Database Theory

Discuss database applications examples in different sectors, such as enterprise information, banking and finance, universities, airlines, telecommunications and navigation systems.

- **Enterprise Information**
  - **Sales:** customers, products, purchases
  - **Accounting:** payments, receipts, assets
  - **Human Resources:** Information about employees, salaries, payroll taxes.
- **Manufacturing:** management of production, inventory, orders, supply chain.
- **Banking:** customer information, accounts, loans, banking transactions, and credit card transactions
- **Finance:** sales and purchases of financial instruments (e.g., stocks and bonds; real-time market data)
- **Universities:** 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
- **Document databases**
  - **Navigation systems:** For maintaining the locations of varies places of interest along with the exact routes of roads, train systems, buses, etc.

Discuss the purpose of Database Systems with relevant examples.

- 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

- Atomicity of updates
  - Failures may leave database in an inconsistent state with partial updates carried out.
  - e.g. 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. E.g. Two people reading a balance (say 100) and updating it by withdrawing money (say 50 each) at the same time.
- Security problems
  - Hard to provide user access to some, but not all, data

Summarize the concepts of super key, candidate key, primary key and foreign key.

- Super key : K is a super-key of R if values for K are sufficient to identify a unique tuple of each possible relation $r(R)$.
  e.g: `{ID}` and `{ID, name}` are both super-keys of instructor.
- Candidate key : Candidate key is a minimal super-key. (nullable)
  e.g: `{ID}` is a candidate-key of instructor.
- Primary key : Primary key is a candidate key selected from among the candidate keys.
- Foreign key : Foreign key links two tables together via the primary key.
  e.g: `dept_name` in instructor is a foreign key from instructor referencing `department`.

Summarize the domain types in SQL with relevant examples.

char(n)	:	Fixed length character string, with user-specified length n.
varchar(n)	:	Variable length character strings, with user-specified maximum length n.
int	:	Integer (a finite subset of the integers that is machine-dependent).
smallint	:	Small integer (a machine-dependent subset of the integer domain type).
numeric(p,d)	:	Fixed point number, with user-specified precision of p digits, with d digits to the right of decimal point. (ex., numeric(3,1), allows 44.5 to be stores exactly, but not 444.5 or 0.32)
real, double precision	:	Floating point and double-precision floating point numbers, with machine-dependent precision.
float(n)	:	Floating point number, with user-specified precision of at least n digits.

Discuss weak entity set with an appropriate example.

A weak entity set is one whose existence is dependent on another entity, called its identifying entity.

Instead of associating a primary key with a weak entity, we use the identifying entity, along with extra attributes called discriminator to uniquely identify a weak entity.

Summarize the following symbols in relational algebra with proper examples:

Selection	σ	: selects tuples that satisfy a given predicate. $\sigma_p(r)$ → e.g: $\sigma_{\text{dept\_name} = \text{"CSE"}}$ (student)
Projection	Π	: retrieve specific attributes from a relation, eliminating duplicates. $\Pi_{A_1, A_2, \ldots A_k}(r)$ → e.g: $\Pi_{\text{ID, name}}$ (student)
Union	$\cup$	: combines tuples of same arity and attribute domains from two relations. $r \cup s$ → e.g: $\sigma_{\text{dept\_name} = \text{"CSE"}}$ (course) $\cup$ $\sigma_{\text{dept\_name} = \text{"EEE"}}$ (course)
Intersection	$\cap$	retrieve tuples that are in both relations. $r \cap s$ → e.g: $\sigma_{\text{dept\_name} = \text{"CSE"}}$ (course) $\cap$ $\sigma_{\text{dept\_name} = \text{"EEE"}}$ (course)
Set difference	$-$	: retrieve tuples that are in one relation but are not in another. $r - s$ → e.g: $\sigma_{\text{dept\_name} = \text{"CSE"}}$ (course) $-$ $\sigma_{\text{dept\_name} = \text{"EEE"}}$ (course)
Cartesian Product	$\times$	: combines each tuple of one relation with every tuple of another relation. $\sigma_\theta (r \times s)$ → e.g: $\sigma_{\text{student.ID} = \text{takes.ID}}$ (student × takes)
Natural Join	$\bowtie$	: Equivalent to Cartesian Product operation $r \bowtie_\theta s$ → e.g: student $\bowtie_{\text{student.ID} = \text{takes.ID}}$ takes

Explain the logical schema, physical schema, instance and physical data independence in database systems.

Schema:

– Schema is the logical structure of the database.

Logical Schema	: the overall logical structure of the database
Physical schema	: the overall physical structure of the database
Instance	: is a snapshot of the data in the database at a given instant in time

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.

