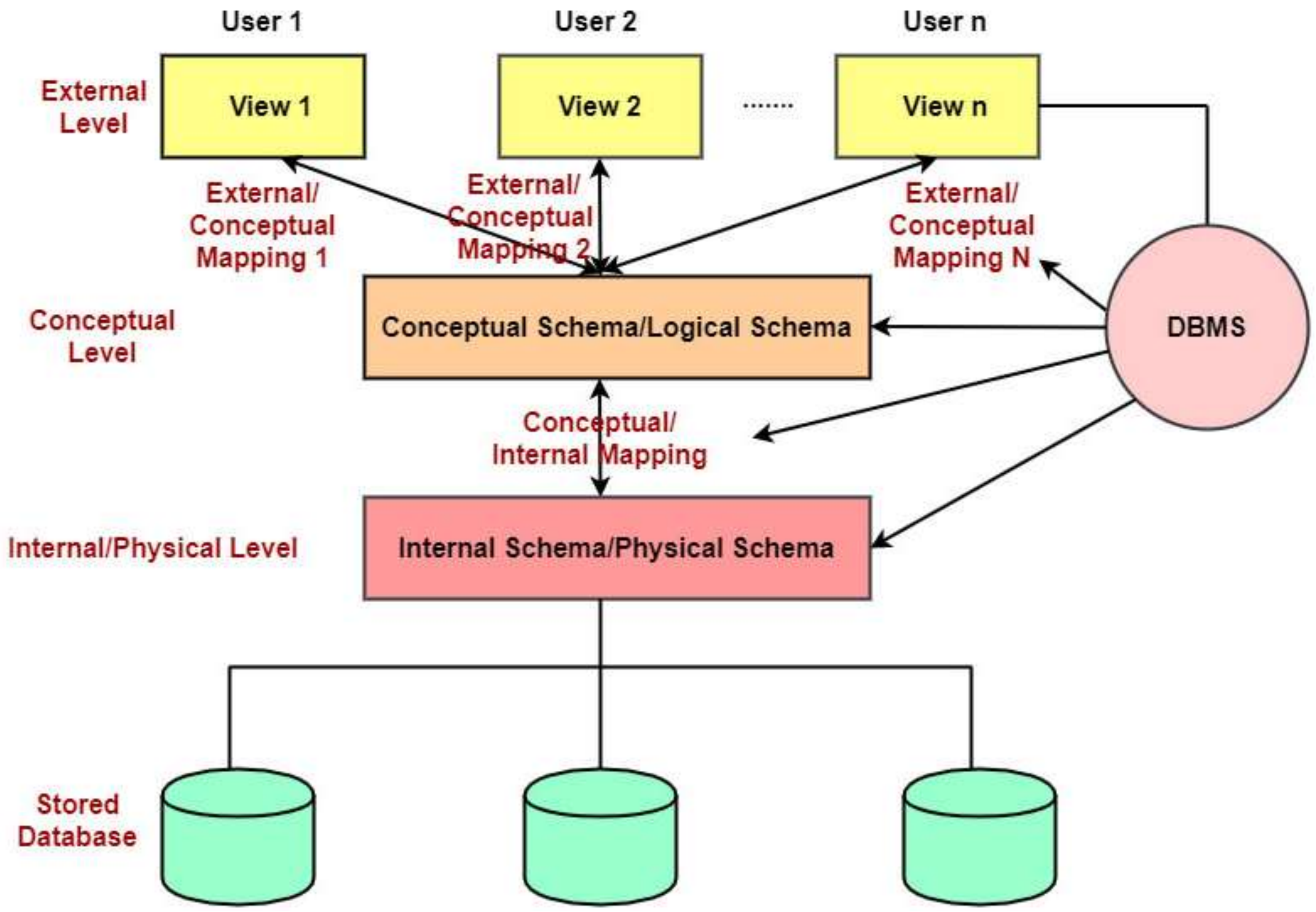
Disadvantages of DBMS

- It increases opportunity for person or groups outside the organization to gain access to information about the firms operation.
- It increases opportunity for fully training person within the organization to misuse the data resources intentionally.
- The data approach is a costly due to higher H/W and S/W requirements.
- Database systems are complex (due to data independence), difficult, and time-consuming to design.
- It is not maintained for all organizations. It is only efficient for particularly large organizations.

Disadvantages of DBMS Cont.

- Damage to database affects virtually all applications programs.
- Initial training required for all programmers and users.

View Levels of DBMS Architecture


User 1
User 2
User n
External Level
View 1
View 2
....
View n
External/ Conceptual Mapping 1
External/ Conceptual Mapping 2
External/ Conceptual Mapping N
Conceptual Level
Conceptual Schema/Logical Schema
DBMS
Conceptual/ Internal Mapping
Internal/Physical Level
Internal Schema/Physical Schema
Stored Database

Levels of Abstraction

- View Levels of DBMS Architecture also called as Level of Data Abstraction in DBMS. It describe the various views and levels of data abstraction in DBMS system which are as Physical Level, Conceptual Level and External Level.

1. Physical Level:

- Physical level describes the physical storage structure of data in database, as well it also describe how the data is stored and retrieve in database.
- It also describe compression and encryption techniques applied on data.
- It is also known as Internal Level.
- This level is very close to physical storage of data.

Levels of Abstraction Cont.

2. Conceptual Level: also called logical or global level

- It describes the structure of the whole database for a group of users.
- It also describe how the data will be appeared to various user and relationship between various data.

3. External Level: is also called view level

- It is related to the data which is viewed by individual end users.
- This level includes a no. of user views or external schemas.
- This level is closest to the user.
- External view describes the segment of the database that is required for a particular user group and hides the rest of the database from that user group.

Levels of Abstraction Cont.

- **Physical level**: describes how a record (e.g., instructor) is stored.
- **Logical level**: describes what data is stored in database, and the relationships among the data.

