**LECTURE 1**

What is data?

Data is some kind of quality/attribute that we care about. There are a lot of characteristics that can be taken for anything. But data is characteristics that we care about.

Data is raw facts that have little meaning unless they have been organized in some logical manner.

What is field?

A character or group of characters that has a specific meaning. It is used to define and store data.

What is record?

A logically connected set of one or more fields that describes a person, place, or thing.

What is file?

A collection of related record.



What is a database?

Database is something that helps you create a design in which you later fill data. Database is a shared, integrated computer structure that stores a collection of:

* End-user data – Raw facts of interest to end user
* Metadata: Data about data, which the end user data are integrated and managed

It describes data characteristics and relationships.

Why DBMS:

Intermediary between user and database

Enables data to be shared

Presents the end user with an integrated view of the data

Receives and translates application requests into operations required to fulfill the requests

Hides database’s internal complexity from the application programs and users.

Adv of DBMS:  
Better data integration and less data inconsistency

Increased end-user productivity

Improved data sharing, data security, data access, decision making

Types of DBs:

Based on user count:

* Single-user database: Supports one user at a time (Desktop database)
* Multiuser database: Supports multiple users at a time (Workgroup, Enterprise)

Based on location:

* Centralized database: Data is located at a single site (old IBM: all stored in one place)
* Distributed database: Data is distributed across different site.
* Cloud database: Created and maintained using cloud data services that provide defined performance measures for the database.

Based on content:

* General-purpose database: Contains a wide variety of data used in multiple disciplines (MySQL, Oracle, SQLite)
* Discipline-specific database: Contains data focused on specific subject areas.

Based on data currency:

* Operational database: Designed to support a company’s day-to-day operations.
* Analytical database: Stored historical data and business metrics used exclusively for tactical or strategic decision making

Based on structure of contained data:

* Unstructured data: It exists in their original state
* Structured data: Processed to some extent
* Extensible Markup Language (XML): Represents data elements in textual format

Types of Data Anomaly:

Update Anomalies

Insertion Anomalies

Deletion Anomaly

Structure means TABLES

DBMS Functions:

Data dictionary management

Data storage management

Data transformation and presentation

Security management

Data Integrity management

Why is structural dependence 'bad'? What are examples?

Structural dependence is considered 'bad' in software design because it creates brittleness, making systems vulnerable to breakage when internal structures change. It hinders adaptability, scalability, and maintainability, as modifications in one component may necessitate widespread adjustments, complicating testing and limiting reusability.

Examples:

* Hard-Coded Paths: A module relying on specific file paths; changing paths requires modifying the module.
* Direct Method References: One module directly using internal methods or variables of another; changes in the second module can disrupt the first.
* Database Schema Dependency: Components relying on a specific database schema; alterations to the schema may necessitate adjustments in dependent components.

DATA MODELING:

Data modeling is an iterative and progressive process of creating a specific data model for a determined problem domain.

Data models are simple representations of complex real world data structures.

Importance:

Communication tool

Gives a overall view of the database

Organize data for various users

Abstraction for creation of good database.

Entity: Unique and distinct object used to collect and store data.

Attribute: Characteristic of an entity.

Relationship: Describes an association among entities:  
one-to-one

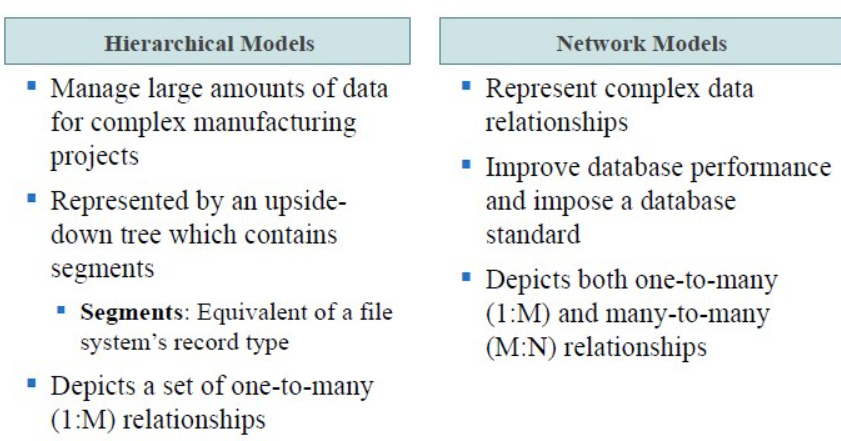
One-to-many

Many-to-many

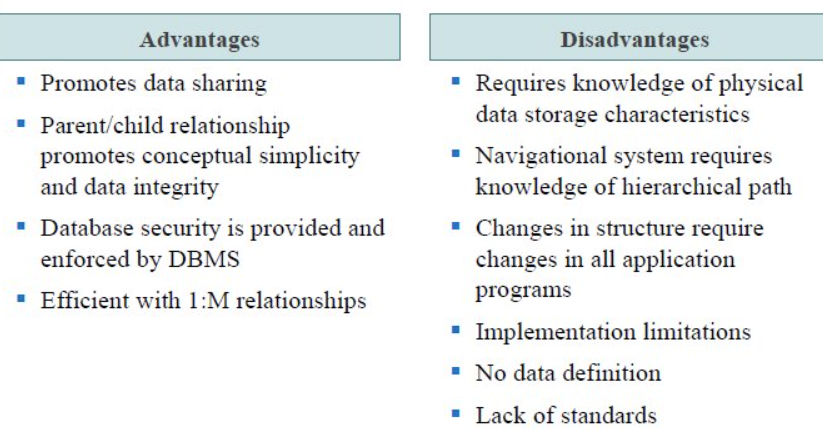
**LECTURE 2**

Hierarchy:

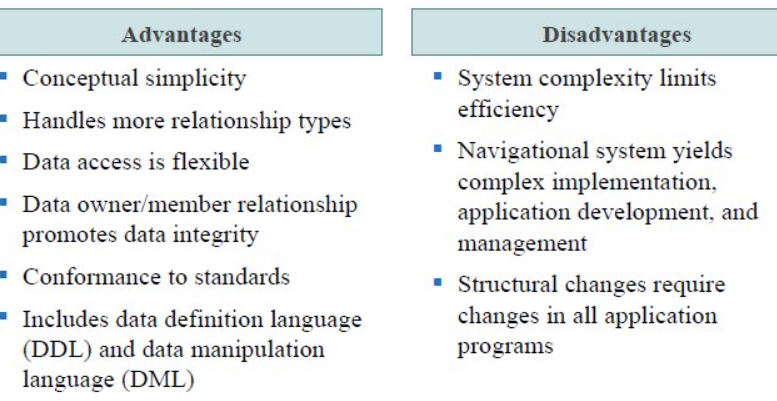
Subclass can only have one superclass.



Hierarchical:



Network:



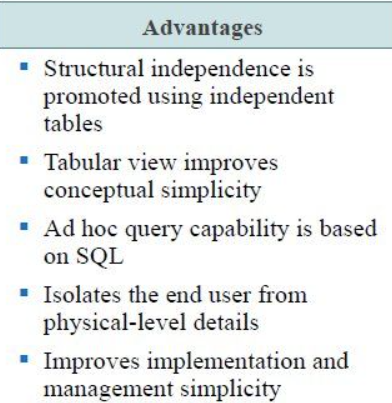
Schema: Conceptual organization of the entire database as viewed by the database administrator.

Sub-schema: Portion of the database seen by the application programs that produce the desired info from the data within the database.

