CSAI 302

# Advanced Database Systems

## Lec 01

## Advanced DBMS and Database System Architectures

# Outline

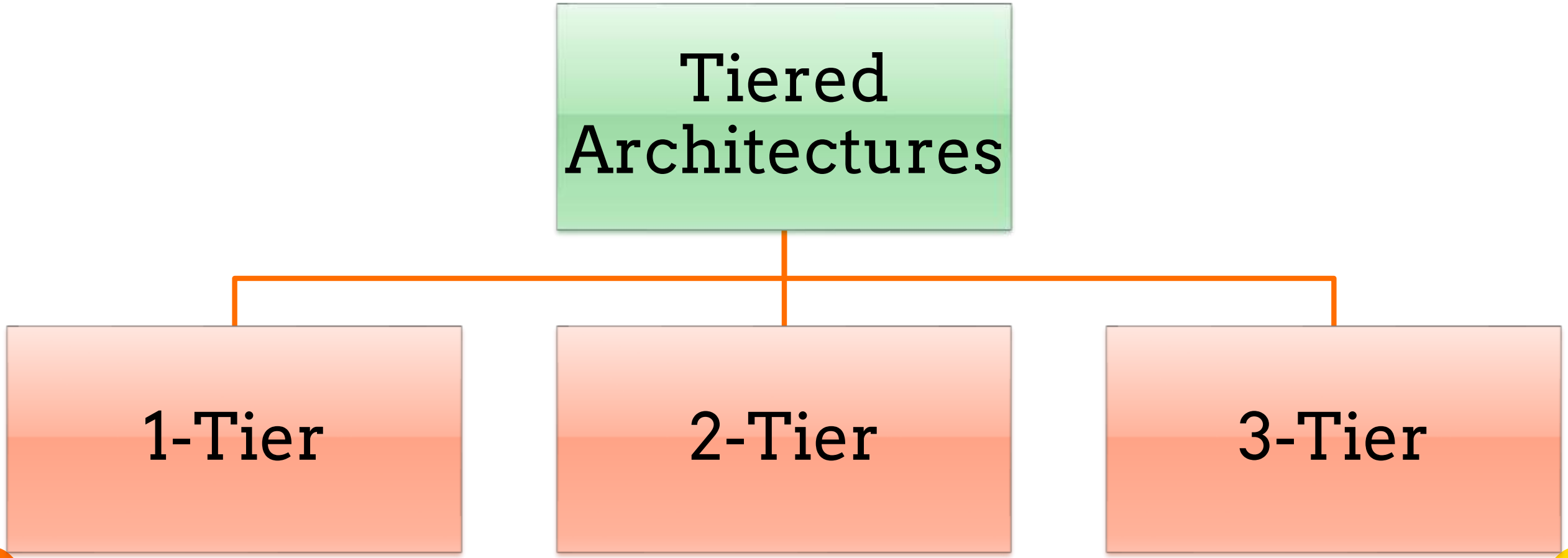
- Database system essentials
- DBMS Architecture Fundamentals
- Evolution of Database System Architectures
- Data Models
- OLTP vs. OLAP Workload Characteristics
- Course Details