DDL vs DML

DDL	DML
It stands for Data **Definition** Language.	It stands for Data **Manipulation** Language.
It is used to create database schema and can be used to define some constraints as well.	It is used to add, retrieve, or update the data. DML also known as query language
It basically defines the column (Attributes)	It adds or updates the row of the table.
Basic commands present in DDL are CREATE, DROP, RENAME, ALTER, etc.	BASIC commands present in DML are UPDATE, INSERT, MERGE etc.

DML classes:

Pure – used for proving properties about computational power and for optimization

- Relational Algebra
- Tuple relational calculus
- Domain relational calculus

Commercial – used in commercial systems

- SQL is the most widely used commercial language

DML types:

- 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.

Distinguish between centralized, client-server, parallel and distributed databases.

Database Architecture

1. Centralized databases
   - One to a few cores, shared memory
2. Client-server,
   - One server machine executes work on behalf of multiple client machines.
3. Parallel databases
   - Many core shared memory
   - Shared disk
   - Shared nothing
4. Distributed databases
   - Geographical distribution
   - Schema/data heterogeneity

Database Application Architecture

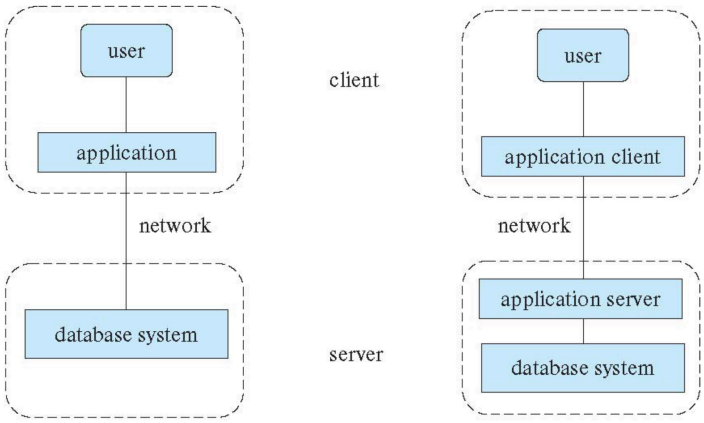
Summarize the features of two-tier and three-tier architectures with relevant examples.

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.



(a) Two-tier architecture (b) Three-tier architecture

Database Users:

Naive users	: End user. Interact with database using previously written application.
Application programmers	: Computer professionals who write application programs.
Sophisticated users	: Interact with the system without writing programs using SQL.
Specialized users	: write specialized database applications that do not fit into the traditional data-processing framework. For example, CAD, graphic data, audio, video.

Relational Algebra:

A procedural language consisting of a set of operations that take one or two relations as input and produce a new relation as their result.

Normalization

Database normalization, is the process of restructuring a relational database in accordance with a series of so-called normal forms in order to reduce data redundancy and improve data integrity.

Why is normalization needed? What anomalies can arise if data is not normalized?

Normalization is needed -

- To free the collection of relations from undesirable insertion, update and deletion dependencies.
- To reduce the need for restructuring the collection of relations, as new types of data are introduced, and thus increase the life span of application programs.
- To make the relational model more informative to users.
- To make the collection of relations neutral to the query statistics, where these statistics are liable to change as time goes by.

The primary anomalies that can arise are -

- **Update Anomaly:** An update anomaly occurs when changes to data are not consistently reflected across all instances where that data appears. This can happen because redundant data is stored in multiple places within the database.
- **Insertion Anomaly:** An insertion anomaly occurs when certain data cannot be inserted into the database without the presence of other data.
- **Deletion Anomaly:** A deletion anomaly occurs when deleting data unintentionally removes additional data leads to unintended loss of information.

Write down the important conditions for each of the 3 normalization methods: 1NF, 2NF, 3NF

A database in first normal form (1NF) if -

- A table must not contain composite or multi-valued attribute
  - each attribute must contain only atomic (indivisible) value from its domain.
- There should be no repeating groups within individual tables
  - each set of related data should be stored in its own table.
- Each row must be unique and identifiable
  - typically ensured by introducing a primary key.

Designs that Violate 1NF

ID	Name	Courses
1	Alice	Math, Science
2	Bob	Math, English, History

Designs that Comply with 1NF

ID	CourseID	Name	Courses
1	101	Alice	Math
1	102	Alice	Science
2	101	Bob	Math
2	103	Bob	English
2	104	Bob	History

A database in second normal form (2NF) if -

- It is in first normal form.
- Every non-prime attribute must be fully functionally dependent on the whole primary key.
  - non-prime attributes cannot be dependent on just a part of a composite primary key.
- Each table must contain data about only one type of thing

Designs that Violate 2NF

ID	CourseID	StudentName	CourseName	InstructorName
1	101	Alice	Math	Prof. Smith
1	102	Alice	Science	Prof. Brown
2	101	Bob	Math	Prof. Smith

Designs that Comply with 2NF

ID	Name
1	Alice
2	Bob

CourseID	CourseName	InstructorName
101	Math	Prof. Smith
102	Science	Prof. Brown

ID	CourseID
1	101
1	102
2	101

A database in third normal form (3NF) if -

- It is in second normal form.
- There should be no transitive functional dependency.
  - A non-prime attribute should not depend on another non-prime attribute.