Schema data definition language (DDL): Enables the database administrator to define the schema components.

Data manipulation language (DML): Environment in which data can be managed and is used to work with the data in the database.

Relational Model:



Rows are called tuples/entity instance/record.

Tables are entities.

Columns are called attributes.

Relational DBMS:

Helps multiple people use the database at the same time. Performs basic functions provided by the hierarchical and network DBMS

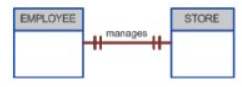
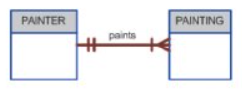
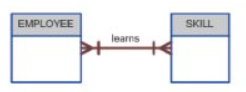
Hides complexities of the relational model from the user.

ER Diagram:

Graphical representation of entities and their relationships in a database structure.

Notations:

One-to-one: One-to-many: Many-to-many:

O-O databases:

Also called 'object stores', these dbs offer(ed) a way to store ("persist") objects on disk. The objects (entity instances) are instanced from classes (entities), like with standard OO programming practice.

Advantages:

'cleaner' design - objects mimic real-world counterparts

inheritance and encapsulation possible

richer datatypes (attributes) available

good for CAD, multimedia..

O-R databases

These are a compromise between RDBs and OODBs - they feature an O-O front-end over a relational architecture. Interfacing applications do so in an O-O way, and queries/modifications are translated to/from relational form ("ORM").

Benefits:

easy to access the data from an O-O application

queries can be simpler (can use objects' structure)

Conceptual model: Basic model

External model: description of tables

Internal model: Table details included (column name, datatype, primary key, etc.)

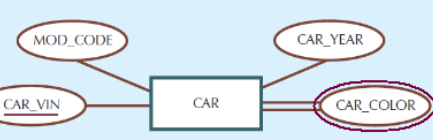
ER DIAGRAM:  
An identifier is called primary key.

Optional attributes can be kept empty. Required attributes should be filled.

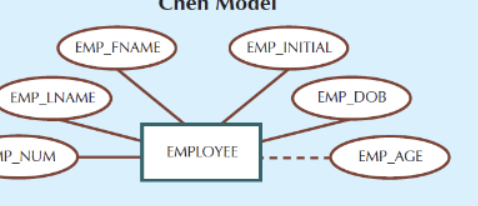
Types of attributes:

* Composite identifier: primary key composed of more than one attribute.
* Compound attribute: Attribute that can be subdivided to yield additional attributes. Eg: Car color: has RGB, 3 values.
* Simple attribute: Attribute that cannot be sub divided.
* Single-valued attribute
* Multi-values attributes.
* Derived attribute: Attribute obtained from other attributes.

Multivalued Attribute: has 2 ellipses.



Derived attribute: has dotted lines.



Dependence vs independence:

Dependence: Entity exists in the database only when it is associated with another related entity occurrence. Eg. if A is EMPLOYEE and B is DEPENDENT, a dependent (eg. child) in B can only exist if there is a corresponding employee (eg. Dad) in A.

Independence: Entity exists apart from all of its related entities. Known as strong entity or regular entity.

Existence independence implies a strong entity; but, existence dependence

Between two entities, a strong relationship, paradoxically, can lead to a weak entity :) Explain the meaning of 'strong' and 'weak' in this context.

- Strong Relationship:

A strong relationship between two entities implies that one entity is heavily dependent on the other. In a strong relationship, the existence of one entity is closely tied to the existence of the other. Typically, this is represented by a foreign key in the dependent entity, referencing the primary key of the other entity. The strong entity is said to have a high level of independence.

- Weak Entity:

A weak entity, on the other hand, is an entity that does not have a primary key attribute of its own. It relies on a related strong entity to provide a portion of its primary key. A weak entity is existence-dependent, meaning it cannot exist unless it is related to a certain strong entity. It often has a partial key, which is a set of attributes that, in combination with the primary key of the related strong entity, forms a unique identifier for the weak entity.

Now, the paradox you mentioned arises when a strong relationship leads to a weak entity. This paradox can occur when the strong entity in the relationship has such a dominant role that the weak entity becomes almost completely dependent on it. Despite being part of a strong relationship, the weak entity's independence is compromised, making it, in a sense, "weak" due to its heavy reliance on the strong entity for existence.

WEAK ENTITY:  
For something to be weak:  
1. It has to be dependent

2. Primary key of the strong table is part of the primary key of the dependent table.

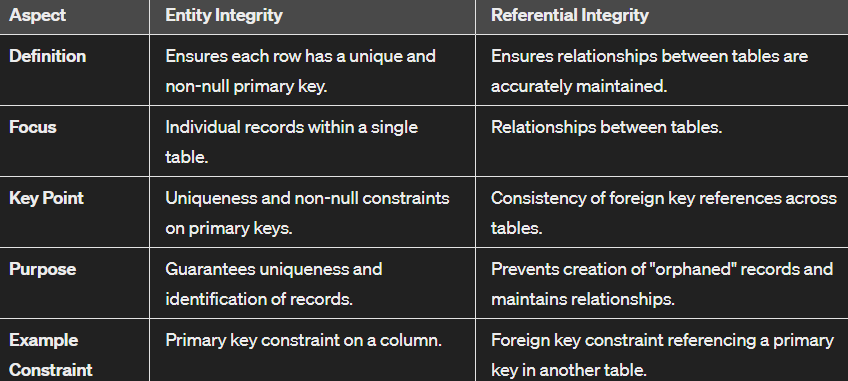
A weak entity implies existence dependence, but existence dependence does not imply a weak entity!

Optional participation: Cardinality is 0

Mandatory participation: not zero



Diff between entity and referential integrity



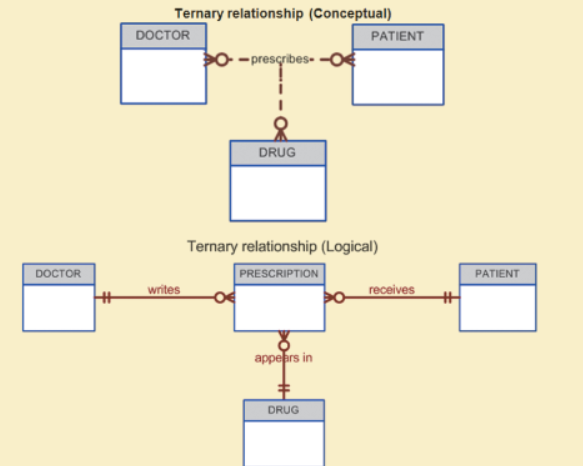
Relationship degree:

Unary relationship: 1 table

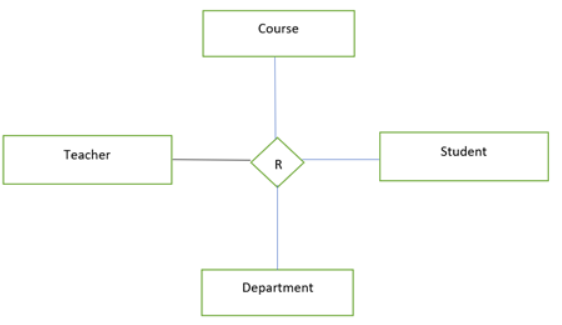
Suppose we have employee table with manager column. Employees are managers. Hence, manager relationship has a unary relationship with employee. 

Binary relationship: 2 tables

Ternary relationship: 3 tables



Quaternary relationships: 4 tables



Entities:

Student (StudentID, Name, etc.)