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

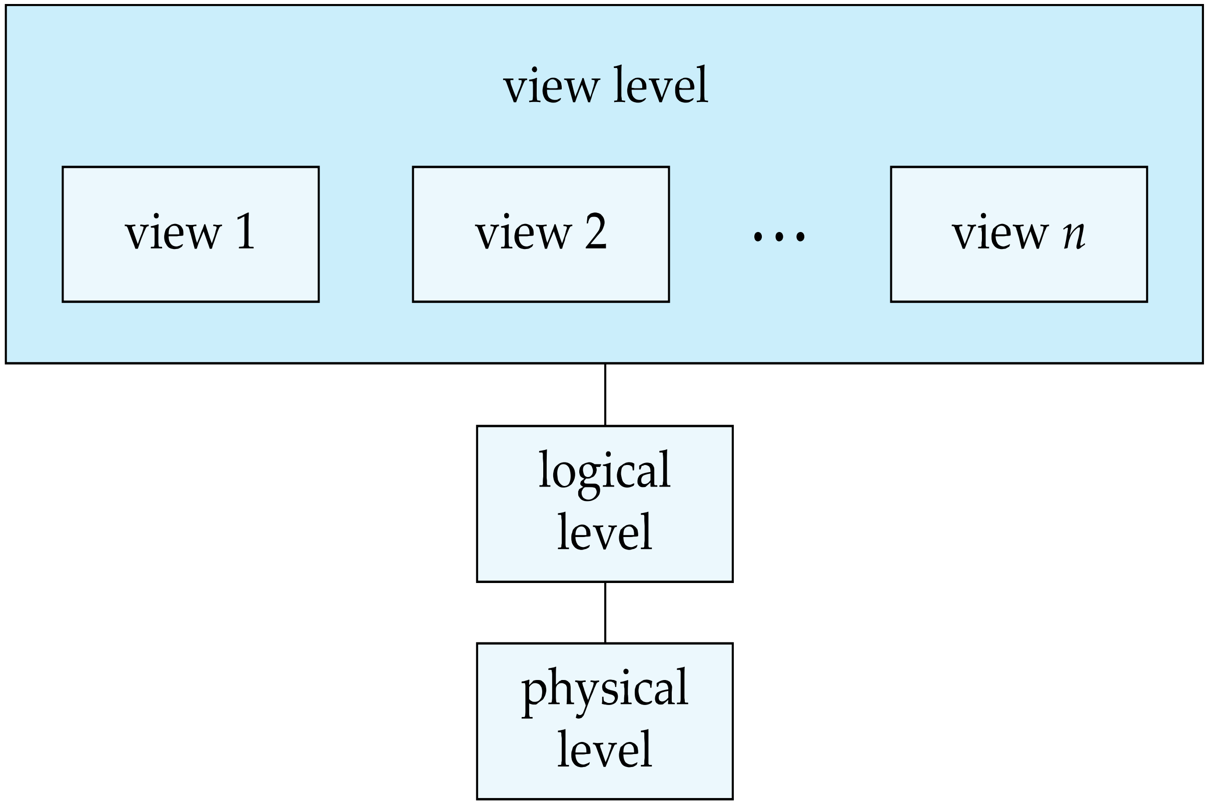
- **View level**: application programs hide details of data types.
  - Views can also hide information (such as an employee's salary) for security purposes.

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.
  – Data models
    - A collection of conceptual tools for describing data, data relationships, data semantics, and consistency constraints.
  – Data abstraction
    - Hide the complexity of data structures to represent data in the database from users through several levels of data abstraction.

View of Data Cont.

- An architecture for a database system



Instances and Schemas

- Similar to types and variables in programming languages.
- **Logical Schema** – the overall logical structure of the database
  - Analogous to type information of a variable in a program.
  - Example: The database consists of information about a set of customers and accounts in a bank and the relationship between them.
  - Customer Schema

Name	CustomerID	Account#	AadhaarID	Mobile#

  - Account Schema

Account#	Account Type	Interest Rate	Min. Bal.	Balance

Instances and Schemas Cont.

- **Physical schema** – the overall physical structure of the database
- **Instance** – the actual content of the database at a particular point in time
  - Analogous to the value of a variable
  - Customer Instance

Name	CustomerID	Account#	AadhaarID	Mobile#
John	6728	917322	182719289372	9830100291
Sam	8912	827183	918291204829	7189203928
Lalit	6617	37292	127837291021	8892021892

Instances and Schemas Cont.

– Account Instance

Account #	Account Type	Interest Rate	Min. Bal.	Balance
917322	Savings	4.0%	5000	150000
372912	Current	0.0%	0	291820
827183	Term Deposit	6.75%	1000	100000

Physical Data Independence

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

Data Models

- Collection of tools for describing
  - Data
  - Data relationships
  - Data semantics
  - Data constraints
- Relational model
- Entity-Relationship data model (mainly for database design)
- Object-based data models (Object-oriented and Object-relational)
- Semi-structured data model (XML)
- Other older models:
  - Network model
  - Hierarchical model

Relational Model

- All the data is stored in various tables.
- Example of tabular data in the relational model

Columns

ID	Name	Dept_Name	Salary
22222	Ram	Ele. Eng.	95000
12121	Vijay	Comp. Sci.	90000
32343	Kumar	Physics	60000
45565	Santosh	Biology	75000
98345	Raghu	History	80000
76766	Lalit	Music	72000
10101	Namita	Physics	65000
58583	Nupur	Finance	92000

Rows

A Sample Relational Database

ID	Name	Dept_Name	Salary
22222	Ram	Ele. Eng.	95000
12121	Vijay	Comp. Sci.	90000
32343	Kumar	Physics	60000
45565	Santosh	Biology	75000
98345	Raghu	History	80000
76766	Lalit	Music	72000
10101	Namita	Physics	65000
58583	Nupur	Ele. Eng.	92000

(a)
The *instructor* table

Dept_Name	Building	Budget
Ele. Eng.	A3	100000
Comp. Sci.	A1	90000
Physics	B1	85000
Biology	B2	80000
History	E	120000
Music	H	70000

(b)
The *department* table

Data Definition Language (DDL)

- Specification notation for defining the database schema
- Allows one to define tables, integrity constraints, assertions, etc.

Example: **create table** *instructor* (

```
ID          int primary key ,
name        varchar(20),
dept_name   varchar(20),
salary      numeric(8,2))
```

- DDL compiler generates a set of table templates stored in a ***data dictionary***
- Data dictionary contains metadata (i.e., data about data)
  - Database schema
  - Integrity constraints
    - Primary key (ID uniquely identifies instructors)
  - Authorization
    - Who can access what

Data Manipulation Language (DML)

- Language for accessing and updating the data organized by the appropriate data model
  - DML also known as query language
- There are basically two types of data-manipulation language
  - **Procedural DML** -- require a user to specify what data are needed and how to get those data.
  - **Declarative DML** -- 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.
- Declarative DMLs are also referred to as non-procedural DMLs.
- The portion of a DML that involves information retrieval is called a **query** language.

SQL Query Language

- The most widely used commercial language.
- SQL query language is nonprocedural. A query takes as input several tables (possibly only one) and always returns a single table.
- Example to find all instructors in Comp. Sci. dept

  select *name*
  from *instructor*
  where *dept_name* = 'Comp. Sci.'

