

**SIDDARTHA INSTITUTE OF SCIENCE AND TECHNOLOGY:PUTTUR
(AUTONOMOUS)**

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING



**DATABASE MANAGEMENT SYSTEM
(20CS0505)**

COURSE OBJECTIVES

- *understand the different issues involved in the design and implementation of a database system.*
- *study the physical and logical database designs, database modelling, relational, hierarchical, and network models*
- *understand and use data manipulation language to query, update, and manage a database*
- *understand the concept of a database transaction and related database facilities.*
- *design and build a simple database system and demonstrate competence with the fundamental tasks involved with modeling, designing, and implementing a dbms.*

COURSE OUTCOMES

on successful completion of this course, the student will be able to

- *develop relational algebra expressions for queries and optimize them.*
- *design the databases using ER method for a given specification of requirements.*
- *apply normalization techniques on given database.*
- *determine the transaction atomicity, consistency, isolation, and durability for a given transaction-processing system.*
- *implement the isolation property, including locking, time stamping based on concurrency control and serializability of scheduling.*
- *understand physical storage media and raid concepts.*

UNIT-1

Introduction: Database System Applications, Purpose Of Database Systems, View Of Data, Data Abstraction, Data Independence, Data Models, Database Languages, Database Architecture, Database Users And Administrators.

Introduction To Data Base Design: Er Diagrams, Entities, Attributes And Entity Sets, Relationships And Relationship Sets.

1.INTRODUCTION TO DBMS

- A database-management system (**DBMS**) is a collection of interrelated data and a set of programs to access those data.
- The collection of data, usually referred to as the *database*, contains information relevant to an enterprise.
- The primary goal of a DBMS is to provide a way to store and retrieve database information that is both *convenient* and *efficient*.

2.DATABASE SYSTEM APPLICATIONS

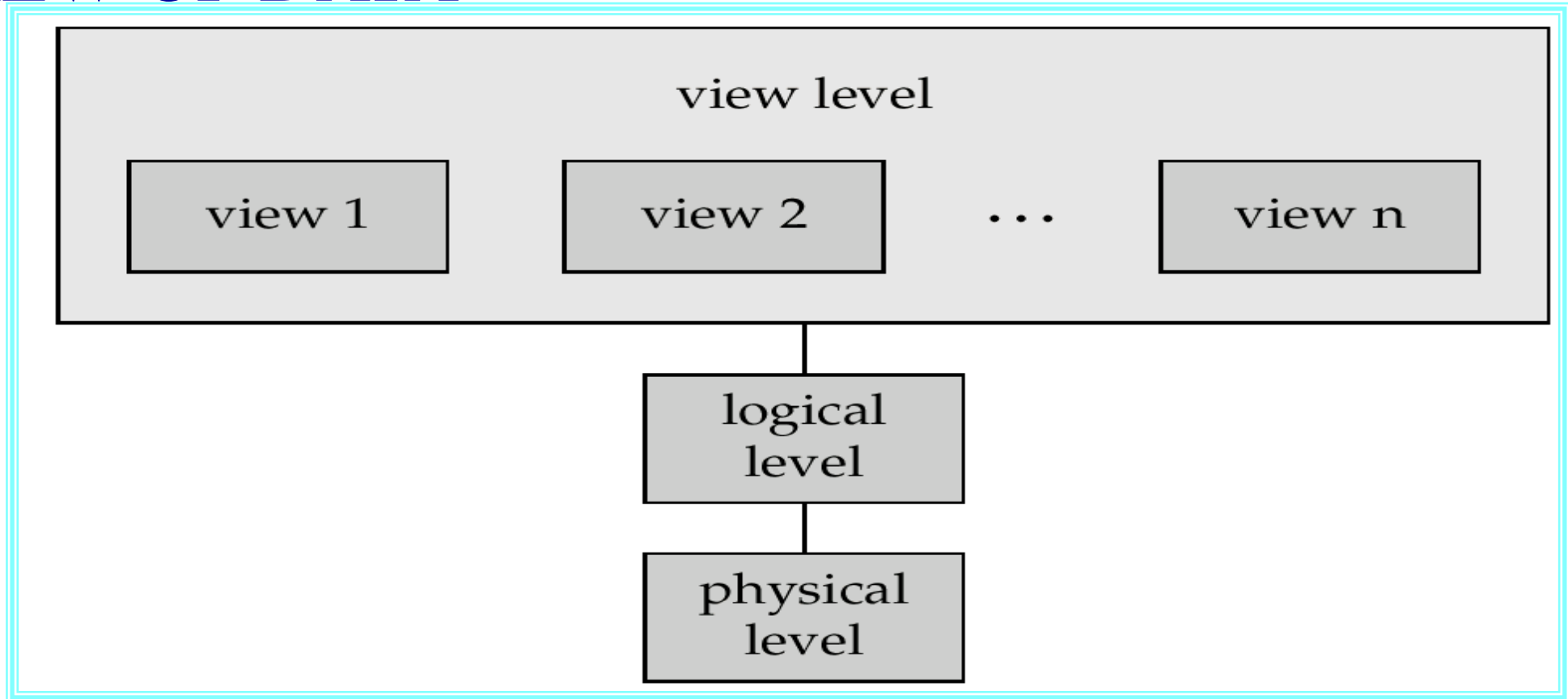
- **Banking:** all transactions
- **Airlines:** reservations, schedules
- **Universities:** registration, grades
- **Sales:** customers, products, purchases
- **Manufacturing:** production, inventory, orders, supply chain
- **Human resources:** employee records, salaries, tax deductions

3.PURPOSE OF DATABASE SYSTEMS

Drawbacks of using file systems to store data:

- Data redundancy and inconsistency
 - Multiple file formats, 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 part of program code
 - Hard to add new constraints or change existing ones

4.VIEW OF DATA



5. ABSTRACTION

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

```
type customer = record  
    name : string;  
    street : string;  
    city : integer;  
end;
```

- **View level:** application programs hide details of data types. Views can also hide information (e.g., salary) for security purposes.

DATA INDEPENDENCE

- *Data Independence*

- These levels of abstraction provide data independence *i.e is all the transactions or changes made at one level are unaffected to other levels*
- DBMS architecture provides two types of data independence
 - *Logical data independence*
 - *Physical data independence*

6.DATA MODELS

- A collection of tools for describing
 - data
 - data relationships
 - data semantics
 - data constraints
- Entity-Relationship model
- Relational model
- Other models:
 - object-oriented model
 - semi-structured data models
 - Older models: network model and hierarchical model

7.DATABASE LANGUAGE

- Data definition language (DDL)
- Data manipulation language (DML)
- Data control language (DCL)
- Transaction control language (TCL)

DATA DEFINITION LANGUAGE (DDL)

- Specification notation for defining the database schema

- E.g.

```
create table account (  
    account-number char(10),  
    balance integer)
```

- DDL compiler generates a set of tables stored in a *data dictionary*
- Data dictionary contains metadata (i.e., data about data)
 - database schema
 - Data *storage and definition* language
 - language in which the storage structure and access methods used by the database system are specified
 - Usually an extension of the data definition language

DATA MANIPULATION LANGUAGE (DML)

- Language for accessing and manipulating the data organized by the appropriate data model
 - DML also known as query language
- Two classes of languages
 - **Procedural** – user specifies what data is required and how to get those data
 - **Nonprocedural** – user specifies what data is required without specifying how to get those data

SQL

- SQL: widely used non-procedural language

- E.g. find the name of the customer with customer-id 192-83-7465

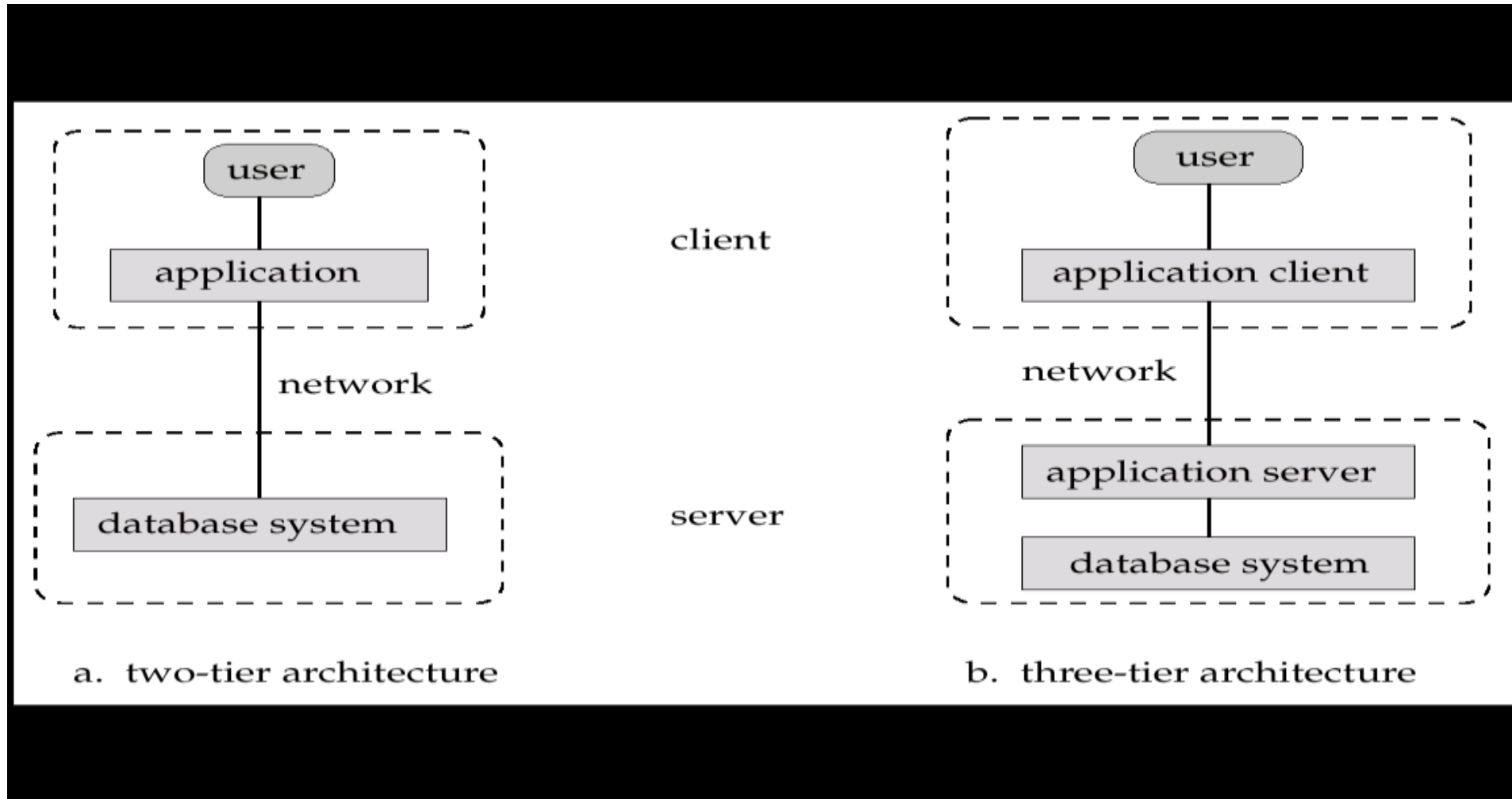
```
select  customer.customer-name  
from    customer  
where customer.customer-id = '192-83-7465'
```

- E.g. find the balances of all accounts held by the customer with customer-id 192-83-7465

```
select  account.balance  
from    depositor, account  
where depositor.customer-id = '192-83-7465' and  
        depositor.account-number = account.account-number
```

- Application programs generally access databases through one of
 - Language extensions to allow embedded SQL
 - Application program interface (e.g. ODBC/JDBC) allow SQL queries to be sent to a database

8.DATABASE THREE TIER ARCHITECTURE



DATABASE USERS

Users are differentiated by the way they expect to interact with the system

- **Application programmers** – interact with system through DML calls
- **Sophisticated users** – form requests in a database query language
- **Specialized users** – write specialized database applications that do not fit into the traditional data processing framework
- **Naïve users** – invoke one of the permanent application programs that have been written previously
 - *E.g. people accessing database over the web, bank tellers, clerical staff*

DATABASE ARCHITECTURE

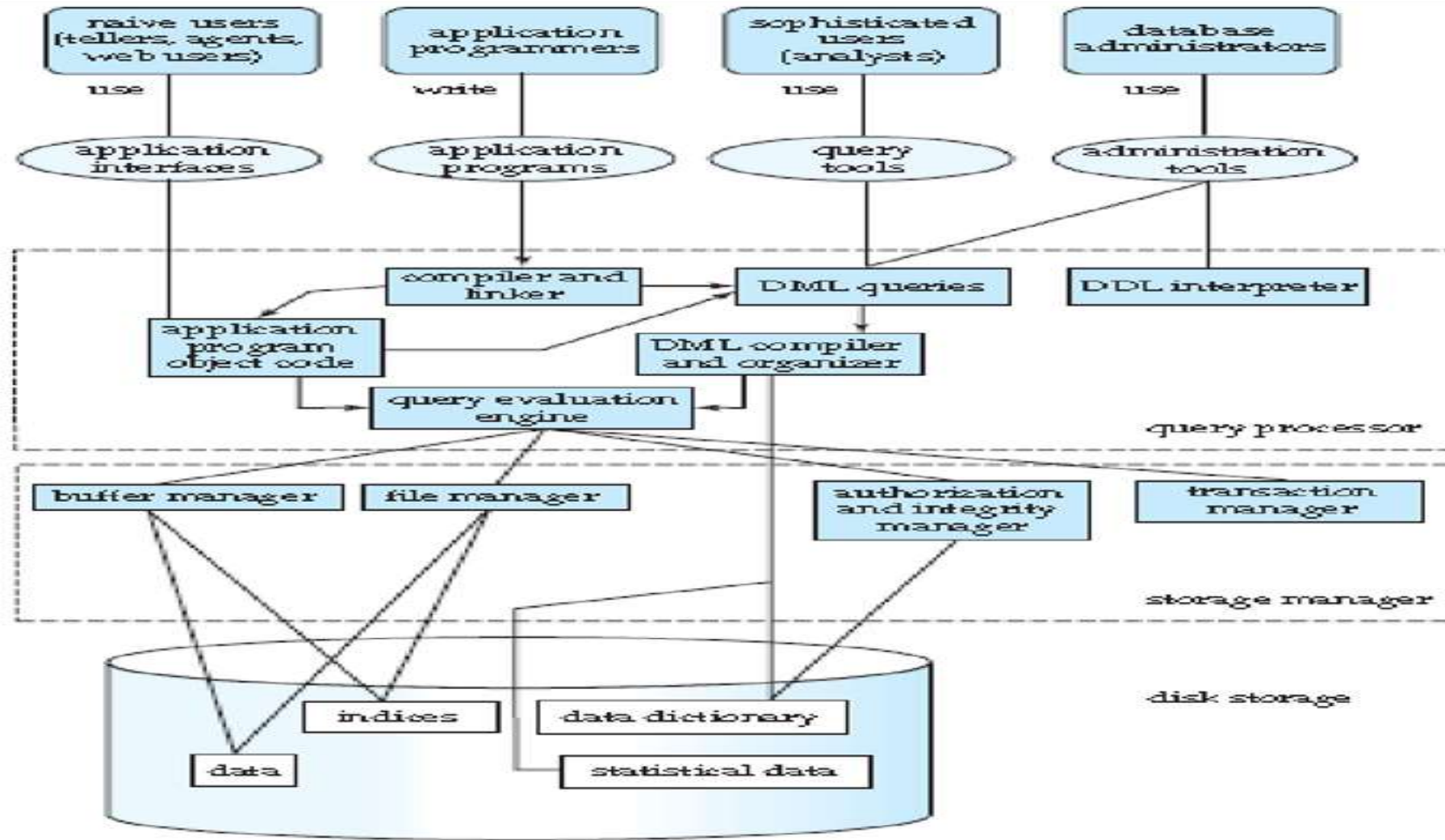


Figure 1.5 System structure.