# Database system essentials

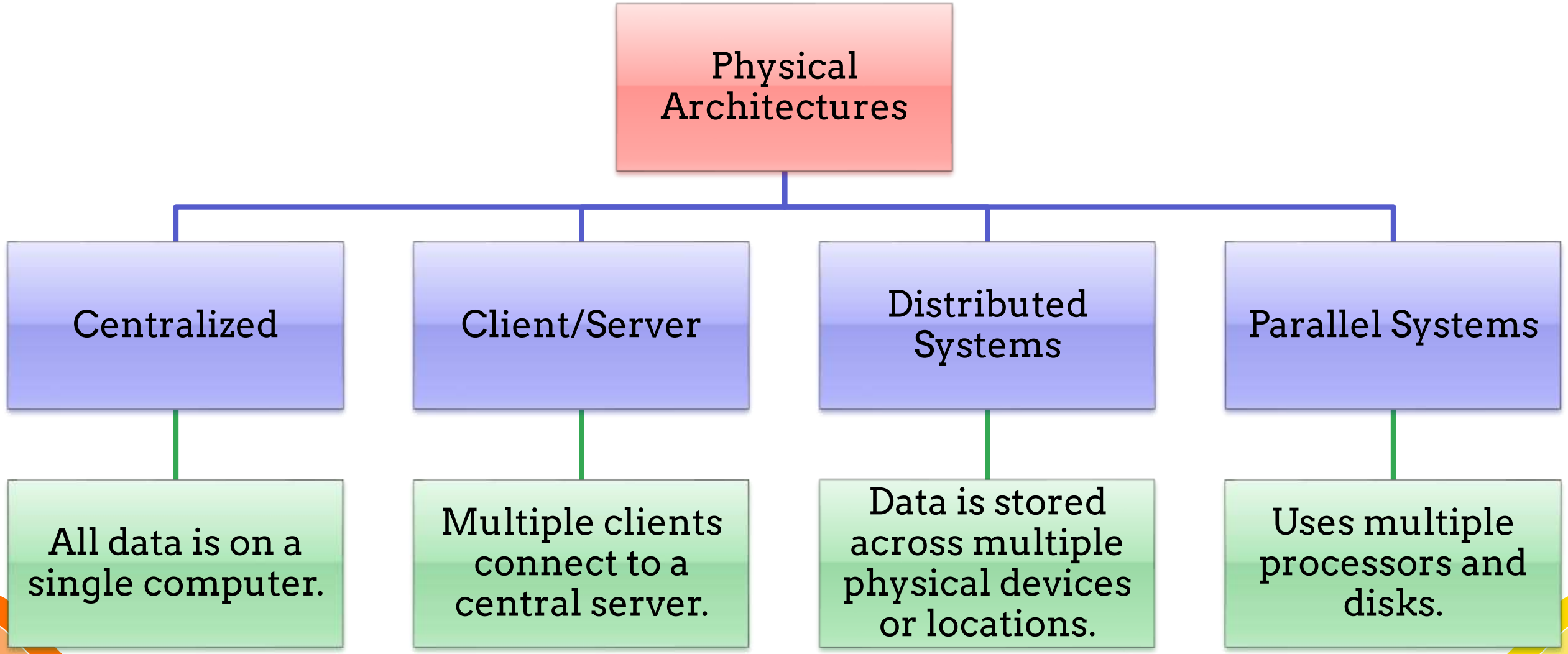
# Database System Architecture

- The overall design and structure of a database system that
  - outlines ***how data is stored***, managed, and accessed.
  - defines ***the components*** of the system
    - how they ***interact***, and how ***data flows*** between them.
- A well-defined architecture ensures Data:
  - Integrity
  - Efficiency
  - Security
  - Scalability.

# Database Architecture Types



# Database Architecture Types

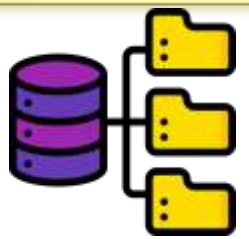


# Schema

- The blueprint for how data is organized in a database
- defining elements like
  - ★tables, fields, data types, and their relationships
- but not the actual data itself.
- ***This defines the structure of data for a data model.***

# Three-Schema Architecture

| External Schema | Conceptual Schema | Internal Schema |
| --- | --- | --- |
| View Level | Logical Level | Physical Level |
| Customized data views | Logical structure of the database | Physical storage structures |



**Hospital management system**

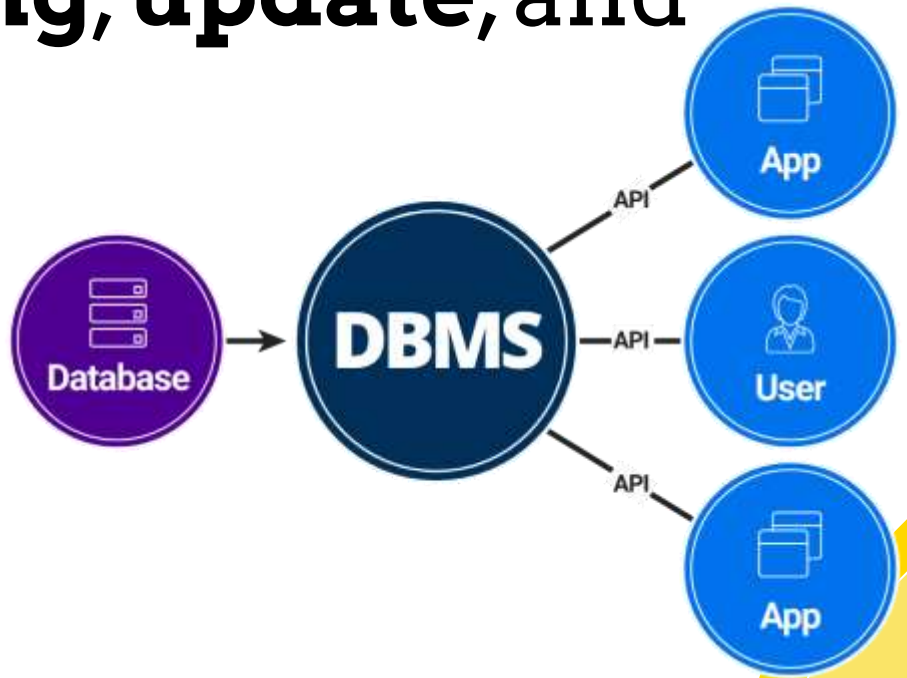
- patient medical records and treatment history
- patient demographic information and insurance details.

- Tables
- Relationships
- Integrity constraints

- B-tree indexes for frequent lookups
- Hash partitioning for parallel processing
- Columnar storage for analytical queries.

# DBMS

- A software that allows ***applications*** to ***store*** and ***analyze*** information in a database.
- A general-purpose DBMS is designed to allow the
  - **definition, creation, querying, update,** and **administration**
    - of databases
  - in **accordance with**
    - some data model.



# DBMS Core Components

# DBMS Key Components

## Database Engine

- Handles ***data processing***
- Enables efficient access and manipulation.

## Database Schema

- The ***blueprint*** or logical structure of the database.

## Query Processor

- Interprets and executes ***user queries***.
- Translates them into actions for the database.