- SQL is **NOT** a Turing machine equivalent language
  - Cannot be used to solve all problems that a C program, for example, can solve
- To be able to compute complex functions SQL is usually embedded in some higher-level language.
- Application programs generally access databases through one of
  - Language extensions to allow embedded SQL
  - Application program interface (e.g., ODBC/JDBC) which allow SQL queries to be sent to a database

Database Design

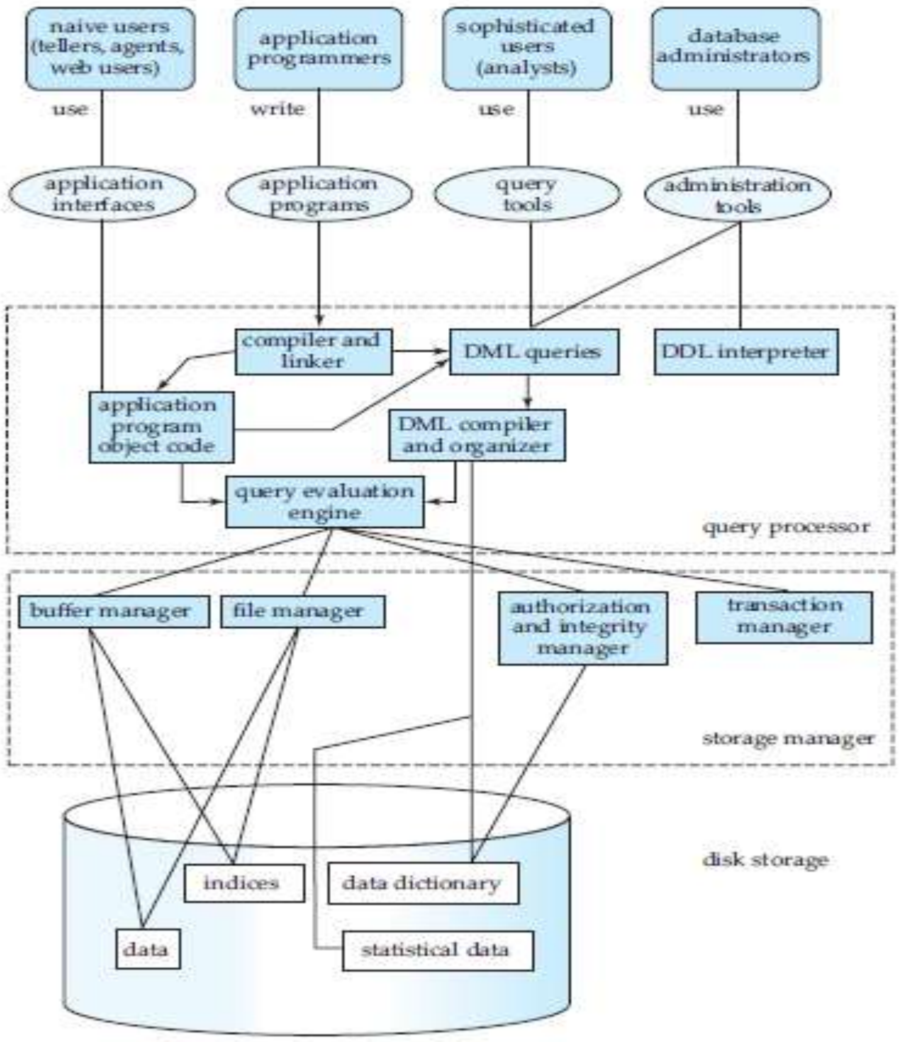
The process of designing the general structure of the database:

- Logical Design – Deciding on the database schema. Database design requires that we find a “good” collection of relation schemas.
  - Business decision – What attributes should we record in the database?
  - Computer Science decision – What relation schemas should we have and how should the attributes be distributed among the various relation schemas?
- Physical Design – Deciding on the physical layout of the database.

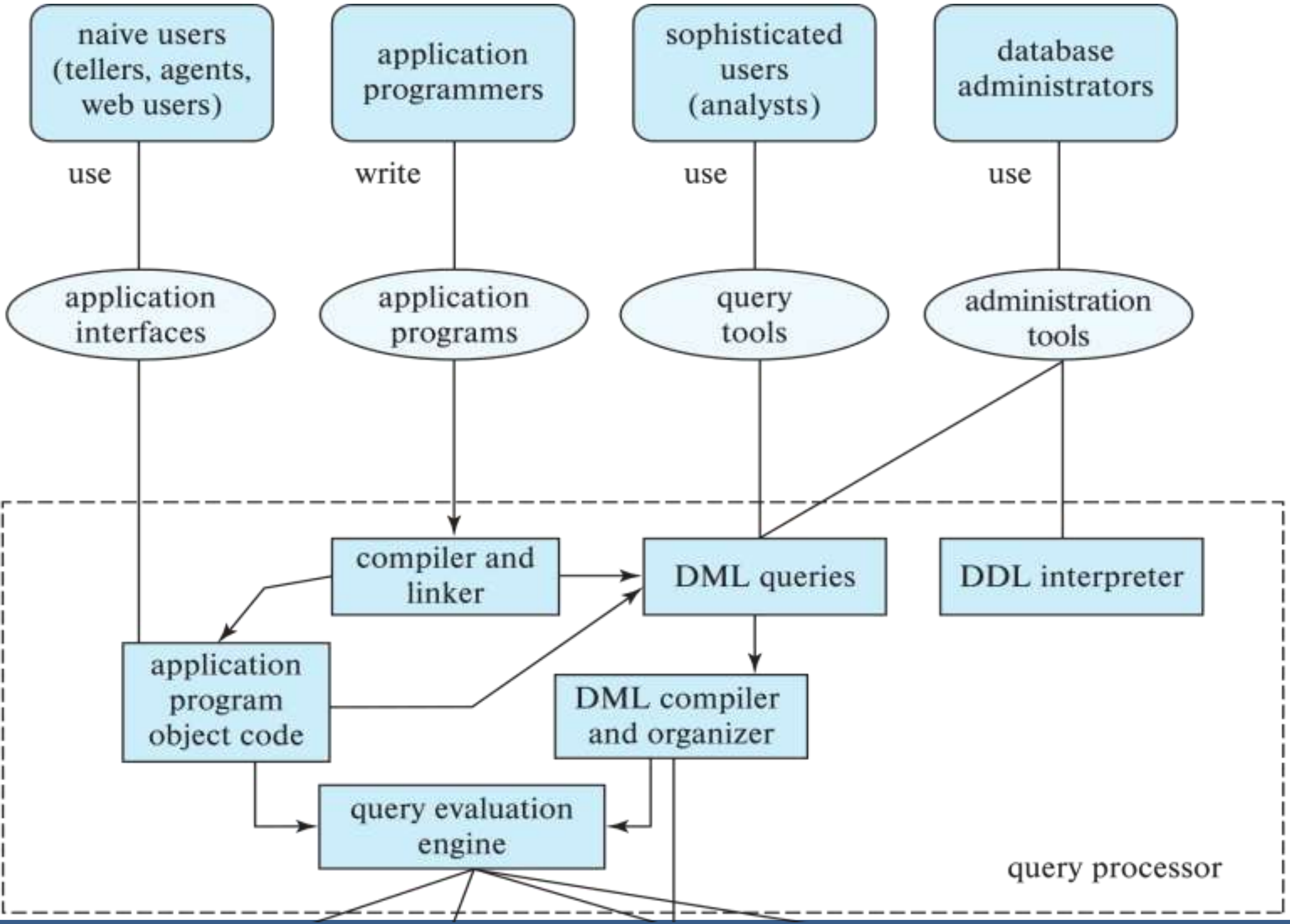
Database System Architecture

- The design of a DBMS depends on its architecture.
- It can be centralized or decentralized or hierarchical. The architecture of a DBMS can be seen as either single tier or multi-tier.
- This architecture is a three layered architecture the bottom layer consist of the Database, middle layer is DBMS and top layer is a view.
- This architecture describe how the interaction between user and various components of DBMS system is done.