DATABASE ADMINISTRATOR

- Coordinates all the activities of the database system;
- database administrator has a good understanding of the enterprise's information resources and needs.
- Database administrator's duties include:
 - Schema definition
 - Storage structure and access method definition
 - Schema and physical organization modification
 - Granting user authority to access the database
 - Specifying integrity constraints
 - Acting as liaison with users
 - Monitoring performance and responding to changes in requirements

9.INTRODUCTION TO DATA BASE DESIGN

- E-R Diagram
- Entities
- Attributes and Entity sets
- Relationships and Relation shipsets

E-R DIAGRAM SYMBOLS

	Represents Entity	
	Represents Attribute	
	Represents Relationship	
	Links Attribute(s) to entity set(s) or Entity set(s) to Relationship set(s)	
	Represents Multivalued Attributes	
	Represents Derived Attributes	
	Represents Total Participation of Entity	
	Represents Weak Entity	
	Represents Weak Relationships	
	Represents Composite Attributes	
	Represents Key Attributes / Single Valued Attributes	

10.ENTITIES

- *A database can be modeled as:*
 - a collection of entities,
 - relationship among entities.
- *An entity is an object that exists and is distinguishable from other objects.*
 - Example: specific person, company, event, plant
- *Entities have attributes*
 - Example: people have *names* and *addresses*
- *An entity set is a set of entities of the same type that share the same properties.*
 - Example: set of all persons, companies, trees, holidays

ENTITY SET

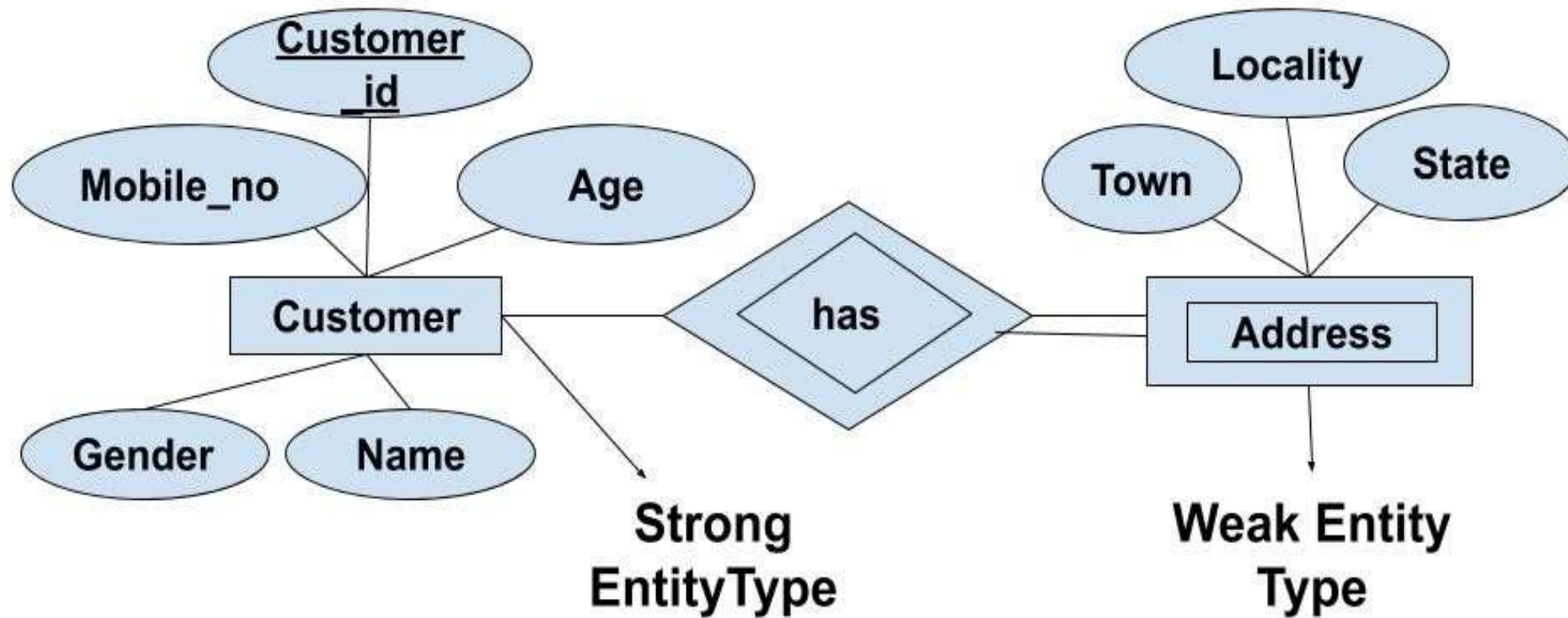
- It is often useful to identify a collection of similar entities.
- Such a collection is called an entity set.

Note:

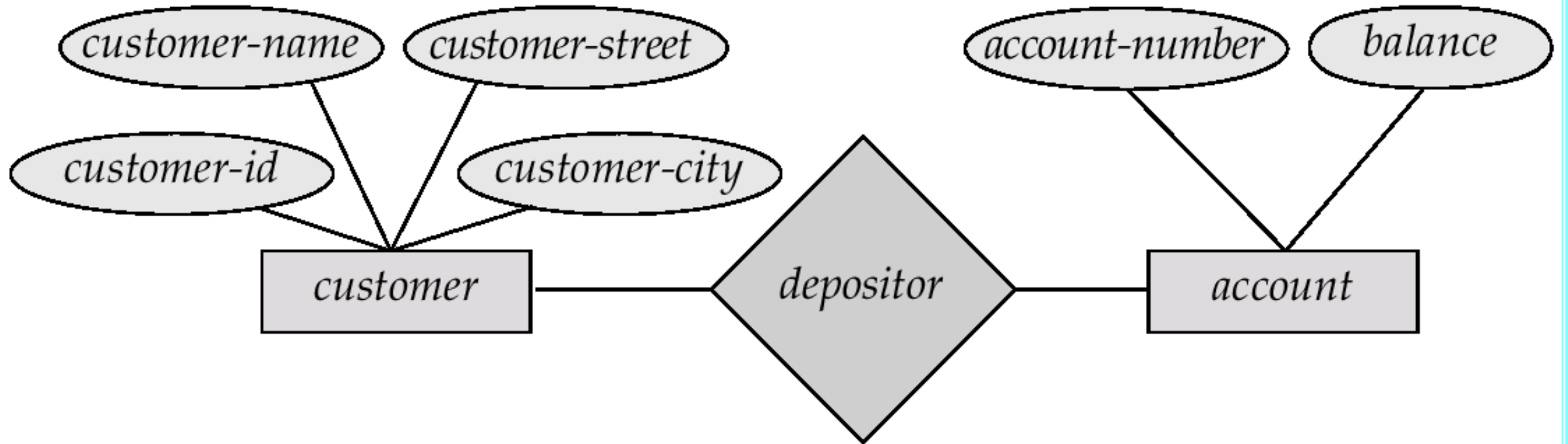
1.Entity sets need not be disjoint

2.The collection of toy department employees and the of appliance department employees may both contain employee.

ENTITY SET



11.ENTITY-RELATIONSHIP MODEL



- Example of tabular data in the relational model

RELATIONAL MODEL

<i>Customer-id</i>	<i>customer-name</i>	<i>customer-street</i>	<i>customer-city</i>	<i>account-number</i>
192-83-7465	Johnson	Alma	Palo Alto	A-101
019-28-3746	Smith	North	Rye	A-215
192-83-7465	Johnson	Alma	Palo Alto	A-201
321-12-3123	Jones	Main	Harrison	A-217
019-28-3746	Smith	North	Rye	A-201

UNIT -2

RELATIONAL ALGEBRA AND CALCULUS

Relational Algebra And Calculus: Relational Algebra - Selection And Projection, Set Operations, Renaming, Joins. Form Of Basic SQL Query- Examples Of Basic SQL Queries, Introduction To Nested Queries, Correlated Nested Queries, Set - Comparison Operators, Aggregate Operators, NULL Values - Comparison Using Null Values, Disallowing NULL Values, Triggers And Active Data Bases.

RELATIONAL ALGEBRA AND CALCULUS

- Relational Algebra
- Selection and Projection
- Set operations
- Renaming
- Joins
 - Form of Basic SQL Query
- Examples of Basic SQL Queries
- Introduction to Nested Queries
- Correlated Nested Queries
- Set - Comparison Operators
- Aggregate Operators
- NULL values
- Comparison using Null values
- Disallowing NULL values
- Triggers and Active Data bases

RELATIONAL ALGEBRA

- Relational algebra is one of the two formal query languages associated with the relational model. Queries in algebra are composed using a collection of operators. A fundamental property is that every operator in the algebra accepts (one or two) relation instances as arguments and returns a relation instance as the result.

SELECTION AND PROJECTION

- Relational algebra includes operators to *select* rows from a relation (σ) and to *project* columns (π). These operations allow us to manipulate data in a single relation. Consider the instance of the Sailors relation, denoted as S_2 . We can retrieve rows corresponding to expert sailors by using the σ operator.
- The expression, $\sigma_{rating > 8}(S_2)$ The selection operator σ specifies the tuples to retain through a *selection condition*. In general, the selection condition is a boolean combination (i.e., an expression using the logical connectives \wedge and \vee) of *terms* that have the form *attribute* op *constant* or *attribute1* op *attribute2*, where op is one of the comparison operators $<$, \leq , $=$, \neq , \geq , or $>$. The projection operator π allows us to extract columns from a relation;

SELECTION OPERATION

- Notation: $\sigma_p(r)$
- p is called the **selection predicate**
- Defined as:

$$\sigma_p(r) = \{t \mid t \in r \text{ and } p(t)\}$$

- Where p is a formula in propositional calculus consisting of **terms** connected by : \wedge (**and**), \vee (**or**), \neg (**not**)
Each **term** is one of:
 - $\langle \text{attribute} \rangle \text{ op } \langle \text{attribute} \rangle$ or $\langle \text{constant} \rangle$
 - where op is one of: $=, \neq, >, \geq, <, \leq$
- Example of selection:

$$\sigma_{\text{branch_name}=\text{"Perryridge"}}(\text{account})$$

PROJECTION OPERATION

- Notation:
- where A_1, A_2 are attribute names and r is a relation name.
- The result is defined as the relation of k columns obtained by erasing the columns that are not listed
- Duplicate rows removed from result, since relations are sets
- Example: To eliminate the *branch_name* attribute of *account*

$$\Pi_{\text{account_number, balance}} (\text{account})$$

SET OPERATIONS

- The set operations **union**, **intersect**, and **except** operate on relations and correspond to the relational algebra operations \cup , \cap , $-$.
- Each of the above operations automatically eliminates duplicates; to retain all duplicates use the corresponding multiset versions **union all**, **intersect all** and **except all**.

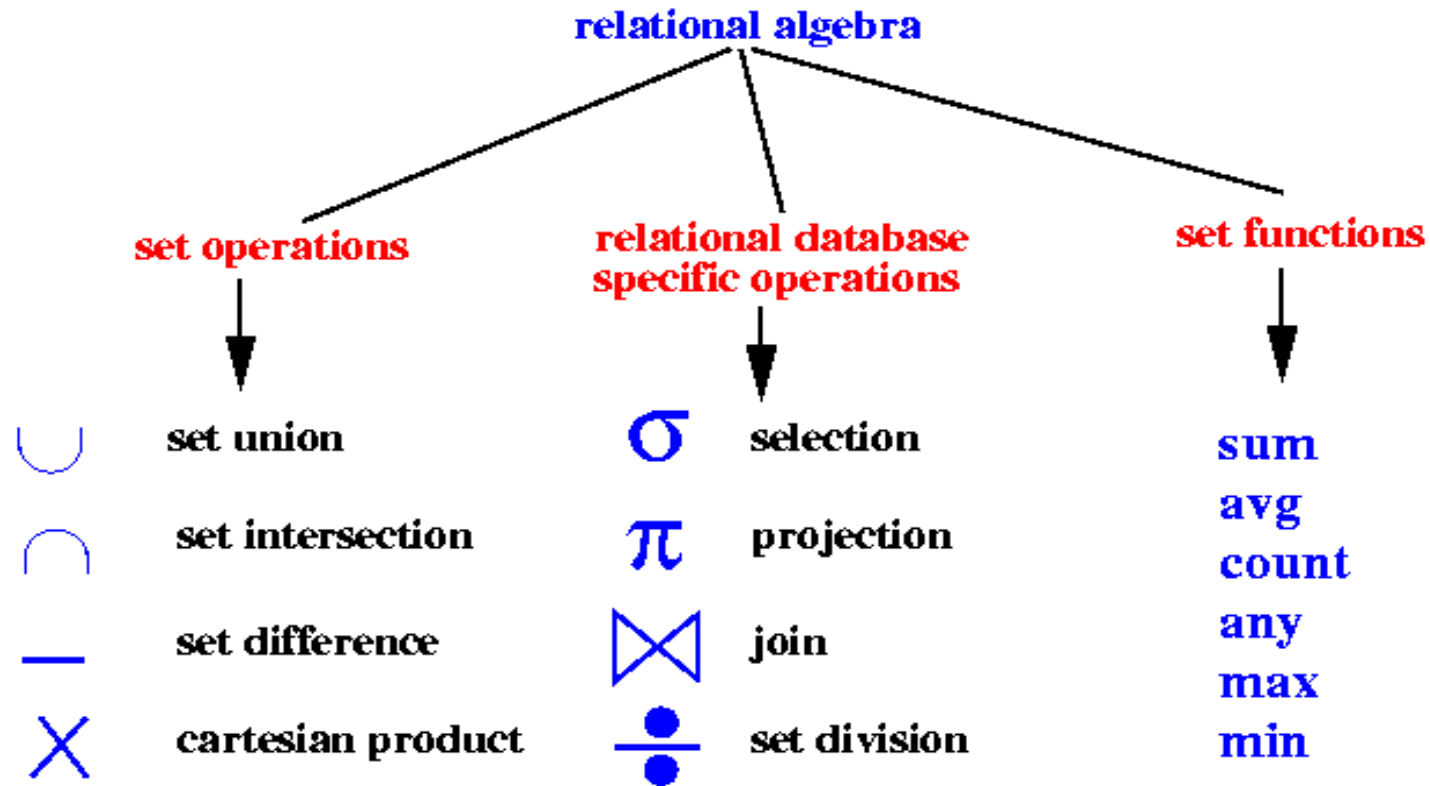
Suppose a tuple occurs m times in r and n times in s , then, it occurs:

- $m + n$ times in r **union all** s
- $\min(m, n)$ times in r **intersect all** s
- $\max(0, m - n)$ times in r **except all** s

SET OPERATIONS

- The following standard operations on sets are also available in relational algebra: *union* (\cup), *intersection* (\cap), *set-difference* ($-$), and *cross-product* (\times).
- **Union:** $R \cup S$ returns a relation instance containing all tuples that occur in *either* relation instance R or relation instance S (or both). R and S must be *union-compatible*, and the schema of the result is defined to be identical to the schema of R .
- **Intersection:** $R \cap S$ returns a relation instance containing all tuples that occur in *both* R and S . The relations R and S must be union-compatible, and the schema of the result is defined to be identical to the schema of R .
- **Set-difference:** $R - S$ returns a relation instance containing all tuples that occur in R but not in S . The relations R and S must be union-compatible, and the schema of the result is defined to be identical to the schema of R .
- **Cross-product:** $R \times S$ returns a relation instance whose schema contains all the fields of R (in the same order as they appear in R) followed by all the fields of (in the same order as they appear in S).

SET OPERATIONS



https://www.google.com/url?sa=i&url=https%3A%2F%2Fslideplayer.com%2Fslide%2F13730025%2F&psig=AOvVaw3tKp3eZGEB0_EJ4MmxmueC&ust=1644243290099000&source=images&cd=vfe&ved=0CAsQjRxqFwoTCNjajry87PUCFQAAAAAdAAAAABAK

UNION OPERATOR (U)-

- Union Operator (U)-
- Let R and S be two relations.
- Then-
- $R \cup S$ is the set of all tuples belonging to either R or S or both.
- In $R \cup S$, duplicates are automatically removed.
- Union operation is both commutative and associative.

INTERSECTION OPERATOR (\cap)-

- Intersection Operator (\cap)-

Let R and S be two relations.

- Then-
- $R \cap S$ is the set of all tuples belonging to both R and S.
- In $R \cap S$, duplicates are automatically removed.
- Intersection operation is both commutative and associative.

DIFFERENCE OPERATOR (-)-

Difference Operator (-)-

Let R and S be two relations.

Then-

- $R - S$ is the set of all tuples belonging to R and not to S.
- In $R - S$, duplicates are automatically removed.
- Difference operation is associative but not commutative.

RENAMING

- The RENAME operation is used to rename the output of a relation.
- Sometimes it is simple and suitable to break a complicated sequence of operations and rename it as a relation with different names. Reasons to rename a relation can be many, like –
- We may want to save the result of a relational algebra expression as a relation so that we can use it later.
- We may want to join a relation with itself, in that case, it becomes too confusing to specify which one of the tables we are talking about, in that case, we rename one of the tables and perform join operations on them.

RENAMING

- The SQL allows renaming relations and attributes using the **as** clause:

old-name as new-name

Find the name, loan number and loan amount of all customers;
rename the column name *loan-number* as *loan-id*.

<https://www.geeksforgeeks.org/rename-operation-in-relational-algebra/>

RENAMING

Renaming

16

- In relational algebra, a **rename** is a unary operation written as $\rho_{a/b}(R)$ where:
 - a and b are attribute names
 - R is a relation
- Examples:
 - 1) $\rho_{\text{employee}}(\text{Emp})$: Changes the name of Emp table to employee.
 - 2) Renaming and Union

Paternity		Maternity	
Father	Child	Mother	Child
Adam	Cain	Eve	Cain
Adam	Abel	Eve	Seth
Abraham	Isaac	Sarah	Isaac
Abraham	Ishmael	Hagar	Ishmael

$\rho_{\text{Father} \rightarrow \text{Parent}}(\text{Paternity}) \cup \rho_{\text{Mother} \rightarrow \text{Parent}}(\text{Maternity})$

Parent	Child
Adam	Cain
Adam	Abel
Abraham	Isaac
Abraham	Ishmael
Eve	Cain
Eve	Seth
Sarah	Isaac
Hagar	Ishmael

JOINS

- Join operations take two relations and return as a result another relation.
- These additional operations are typically used as subquery expressions in the **from** clause
- Join condition – defines which tuples in the two relations match, and what attributes are present in the result of the join.
- Join type – defines how tuples in each relation that do not match any tuple in the other relation (based on the join condition) are treated.