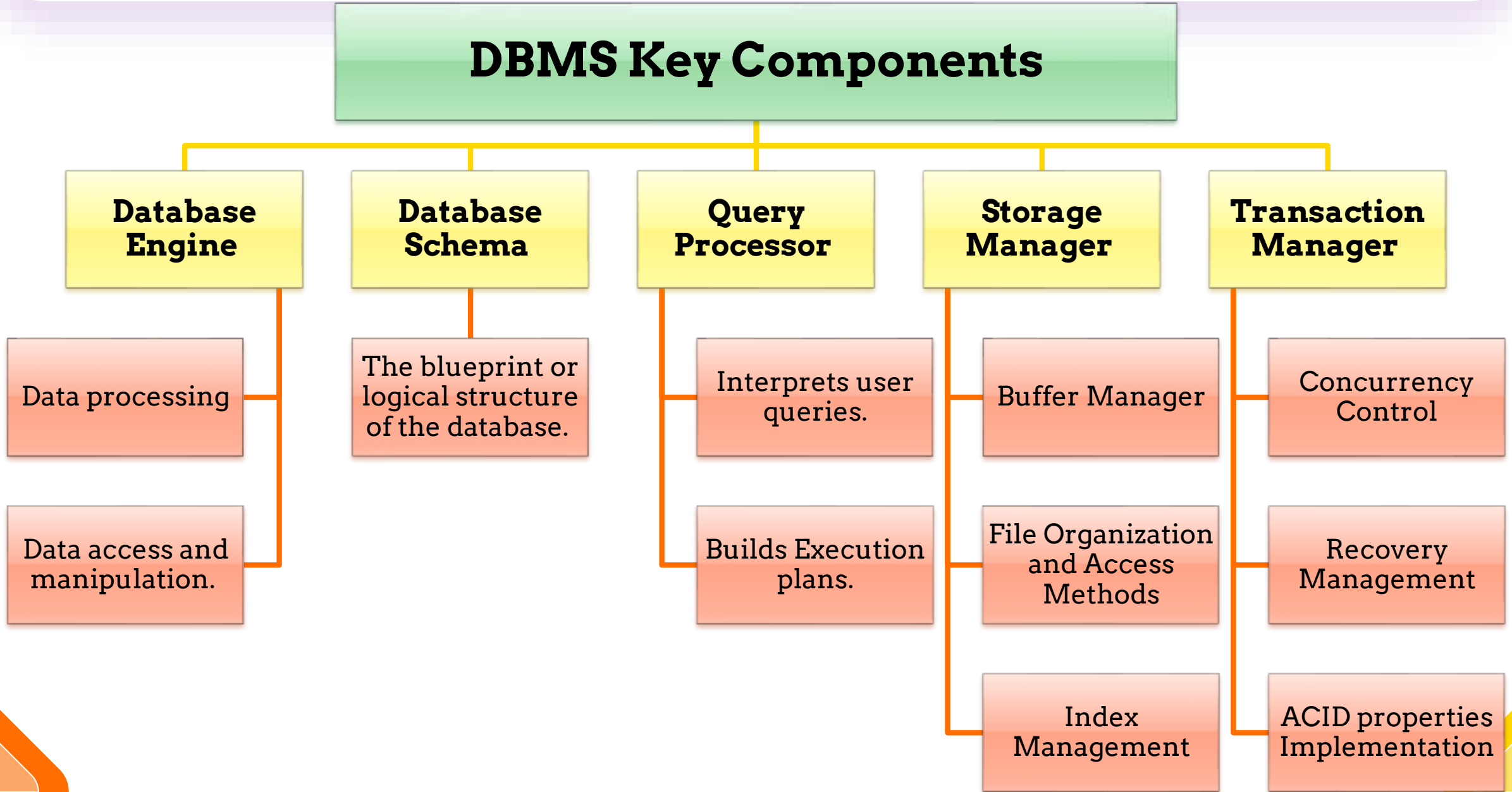
## Storage Manager

- Manages the ***physical storage*** of data on various media
- Handles ***data access***.