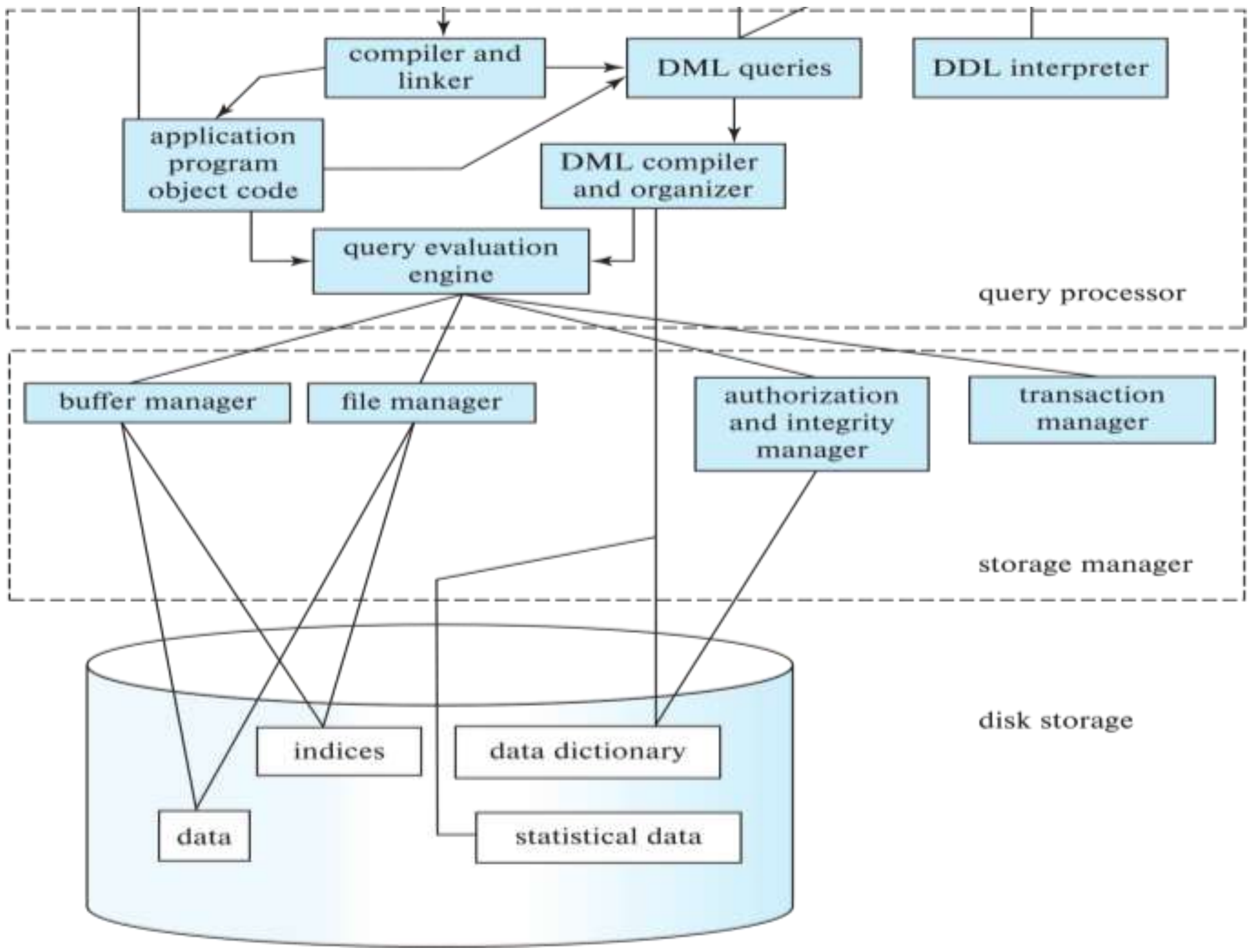
Database System Architecture Cont.



Database System Architecture Cont.



Database System Architecture Cont.



Database System Architecture Cont.

DBMS Layer: is a middle layer consisting of two major components as follow.

- **Query Processor:** Responsible for executing users query
- Following are the components of the query processor;
- **DML Pre-compiler:** It translates DML statements in a query language into low level instructions that query evaluation engine understands.
- **Embedded DML Pre-compiler:** It converts DML statements embedded in an application program to normal procedure calls in the host language.
- **DDL Interpreter:** It interprets the DDL statements and records them in a set of tables containing meta data or data dictionary.
- **Query Evaluation Engine:** It executes low-level instructions generated by the DML compiler.

Database System Architecture Cont.

- **Storage Manager:** A storage manager is responsible for storing, retrieving and updating data in the database.
- Following are the components of the storage manager;
- **Authorization and Integrity Manager:** It tests the integrity constraints and checks the authorization of users to access data.
- **Transaction Manager:** It ensures that no kind of change will be brought to the database until a transaction has been completed totally.
- **File Manager:** It manages the allocation of space on disk storage and the data structures used to represent information stored on disk.
- **Buffer Manager:** It decides which data is in need to be cached in main memory and then fetch it up in main memory.

Database System Architecture Cont.