Course (CourseID, Title, etc.)

Instructor (InstructorID, Name, etc.)

Classroom (ClassroomID, RoomNumber, etc.)

Relationships:

Enroll (StudentID, CourseID, InstructorID, ClassroomID)

Quinary relationship: 5 tables

Entities:

Student (StudentID, Name, etc.)

Course (CourseID, Title, etc.)

Instructor (InstructorID, Name, etc.)

Classroom (ClassroomID, RoomNumber, etc.)

Semester (SemesterID, Term, Year, etc.)

Relationships:

Enroll (StudentID, CourseID, InstructorID, ClassroomID, SemesterID)

Bridge table:

It makes the many-to-many relationship into one-to-many relationship. Bridge has primary key of all the tables connected.

**LECTURE 3**

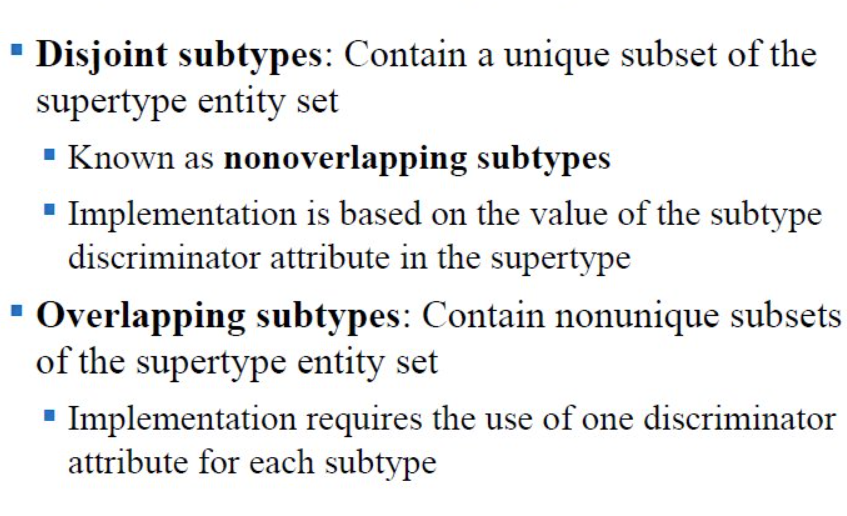
EER DIAGRAM:

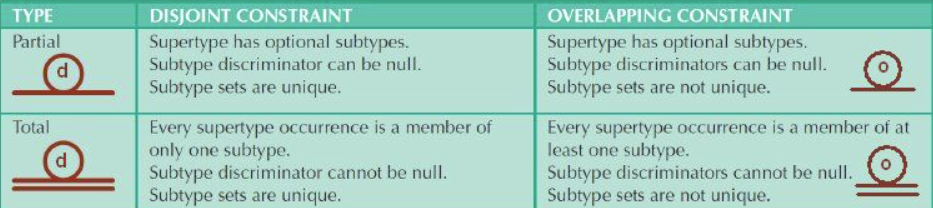
Entity supertype: Generic entity type related to one or more entity subtypes

Entity subtype: Contains unique characteristics of each entity subtype.

The concept of EER is analogous to what, in software development?

The concept of EER (Entity-Relationship) in database design is analogous to object-oriented modeling in software development, as both focus on representing entities, relationships, and attributes in a structured and reusable manner.





Default comparison is equality, but we can use <, >, <=, >=,!= as well. These are called theta joins.

Why is full functional dependence a good thing?

Full functional dependency is beneficial because it ensures that every non-prime attribute in a relational table is fully dependent on the entire primary key, leaving no partial dependencies. This helps in maintaining data integrity and avoids anomalies that can arise when updating, inserting, or deleting records in the database. By adhering to full functional dependency, redundancy is minimized, and the database design is more robust, making it easier to manage and maintain.

RELATIONAL MODELING:

Any cell in a table is also called a relation: - between that row and column.

Table is called a relation.



Examples of 'blind' (non-identifying) keys in RL

In relational databases, a "blind" or non-identifying relationship is one in which the child table's foreign key does not include all columns of the parent table's primary key. Here are a few examples:

Customer and Order Tables:

Customer Table (Parent):

CustomerID (Primary Key)

Name

Email

Order Table (Child):

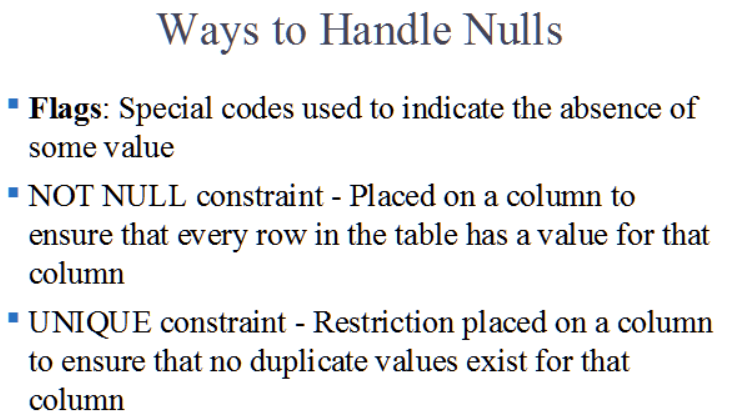
OrderID (Primary Key)

OrderDate

CustomerID (Foreign Key referencing CustomerID in Customer Table)

In this case, the foreign key in the Order table (CustomerID) does not include all columns of the primary key in the Customer table. The relationship is non-identifying because it doesn't uniquely identify a record in the Order table.

NULLS: -



**Operations on tables:**

There are (only) EIGHT 'relational set operators' (defined by Ed Codd, at IBM, in 1970), which are all used to operate ("perform relational algebra") on tables: Select, Project, Union, Intersect, Difference, Product, Divide, Join

Select and Project can be unary.

Select gives rows; Project gives columns.

In difference, order matters. A-B won’t give you same output as B-A.

Why is closure a good property?

Closure breaks the chain. It won’t let you continue.

Closure is a valuable property in databases as it enables the inference of additional functional dependencies, aiding in comprehensive understanding of data relationships.

* Inference: Closure allows the derivation of additional functional dependencies from the given set, enhancing understanding.
* Query Optimization: Helps optimize query processing by providing insights into relationships and dependencies.
* Data Integrity: Facilitates maintaining data consistency and avoiding anomalies during database operations.
* Normalization: Aids in the normalization process by identifying and eliminating redundant data.
* Schema Design: Contributes to creating a robust and flexible database schema for organized data management.

Closure - where else?

Complex number modulus has closure. Addition, subtraction of complex number gives complex number. But modulus of complex number does not give a complex number. It gives a single number.

String:

If you perform substring, concatenate, upper/lower case gives string. But if you ask for number of characters in a string, it returns number. Hence, it closes the flow.

**Lecture 4**

AUB = (A-B) U (AnB) U (B-A)

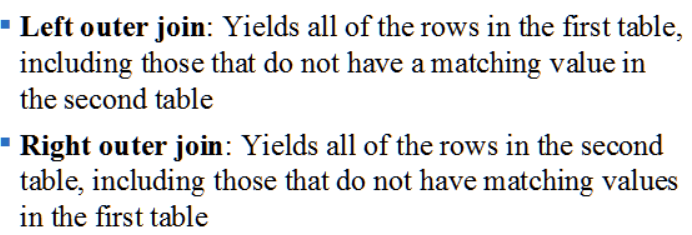
JOINS:

Types:

* Natural Join: Links tables by selecting only the rows with common values in their common attributes.
* Equijoin: Links tables on the basis of an equality condition that compares specified columns of each table.
* Theta join: Extension of natural join, denoted by adding a theta.