## Transaction Manager

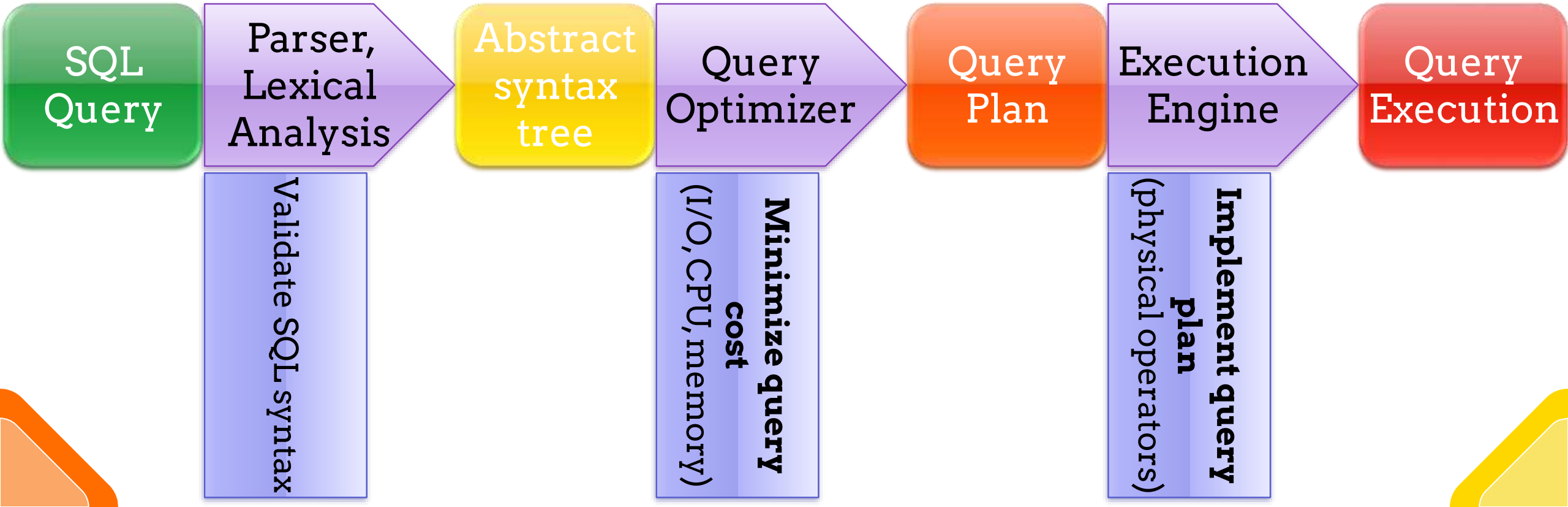
- Ensures the ***integrity and consistency*** of data by
  - Managing ***transactions***
  - Enforcing ***rules*** like ACID properties.

# DBMS Key Components



# Query Processing Engine

- Transforms high-level SQL statements into efficient execution plans through several critical phases.



# Evolution of Database System Architectures

# Historical Progression

- The evolution of database architectures reflects the changing demands of data processing, from simple record-keeping to complex analytical workloads handling massive datasets.
- Architectural evolution Drivers
  - Scalability Requirements
  - Data Complexity
  - Performance Expectations
  - Reliability Demands

# File-Based Systems (1960s–1970s)

- Early computing relied on flat file systems
- Each application maintained its own data files.
- **I**ntegrated **D**ata **S**tore
  - Network data model.
  - Tuple-at-a-time queries.
  - Transaction-oriented
  - Dedicated working storage area for each record type
- This approach suffered from data redundancy, inconsistency, and lack of standardized access methods.

# Hierarchical and Network Models (70s-80s)

- IBM's Information Management System (IMS) introduced hierarchical structures resembling tree organizations,
- The Conference on Data Systems Languages (CODASYL) developed network models allowing more complex relationships.
- These systems provided better data organization but required programmers to navigate complex pointer structures manually.
  - Hierarchical data model.
  - Programmer-defined physical storage format.
  - Tuple-at-a-time queries.

# Relational Model (80s–90s)

- Relational model introduced mathematical foundations based on set theory and first-order predicate logic.
- Allowed users to focus on what data they wanted rather than how to retrieve it
- leading to the development of SQL and modern RDBMS systems like Oracle, DB2, and SQL Server.
- Database abstraction to avoid this maintenance:
  - Store database in simple data structures.
  - Access data through high-level language.
  - Physical storage left up to implementation.

IBM DB2
ORACLE
Informix
TANDEM
SYBASE
TERADATA
INGRES
InterBase

# Object-Oriented-Systems (2000s)

- As applications became more complex, the need to store complex data types led to object-oriented databases and object-relational extensions.
- PostgreSQL exemplifies this evolution, supporting both traditional relational operations and complex data types like arrays, JSON, and user-defined types.




# 2000s – Internet boom

- All the big players were heavyweight and expensive.
- Open-source databases were missing important features.
- Many companies wrote their own custom middleware to scale out database across single-node DBMS instances.

# 2000s – Data warehouses

- Rise of the special purpose **OLAP DBMSs**.
  - Distributed / Shared-Nothing
  - Relational / SQL
  - Usually closed-source.
- Significant performance benefits from using ***columnar data storage model***.






# 2000s – NoSQL Systems

- Focus on high-availability & high-scalability:
  - Schema-less (i.e., "Schema Last")
  - Non-relational data models (document, key/value, etc)
  - No ACID transactions
  - Custom APIs instead of SQL
  - Usually open-source

# 2010s – NewSQL

- Provide same performance for **OLTP workloads** as **NoSQL** DBMSs without giving up **ACID**:
  - Relational / SQL
  - Distributed
  - Usually closed-source

# 2010s – Hybrid systems

- **H**ybrid **T**ransactional-**A**nalytical **P**rocessing.
- Execute fast **OLTP** like a **NewSQL** system while also executing complex **OLAP** queries like a **data warehouse** system.
  - ★ Distributed / Shared-Nothing
  - ★ Relational / SQL
  - ★ Mixed open/closed-source.



splice MACHINE



# 2010s – Cloud systems

- First database-as-a-service (**DBaaS**) offerings were "containerized" versions of existing DBMSs.
- There are new DBMSs that are designed from scratch explicitly for running in a cloud environment.

# 2010s – Shared-disk engines

- Instead of writing a custom storage manager, the DBMS relies on third-party distributed storage (object stores) .
  - Scale execution layer independently of storage.
  - Favors log-structured approaches.
- This is what most people think of when they talk about a ***data lake***.