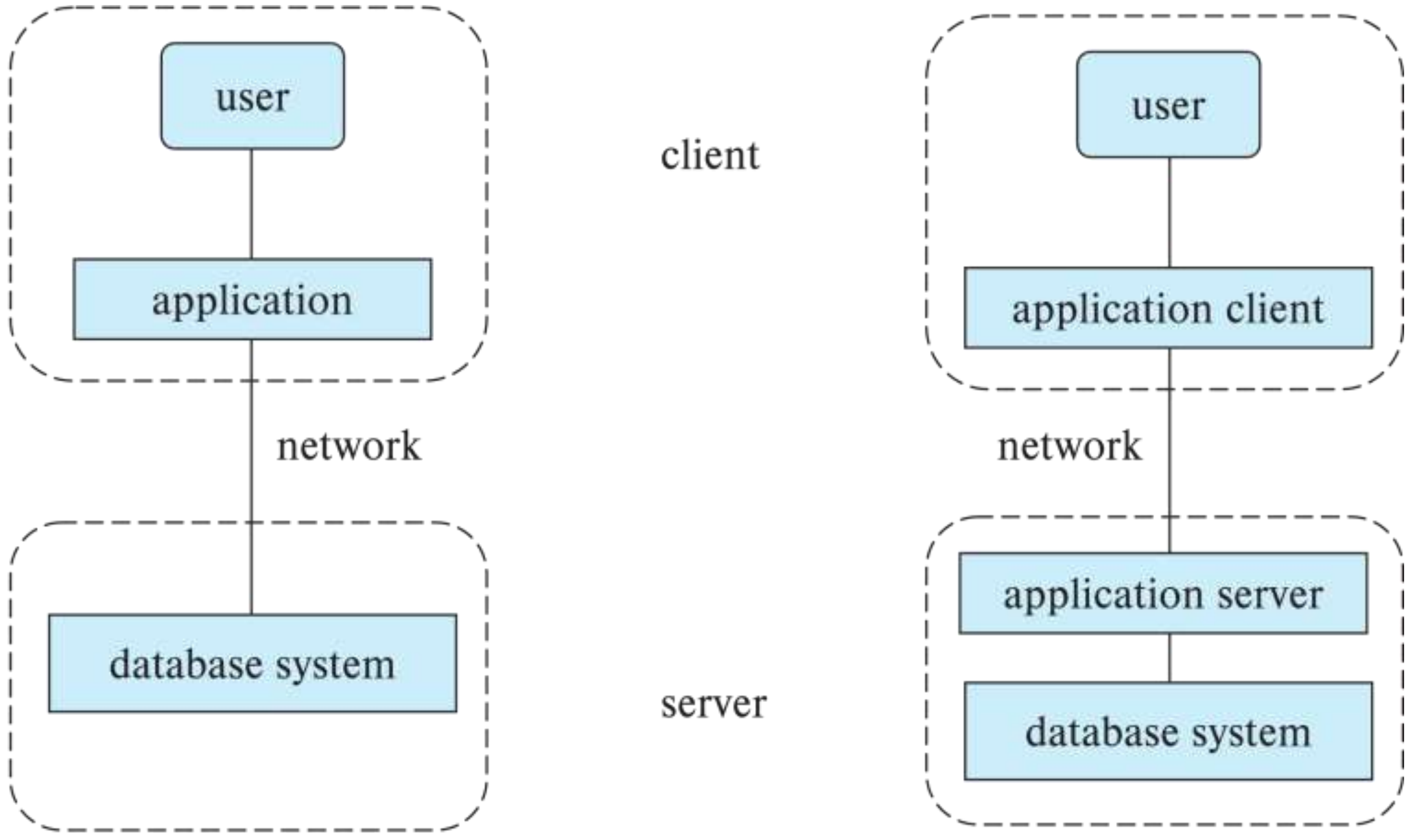
- **Database Layer:** It is the Bottom layer consisting of following sub components.
- **Data Dictionary**: data dictionary is a file or a set of files that contains a database's metadata. The data dictionary contains records about other objects in the database, such as data ownership, data relationships to other objects, and other data.
- **Data Files**: A **data file** is file which stores data to be used by a application or user .
- **DBMS Indexing**: We know that data is stored in the form of records. Every record has a key field, which helps it to be recognized uniquely.

Database Applications

Database applications are usually partitioned into two or three parts

- Two-tier architecture -- the application resides at the client machine, where it invokes database system functionality at the server machine
- Three-tier architecture -- the client machine acts as a front end and does not contain any direct database calls.
  - The client end communicates with an application server, usually through a forms interface.
  - The application server in turn communicates with a database system to access data.

Two-tier and three-tier architectures

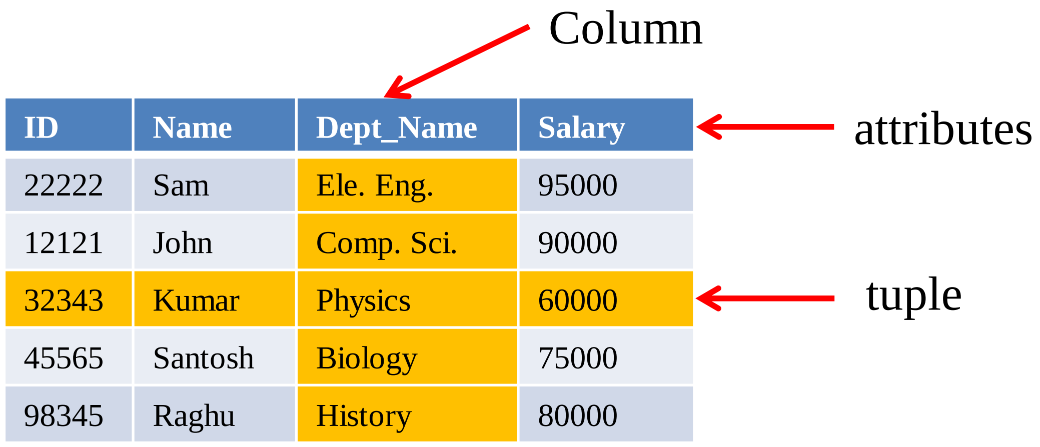


(a) Two-tier architecture

(b) Three-tier architecture

Relational Data Model

- This model is the most popular model and the most extensively used model.
- In this data can be stored in the tables and this storing is called as relation.
- Each row in a relation contains unique value and it is called as tuple, each column contains value from same domain and it is called as attribute.


Column
attributes
tuple

ID	Name	Dept_Name	Salary
22222	Sam	Ele. Eng.	95000
12121	John	Comp. Sci.	90000
32343	Kumar	Physics	60000
45565	Santosh	Biology	75000
98345	Raghu	History	80000

Example of a *Instructor* Relation

Attributes
or Columns

ID	Name	Dept_Name	Salary
22222	Sam	Ele. Eng.	95000
12121	John	Comp. Sci.	90000
32343	Kumar	Physics	60000
45565	Santosh	Biology	75000
98345	Raghu	History	80000
76766	Lalit	Music	72000
10101	Namita	Physics	65000
58583	Nupur	Finance	92000

Tuples
or Rows

Relation Schema and Instance

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

  Example:

  instructor = (ID, name, dept_name, salary)

- A relation instance r defined over schema R is denoted by r(R).
- The current values a relation are specified by a table
- An element **t** of relation **r** is called a tuple and is represented by a row in a table

Attributes

- The set of allowed values for each attribute is called the **domain** of the attribute.
- Attribute values are (normally) required to be **atomic**; that is, indivisible.
- The special value ***null*** is a member of every domain. Indicated that the value is "unknown".
- The null value causes complications in the definition of many operations.