Inner outer joins deal with null values.

* Inner join: Only returns matched records from the tables that are being joined. If rows have nulls, they are ignored.
* Outer join: Matched pairs are retained and unmatched values in the other table are left null.



NORMALIZATION:

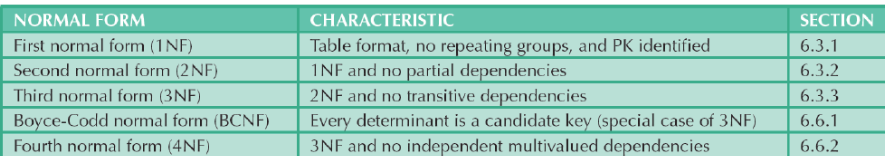
You do normalization on one table at a time. When a table goes in, you will get more tables by breaking table by columns. Idea is to reduce redundancies.

Eg: If one table has 4 columns. After normalization, it may split into 2 tables with 2 columns and 3 columns each. This can be done to improve the design.

1NF, 2NF, 3NF, BCNF, 4NF.

BCNF is 3 and a half NF.

Properly designed 3NF can be same as 4NF.



Normalization how-to, in one sentence: work on one relation (table) at a time: identify dependencies, then 'normalize' - progressively break it down into smaller relations (tables), based on the dependencies we identify in the original relation.

\* 1NF: eliminate repeating groups (partial:y, transitive:y)

Table should not contain any multivalued attribute.

If so, to convert it into 1st normal form, we have 3 solutions:  
1. Split the multivalued attributes into rows. Suppose there are 2 courses for student A, then create two rows for student A with each course in different line.

2. Create different columns. For the previous example, the two courses names will have 2 columns. Problem with this is: NULL Values increase.

3. Divide the table int 2: eg: - base table (has primary key roll number) has student name and roll number. 2nd table contains roll number, courses.

\* 2NF: eliminate redundant data (partial:n, transitive:y)

\* 3NF: eliminate fields not dependent on key fields (partial:n, transitive:n)

**LECTURE 5**

SQL: -

Select column names

from table name;

Update table name

Set condition;

Delete from table name

Where condition;

Insert into table name(column names)

Values ….;

<> can also be used to represent !=.

Can use OR:

Select column names

From table name

Where condition1 OR condition2;

Can use and:

Select column names

From table name

Where condition1

And condition2;

Insert table rows with select subquery:  
Insert into table name

Select column names

From tablename;

BETWEEN:  
Select column names

From table name

Where column name BETWEEN value1 and value2;

ISNULL:

Select column names

From table name

Where column name IS NULL;

LIKE:

Select column names

From table name

Where column name LIKE ‘value%’;

IN:

Select column names

From table name

Where column name IN (range or another select statement);

ALTER:

Alter table tablename

ADD/MODIFY/DROP column name or condition;

DISTINCT:

Select DISTINCT column names

From table name;

Aggregate:

Select COUNT/MIN/MAX/SUM/AVG(column names)

From table name;

GROUP BY/ORDER BY/HAVING:

Select column names

From table name

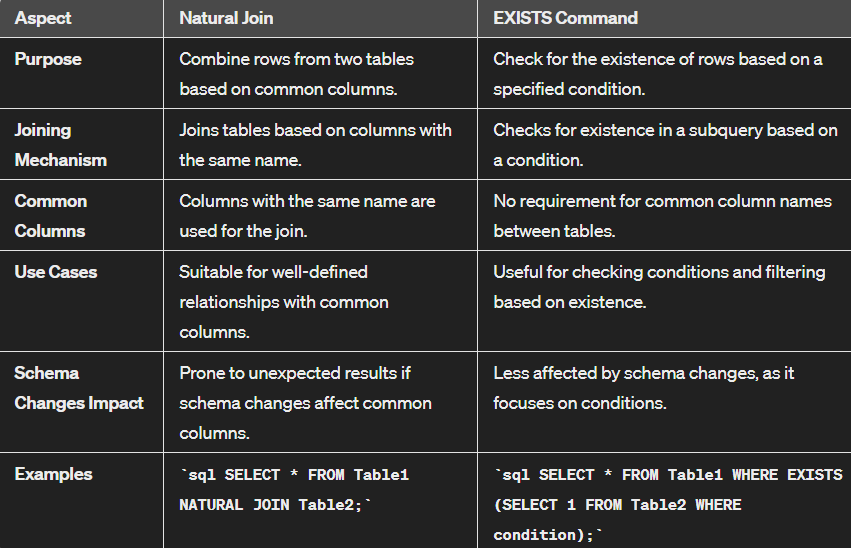
WHERE conditions

GROUP BY column names

HAVING conditions

ORDER BY conditions ASC/DESC;

EXISTS:  
In SQL, in what sense are these similar: a natural join, and the 'EXISTS' command? Discuss, with examples.



Format:

SELECT columns

FROM table1

WHERE EXISTS (SELECT 1 FROM table2 WHERE condition);

COMMIT:

The COMMIT command is used to permanently save the changes made during the current transaction to the database.

Once a COMMIT is issued, the changes become permanent, and they are visible to other transactions.

ROLLBACK:  
The ROLLBACK command is used to undo the changes made during the current transaction.

If any error occurs during the transaction or if the user decides to discard the changes, a ROLLBACK can be issued to revert the database to its state before the transaction started.

It cancels all the modifications made during the transaction and leaves the database in a consistent state.

Here is a (SQL) query ['sno' and 'sname' are number and name]:

SELECT sno, sname

FROM Suppliers

WHERE 100 > (SELECT SUM(quantity)

FROM Shipments

WHERE Shipments.sno = Suppliers.sno);

What is such a query called?

What does the above, do? Be specific, and, explain your answer.

The given SQL query is an example of a correlated subquery. In this specific query, it retrieves the supplier numbers (`sno`) and names (`sname`) from the "Suppliers" table where the sum of quantities in the "Shipments" table for each supplier is less than 100.

Explanation:

1. The subquery `(SELECT SUM(quantity) FROM Shipments WHERE Shipments.sno = Suppliers.sno)` calculates the total quantity of shipments for each supplier in the "Shipments" table, correlated to the outer query by matching supplier numbers (`sno`).

2. The main query `SELECT sno, sname FROM Suppliers WHERE 100 > ...` filters the suppliers based on the condition that the sum of quantities from the subquery is less than 100.

In summary, the query retrieves suppliers whose total shipment quantity is less than 100, utilizing a correlated subquery to calculate the sum of quantities for each supplier individually.

When we 'mine' data for insights, we are looking for something new (the 'gold') the data can provide us. How could 'GROUP BY'(the SQL command) help with this? You can provide a general/overview answer, NO need for code or an algorithm. Simply give it some thought, and write them down.

The `GROUP BY` SQL command is instrumental in data mining for insights as it allows us to group data based on specific attributes or columns. This grouping facilitates the aggregation of information, enabling the extraction of patterns, trends, and summary statistics from the data. By applying functions like COUNT, SUM, AVG, etc., within the `GROUP BY` framework, we can analyze the distribution of values, identify outliers, and discern meaningful patterns that might not be apparent when examining the data as a whole. This command is particularly useful when exploring large datasets and trying to uncover hidden insights within different categories or dimensions of the data.

**LECTURE 6**

For std dev = (Xi – AVG)2 for all values. Add them and divide by n. Take a root.