# 2010s – Stream processing

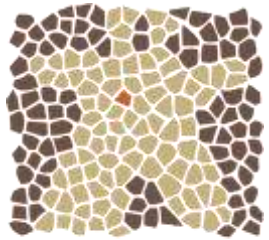
- Execute continuous queries on streams of tuples.
- Extend processing semantics to include notion of windows.
- Often used in combination of batch-oriented systems in a ***lambda architecture*** deployment.






# 2010s – Graph systems

- Systems for storing and querying graph data.
- Their main advantage over other data models is to provide a graph-centric query API
  - ★ Recent research demonstrated that is unclear whether there is any benefit to using a graph-centric execution engine and storage manager.

# 2010s – Timeseries systems

- Specialized systems that are designed to store timeseries / event data.
- The design of these systems make deep assumptions about the distribution of data and workload query patterns.






# 2020s - LAKEHOUSE SYSTEMS

- Middleware for data lakes that adds support for  better schema control / versioning with transactional CRUD operations.
  - Store changes in row-oriented log-structured files with  indexes.
  - Periodically compact recently added data into read-only columnar files.

databricks Apache hudi ICEBERG snowflake

# Data Models

*A collection of concepts for describing the data in database.*

# Common Data Models

## Relational

- (Most DBMSs)

## Key/Value

- (NoSQL)

## Graph

- (NoSQL)

## Document / XML / Object

- (NoSQL)

## Wide-Column / Column-family

- (NoSQL)

## Array / Matrix / Vector

- (Machine Learning)

## Hierarchical / Network / Multi-Value

- (Obsolete/ Rare)

# Relational Model

# Relational Model

- The relational model defines a database abstraction based on relations to avoid maintenance overhead.
- It has three key ideas:
  - Store database in simple data structures (relations)
  - Physical storage left up to the DBMS implementation
  - Access data through a high-level (declarative) language, where the DBMS figures out best execution strategy

# Relational Model concepts

## Structure

The definition of relations and their contents independent of their physical representation.

Each relation has a set of attributes. Each attribute has a domain of values.

## Integrity

Ensure the database's contents satisfy certain constraints.

An example of a constraint would be that the age of a person cannot be a negative number.

## Manipulation

An interface for accessing and modifying a database's contents (e.g. SQL, Dataframes)

# Relational Model definitions

- Relation
- Tuple
- Primary key
- Foreign key
- Constraint

# Document Model

# Document Data Model

- A collection of record documents containing a hierarchy of named field/value pairs.
  - A field's value can be either a scalar type, an array of values, or another document.
  - Modern implementations use JSON.
    - Older systems use XML or custom object representations.
- Avoid "relational-object impedance mismatch" by tightly coupling objects and database.

# Relational vs. Document

**SUPPLIER**
(sno, sname, scity)

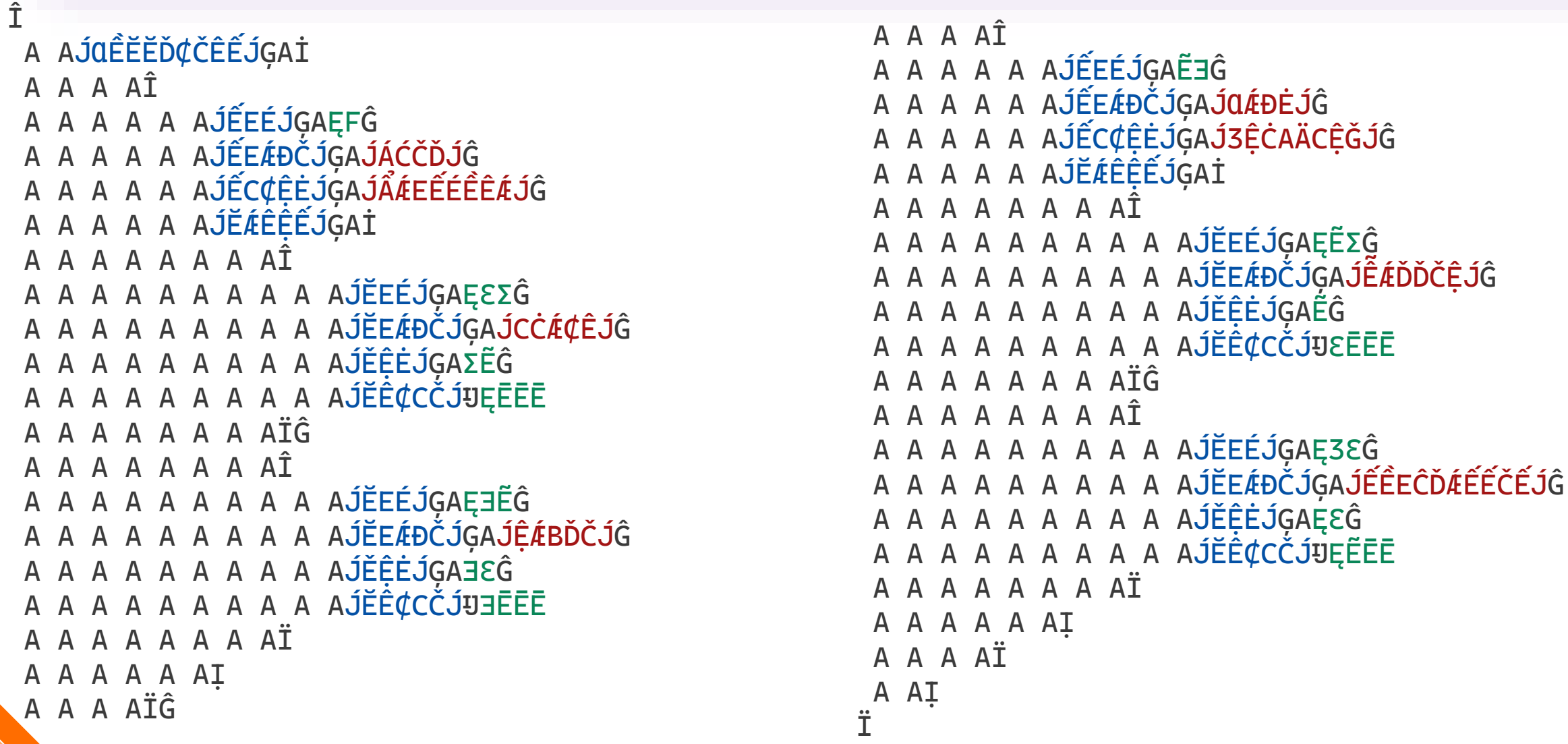
**SUPPLY**
(sno, pno, qty, price)