TYPES OF JOINS

- **inner join**
- **left outer join**
- **right outer join**
- **full outer join**

INNER JOIN

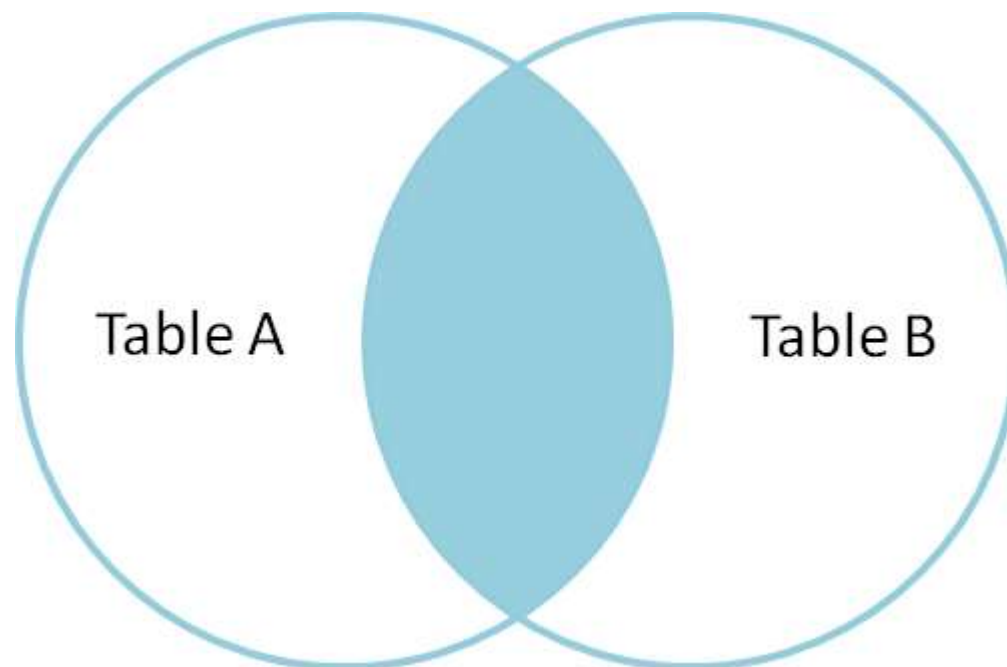
- **INNER JOIN:** The INNER JOIN keyword selects all rows from both the tables as long as the condition satisfies. This keyword will create the result-set by combining all rows from both the tables where the condition satisfies i.e value of the common field will be same.
- <https://www.tutorialspoint.com/sql/sql-inner-joins.htm>

INNER JOIN

Syntax:

```
SELECT  
table1.column1,table1.column2,table2.column1,....  
FROM table1  
INNER JOIN table2  
ON table1.matching_column =  
table2.matching_column;  
table1: First table.  
table2: Second table  
matching_column: Column common to both the tables.
```

INNER JOIN



LEFT JOIN

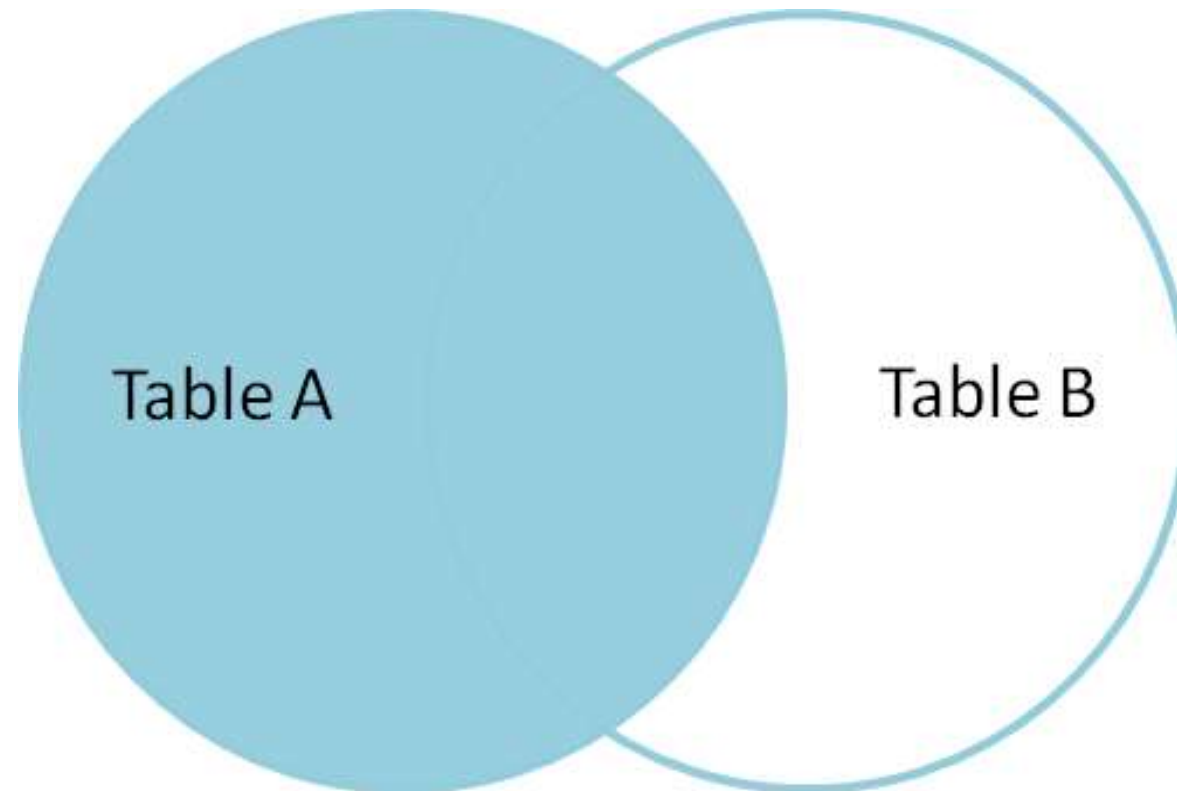
- **LEFT JOIN:** This join returns all the rows of the table on the left side of the join and matching rows for the table on the right side of join. The rows for which there is no matching row on right side, the result-set will contain *null*. LEFT JOIN is also known as LEFT OUTER JOIN.

LEFT JOIN

Syntax:

```
SELECT table1.column1,table1.column2,table2.column1,....  
FROM table1 LEFT JOIN table2  
ON table1.matching_column = table2.matching_column;  
table1: First table.  
table2: Second table  
matching_column: Column common to both the tables.
```


LEFT JOIN



RIGHT JOIN

- **RIGHT JOIN:** RIGHT JOIN is similar to LEFT JOIN. This join returns all the rows of the table on the right side of the join and matching rows for the table on the left side of join. The rows for which there is no matching row on left side, the result-set will contain *null*. RIGHT JOIN is also known as RIGHT OUTER JOIN.

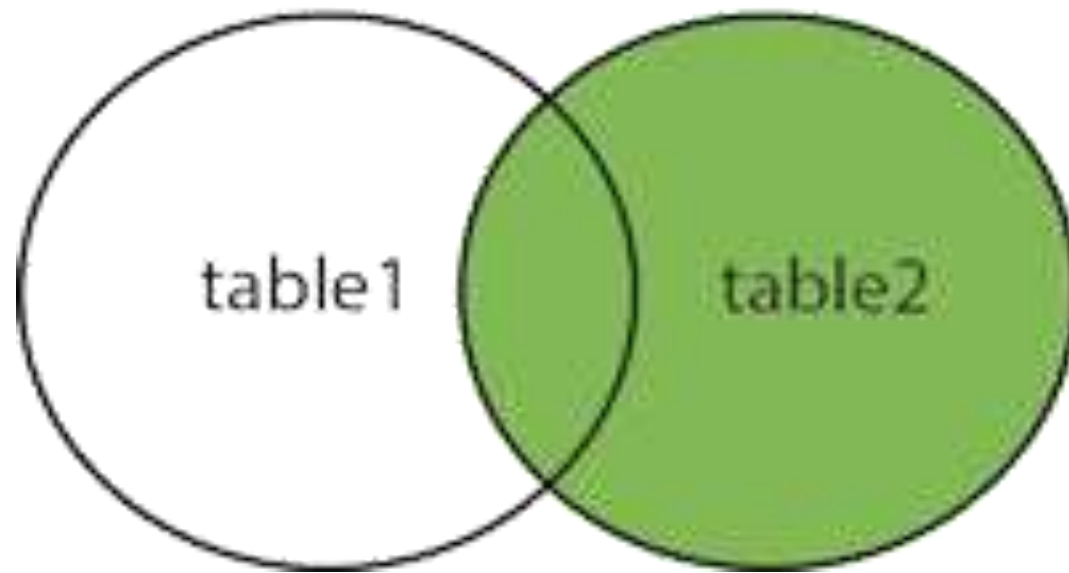
RIGHT JOIN

Syntax:

```
SELECT  
table1.column1,table1.column2,table2.column1,..  
..  
FROM table1 RIGHT JOIN table2  
ON table1.matching_column =  
table2.matching_column;  
table1: First table.  
table2: Second table  
matching_column: Column common to both the  
tables.
```

RIGHT JOIN

RIGHT JOIN



FULL JOIN

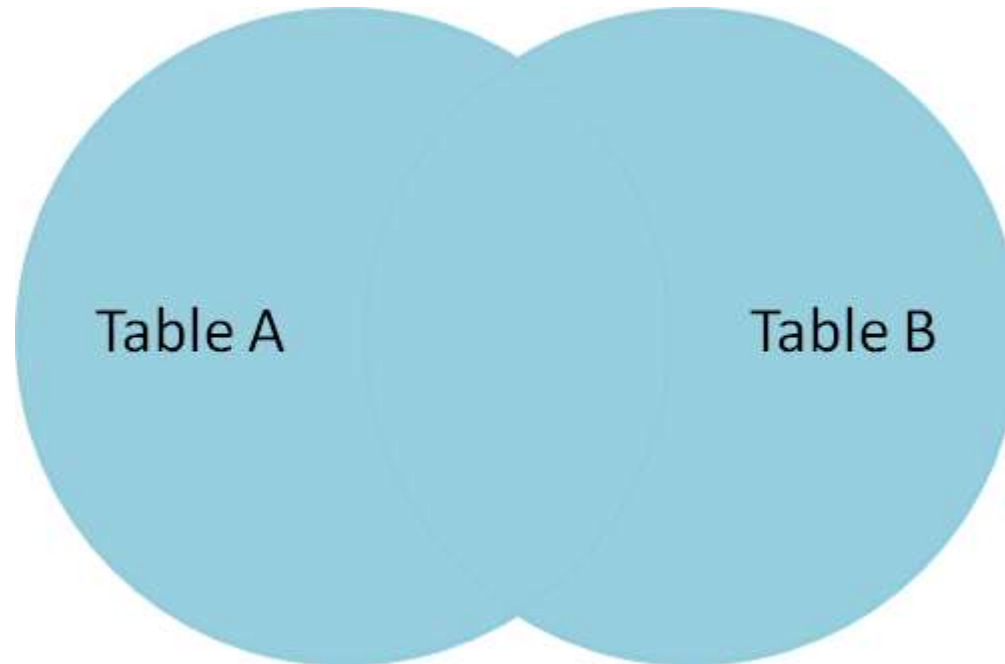
- **FULL JOIN:** FULL JOIN creates the result-set by combining result of both LEFT JOIN and RIGHT JOIN. The result-set will contain all the rows from both the tables. The rows for which there is no matching, the result-set will contain *NULL* values.

FULL JOIN

Syntax:

```
SELECT  
table1.column1,table1.column2,table2.column1,....  
  FROM table1 FULL JOIN table2  
  ON table1.matching_column =  
table2.matching_column;  
table1: First table.  
table2: Second table  
  matching_column: Column common to both the  
tables.
```

FULL JOIN



FORM OF BASIC SQL QUERY

- The **from** clause lists the relations involved in the query
 - corresponds to the Cartesian product operation of the relational algebra.
- Find the Cartesian product *borrower* x *loan* **select** *
from *borrower, loan*
- Find the name, loan number and loan amount of all customers having a loan at the Perryridge branch.
- **select** *customer-name, borrower.loan-number, amount*
from *borrower, loan*
where *borrower.loan-number = loan.loan-number and*
branch-name = 'Perryridge'

EXAMPLES OF BASIC SQL QUERIES

- From within a host language, find the names and cities of customers with more than the variable *amount* dollars in some account.
- Specify the query in SQL and declare a *cursor* for it
- EXEC SQL
- **declare** *c cursor for*
select *customer-name, customer-city*
from *depositor, customer, account*
where *depositor.customer-name = customer.customer-name*
and *depositor account-number = account.account-number*
and *account.balance > :amount*

INTRODUCTION TO NESTED QUERIES

- SQL provides a mechanism for the nesting of subqueries.
- A subquery is a **select-from-where** expression that is nested within another query.
- A common use of subqueries is to perform tests for set membership, set comparisons, and set cardinality.

CORRELATED NESTED QUERIES

In the nested queries that we have seen thus far, the inner subquery has been completely independent of the outer query:

```
SELECT S.sname
FROM Sailors S
WHERE EXISTS ( SELECT *
FROM Reserves R
WHERE R.bid = 103
AND R.sid = S.sid )
```

SET - COMPARISON OPERATORS

- The set operations **union**, **intersect**, and **except** operate on relations and correspond to the relational algebra operations \cup , \cap , $-$.
- Each of the above operations automatically eliminates duplicates; to retain all duplicates use the corresponding multiset versions **union all**, **intersect all** and **except all**.

Suppose a tuple occurs m times in r and n times in s , then, it occurs:

- $m + n$ times in r **union all** s
- $\min(m, n)$ times in r **intersect all** s
- $\max(0, m - n)$ times in r **except all** s

AGGREGATE OPERATORS

We now consider a powerful class of constructs for computing aggregate values such as MIN and SUM.

1. COUNT ([DISTINCT] A): The number of (unique) values in the A column.
2. SUM ([DISTINCT] A): The sum of all (unique) values in the A column.
3. AVG ([DISTINCT] A): The average of all (unique) values in the A column.
4. MAX (A): The maximum value in the A column.
5. MIN (A): The minimum value in the A column.

NULL VALUES

- It is possible for tuples to have a null value, denoted by *null*, for some of their attributes
- *null* signifies an unknown value or that a value does not exist.
- The predicate **is null** can be used to check for null values.
 - E.g. Find all loan number which appear in the *loan* relation with null values for *amount*.
- **select** *loan-number*
from *loan*
where *amount is null*
- The result of any arithmetic expression involving *null* is *null*
 - E.g. $5 + \text{null}$ returns null
- However, aggregate functions simply ignore nulls
 - more on this shortly

NULL VALUES

- Any comparison with *null* returns *unknown*
 - E.g. $5 < \text{null}$ or $\text{null} <> \text{null}$ or $\text{null} = \text{null}$
- Three-valued logic using the truth value *unknown*:
 - OR: $(\text{unknown} \text{ or } \text{true}) = \text{true}$, $(\text{unknown} \text{ or } \text{false}) = \text{unknown}$
 $(\text{unknown} \text{ or } \text{unknown}) = \text{unknown}$
 - AND: $(\text{true} \text{ and } \text{unknown}) = \text{unknown}$, $(\text{false} \text{ and } \text{unknown}) = \text{false}$,
 $(\text{unknown} \text{ and } \text{unknown}) = \text{unknown}$
 - NOT: $(\text{not } \text{unknown}) = \text{unknown}$
 - “*P* is **unknown**” evaluates to true if predicate *P* evaluates to *unknown*
- Result of **where** clause predicate is treated as *false* if it evaluates to *unknown*

COMPARISONS USING NULL VALUES

Consider a comparison such as `rating = 8`. If this is applied to the row for Dan, is this condition true or false? Since Dan's rating is unknown, it is reasonable to say that this comparison should evaluate to the value unknown. SQL also provides a special comparison operator `IS NULL` to test whether a column value is null; for example, we can say `rating IS NULL`, which would evaluate to true on the row representing Dan. We can also say `rating IS NOT NULL`, which would evaluate to false on the row for Dan.



TRIGGERS

A trigger is a procedure that is automatically invoked by the DBMS in response to specified changes to the database, and is typically specified by the DBA. A database that has a set of associated triggers is called an active database. A trigger description contains three parts:

Event: A change to the database that activates the trigger.

Condition: A query or test that is run when the trigger is activated.

Action: A procedure that is executed when the trigger is activated and its condition is true .

A trigger action can examine the answers to the query in the condition part of the trigger, refer to old and new values of tuples modified by the statement activating the trigger, execute new queries, and make changes to the database.

<https://www.geeksforgeeks.org/sql-trigger-student-database/>

TRIGGERS

Triggers

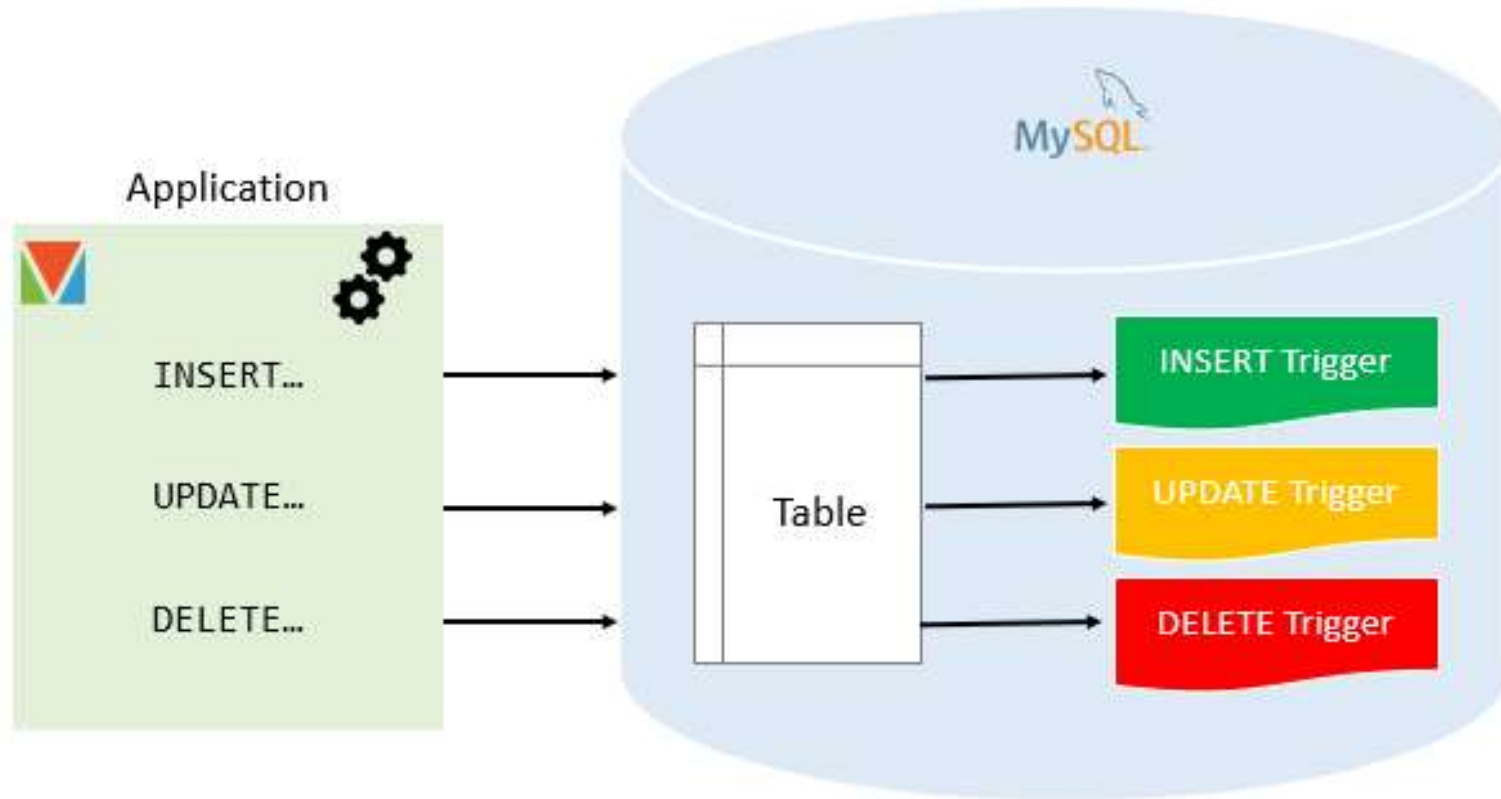
- What is a trigger?