Roll #	First Name	Last Name	DoB	Passport #	Aadhaar #	Department
15CS10026	Lalit	Dubey	27-Mar-1997	L4032464	1728-6174-9239	Computer
16EE30029	Jatin	Chopra	17-Nov-1996	null	3917-1836-3816	Electrical

Relations are Unordered

- Order of tuples is irrelevant (tuples may be stored in an arbitrary order)
- Example: *instructor* relation with unordered tuples

ID	Name	Dept_Name	Salary
22222	Sam	Ele. Eng.	95000
12121	John	Comp. Sci.	90000
32343	Kumar	Physics	60000
45565	Santosh	Biology	75000
98345	Raghu	History	80000
76766	Lalit	Music	72000
10101	Namita	Physics	65000
58583	Nupur	Finance	92000

Database Schema

- Database schema -- is the logical structure of the database.
- Database instance -- is a snapshot of the data in the database at a given instant in time.
- Example:
  - schema: *instructor* (*ID*, *name*, *dept_name*, *salary*)
  - Instance:

ID	Name	Dept_Name	Salary
22222	Sam	Ele. Eng.	95000
12121	John	Comp. Sci.	90000
32343	Kumar	Physics	60000
45565	Santosh	Biology	75000
98345	Raghu	History	80000
76766	Lalit	Music	72000
10101	Namita	Physics	65000
58583	Nupur	Finance	92000

Fundamental Concepts

Relation:

- A *relation*, also known as a *table* or *file*, is a subset of the Cartesian product of a list of domains characterized by a name. And within a table, each row represents a group of related data values. A *row*, or record, is also known as a *tuple*. The columns in a table is a field and is also referred to as an attribute. You can also think of it this way: an attribute is used to define the record and a record contains a set of attributes.
- The steps below outline the logic between a relation and its domains.

1. Given *n* domains are denoted by D1, D2, … Dn
2. And *r* is a relation defined on these domains
3. Then $r \subseteq D1 \times D2 \times \ldots \times Dn$

Fundamental Concepts Cont.

Table:

- A database is composed of multiple tables and each table holds the data. Figure shows a database that contains three tables.

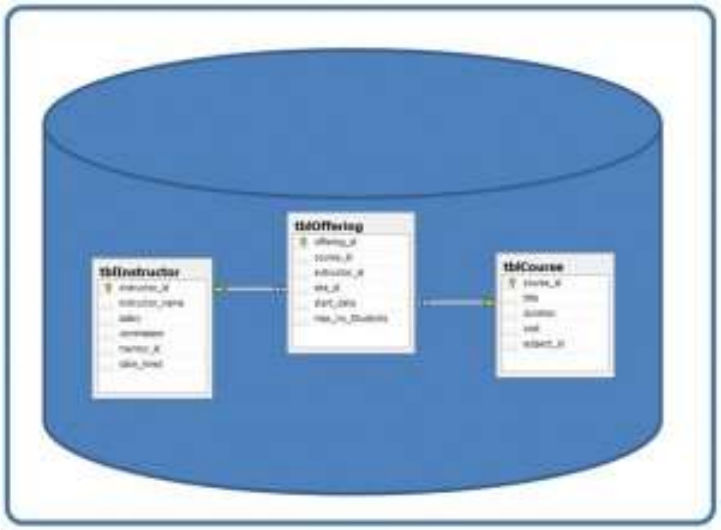


Figure: Database with three tables

Fundamental Concepts Cont.

Column:

- A database stores pieces of information or facts in an organized way. Understanding how to use and get the most out of databases requires us to understand that method of organization.
- The principal storage units are called *columns* or *fields* or *attributes*. These house the basic components of data into which your content can be broken down. When deciding which fields to create, you need to think generically about your information, for example, drawing out the common components of the information that you will store in the database and avoiding the specifics that distinguish one item from another.
- Look at the example of an ID card in Figure to see the relationship between fields and their data.

Fundamental Concepts Cont.

Field Name	Data
First Name	Sam
Family Name	Dubey
Nationality	Indian
Salary	100,000
Date of Birth	01 June 1988
Marital Status	Married
Shift	Mon, Tue
Place	Pune

Figure: Example of an ID card

Fundamental Concepts Cont.

Domain:

- A *domain* is the original sets of atomic values used to model data. By *atomic value*, we mean that each value in the domain is indivisible as far as the relational model is concerned. For example:
- The domain of Marital Status has a set of possibilities: Married, Single, Divorced.
- The domain of Shift has the set of all possible days: {Mon, Tue, Wed…}.
- The domain of Salary is the set of all floating-point numbers greater than 0 and less than 200,000.
- The domain of First Name is the set of character strings that represents names of people.

Fundamental Concepts Cont.

Records:

- Just as the content of any one document or item needs to be broken down into its constituent bits of data for storage in the fields, the link between them also needs to be available so that they can be reconstituted into their whole form. Records allow us to do this. *Records* contain fields that are related, such as a customer or an employee. As noted earlier, a tuple is another term used for record.
- Records and fields form the basis of all databases. A simple table gives us the clearest picture of how records and fields work together in a database storage project.

Fundamental Concepts Cont.

Columns

ID	Name	Dept_Name	Salary
22222	Sam	Ele. Eng.	95000
12121	John	Comp. Sci.	90000
32343	Kumar	Physics	60000
45565	Santosh	Biology	75000
98345	Raghu	History	80000
76766	Lalit	Music	72000
10101	Namita	Physics	65000
58583	Nupur	Finance	92000

Rows

Figure: Example of a simple table

Fundamental Concepts Cont.

Degree:

- The *degree* is the number of attributes in a table. In the above example in Figure, the degree is 4.

Relational Data Model

Advantages

- Structural independence – changes in the relational data structure do not affect the DBMS's data access in any way
- Improved conceptual simplicity by concentrating on the logical view
- Easier database design, implementation, management, and use
- Powerful database management system.

Disadvantages

- Can facilitate poor design and implementation

Relational Algebra

- Relational database systems are expected to be equipped with a query language that can assist its users to query the database instances. There are two kinds of query languages − relational algebra and relational calculus.

Relational Algebra Cont.

1. Relational Algebra

- Relational algebra is a procedural query language, which takes instances of relations as input and yields instances of relations as output.
- It uses operators to perform queries.
- An operator can be either **unary** or **binary**.
- They accept relations as their input and yield relations as their output.
- Relational algebra is performed recursively on a relation and intermediate results are also considered relations.

Relational Algebra Cont.