**PART**
(pno, pname)

**SUPPLIER**
(sno, sname, scity, sstate, parts[])

**PART**
(pno, pname, qty, price)

# JSON file



# JSON table

| Suppliers | sno | 17 | | | 53 | | |
|---|---|---|---|---|---|---|---|
| | sname | Adel | | | Samy | | |
| | scity | Mansoura | | | 6th Oct. | | |
| | parts | pno | 124 | 135 | pno | 154 | 162 |
| | | pname | chair | table | pname | wallet | sunglasses |
| | | qty | 45 | 32 | qty | 5 | 12 |
| | | price | 1000 | 3000 | price | 2000 | 1500 |

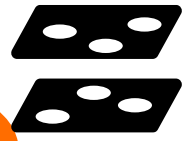
# XML file

```
ŞHËÐĎAĚČÊĚ¢ÉEŞJĘĞĒJAČECÉĆ¢ECŞJĄĀẢÌGJHŜ
ŞÊÉÉĘŜ
 A AŞΩỀĔĔĎ¢ČÊĚŜ
 A A A AŞĚEÉŜĘFảãĚEÉŜ
 A A A AŞĚEÆÐČŜÁĆČĎảãĚEÆÐČŜ
 A A A AŞĚC¢ĘĖŜẢÆEĚÉỀÊÁảãĚC¢ĘĖŜ
 A A A AŞĔÆÊĘĚŜ
 A A A A A AŞĔEÉŜĘƐΣảãĔEÉŜ
 A A A A A AŞĔEÆÐČŜCĊÁ¢Êảã̃ĔEÆÐČŜ
 A A A A A AŞĔĘĖŜΣẼảãĔĘĖŜ
 A A A A A AŞĔÊ¢CČŜĘĒĒĒảãĔÊ¢CČŜ
 A A A Aảã̃ĔÆÊĘĚŜ
 A A A AŞĔÆÊĘĚŜ
 A A A A A AŞĔEÉŜĘƎẼảãĔEÉŜ
 A A A A A AŞĔEÆÐČŜĘÁBĎČảãĔEÆÐČŜ
 A A A A A AŞĔĘĖŜƎƐảãĔĘĖŜ
 A A A A A AŞĔÊ¢CČŜƎĒĒĒảãĔÊ¢CČŜ
 A A A Aảã̃ĔÆÊĘĚŜ
 A Aảã̃ΩỀĔĔĎ¢ČÊĚŜ
 A AŞΩỀĔĔĎ¢ČÊĚŜ
 A A A AŞĚEÉŜẼƎảãĚEÉŜ
 A A A AŞĚEÆÐČŜΩÁÐĖảãĚEÆÐČŜ
 A A A AŞĚC¢ĘĖŜƷĘĊAÄCĘĞảãĚC¢ĘĖŜ
 A A A AŞĔÆÊĘĚŜ
 A A A A A AŞĔEÉŜĘẼΣảãĔEÉŜ
 A A A A A AŞĔEÆÐČŜẼÆĎĎČĘảãĔEÆÐČŜ
 A A A A A AŞĔĘĖŜẼảãĔĘĖŜ
 A A A A A AŞĔÊ¢CČŜƐĒĒĒảãĔÊ¢CČŜ
 A A A Aảã̃ĔÆÊĘĚŜ
 A A A AŞĔÆÊĘĚŜ
 A A A A A AŞĔEÉŜĘ3ƐảãĔEÉŜ
 A A A A A AŞĔEÆÐČŜẼỀEĈĎÆẼĚČẼảãĔEÆÐČŜ
 A A A A A AŞĔĘĖŜĘƐảãĔĘĖŜ
 A A A A A AŞĔÊ¢CČŜĘẼĒĒảãĔÊ¢CČŜ
 A A A Aảã̃ĔÆÊĘĚŜ
 A Aảã̃ΩỀĔĔĎ¢ČÊĚŜ
ảã̃ÊÉÉĘŜ
```