Trigger is like a procedure that is automatically invoked by the DBMS in response to specified changes to data base

Trigger is like a 'Daemon that monitors a data base, and is executed when the data base is modified in a way that matches the event specification

A data base that has a set of associated triggers is called an active data base

TRIGGERS



TRIGGERS

Triggers in SQL



- DML Triggers

- DDL Triggers

- Logon Triggers

www.educba.com

<https://www.google.com/imgres?imgurl=http%3A%2F%2Fd1jnx9ba8s6j9r.cloudfront.net%2Fblog%2Fwp-content%2Fuploads%2F2019%2F10%2FTriggers-in-SQL.png&imgrefurl=https%3A%2F%2Fwww.edureka.co%2Fblog%2Ftriggers-in-sql%2F&tbnid=06rm3xlws4uejM&vet=12ahUKEwiNhMG1pOv1AhXHBbcAHfirDb8QMygBegUIARDFAQ..i&docid=oAQTiUF5XO2vGM&w=600&h=350&q=trigger%20in%20dbms&hl=en-GB&ved=2ahUKEwiNhMG1pOv1AhXHBbcAHfirDb8QMygBegUIARDFAQ>

ACTIVE DATABASES

[HTTPS://EN.WIKIPEDIA.ORG/WIKI/ACTIVE_DATABASE](https://en.wikipedia.org/wiki/Active_Database)

Active Database is a database consisting of set of triggers. These databases are very difficult to be maintained because of the complexity that arises in understanding the effect of these triggers. In such database, DBMS initially verifies whether the particular trigger specified in the statement that modifies the database) is activated or not, prior to executing the statement.

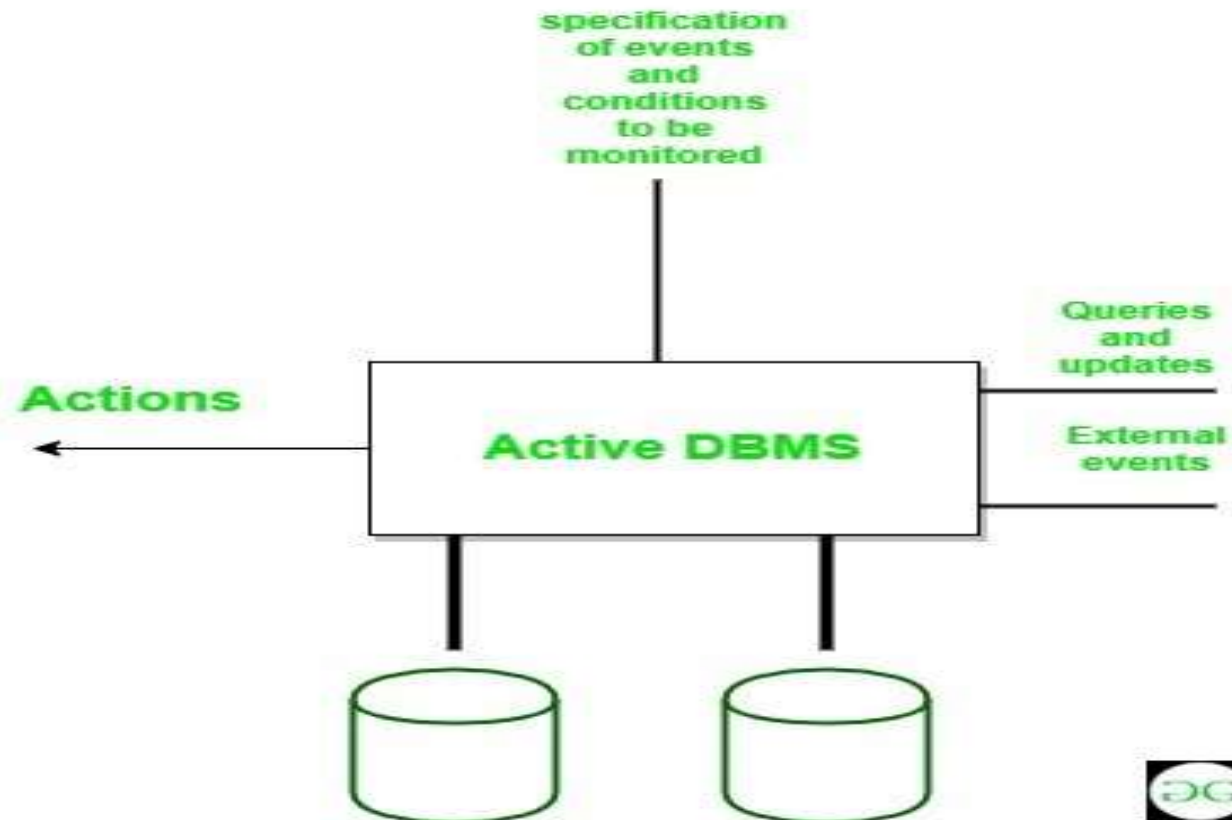
If the trigger is active then DBMS executes the condition part and then executes the action part only if the specified condition is evaluated to true. It is possible to activate more than one trigger within a single statement.

ACTIVE DATABASES

ACTIVE DATABASE

- ✦ **Traditional database systems are passive: respond to applications**
- ✦ **We may wish to extend the database systems by allowing them to invoke applications in response to an event**
- ✦ **For example, an inventory database system may initiate re-order transaction if the inventory level falls below specified level**

ACTIVE DATABASES



UNIT-3

INTRODUCTION TO SCHEMA REFINEMENT

Introduction To Schema Refinement- Problems Caused By Redundancy, Functional Dependencies, Normal Forms - FIRST, SECOND, THIRD Normal Forms – BCNF

Properties Of Decompositions: Lossless Join Decomposition, Dependency Preserving Decomposition - FOURTH Normal Form, FIFTH Normal Form.

INTRODUCTION TO SCHEMA REFINEMENT

- Problems Caused by redundancy
- Functional Dependencies
- Normal Forms
- FIRST, SECOND, THIRD Normal forms
- BCNF
- Properties of Decompositions
- Lossless join Decomposition
- Dependency preserving Decomposition
- FOURTH Normal Form
- FIFTH Normal form.

PROBLEMS CAUSED BY REDUNDANCY

- *Redundancy* is at the root of several problems associated with relational schemas:
 - *redundant storage, insert/delete/update anomalies*
- Integrity constraints, in particular *functional dependencies*, can be used to identify schemas with such problems and to suggest refinements.
- Main refinement technique: decomposition (replacing ABCD with, say, AB and BCD, or ACD and ABD).
- Decomposition should be used judiciously:
 - Is there reason to decompose a relation?
 - What problems (if any) does the decomposition cause?

PROBLEMS CAUSED BY REDUNDANCY

Problems of redundancy

If redundancy exists then this can cause problems during normal database operations:

- ④ Updates - when some data needs altering, we find it's in several places.
- ④ Insertion - When we want to insert something, we can't because some other data is also required which we don't have.
- ④ Deletion - When something is deleted, other data is lost as well.

FUNCTIONAL DEPENDENCIES

- Constraints on the set of legal relations.
- Require that the value for a certain set of attributes determines uniquely the value for another set of attributes.
- A functional dependency is a generalization of the notion of a *key*.

FUNCTIONAL DEPENDENCIES

FUNCTIONAL DEPENDENCIES

CONSIDER THE FOLLOWING RELATION

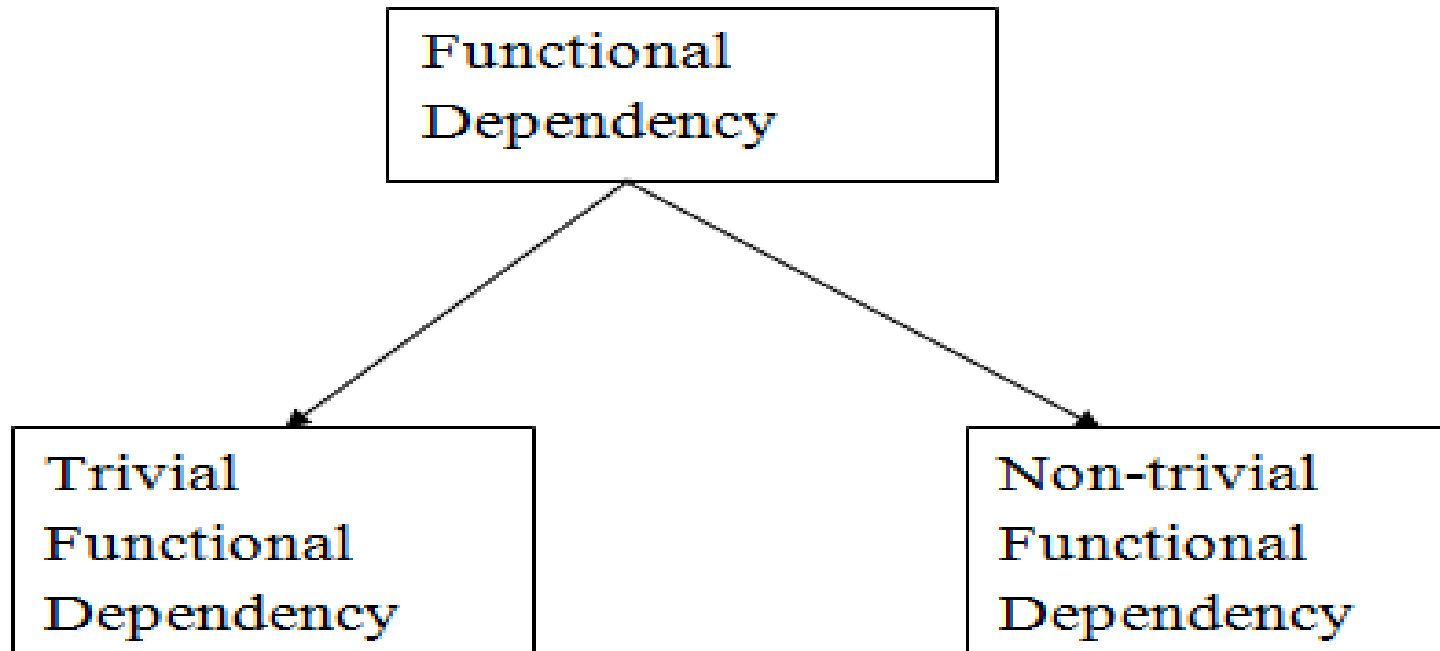
REPORT (STUDENT#, COURSE#, COURSENAME, INAME, ROOM#, MARKS, GRADE)

- **STUDENT#** - STUDENT NUMBER
- **COURSE#** - COURSE NUMBER
- **COURSENAME** - COURSE NAME
- **INAME** - NAME OF THE INSTRUCTOR WHO DELIVERED THE COURSE
- **ROOM#** - ROOM NUMBER WHICH IS ASSIGNED TO RESPECTIVE INSTRUCTOR
- **MARKS** - SCORED IN COURSE COURSE# BY STUDENT STUDENT#
- **GRADE** - OBTAINED BY STUDENT STUDENT# IN COURSE COURSE#

FUNCTIONAL DEPENDENCIES

- We use functional dependencies to:
 - test relations to see if they are legal under a given set of functional dependencies.
 - If a relation r is legal under a set F of functional dependencies, we say that r satisfies F .
 - specify constraints on the set of legal relations
 - We say that F holds on R if all legal relations on R satisfy the set of functional dependencies F .
- Note: A specific instance of a relation schema may satisfy a functional dependency even if the functional dependency does not hold on all legal instances.
 - For example, a specific instance of *Loan-schema* may, by chance, satisfy
 $loan-number \rightarrow customer-name$.

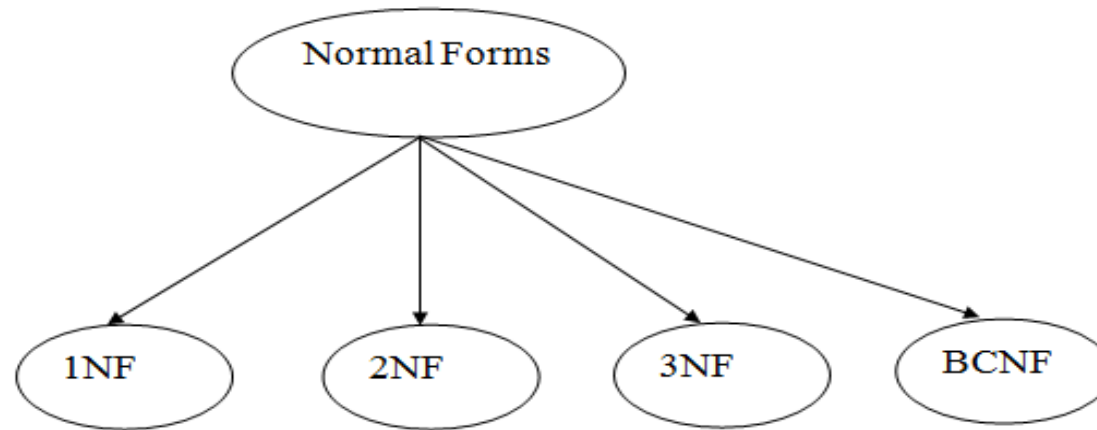
FUNCTIONAL DEPENDENCIES



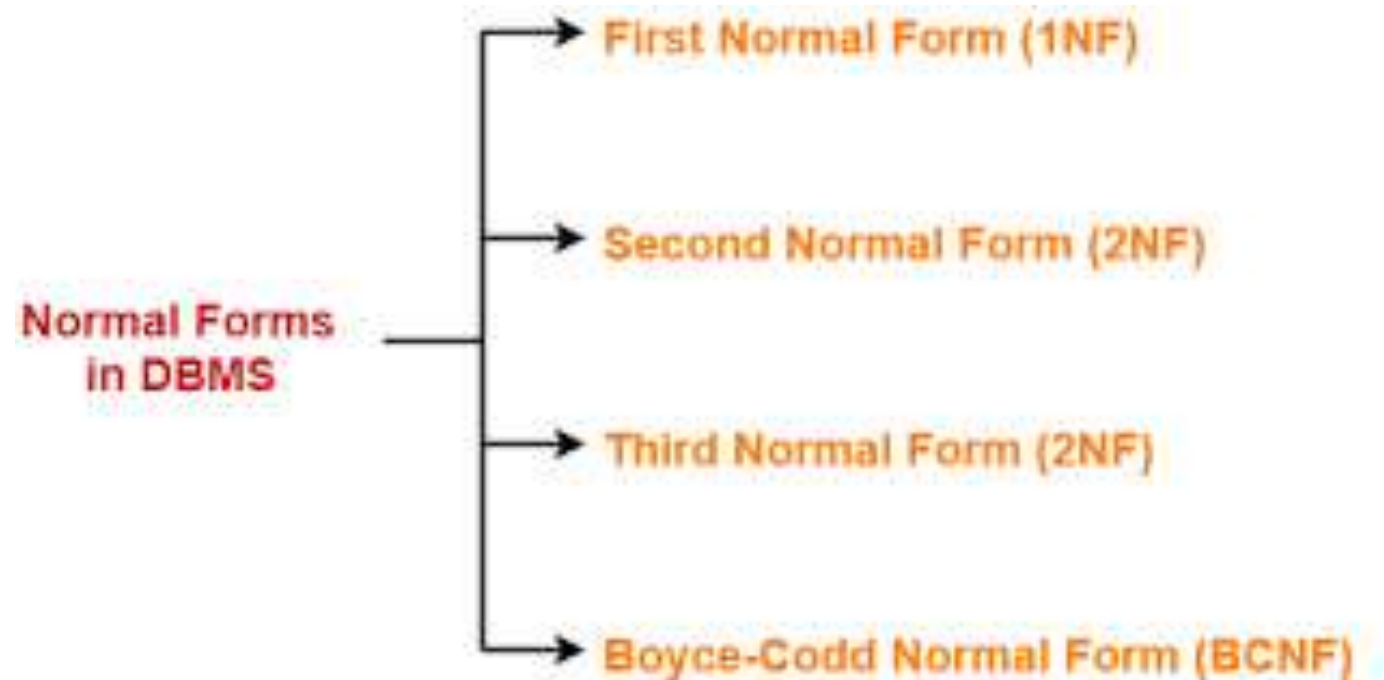
NORMAL FORMS

Types of Normal Forms

There are the four types of normal forms:



NORMAL FORMS



FIRST NORMAL FORMS

First Normal Form (1NF)

[< Prev](#)[Next >](#)

- A relation will be 1NF if it contains an atomic value.
- It states that an attribute of a table cannot hold multiple values. It must hold only single-valued attribute.
- First normal form disallows the multi-valued attribute, composite attribute, and their combinations.