The fundamental operations of relational algebra are as follows −

Unary Relational Operations

- SELECT (symbol: σ)
- PROJECT (symbol: π)
- RENAME (symbol: ρ)

Relational Algebra Operations From Set Theory

- UNION (∪)
- INTERSECTION (∩),
- DIFFERENCE (-)
- CARTESIAN PRODUCT (x)

Binary Relational Operations

- JOIN
- DIVISION

Unary Relational Operations

1. SELECT (σ)

- The SELECT operation is used for selecting a subset of the tuples according to a given selection condition.
- Sigma(σ) Symbol denotes it.
- It is used as an expression to choose tuples which meet the selection condition.
- Select operator selects tuples that satisfy a given predicate.
- $\sigma_p(r)$
- p is the predicate
- r stands for relation which is the name of the table
- p is prepositional logic

Unary Relational Operations Cont.

Example 1:

- $\sigma_{\text{course = “DBMS"}}$ (Courses)
- **Output** - Selects tuples from Courses where course = ‘DBMS'.

Example 2:

- $\sigma_{\text{course = “DBMS" and author = “Korth"}}$(Courses)
- **Output** - Selects tuples from Courses where the course is ‘DBMS' and author is ‘Korth’.

Example 3:

- $\sigma_{\text{sales > 50000}}$ (Customers)
- **Output** - Selects tuples from Customers where sales is greater than 50000

Unary Relational Operations Cont.

2. Project Operation (∏)

- It projects column(s) that satisfy a given predicate.
- Notation – $\prod_{A1,\, A2,\, An} (r)$
- Where A_1, A_2 , A_n are attribute names of relation **r**.
- Duplicate rows are automatically eliminated, as relation is a set.

Unary Relational Operations Cont.

Example:

- Consider the customers table.

CustomerID	CustomerName	Status
1	Google	Active
2	Amazon	Active
3	Apple	Inactive
4	Alibaba	Active

- Here, the projection of CustomerName and status will give
- $\Pi_{\text{CustomerName, Status}}$ (Customers)

CustomerName	Status
Google	Active
Amazon	Active
Apple	Inactive
Alibaba	Active

Unary Relational Operations Cont.

3. Rename (ρ)

- Rename is a unary operation used for renaming attributes of a relation.
- ρ (a/b)r will rename the attribute 'b' of relation r by 'a'.

- The results of relational algebra are also relations but without any name. The rename operation allows us to rename the output relation. 'rename' operation is denoted with small Greek letter **rho** ρ.
- **Notation** – ρ_{x} (r)
- Where the result of expression **r** is saved with name of **x**.

Operations From Set Theory

1. Union Operation (∪)

- It performs binary union between two given relations and is defined as –
- r ∪ s = { t | t ∈ r or t ∈ s}
- **Notation** – r U s
- Where **r** and **s** are either database relations or relation result set (temporary relation).
- For a union operation to be valid, the following conditions must hold –
- **r**, and **s** must have the same number of attributes.
- Attribute domains must be compatible.
- Duplicate tuples are automatically eliminated.

Operations From Set Theory Cont.

Example 1:

- $\prod_{\text{author}}$ (Books) ∪ $\prod_{\text{author}}$ (Articles)
- **Output** – Projects the names of the authors who have either written a book or an article or both.

Example 2:

- **AUB**

Table A			**Table B**	
column 1	**column 2**		**column 1**	**column 2**
1	1		1	1
1	2		1	3

Table A ∪ B	
column 1	**column 2**
1	1
1	2
1	3

Operations From Set Theory Cont.

2. Set Difference (−)

- The result of set difference operation is tuples, which are present in one relation but are not in the second relation.
- **Notation − r − s**
- Finds all the tuples that are present in **r** but not in **s**.
- For this operation to be valid, the following conditions must hold −
- The attribute name of r has to match with the attribute name in s.
- The two-operand relations r and s should be either compatible or Union compatible.
- It should be defined relation consisting of the tuples that are in relation r, but not in s.

Operations From Set Theory Cont.

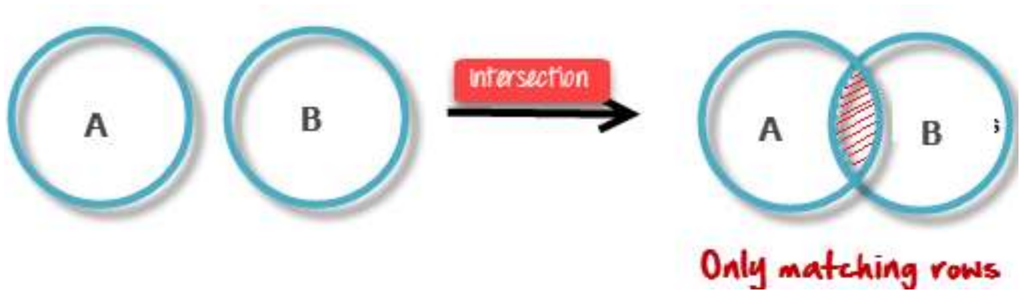
Example 1:

- $\prod_{\text{author}}$ (Books) − $\prod_{\text{author}}$ (Articles)
- **Output** − Provides the name of authors who have written books but not articles.

Operations From Set Theory Cont.

3. Intersection

- An intersection is defined by the symbol ∩
- A ∩ B
- Defines a relation consisting of a set of all tuple that are in both A and B. However, A and B must be union-compatible.



- **Example:**
- A ∩ B

Table A ∩ B	
column 1	column 2
1	1

Operations From Set Theory Cont.

4. Cartesian Product (X)

- Combines information of two different relations into one.
- **Notation** − r X s
- Where **r** and **s** are relations and their output will be defined as –
- r X s = { q t | q ∈ r and t ∈ s}

Example

- $\sigma_{\text{author = 'korth'}}$(Books X Articles)
- **Output** − Yields a relation, which shows all the books and articles written by ‘korth’.

Binary Relational (Join) Operations

- Join operation is essentially a cartesian product followed by a selection criterion.
- Join operation denoted by ⋈.
- JOIN operation also allows joining variously related tuples from different relations.
- **Types of JOIN:**
- Various forms of join operation are:
- Inner Joins:
  - Theta join
  - EQUI join
  - Natural join

Binary Relational (Join) Operations

- Outer join:
  - Left Outer Join
  - Right Outer Join
  - Full Outer Join

Binary Relational (Join) Operations

- **Inner Join:**
- In an inner join, only those tuples that satisfy the matching criteria are included, while the rest are excluded. Let's study various types of Inner Joins:

1. Theta Join:

- The general case of JOIN operation is called a Theta join. It is denoted by symbol **θ**
- Example
- $A \bowtie_{\theta} B$
- Theta join can use any conditions in the selection criteria.
- For example:
- $A \bowtie_{\text{A.column 2} > \text{B.column 2}} (B)$

Binary Relational (Join) Operations

2. EQUI join:

- When a theta join uses only equivalence condition, it becomes a equi join.
- For example:
- $A \bowtie_{A.\text{column } 2 = B.\text{column } 2} (B)$

A ⋈ A.column 2 = B.column 2 (B)	
column 1	column 2
1	1

- EQUI join is the most difficult operations to implement efficiently using SQL in an RDBMS and one reason why RDBMS have essential performance problems.