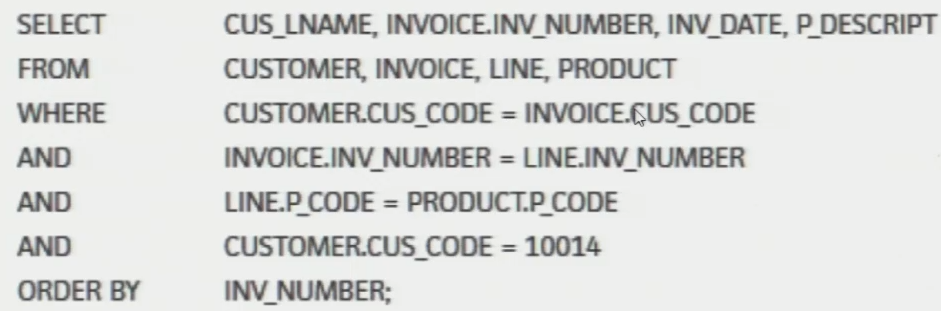
Group by works iff you have COUNT/MIN/MAX/SUM/AVG command.

JOIN:

Select columns

From table1, table2

Where table1.column = table2.column;





SEQUENCE:  
Objects are called sequence.

Create sequence sequence name

Start With 4000 Nocache;

Insert into table name

Values (sequencename.NEXTVAL, val1, val2);

CURRVAL: - Current value

NEXTVAL: - Next value

NEXTVAL returns the current value, then does ++ (ie. it does 'post increment', ie. C++ as opposed to ++C); CURRVAL on the other hand, just fetches the current value (does not ++ it).

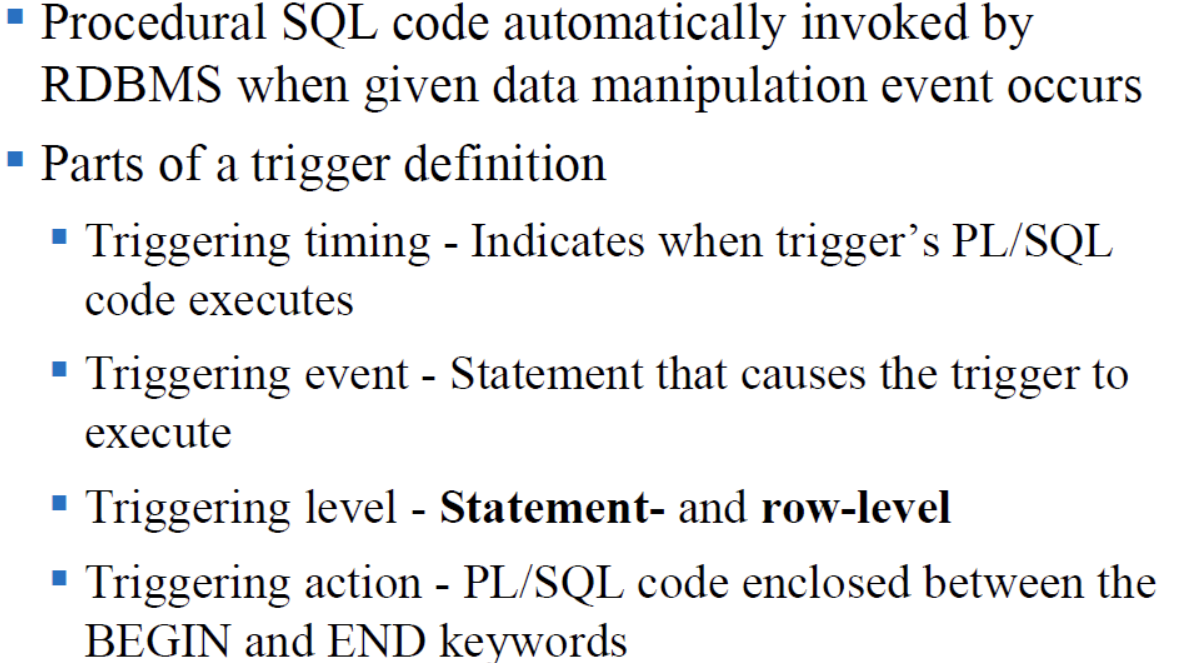
PL/SQL:

PL/SQL involves extra (augmented) syntax that lets us do looping, branching, variable declaration and function declaration - these are of course not possible using 'plain' SQL.

PL/SQL can be used to create:

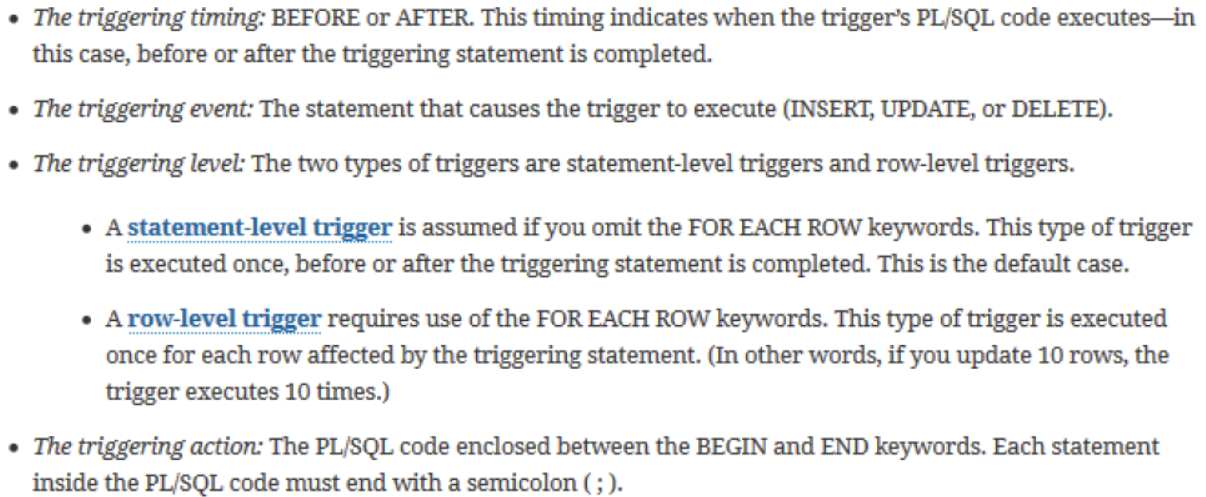
* blocks of code for one-time execution
* triggers - callbacks to invoke
* stored procedures - named procedures (no return values) for repeated calling
* stored functions - named functions (with return values) for repeated calling