Example: Relation EMPLOYEE is not in 1NF because of multi-valued attribute EMP_PHONE.

FIRST NORMAL FORM

EMPLOYEE table:

EMP_ID	EMP_NAME	EMP_PHONE	EMP_STATE
14	John	7272826385, 9064738238	UP
20	Harry	8574783832	Bihar
12	Sam	7390372389, 8589830302	Punjab

FIRST NORMAL FORM

The decomposition of the EMPLOYEE table into 1NF has been shown below:

EMP_ID	EMP_NAME	EMP_PHONE	EMP_STATE
14	John	7272826385	UP
14	John	9064738238	UP
20	Harry	8574783832	Bihar
12	Sam	7390372389	Punjab
12	Sam	8589830302	Punjab

SECOND NORMAL FORM

Second Normal Form (2NF)

[< Prev](#)[Next >](#)

- In the 2NF, relational must be in 1NF.
- In the second normal form, all non-key attributes are fully functional dependent on the primary key

Example: Let's assume, a school can store the data of teachers and the subjects they teach. In a school, a teacher can teach more than one subject.

SECOND NORMAL FORM

TEACHER table

TEACHER_ID	SUBJECT	TEACHER_AGE
25	Chemistry	30
25	Biology	30
47	English	35
83	Math	38
83	Computer	38

In the given table, non-prime attribute TEACHER_AGE is dependent on TEACHER_ID which is a proper subset of a candidate key. That's why it violates the rule for 2NF.

To convert the given table into 2NF, we decompose it into two tables:

SECOND NORMAL FORM

TEACHER_DETAIL table:

TEACHER_ID	TEACHER_AGE
25	30
47	35
83	38

TEACHER_SUBJECT table:

TEACHER_ID	SUBJECT
25	Chemistry
25	Biology
47	English
83	Math
83	Computer

SECOND NORMAL FORM



THIRD NORMAL FORM (3NF)

- Reln R with FDs F is in 3NF if, for all $X \twoheadrightarrow A$ in F
 - $A \in X$ (called a *trivial* FD), or
 - X contains a key for R , or
 - A is part of some key for R .
- *Minimality* of a key is crucial in third condition above!
- If R is in BCNF, obviously in 3NF.
- If R is in 3NF, some redundancy is possible. It is a compromise, used when BCNF not achievable (e.g., no “good” decomp, or performance considerations).
 - *Lossless-join, dependency-preserving decomposition of R into a collection of 3NF relations always possible.*

THIRD NORMAL FORM (3NF)

Third Normal Form

Teacher Details

ID	NAME	SUBJECT	STATE
29	Lalita	English	Gujrat
33	Ramesh	Geography	Punjab
49	Sarita	Mathematics	Maharashtra
78	Zayed	History	Bihar

Country Details

STATE	COUNTRY
Gujrat	INDIA
Punjab	INDIA
Maharashtra	INDIA
Bihar	INDIA

BCNF

- a relation schema R is in BCNF with respect to a set F of functional dependencies if for all functional dependencies in F^+ of the form $\alpha \twoheadrightarrow \beta$, where $\alpha \subseteq R$ and $\beta \subseteq R$, at least one of the following holds:
 - $\alpha \twoheadrightarrow \beta$ is trivial (i.e., $\beta \subseteq \alpha$)
 - α is a superkey for R
-
- <https://www.javatpoint.com/dbms-boyce-codd-normal-form>

BCNF

- Boyce and Codd Normal Form is a higher version of the Third Normal form. This form deals with certain type of anomaly that is not handled by 3NF. A 3NF table which does not have multiple overlapping candidate keys is said to be in BCNF. For a table to be in BCNF, following conditions must be satisfied:
- R must be in 3rd Normal Form
- and, for each functional dependency ($X \rightarrow Y$), X should be a super Key.
- A relation is said to undergo BCNF normal form if it is in 3rd normal form and if the following systems are noticed.
- The relation will have multiple composite keys
- Further the composite keys will have common attribute
- And the key of first composite key is functionally dependent on key of other composite key.

BCNF

Boyce code normal form

- BCNF is a normal form used in database normalization.
- BCNF was developed in 1974 by Raymond F. Boyce and Edgar F. Codd
- Normalization is the process of efficiently organizing data in a database .

BCNF

Consider the following relationship : **R (A,B,C,D)**

and following dependencies :

A -> BCD

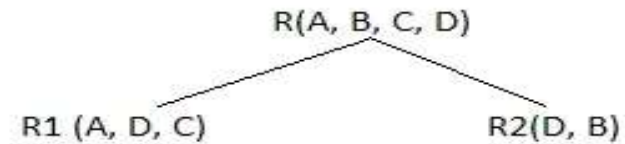
BC -> AD

D -> B

Above relationship is already in 3rd NF. Keys are **A** and **BC**.

Hence, in the functional dependency, **A -> BCD**, A is the super key.
in second relation, **BC -> AD**, BC is also a key.
but in, **D -> B**, D is not a key.

Hence we can break our relationship R into two relationships **R1** and **R2**.



Breaking, table into two tables, one with A, D and C while the other with D and B.

PROPERTIES OF DECOMPOSITIONS

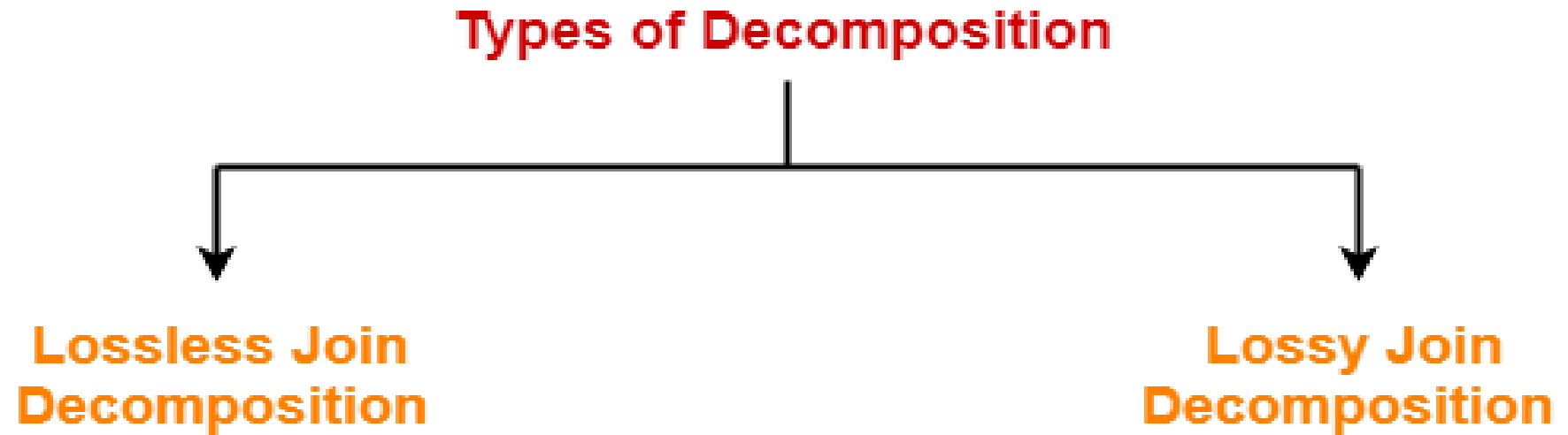
- There are three potential problems to consider:
 - Some queries become more expensive.
 - e.g., How much did sailor Joe earn? (salary = $W * H$)
- Given instances of the decomposed relations, we may not be able to reconstruct the corresponding instance of the original relation!
 - Fortunately, not in the SNLRWH example.
- Checking some dependencies may require joining the instances of the decomposed relations.
 - Fortunately, not in the SNLRWH example.
- Tradeoff: Must consider these issues vs. redundancy.

LOSSLESS JOIN DECOMPOSITION

- Decomposition of R into X and Y is lossless-join w.r.t. a set of FDs F if, for every instance r that satisfies F:
 - $\pi_X(r) \bowtie \pi_Y(r) = r$
- Definition extended to decomposition into 3 or more relations in a straightforward way.
- *It is essential that all decompositions used to deal with redundancy be lossless! (Avoids Problem (2).)*

<https://www.geeksforgeeks.org/lossless-decomposition-in-dbms/>

PROPERTIES OF DECOMPOSITIONS



PROPERTIES OF DECOMPOSITIONS

Properties of decomposition

- Lossless join decomposition
 - Enable us to recover any instance of the decomposed relation from corresponding instances of the smaller relations.
- Dependency-preservation decomposition
 - Enable us to enforce any constraints on the original relation by simple enforcing some constraints on each of smaller relations.
 - Key constrains

S	N	L	R	H
123-22-3666	Attishoo	48	8	40
231-31-5368	Smiley	22	8	30
131-24-3650	Smethurst	35	5	30
434-26-3751	Guldu	35	5	32
612-67-4134	Madayan	35	8	40

Hourly_Emps2

R	W
8	10
5	7

Wages

https://www.google.com/url?sa=i&url=https%3A%2F%2Fslideplayer.com%2Fslide%2F11664980%2F&psig=AOvVaw3S84oT_bvRai8WIFfNfjHq&ust=1644245419934000&source=images&cd=vfe&ved=0CAsQjRxqFwoTCNDh6PPD7PUCFQAAAAAdAAAAABAD

PROPERTIES OF DECOMPOSITIONS

Dependency preservation

Getting lossless decomposition is necessary. But of course, we also want to keep dependencies, since losing a dependency means, that the corresponding constraint can be checked only through natural join of the appropriate resultant relation in the decomposition. This would be very expensive, so, our aim is to get a lossless dependency preserving decomposition.

Example:

$R=(A, B, C)$, $F=\{A \rightarrow B, B \rightarrow C\}$

Decomposition of R : $R_1=(A, C)$ $R_2=(B, C)$

Does this decomposition preserve the given dependencies?

Solution:

In R_1 the following dependencies hold: $F_1'=\{A \rightarrow A, C \rightarrow C, A \rightarrow C, AC \rightarrow AC\}$

In R_2 the following dependencies hold: $F_2'=\{B \rightarrow B, C \rightarrow C, B \rightarrow C, BC \rightarrow BC\}$

The set of nontrivial dependencies hold on R_1 and R_2 : $F'=\{B \rightarrow C, A \rightarrow C\}$

$A \rightarrow B$ can not be derived from F' , so this decomposition is NOT dependency preserving.

DEPENDENCY PRESERVING DECOMPOSITION

A decomposition of a relation R into $R_1, R_2, R_3, \dots, R_n$ is dependency preserving decomposition with respect to the set of Functional Dependencies F that hold on R only if the following is hold;

- $(F_1 \cup F_2 \cup F_3 \cup \dots \cup F_n)^+ = F^+$ where,
- $F_1, F_2, F_3, \dots, F_n$ – Sets of Functional dependencies of relations $R_1, R_2, R_3, \dots, R_n$.
- $(F_1 \cup F_2 \cup F_3 \cup \dots \cup F_n)^+$ - Closure of Union of all sets of functional dependencies.
- F^+ - Closure of set of functional dependency F of R .
- If the closure of set of functional dependencies of individual relations $R_1, R_2, R_3, \dots, R_n$ are equal to the set of functional dependencies of the main relation R (before decomposition), then we would say the decomposition D is lossless dependency preserving decomposition.

FOURTH NORMAL FORM

- A relation schema R is in 4NF with respect to a set D of functional and multivalued dependencies if for all multivalued dependencies in D^+ of the form $\alpha \twoheadrightarrow \beta$, where $\alpha \subseteq R$ and $\beta \subseteq R$, at least one of the following hold:
 - $\alpha \twoheadrightarrow \beta$ is trivial (i.e., $\beta \subseteq \alpha$ or $\alpha \cup \beta = R$)
 - α is a superkey for schema R
- If a relation is in 4NF it is in BCNF

FOURTH NORMAL FORM

Fourth Normal Form

Course	Instructor	TextBook_Author
Management	X	Churchil
Management	Y	Peters
Management	Z	Peters
Finance	A	Weston
Finance	A	Gilbert

Course

Course	Instructor
Management	X
Management	Y
Management	Z
Finance	A

Textbook_Author

Course	TextBook_Author
Management	Churchil
Management	Peters
Finance	Weston
Finance	Gilbert

FIFTH NORMAL FORM

- If a relation schema is in 3NF and each of its keys consists of a single attribute, it is also in 5NF.
- The conditions identified in this result are sufficient for a relation to be in 5NF but not necessary. The result can be very useful in practice because it allows us to conclude that a relation is in 5NF 'Without ever ' identifying the MVDs and JDs that may hold over the relation.
- <https://www.javatpoint.com/dbms-fifth-normal-form>

FIFTH NORMAL FORM

- The second condition deserves same explanation, since we have not presented inference rules for FDs and FDs taken together. Intuitively, we must be able to show that the decomposition of R into $\{R_1, \dots, R_n\}$ is lossless-join whenever the key dependencies hold. $JD \rightarrow J \{R_1, \dots, R_n\}$ is a trivial JD if $R_i = R$ for some i ; such a JD always holds. The following result, also due to Date and Fagin, identifies conditions. Again, detected using only FD information---under which we can safely ignore JD information:

FIFTH NORMAL FORM

Fifth Normal Form (5NF)

- A relation that has no join dependency.

(a) PropertyItemSupplier (Illegal state)

propertyNo	itemDescription	supplierNo
PG4	Bed	S1
PG4	Chair	S2
PG16	Bed	S2

When this tuple is added to relation.

(b) PropertyItemSupplier (Legal state)

propertyNo	itemDescription	supplierNo
PG4	Bed	S1
PG4	Chair	S2
PG16	Bed	S2
PG4	Bed	S2

This new tuple must also be added to exist in any legal state of the relation.

https://www.google.com/url?sa=i&url=https%3A%2F%2Fwww.toppr.com%2Fask%2Fquestion%2F5-nf-is-related-to%2F&psig=AOvVaw2GLCk88VR_E_zPZYYecNuv&ust=1644245537126000&source=images&cd=vfe&ved=0CAsQjRxqFwoTCNC3y6vE7PUCFQAAAdAAAAABAD



UNIT-4

TRANSACTION AND CONCURRENCY

Transaction And Concurrency: Transaction Concepts – ACID Properties – Transactions And Schedules- Transaction States - Concurrent Execution, Serializability - Types Of Failures.

TRANSACTION AND CONCURRENCY

- Transaction Concepts
- ACID Properties
- Transactions and Schedules
- Transaction States
- Concurrent Execution
- Serializability
- Types of Failures

WHAT IS A TRANSACTION?

- A logical unit of work on a database
 - An entire program
 - A portion of a program
 - A single command
- The entire series of steps necessary to accomplish a logical unit of work
- Successful transactions change the database from one CONSISTENT STATE to another
- (One where all data integrity constraints are satisfied)

TRANSACTION CONCEPTS

- A transaction is a unit of program execution that accesses and possibly updates various data items.
- Usually, a transaction is initiated by a user program written in a high-level data-manipulation language (typically SQL), or programming language (for example, C++, or Java), with embedded database accesses in JDBC or ODBC. A transaction is delimited by statements (or function calls) of the form *begin transaction* and *end transaction*.
- The transaction consists of all *operations* executed between the *begin transaction* and *end transaction*.