Binary Relational (Join) Operations

3. NATURAL JOIN (⋈)

- Natural join can only be performed if there is a common attribute (column) between the relations. The name and type of the attribute must be same.

Example:

- Consider the following two tables
- C ⋈ D

C	
Num	**Square**
2	4
3	9

D	
Num	**Cube**
2	8
3	27

C ⋈ D		
Num	**Square**	**Cube**
2	4	8
3	9	27

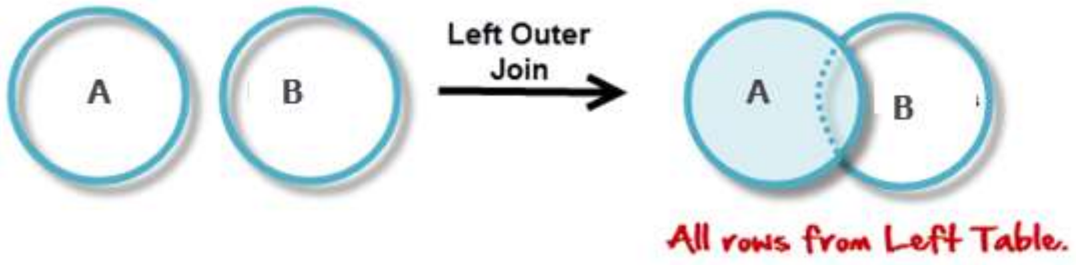
Binary Relational (Join) Operations

OUTER JOIN

- In an outer join, along with tuples that satisfy the matching criteria, we also include some or all tuples that do not match the criteria.

1. Left Outer Join(A⟕B)

- In the left outer join, operation allows keeping all tuple in the left relation.
- However, if there is no matching tuple is found in right relation, then the attributes of right relation in the join result are filled with null values



Binary Relational (Join) Operations

- Consider the following 2 Tables
- A⟕B

A	
Num	**Square**
2	4
3	9
4	16

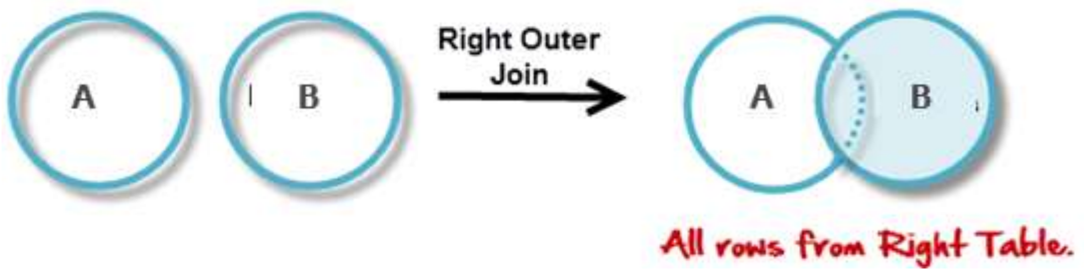
B	
Num	**Cube**
2	8
3	27
5	125

A ⟕ B			
Num	**Square**	**Num**	**Cube**
2	4	2	4
3	9	3	27
4	16	Null	Null

Binary Relational (Join) Operations

2. Right Outer Join: (A⟖B)

- In the right outer join, operation allows keeping all tuple in the right relation.
- However, if there is no matching tuple is found in the left relation, then the attributes of the left relation in the join result are filled with null values.



Binary Relational (Join) Operations

Example:

(A ⟖ B)

A ⟖ B			
Num	**Square**	**Num**	**Cube**
2	4	2	8
3	9	3	27
Null	Null	5	125

Binary Relational (Join) Operations

3. Full Outer Join: (A ⟗ B)

- In a full outer join, all tuples from both relations are included in the result, irrespective of the matching condition.
- **Example:**

A⟗B			
Num	**Square**	**Num**	**Cube**
2	4	2	8
3	9	3	27
4	16	Null	Null
Null	Null	5	75

Tuple Relational Calculus (TRC)

- Filtering variable ranges over tuples
- **Notation** − {T | Condition}
- Returns all tuples T that satisfies a condition.
- **For example** −
- { T.name | Author(T) AND T.article = 'database' }
- **Output** − Returns tuples with 'name' from Author who has written article on 'database'.
- TRC can be quantified. We can use Existential (∃) and Universal Quantifiers (∀).
- **For example** −
- { R | ∃T ∈ Authors(T.article='database' AND R.name=T.name)}
- **Output** − The above query will yield the same result as the previous one.

Tuple Relational Calculus (TRC)

- **Example:**
- **Student**

Roll	Name	Marks
32301	A	90
32302	B	95
32303	C	99

- Select all tuples from student table
- $\{ t \mid t \in Student \}$

- Select all tuples having marks less than 95
- $\{ t \mid t \in Student \wedge t[Marks] < 95 \}$

Roll	Name	Marks
32301	A	90

Tuple Relational Calculus (TRC)

Student

- **Example:**

Roll	Name	Marks
32301	A	90
32302	B	95
32303	C	99

- Select tuples for some attributes from student table
- Select Roll numbers for students
- { t | ∃ s ∈ (Students(Roll) = s[Roll])}

Roll
32301
32302
32303

- Select Roll number of Students having marks less than 95
- { t | ∃ s ∈ (Students(Roll) = s[Roll] ˄ s[Marks] < 95)}

Roll	Name	Marks
32301	A	90

Relational Algebra

Summary

Operation(Symbols)	Purpose
Select(σ)	The SELECT operation is used for selecting a subset of the tuples according to a given selection condition
Projection(π)	The projection eliminates all attributes of the input relation but those mentioned in the projection list.
Union Operation(∪)	UNION is symbolized by symbol. It includes all tuples that are in tables A or in B.
Set Difference(-)	- Symbol denotes it. The result of A - B, is a relation which includes all tuples that are in A but not in B.

Relational Algebra

Summary

Intersection(∩)	Intersection defines a relation consisting of a set of all tuple that are in both A and B.
Cartesian Product(X)	Cartesian operation is helpful to merge columns from two relations.
Inner Join	Inner join, includes only those tuples that satisfy the matching criteria.
Theta Join(θ)	The general case of JOIN operation is called a Theta join. It is denoted by symbol θ.
EQUI Join	When a theta join uses only equivalence condition, it becomes a equi join.

Relational Algebra

Summary

Natural Join(⋈)	Natural join can only be performed if there is a common attribute (column) between the relations.
Outer Join	In an outer join, along with tuples that satisfy the matching criteria.
Left Outer Join( )	In the left outer join, operation allows keeping all tuple in the left relation.
Right Outer join()	In the right outer join, operation allows keeping all tuple in the right relation.
Full Outer Join()	In a full outer join, all tuples from both relations are included in the result irrespective of the matching condition.

Thank You ..!!