Designs that Violate 3NF

ID	CourseID	StudentName	CourseName	InstructorName	InstructorContact
1	101	Alice	Math	Prof. Smith	01234-567895
1	102	Alice	Science	Prof. Brown	01234-789556
2	101	Bob	Math	Prof. Smith	01234-957856

Designs that Comply with 3NF

ID	Name
1	Alice
2	Bob

CourseID	CourseNam	InstructorID
101	Math	I201
102	Science	I202

ID	CourseID
1	101
1	102
2	101

InstructorID	InstructorName	InstructorContact
I201	Prof. Smith	01234-567895
I202	Prof. Brown	01234-789556

Transaction

A database transaction is a sequence of multiple operations performed on a database, and all served as a single logical unit of work — taking place wholly or not at all.

Discuss the four ACID properties in transaction management.

ACID properties:

- **Atomicity:** A transaction must be treated as an indivisible unit. This means that either all the operations within the transaction are executed completely, or none are executed at all. If any part of the transaction fails, all changes made during the transaction are undone. This property is critical because partial transactions can leave the database in an inconsistent state.

- **Consistency:** This property guarantees that a transaction will bring the database from one valid state to another, maintaining all defined rules, such as integrity constraints. This means that the transaction will not violate any database constraints, ensuring that the database remains in a consistent state before and after the transaction.

- **Isolation:** Transactions must operate independently of each other without interference. Even though transactions may execute concurrently, the outcome must be the same as if the transactions were executed sequentially, one after another.

- **Durability:** Once a transaction has been successfully completed, its changes must be permanent. These changes should persist even in the case of a system crash or other failures, ensuring that completed transactions are never lost.

Summarize the atomicity and concurrent access issues solved by database systems.

Atomicity

- **Key issue:** During a transaction, the database may temporarily enter an inconsistent state (e.g., account A is debited but account B is not yet credited).

- **Solution:** The database system keeps a log of old values before any updates are made. If a transaction fails, the system can use this log to restore the database to its state before the transaction began, effectively undoing any partial changes.

- **Responsible component** - A specialized component of the database (the recovery system) is responsible for tracking and restoring changes to ensure atomicity.

Concurrent Access

- **Key issue:** Concurrently executing transactions can interleave in ways that lead to inconsistent database states. (e.g., one transaction might read a partially updated state from another transaction, leading to incorrect computations and updates).

- **Solution:** To ensure consistency and avoid conflicts, the database employs mechanisms such as - **Serial execution** - Executing transactions one after the other, though this approach can be inefficient and **Isolation** - The isolation property of a transaction ensures that the concurrent

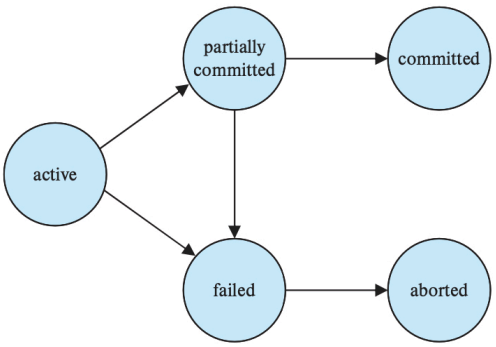
execution of transactions results in a state that is equivalent to a state obtained by some serial order of execution.

- **Responsible component** - The concurrency-control system of the database is responsible for ensuring isolation.

Discuss the states of a transaction.

A transaction in a database system can exist in one of the following states:

- **Active:** This is the initial state where the transaction is actively executing.
- **Partially Committed:** After the transaction has executed its final statement but before the transaction's effects have been made permanent in the database. At this point, the transaction is not yet fully committed, as it may still be aborted due to system failures.



- **Failed:** If an error is detected during the execution of the transaction, the transaction enters the failed state. It cannot proceed normally and must be rolled back.
- **Aborted:** After the failed transaction has been rolled back, undoing any changes it made to the database, it enters the aborted state. The transaction may either be restarted (if the failure was due to a transient error) or terminated permanently.
- **Committed:** After all the necessary updates have been successfully written to disk, the transaction enters the committed state. At this point, the changes are permanent and will persist even in the event of a system failure.

A transaction is said to have terminated when it has either committed or aborted.