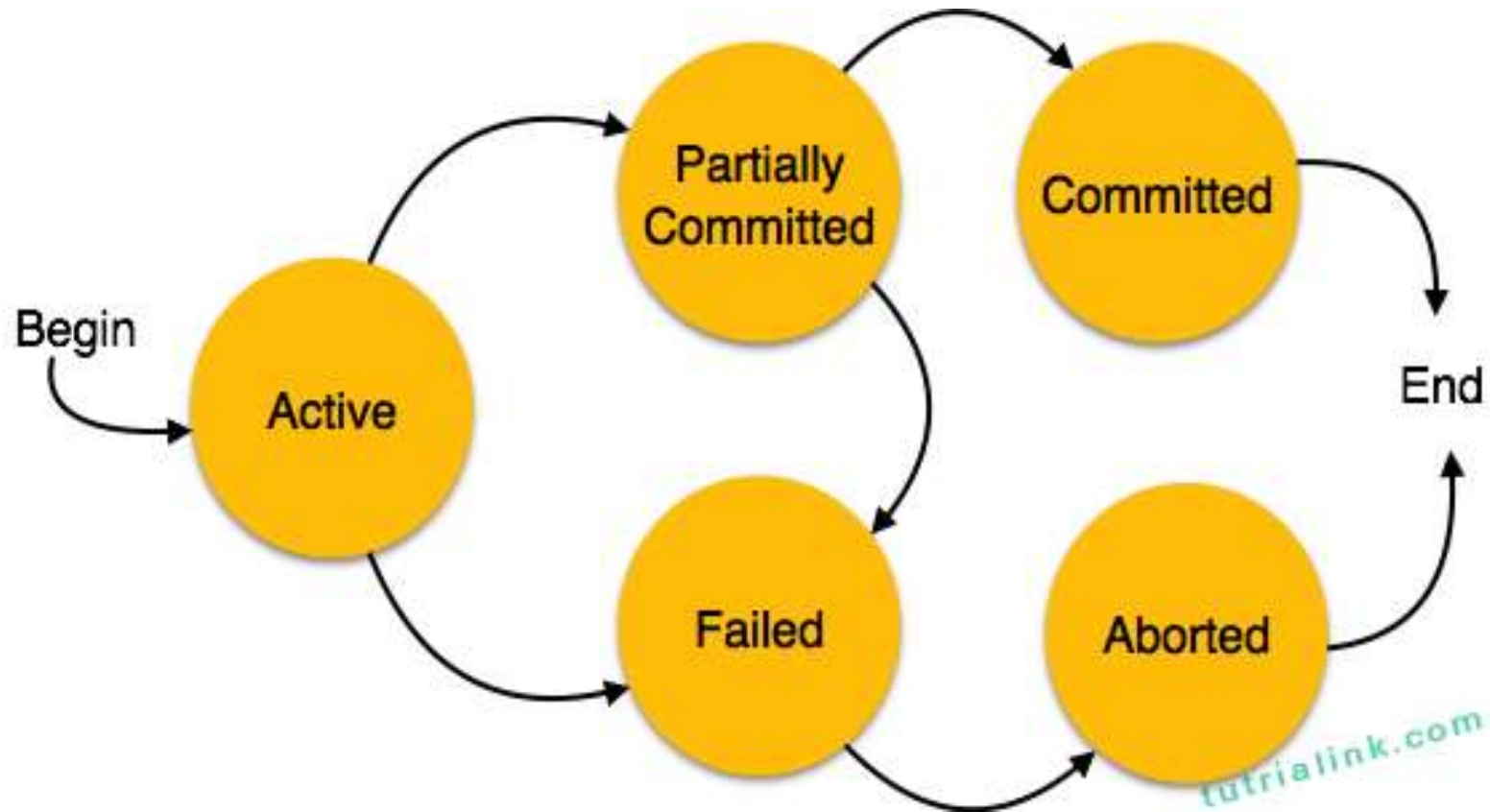
TRANSACTION CONCEPTS

- **Y's Account**
- `Open_Account(Y)`
- `Old_Balance = Y.balance`
- `New_Balance = Old_Balance + 800`
- `Y.balance = New_Balance`
- `Close_Account(Y)`

TRANSACTION CONCEPTS

- **X's Account**
- `Open_Account(X)`
- $\text{Old_Balance} = \text{X.balance}$
- $\text{New_Balance} = \text{Old_Balance} - 800$
- $\text{X.balance} = \text{New_Balance}$
- `Close_Account(X)`

TRANSACTION CONCEPTS

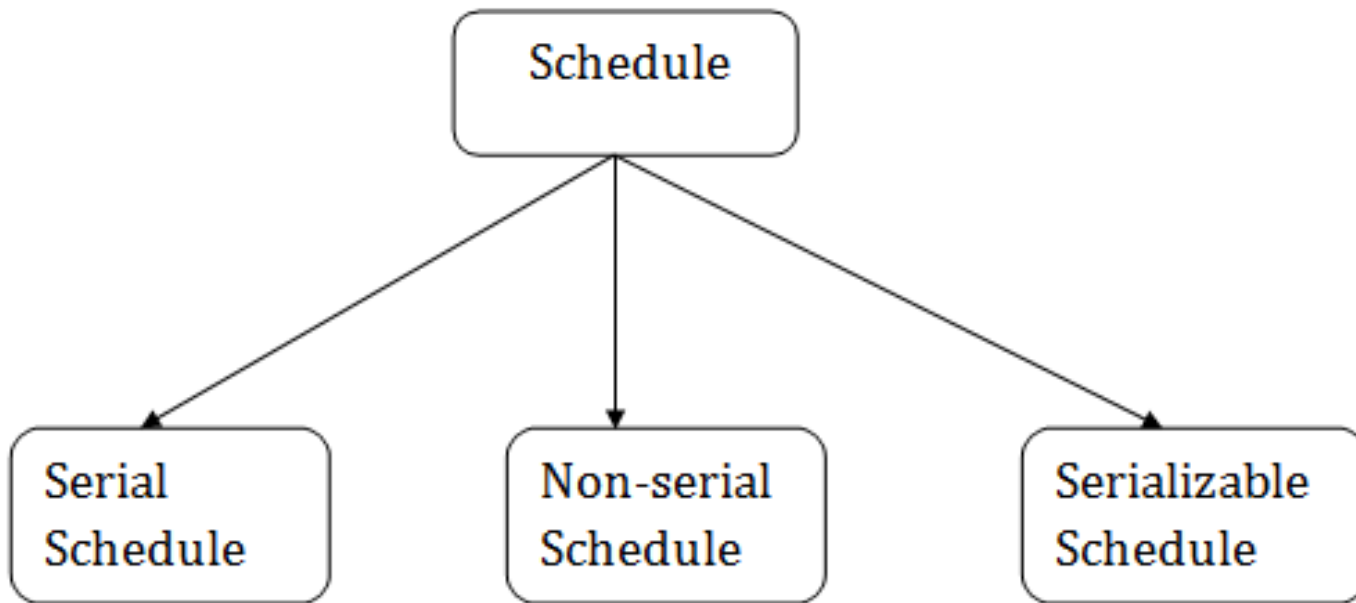


ACID PROPERTIES

- A Transaction is a single logical unit of work which accesses and possibly modifies the contents of a database. Transactions access data using read and write operations.

In order to maintain consistency in a database, before and after the transaction, certain properties are followed. These are called **ACID** properties.

ACID PROPERTIES



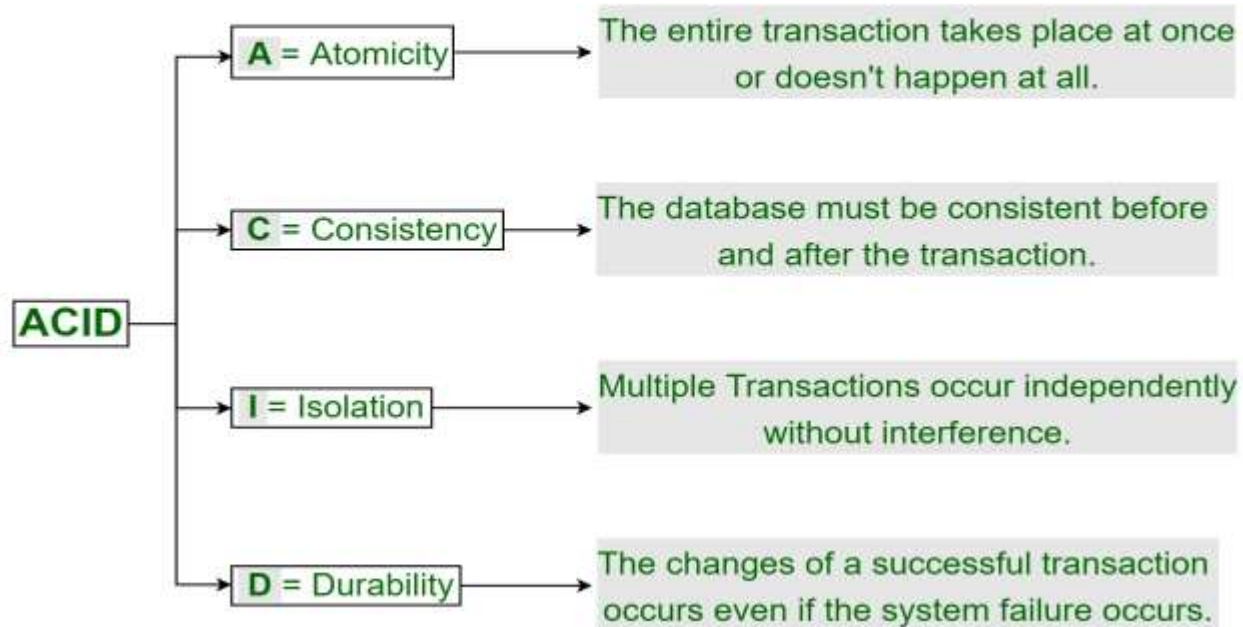
https://www.google.com/imgres?imgurl=https%3A%2F%2Fstatic.javatpoint.com%2Fdbms%2Fimages%2Facid-properties-in-dbms.png&imgrefurl=https%3A%2F%2Fwww.javatpoint.com%2Facid-properties-in-dbms&tbnid=4NUrVtUJ9e_qZM&vet=12ahUKEwiR7t-Eq-v1AhXGYmwGHTDSCI8QMygMegUIARDWAQ..i&docid=Zp9J8R7fFm4m3M&w=700&h=239&q=acid%20properties%20in%20dbms&hl=en-GB&ved=2ahUKEwiR7t-Eq-v1AhXGYmwGHTDSCI8QMygMegUIARDWAQ

ACID PROPERTIES

- **Abort:** If a transaction aborts, changes made to database are not visible.
- **Commit:** If a transaction commits, changes made are visible.
Atomicity is also known as the 'All or nothing rule'. Consider the following transaction **T** consisting of **T1** and **T2**: Transfer of 100 from account **X** to account **Y**.
- <https://www.geeksforgeeks.org/acid-properties-in-dbms/>

ACID PROPERTIES

ACID Properties in DBMS



ACID PROPERTIES

Before: X : 500	Y: 200
Transaction T	
T1	T2
Read (X) X: = X - 100 Write (X)	Read (Y) Y: = Y + 100 Write (Y)
After: X : 400	Y : 300

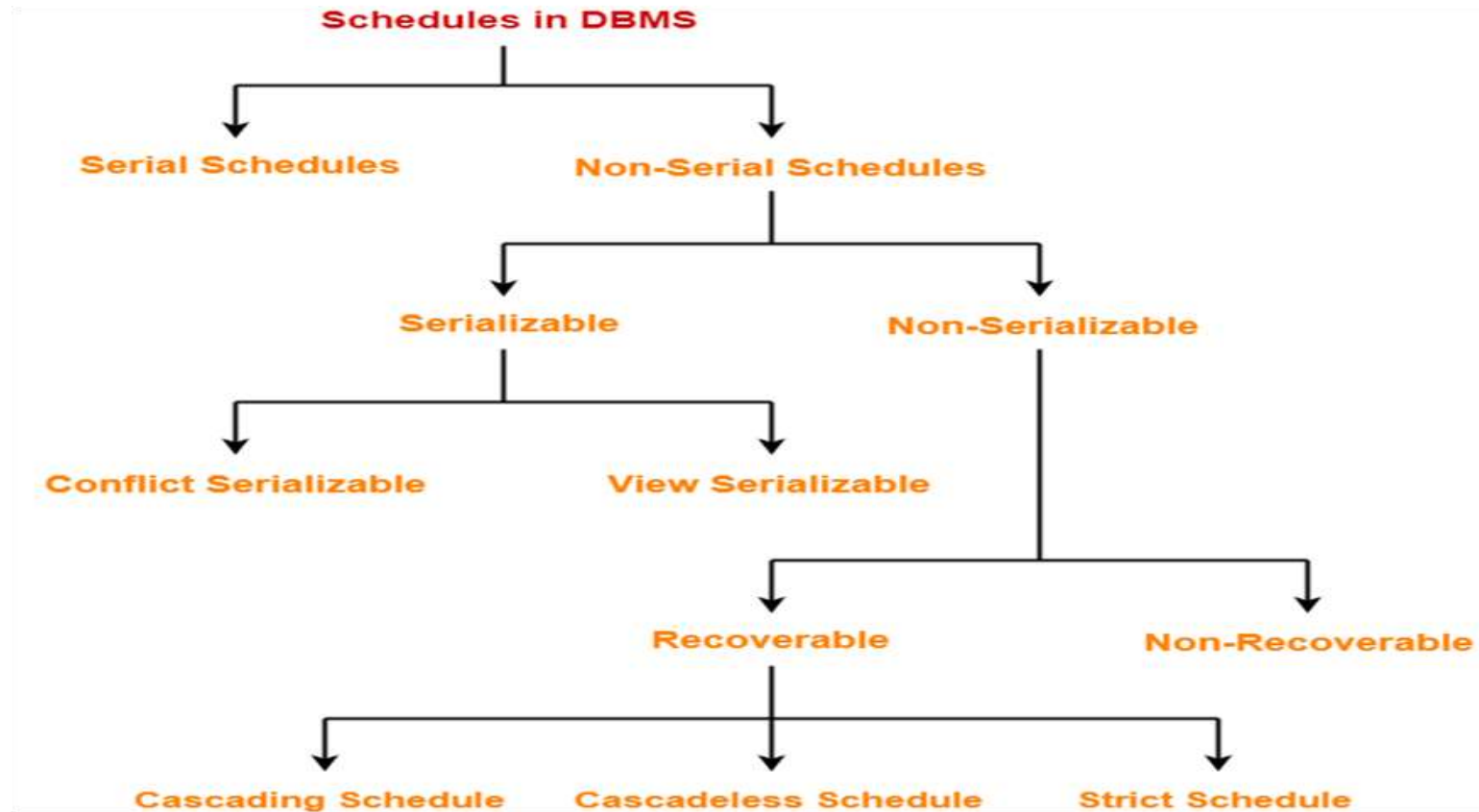
TRANSACTIONS AND SCHEDULES

- A series of operation from one transaction to another transaction is known as schedule. It is used to preserve the order of the operation in each of the individual transaction.
- special DBMS program: establishes order of operations within which concurrent transactions are executed
- Interleaves the execution of database operations to ensure serializability and isolation of transactions
 - To determine the appropriate order, the scheduler bases its actions on concurrency control algorithms such as locking and time stampingSp

TRANSACTIONS AND SCHEDULES

- Special DBMS Program to establish the order of operations in which concurrent transactions are executes
- Interleaves the execution of database operations to ensure:
- Serializability
- Isolation of Transactions

TYPES OF SCHEDULES



TRANSACTION STATE

- States through which a transaction goes during its lifetime.
These are the states which tell about the current state of the Transaction and also tell how we will further do the processing in the transactions. These states govern the rules which decide the fate of the transaction whether it will commit or abort.
- <https://www.javatpoint.com/dbms-states-of-transaction>