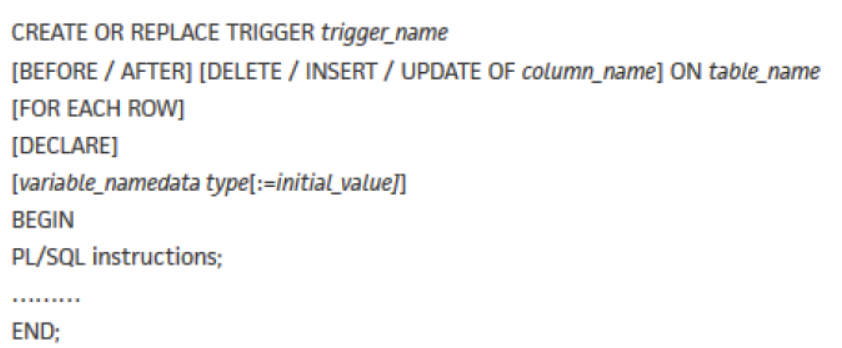
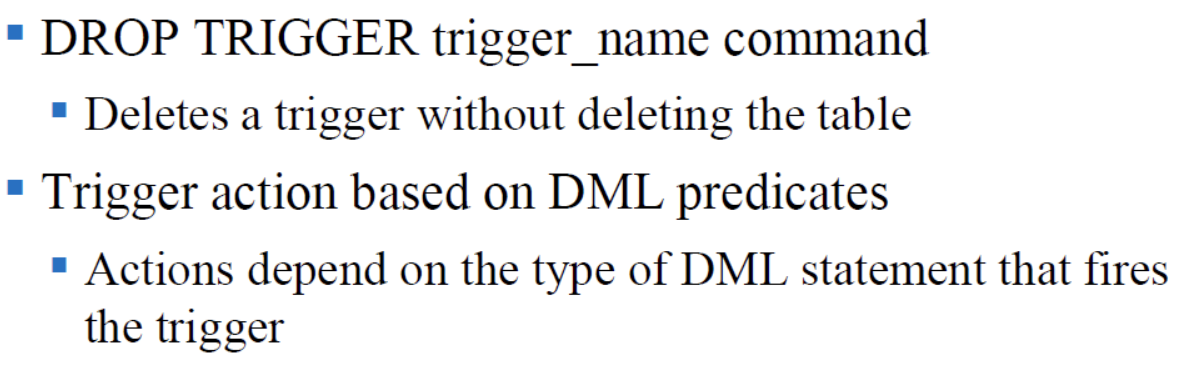
TRIGGERS:



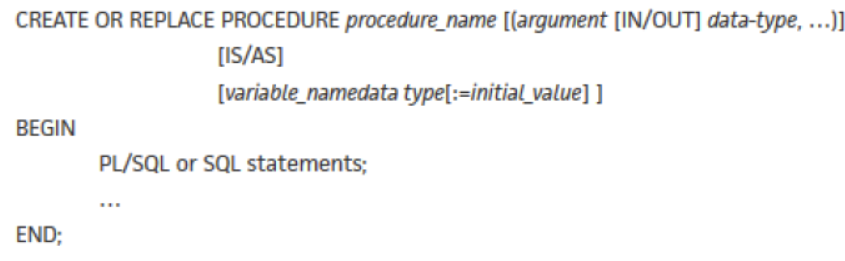
Timing: Before or after

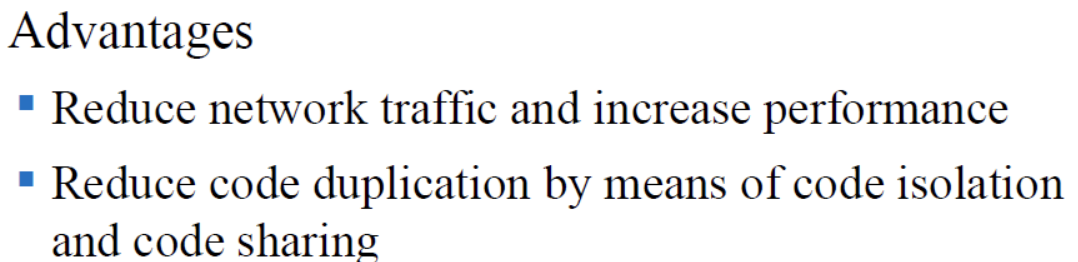
Event: DELETE/INSERT/UPDATE



STORED PROCEDURE:

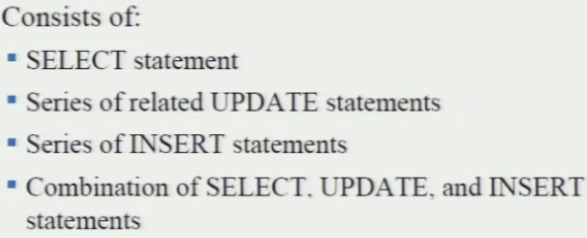


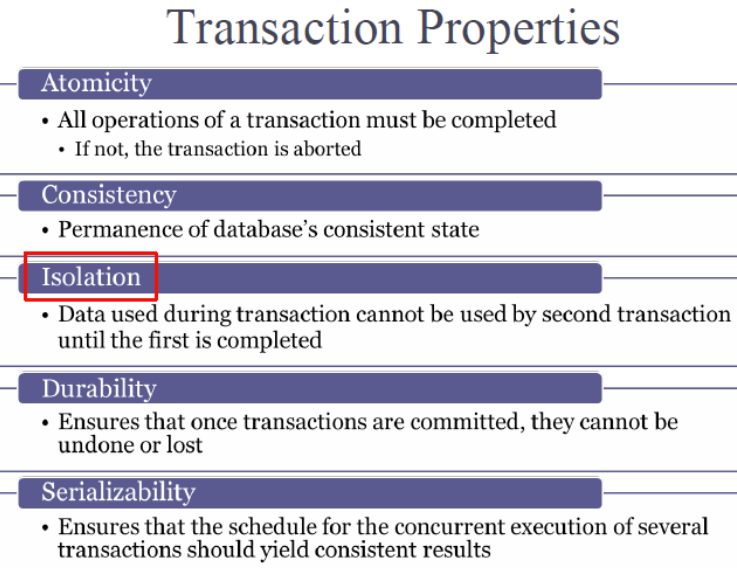


**LECTURE 7**

TRANSACTION MANAGEMENT:  
What is transaction?

Transaction is a logical unit of work (bunch of sql statements) that must be entirely completed or aborted.





Commit:

Definition: Committing a transaction means making all changes made during the transaction permanent.

Purpose: It signifies that the transaction has been successfully completed, and the changes are now durable and visible to other transactions.

Example (SQL): COMMIT;

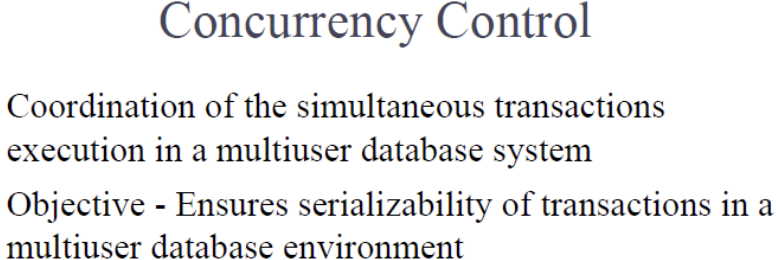
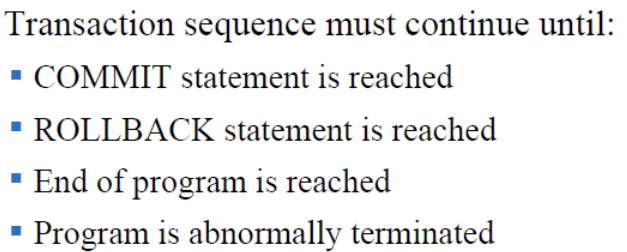
Rollback:

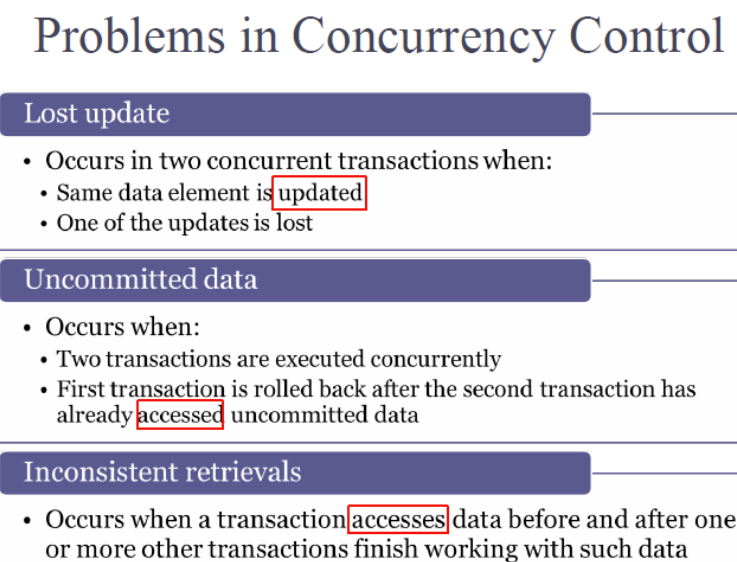
Definition: Rolling back a transaction means undoing or discarding all changes made during the transaction.