# Vector data Model

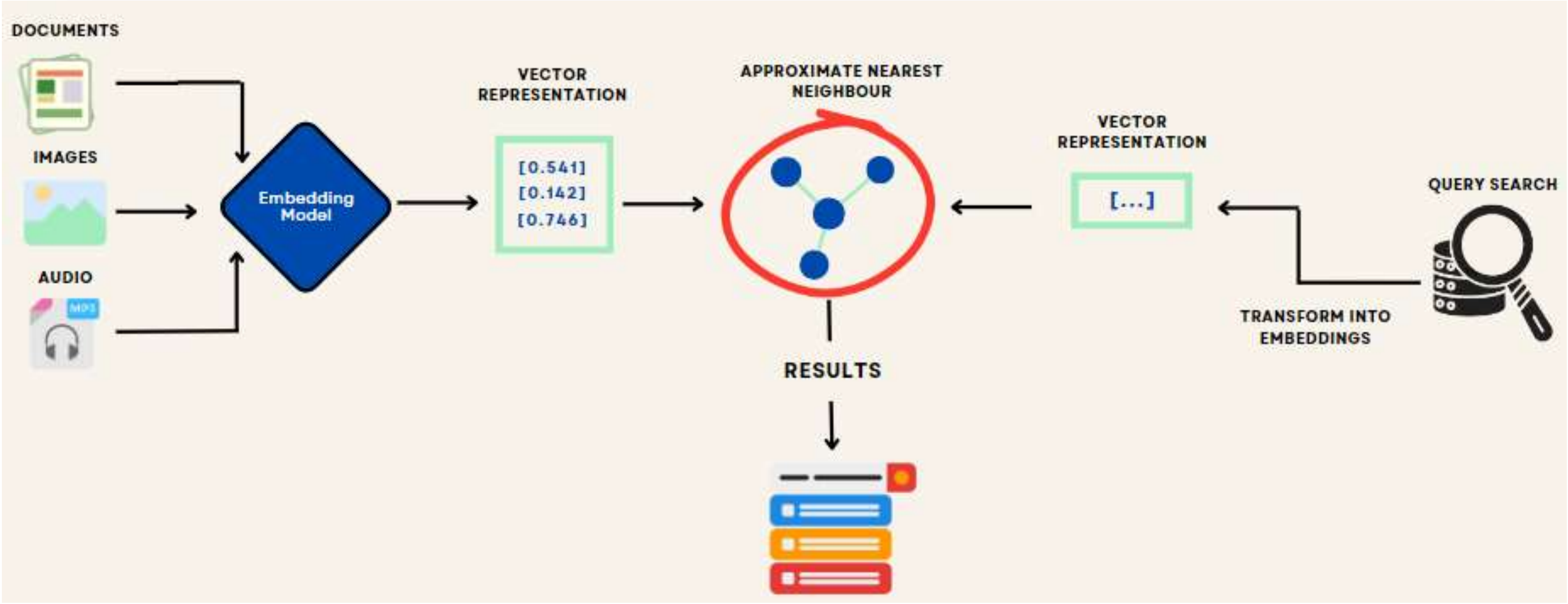
# Vector data model

- One-dimensional arrays used for nearest-neighbor search (exact or approximate).
  - ★ Used for semantic search on embeddings generated by ML-trained transformer models (think ChatGPT).
  - ★ Native integration with modern ML tools and APIs (e.g., LangChain, OpenAI).
- At their core, these systems use specialized indexes to perform NN searches quickly.



turbopuffer <(°O°)>

# Content-Based Retrieval Workflow Using Vector Space Embeddings



# OLTP vs. OLAP Workload Characteristics

# Online Transaction Processing (OLTP)

# OLTP

- OLTP systems handle operational business processes:
  - requiring fast, consistent transaction processing
  - with high concurrency levels.
- A class of systems designed for managing real-time, day-to-day business transactions.
- Examples
  - ATM transactions
  - online banking
  - credit card processing
  - e-commerce purchases.
- These systems are characterized by
  - high volume
  - short transactions
  - need for quick response times.

# Characteristics and Requirements

- OLTP workloads involve
  - short-duration transactions
  - accessing small amounts of current data.
- Response times must be predictable and fast, usually under 100 milliseconds.
- Consider an e-commerce checkout process
  - Multiple users simultaneously place orders, update inventory, and process payments.
  - Each transaction involves few records but requires immediate consistency to prevent overselling products.

# Data Access Patterns

- Use indexed access to retrieve specific records or small record sets.
- Queries involve
  - primary key lookups
  - simple range scans.
- Update operations modify
  - individual records
  - small groups of related records.
- The workload mix typically includes
  - 70-80% read operations
  - 20-30% write operations
    - distributed across many different tables and records.

# System Optimizations

- ***Row-oriented storage*** optimizes OLTP performance by storing complete records together, minimizing I/O for transactions accessing multiple attributes of the same entity.
- ***B+ tree indexes*** provide efficient access paths for both equality and range queries.
- Connection pooling and prepared statements reduce overhead for repetitive operations.
- Partitioning strategies often use range or hash partitioning based on transaction date or customer ID to distribute load evenly.

# Online Analytical Processing (OLAP)

# OLAP

- OLAP systems support
  - complex analytical queries and business intelligence applications
  - requiring sophisticated data aggregation and analysis capabilities.
- OLAP systems enable users to analyze large datasets from multiple perspectives.