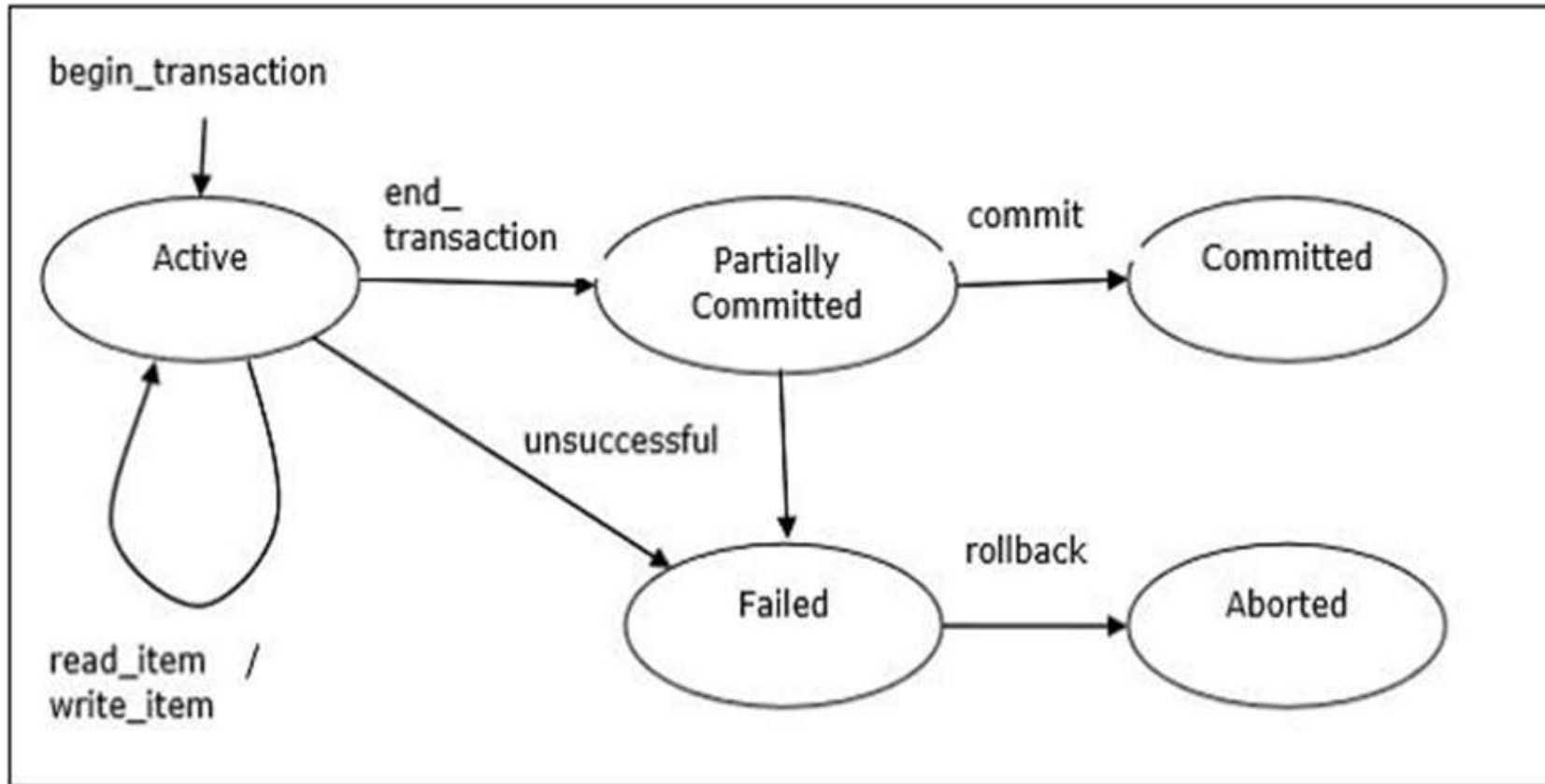
TRANSACTION STATE

Transaction State

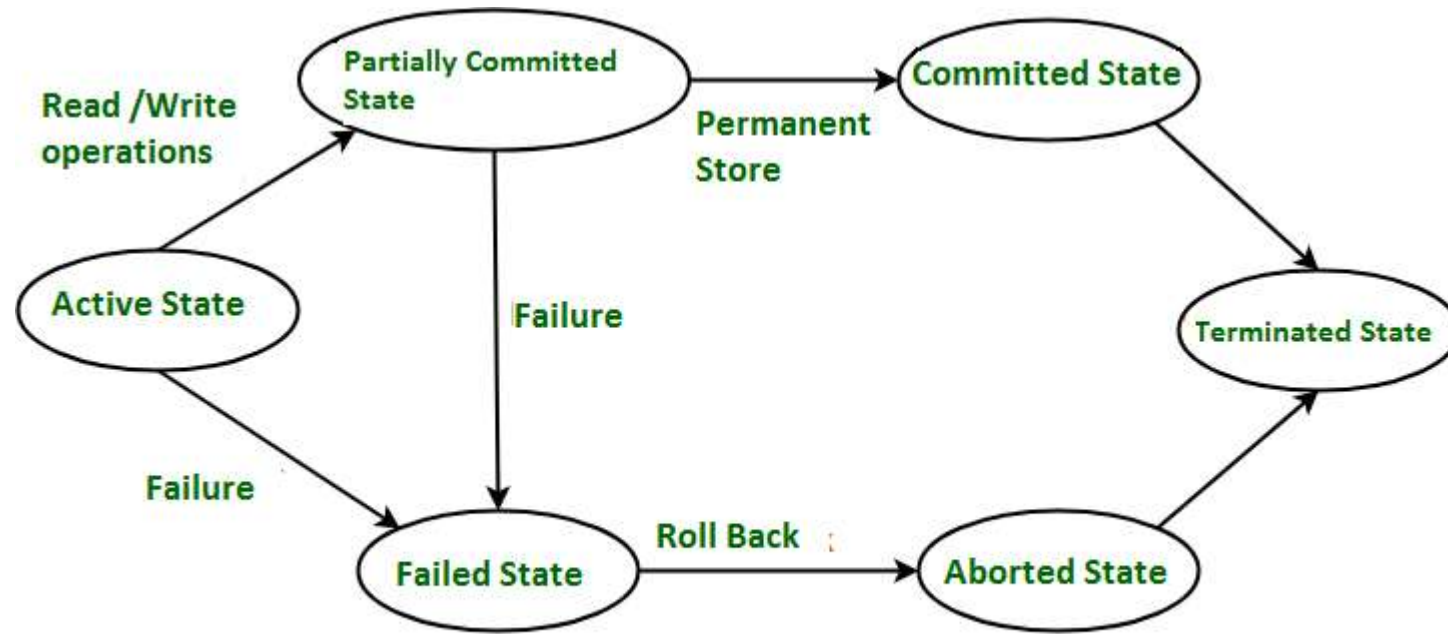
- “ **Active** – the initial state; the transaction stays in this state while it is executing
- “ **Partially committed** – after the final statement has been executed.
- “ **Failed** – after the discovery that normal execution can no longer proceed.
- “ **Aborted** – after the transaction has been rolled back and the database restored to its state prior to the start of the transaction. Two options after it has been aborted:
 - “ restart the transaction
 - “ can be done only if no internal logical error
 - “ kill the transaction
- “ **Committed** – after successful completion.



TRANSACTION STATE



TRANSACTION STATE



Transaction States in DBMS

TRANSACTION STATE

- **Active** – In this state, the transaction is being executed. This is the initial state of every transaction.
- ☐ **Partially Committed** – When a transaction executes its final operation, it is said to be in a partially committed state.
- ☐ **Failed** – A transaction is said to be in a failed state if any of the checks made by the database recovery system fails. A failed transaction can no longer proceed further.
- ☐ **Aborted** – If any of the checks fails and the transaction has reached a failed state, then the recovery manager rolls back all its write operations on the database to bring the database back to its original state where it was prior to the execution of the transaction. Transactions in this state are called aborted. The database recovery module can select one of the two operations after a transaction aborts –
 - o Re-start the transaction
 - o Kill the transaction
- ☐ **Committed** – If a transaction executes all its operations successfully, it is said to be committed. All its effects are now permanently established on the database system.

CONCURRENT EXECUTION

In the transaction process, a system usually allows executing more than one transaction simultaneously. This process is called a concurrent execution.

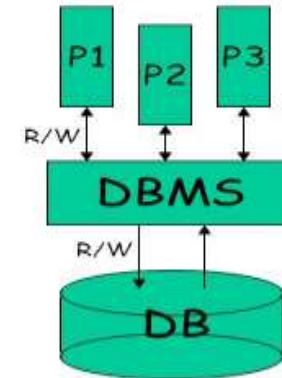
Advantages of concurrent execution of a transaction

- Decrease waiting time or turnaround time.
- Improve response time
- Increased throughput or resource utilization.

CONCURRENT EXECUTION

Concurrent Execution

- Concurrent execution of user programs is essential for good DBMS performance.
 - Disk accesses are frequent, and relatively slow.
 - Want to keep the CPU working on several user programs concurrently.
- **Challenges**
 - Concurrency Control: How do the DBMSs handle concurrent transactions?
 - Crash Recovery: How do the DBMSs handle partial transactions because of machine crashes or users abort the transactions ?



SERIALIZABILITY

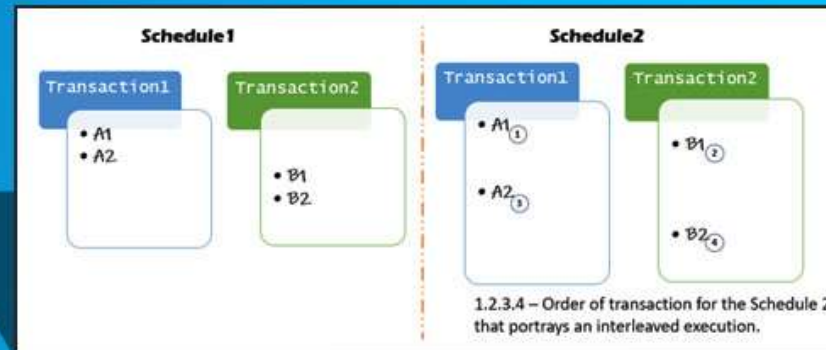
- When multiple transactions are being executed by the operating system in a multiprogramming environment, there are possibilities that instructions of one transactions are interleaved with some other transaction.
- If 2 Transactions are only reading data items – They do not conflict ☐
Order is unimportant
- If 2 Transactions operate (Read/Write) on Separate Data Items
- – They do not conflict ☐ Order is unimportant
- If 1 Transaction Writes to a Data Item and Another Reads or Writes to the Same Data Item ☐ The Order of Execution IS Important
- <https://www.tutorialspoint.com/what-is-the-term-serializability-in-dbms>

VIEW OF SERIALIZABLE

- **Schedule** – A chronological execution sequence of a transaction is called a schedule. A schedule can have many transactions in it, each comprising of a number of instructions/tasks.
- **Serial Schedule** – It is a schedule in which transactions are aligned in such a way that one transaction is executed first. When the first transaction completes its cycle, then the next transaction is executed. Transactions are ordered one after the other. This type of schedule is called a serial schedule, as transactions are executed in a serial manner.
- To resolve this problem, we allow parallel execution of a transaction schedule, if its transactions are either serializable or have some equivalence relation among them.
- **Equivalence Schedules:**
- If two schedules produce the same result after execution, they are said to be result equivalent. They may yield the same result for some value and different results for another set of values. That's why this equivalence is not generally considered significant.

SERIALIZABILITY

Serializability in DBMS



SERIALIZABILITY

View Serializability

1. Initial Read

T1	T2
Read(A)	Write(A)

Schedule 1

T1	T2
Read(A)	Write(A)

Schedule 2

2. Updated Read

T1	T2	T3
Write(A)	Write(A)	Read(A)

Schedule 1

T1	T2	T3
Write(A)	Write(A)	Read(A)

Schedule 2

3. Final Write

T1	T2	T3
Write(A)	Read(A)	Write(A)

Schedule 1

T1	T2	T3
Write(A)	Read(A)	Write(A)

Schedule 2

TYPES OF FAILURES

- To find that where the problem has occurred, we generalize a failure into the following categories:
- Transaction failure
- System crash
- Disk failure

TYPES OF FAILURES

1. Transaction failure

- The transaction failure occurs when it fails to execute or when it reaches a point from where it can't go any further. If a few transaction or process is hurt, then this is called as transaction failure.

2. System Crash

- System failure can occur due to power failure or other hardware or software failure. **Example:** Operating system error.

3. Disk Failure

- It occurs where hard-disk drives or storage drives used to fail frequently. It was a common problem in the early days of technology evolution.
- Disk failure occurs due to the formation of bad sectors, disk head crash, and unreachability to the disk or any other failure, which destroy all or part of disk storage.

TRANSACTION FAILURE

Classification

Transaction failure

Types of errors that may cause transaction failure :

- ➔ • **Logical errors**: transaction cannot complete due to some internal error condition (e.g. division by zero)
- ➔ • **System errors**: the database system must terminate an active transaction due to an error condition (e.g., deadlock)

TRANSACTION FAILURE

1. Failure classification

A. System crash: caused by

power failure, hardware or software failure

The non-volatile storage content is not corrupted

B. Disk failure: head crash, surface scratch

The disk storage is destroyed completely or in part

C. Transaction failure:

1. **Logical errors:** transaction cannot complete due to
some internal error condition

2. **System errors:** DBMS must terminate an active
transaction due to an error condition
(e.g. deadlock)

SYSTEM CRASH DISK FAILURE



2. System Crash

- There are problems – external to the system – that may cause the system to stop abruptly and cause the system to crash.

3. Disk Failure

- Disk failures include formation of bad sectors, unreachability to the disk, disk head crash or any other failure, which destroys all or a part of disk storage.



UNIT -5

Transaction And Concurrency: Transaction Concepts – ACID Properties – Transactions And Schedules- Transaction States - Concurrent Execution, Serializability - Types Of Failures.

RECOVERABILITY

- System Recovery
- Media Recovery
- Two Phase locking
- Deadlock-Detection
- Recovery and Prevention.
- Physical Storage and Database Concepts:
- Overview of Physical Storage Media and RAID

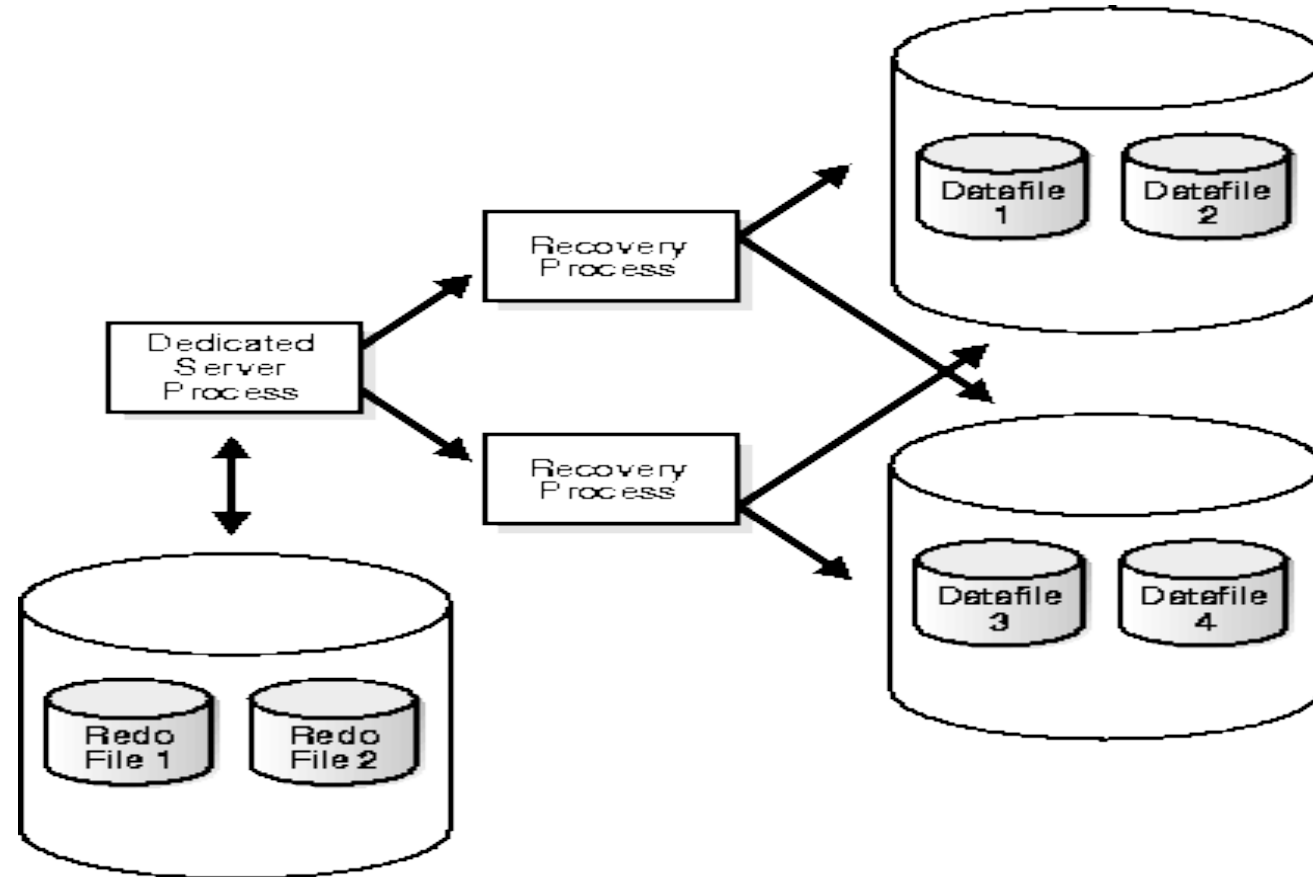
RECOVERY

- Database recovery
 - Restores database from a given state, usually inconsistent, to a previously consistent state
 - Based on the atomic transaction property
 - All portions of the transaction must be treated as a single logical unit of work, in which all operations must be applied and completed to produce a consistent database
 - If transaction operation cannot be completed, transaction must be aborted, and any changes to the database must be rolled back (undone)

SYSTEM RECOVERY

- When the system recovers from failure, it can restore the latest dump.
- It can maintain redo-list and undo-list as in checkpoints.
- It can recover the system by consulting undo-redo lists to restore the state of all transaction
- up to last checkpoint

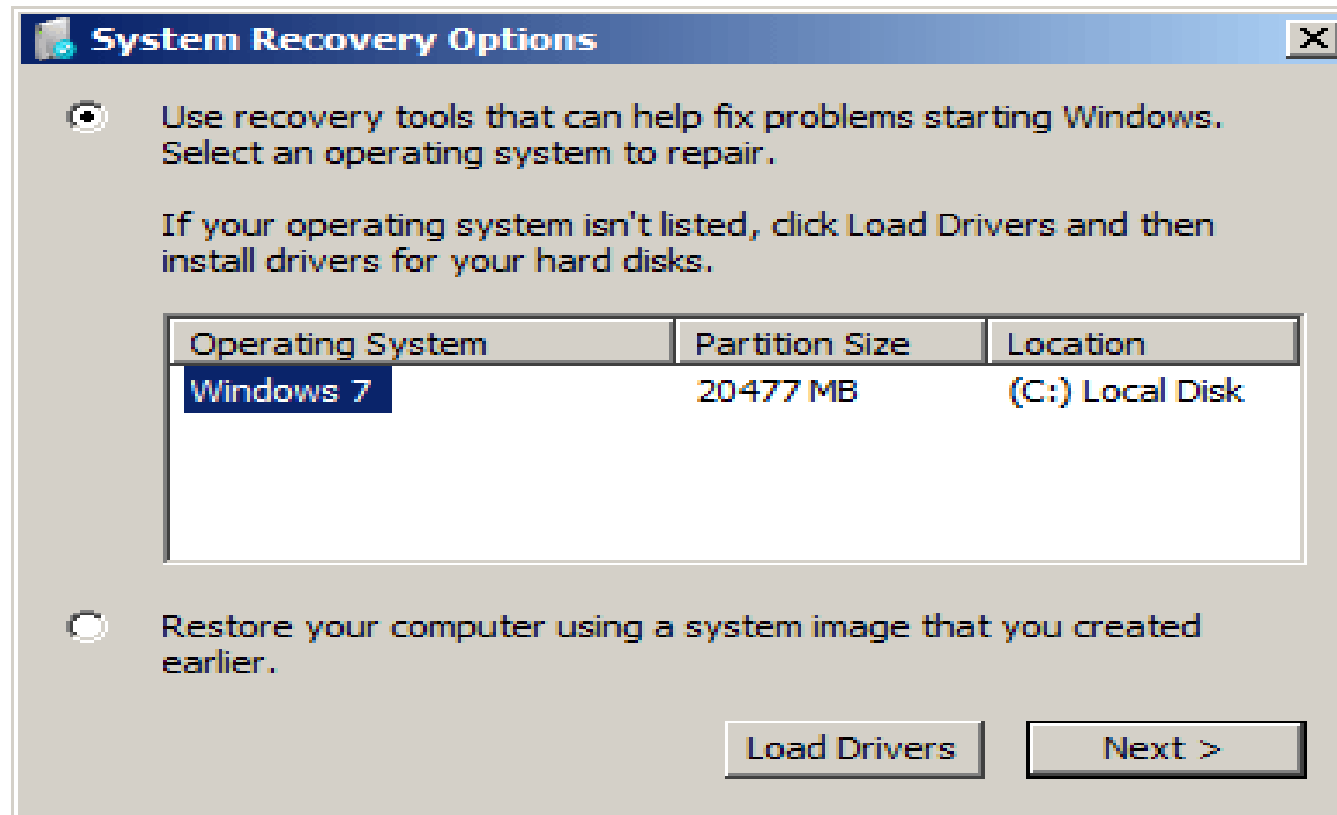
SYSTEM RECOVERY



SYSTEM RECOVERY



SYSTEM RECOVERY



The image shows a Windows System Recovery Options dialog box. It has a title bar with the text "System Recovery Options" and a close button. There are two radio buttons at the top. The first radio button is selected and is accompanied by the text "Use recovery tools that can help fix problems starting Windows. Select an operating system to repair." Below this text is a paragraph: "If your operating system isn't listed, click Load Drivers and then install drivers for your hard disks." Underneath the paragraph is a table with three columns: "Operating System", "Partition Size", and "Location". The table contains one row with the text "Windows 7", "20477 MB", and "(C:) Local Disk". The second radio button is unselected and is accompanied by the text "Restore your computer using a system image that you created earlier." At the bottom right of the dialog box are two buttons: "Load Drivers" and "Next >".