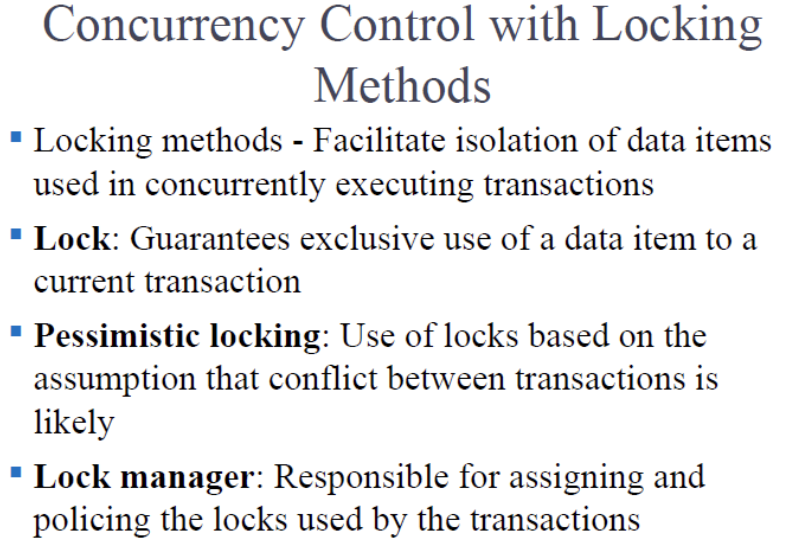
Purpose: It is used when an error occurs or if there's a need to discard the transaction for any reason.

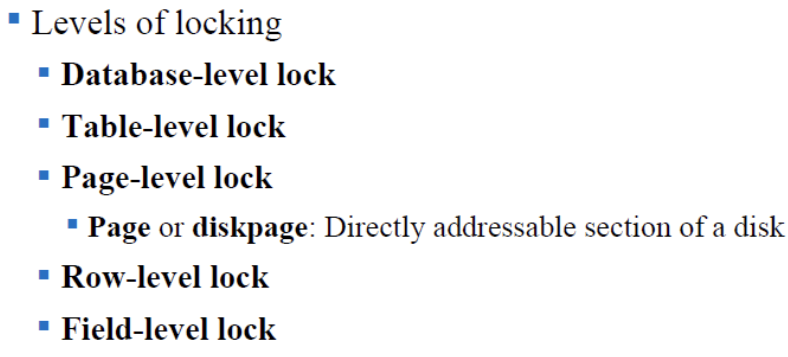
Example (SQL): ROLLBACK;

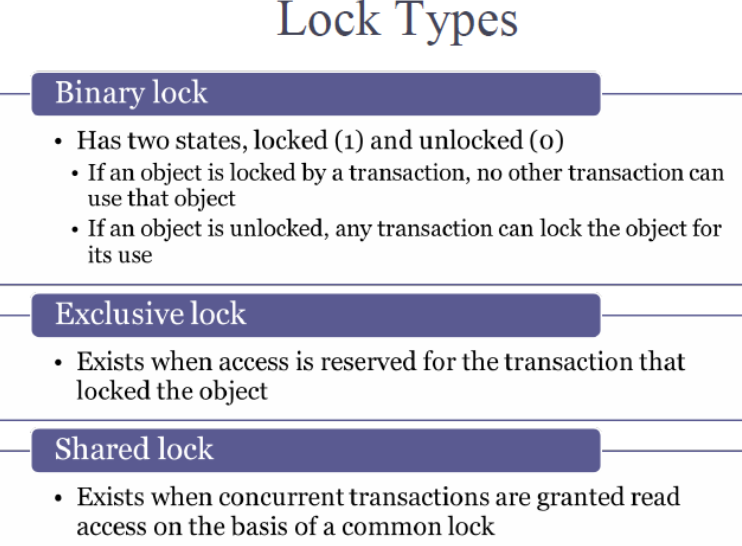
Rollbacks are local, Commits are permanent.