# OLAP Example

- Analyzing sales data for a retail company using an OLAP cube.
  - The cube could organize sales by
    - product, time, and location,
  - allowing users to drill down into specific
    - regions, products, or time periods
  - to identify
    - trends, such as which products are most popular in a particular city during a specific month.

# Characteristics and Requirements

- OLAP workloads involve ***long-running*** queries that scan large portions of ***historical data*** to compute ***aggregations***, ***trends***, and ***statistical summaries***.
- *Response times* range from seconds to minutes, but *query complexity* and data volume are substantially higher than OLTP systems.
- Consider a retail analytics query calculating seasonal sales trends across multiple product categories, geographical regions, and customer segments over several years.

# Data Access Patterns

- OLAP queries typically involve ***full table scans*** or large range scans, often accessing millions or billions of records.
- ***Aggregation operations*** like SUM, COUNT, and AVG are common, frequently combined with GROUP BY clauses creating multi-dimensional summaries.
- ***Joins*** often involve large fact tables with smaller dimension tables in star or snowflake schema configurations.

# System Optimizations

- ***Columnar storage*** dramatically improves OLAP performance by storing each column separately, enabling efficient compression and reducing I/O for queries accessing few columns.
- ***Bitmap indexes*** support complex Boolean queries common in analytical workloads.
- ***Materialized views*** pre-compute common aggregations, trading storage space for query performance.
- ***Parallel processing*** distributes query execution across multiple processors or machines.

# Hybrid Approaches

# Hybrid Approaches

- HTAP (Hybrid Transactional/Analytical Processing)
  - Modern systems increasingly support mixed workloads combining OLTP and OLAP requirements.
  - SAP HANA exemplifies this approach using in-memory storage and columnar organization to support both transaction processing and real-time analytics on the same data.
- Data Replication Strategies
  - Many organizations maintain separate OLTP and OLAP systems connected through data replication mechanisms.
  - Change Data Capture (CDC) identifies and propagates OLTP system changes to analytical systems with minimal impact on operational performance.

# System Optimizations

# Physical Design Optimizations

- Index Selection and Design
- Partitioning Strategies

# Query Optimization Techniques

- Cost-Based Optimization
- Advanced Optimization Techniques

# Memory and Caching Strategies

- Buffer Pool Management
- Query Result Caching

# Course Details

# Course plan

| W | LECTURE | LAB | Coursework |
|---|---|---|---|
| 1 | Advanced DBMS and Database System Architectures | ▪ Environment setup | |
| 2 | Storage Management | ▪ Implementing a buffer pool manager.<br>▪ Benchmarking different buffer pool strategies under various workloads | **Assignment** 1<br>▪ Storage Manager and Buffer Pool Implementation with Different Replacement Policies |
| 3 | Storage Models - Row-Oriented and Columnar Systems | ▪ Implement storage engines.<br>▪ Convert row-store to columnar | |
| 4 | Indexing Structures and Implementation | ▪ Build a B+ tree index;<br>▪ Implement extendible hashing. | **Assignment** 2<br>▪ Indexing structures (b+ trees and lsm trees) |
| 5 | Advanced Programming and Query Processing | ▪ Build inventory-auditing triggers.<br>▪ Atomic SPs with error handling<br>▪ Implementing join algorithms and vectorized execution operators | **Quiz 1** |
| 6 | Query Optimization | ▪ Implementing a cost-based query optimizer.<br>▪ Tune query plans. | **Assignment** 3<br>▪ Query Processing and Optimization |
| 7 | Transaction Processing and Concurrency Control | ▪ Deadlock simulation; lock contention analysis.<br>▪ Analyzing isolation level bugs in production systems | |
| 8 | **Mid Term Exam** | | |

# Course plan

| W | LECTURE | LAB | Coursework |
|---|---|---|---|
| 9 | Crash Recovery | ▪ Implementing WAL and recovery procedure | **Assignment** 4<br>▪ Transaction Processing with Concurrency Control and Recovery |
| 10 | Parallel and Distributed Database Architectures | ▪ Implement cross-shard ACID transactions<br>▪ Distributed transaction coordination | |
| 11 | NoSQL Databases and Big Data Storage Systems | ▪ Use MongoDB's query language to explore document-based databases. | **Assignment** 5<br>▪ Implement NoSQL queries |
| 12 | Advanced NoSQL Models. | ▪ Implement core components of different NoSQL models | |
| 13 | Caching Strategies, ORM Integration, and Modern Database Trends | ▪ Session data caching<br>▪ API caching<br>▪ E-commerce platforms | **Assignment** 6<br>▪ Build a Simple Web-site Integrating an ORM Framework |
| 14 | Database Security | ▪ Implement column-level encryption; audit log triggers | |
| 15 | **Course Wrap up** | **Assignments Presentation as a Whole Project** | |

# Assessment Strategy

| Assessment Method | Degrees |
|---|---|
| Quizzes (best 2 of 3) (week 5, 10, 14) | 10 |
| Midterm Exam (week 8) | 20 |
| Lecture participation | 8 |
| Lab participation | 7 |
| Term Project | 15 |
| Final Exam | 40 |
| Total | 100 |

# THANK YOU