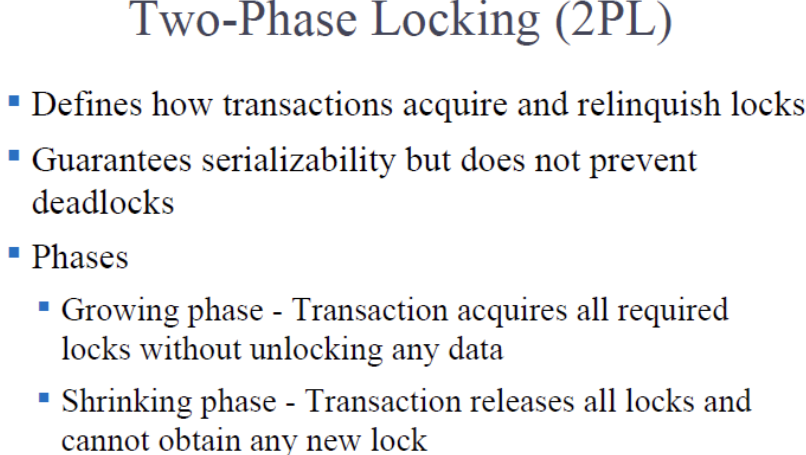


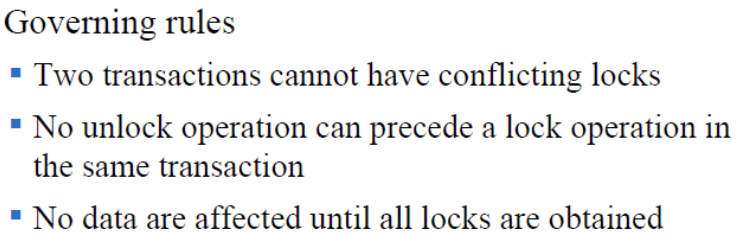








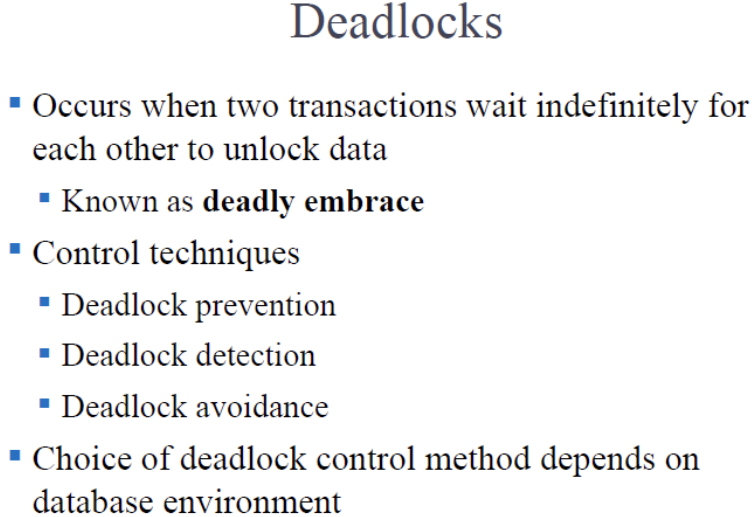


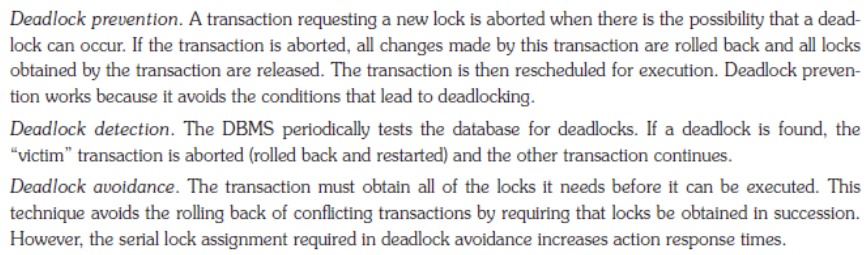


Phases are:

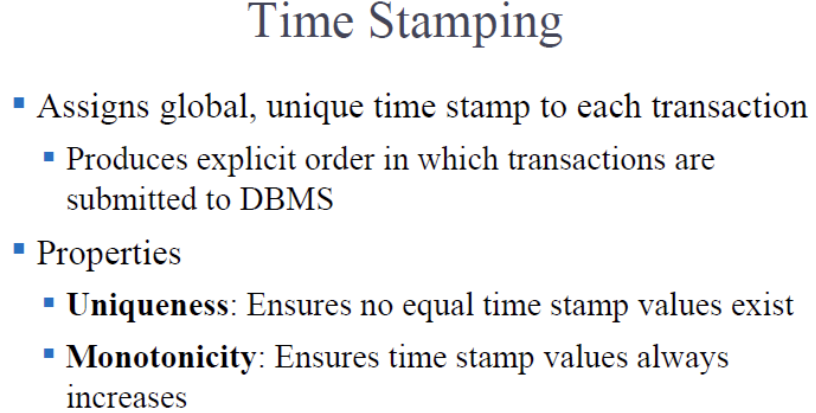
* Lock phase
* Execution
* Unlock phase

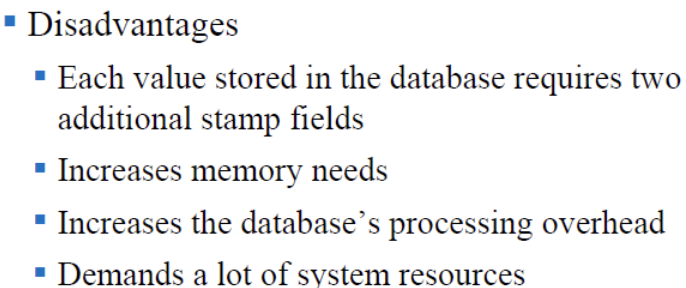
Should be known as 3 phase, but is called 2 phase (lock and unlock)

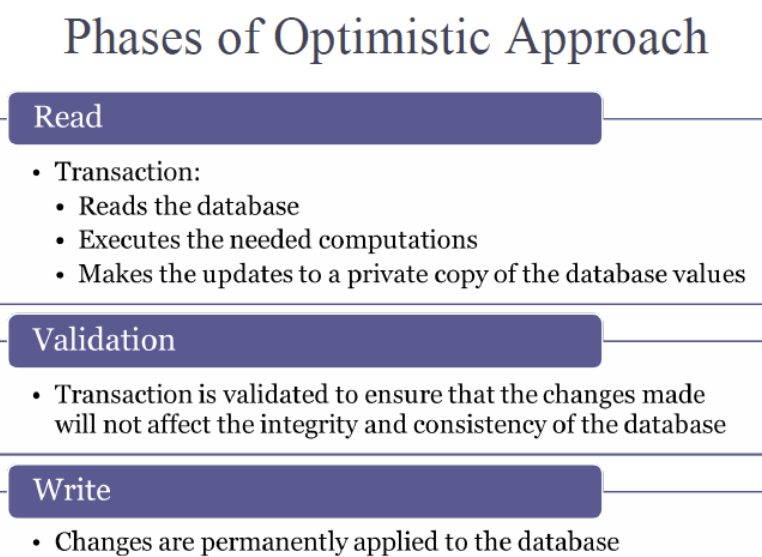




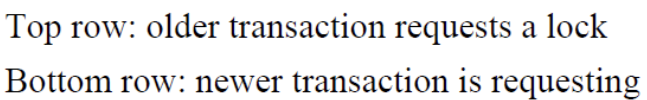
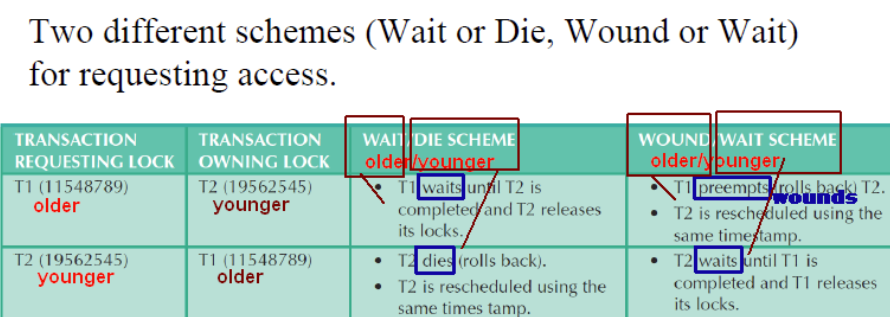
Timestamping-based schemes prevent deadlocks; 2PL avoids deadlocks; detection is used in both.







Doesn’t work for Amazon/Alibaba. But it works for databases that update rarely.



DISTRIBUTED DATABASE:  
Two Phase Commit Protocol(2PC):

Implementation

Final commit

What is the logic behind 2PC?

The Two-Phase Commit (2PC) protocol ensures atomicity in distributed transactions by coordinating a prepare phase where all participants vote on their ability to commit, followed by a commit phase where the coordinator decides and communicates whether to commit or abort the transaction based on the votes.

Explain how 2PC works, in your own words.

In the two phase commit protocol, as the name implies, there are two phases:

phase 1: commit request phase, aka 'voting' phase: coordinator sends a 'commit or abort?' (aka 'query to commit' or 'prepare to commit') message to each participating transaction node; each node responds (votes), with a 'can commit' (aka 'ready') or 'need to abort' (aka 'abort') message

phase 2: commit phase, aka 'completion' phase:

if in phase 1, all nodes responded with 'can commit', the coordinator sends a 'commit' phase to each node; each node commits, and sends an acknowledgment to the coordinator

or, if in phase 1, any node responded with 'need to abort', the coordinator sends an 'abort' message to each node; each node aborts, and sends an acknowledgment to the coordinator

Two things that can possibly go wrong in Two-Phase Commit (2PC) are:

Blocking: Participants may be left in a blocked state if the coordinator fails after the prepare phase.

Coordinator Failure: If the coordinator fails after participants commit, the system may end up in an inconsistent state without proper recovery mechanisms.