System Recovery Options

☒ Use recovery tools that can help fix problems starting Windows. Select an operating system to repair.

If your operating system isn't listed, click Load Drivers and then install drivers for your hard disks.

Operating System	Partition Size	Location
Windows 7	20477 MB	(C:) Local Disk

☐ Restore your computer using a system image that you created earlier.

Load Drivers Next >

MEDIA RECOVERY

- Media recovery can be a complete recovery or a point-in-time recovery. Complete recovery can apply to individual data files, table spaces, or the entire database. Point-in-time recovery applies to the whole database (and also sometimes to individual table spaces, with automation help from Oracle Recover Manager (RMAN)).
- RMAN enables you to perform both a complete and a point-in-time recovery of your database. However, this documentation focuses on complete recovery.

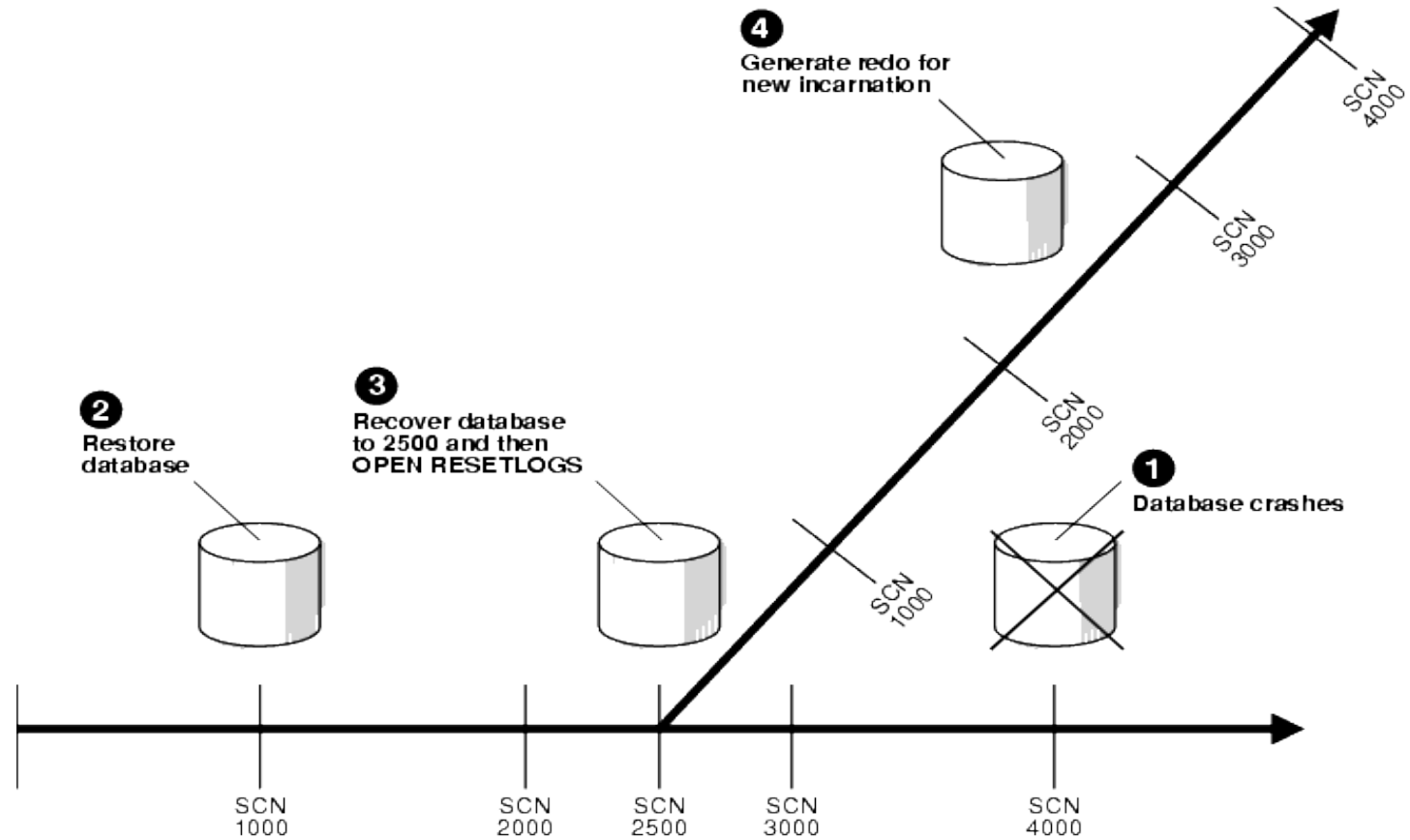
MEDIA RECOVERY

Media Recovery

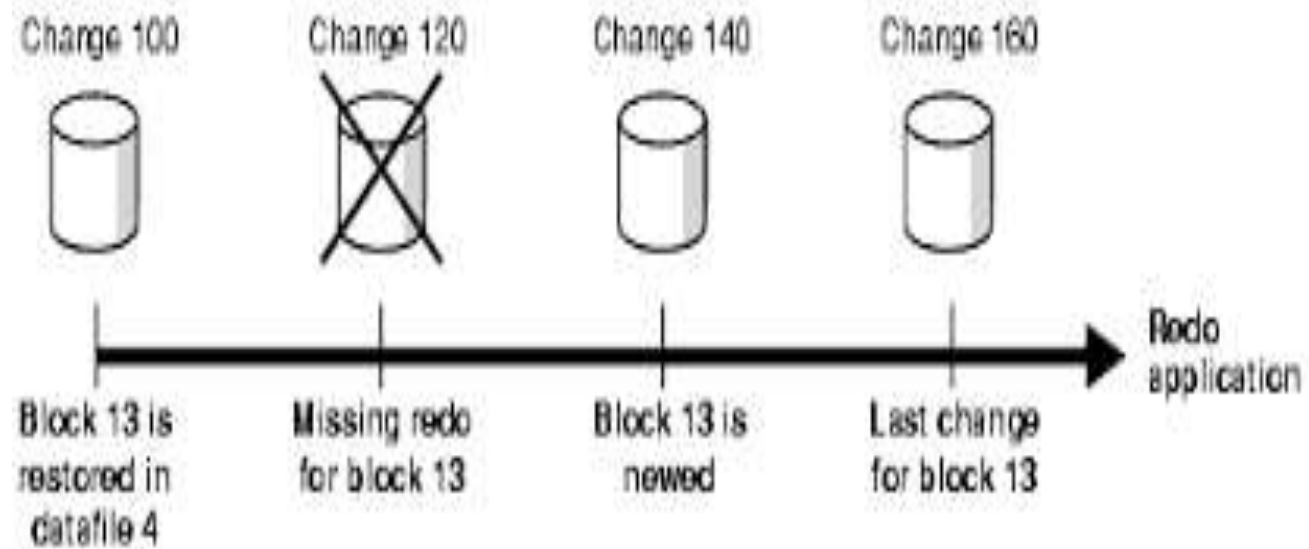
- Media recovery is based on periodically making copy of the database.
- Because copying a large database object such as a file can take a long time and the DBMS must be allowed to continue with its operation in the meantime, creating a copy is handled in manner similar to taking a fuzzy checkpoint.

<https://www.google.com/url?sa=i&url=https%3A%2F%2Fslideplayer.com%2Fslide%2F8029790%2F&psig=AOvVaw3Nt6d6SyJBpx6hfOyrHRqB&ust=1644246197673000&source=images&cd=vfe&ved=0CAsQjRxqFwoTCLjA1efG7PUCFQAAAAAdAAAAABAD>

MEDIA RECOVERY



MEDIA RECOVERY



TWO PHASE LOCKING

- Defines how transactions Acquire and Relinquish Locks
- Growing Phase – The transaction acquires all locks (doesn't unlock any data)
- Shrinking Phase – The transaction releases locks (doesn't lock any additional data)
- Transactions acquire all locks it needs until it reaches locked point
- When locked, data is modified and locks are released

TWO-PHASE LOCKING

- Defines how transactions acquire and relinquish locks
- Guarantees serializability, but it does not prevent deadlocks
 - *Growing phase*, in which a transaction acquires all the required locks without unlocking any data
 - *Shrinking phase*, in which a transaction releases all locks and cannot obtain any new lock

TWO-PHASE LOCKING

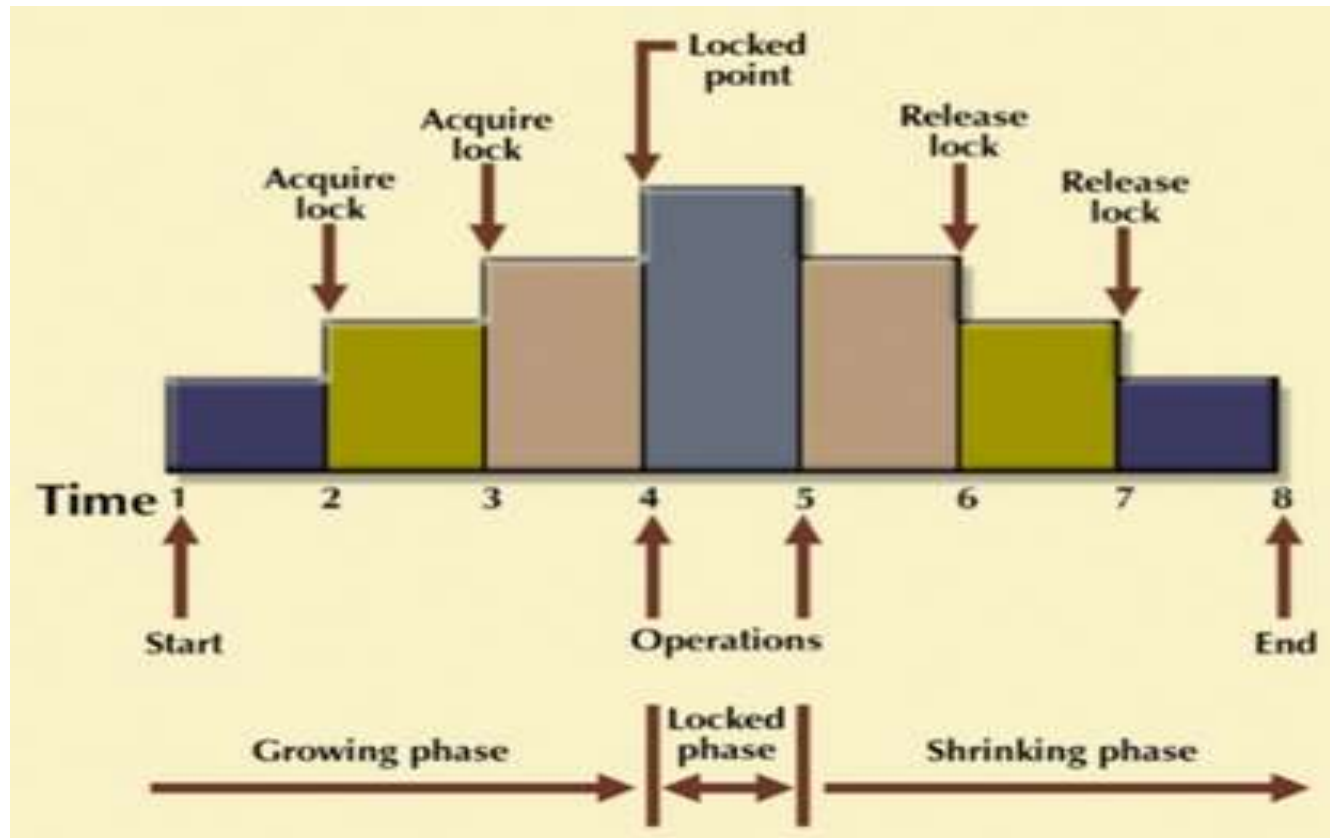
	T ₁	T ₂	T ₃
1	Lock-X(A)		
2	Read(A)		
3	Write(A)		
4	Lock-S(B) → LP	Rollback	
5	Read(B)		Rollback
6	Unlock(A),Unlock(B)		
7		Lock-X(A) → LP	
8		Read(A)	
9		Write(A)	
10		Unlock(A)	
11			Lock-S(A) → LP
12			Read(A)

FAIL Rollback

LP - Lock Point

Read(A) in T₂ and T₃ denotes Dirty Read because of Write(A) in T₁.

TWO PHASE LOCKING



TWO PHASE LOCKING

T'1	T'2	Result
read_lock (Y);	read_lock (X);	T1 and T2 follow two-phase policy but they are subject to deadlock, which must be dealt with.
read_item (Y);	read_item (X);	
write_lock (X);	Write_lock (Y);	
unlock (Y);	unlock (X);	
read_item (X);	read_item (Y);	
X:=X+Y;	Y:=X+Y;	
write_item (X);	write_item (Y);	
unlock (X);	unlock (Y);	

DEADLOCKS

- Occur when 2 transactions exist in the following mode:
- T1 = access data item X and Y
- T2 = Access data items Y and X

If T1 does not unlock Y, T2 cannot begin

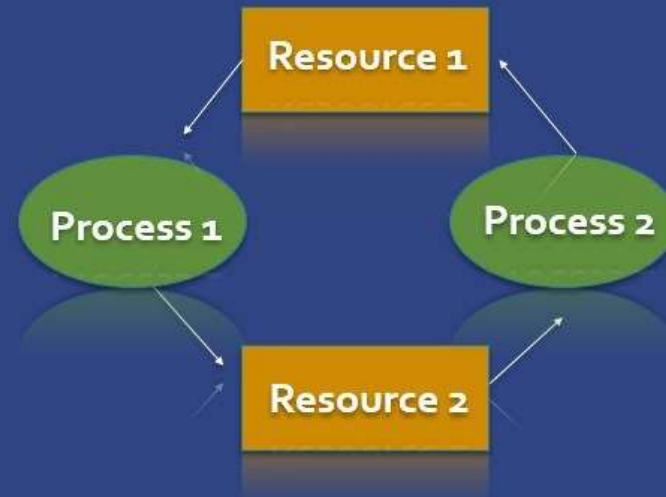
- If T2 does not unlock X, T1 cannot continue

T1 & T2 wait indefinitely for each other to unlock data

- Deadlocks are only possible if a transactions wants an Exclusive Lock (No Deadlocks on Shared Locks)
- <https://www.geeksforgeeks.org/deadlock-in-dbms/>

DEADLOCKS

Deadlock in DBMS



DEADLOCKS

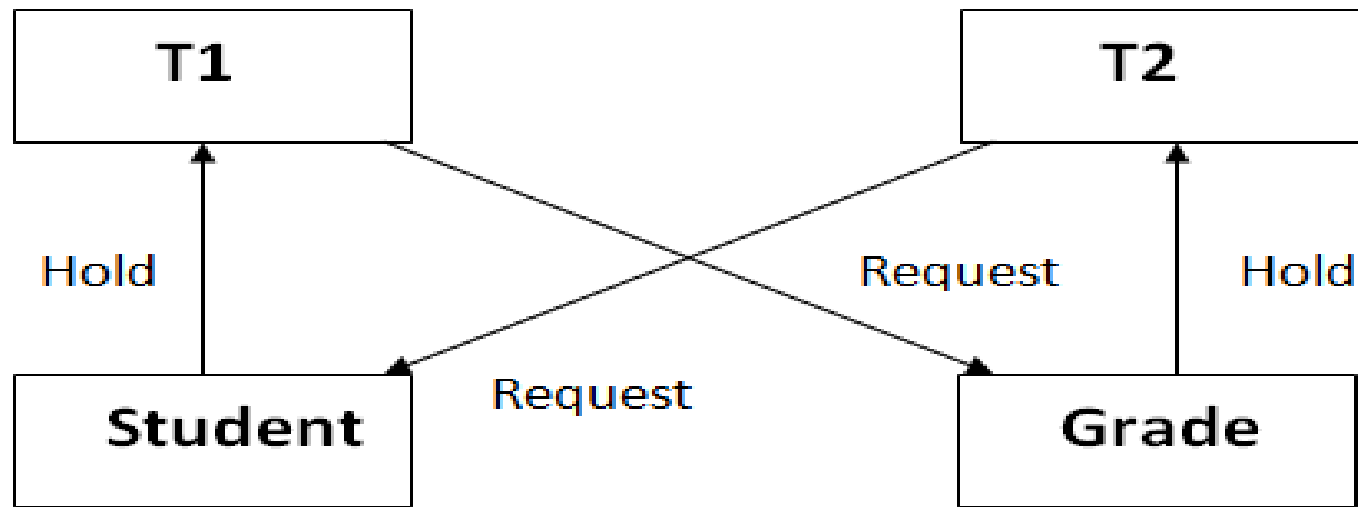


Figure: Deadlock in DBMS

DEADLOCKS-DETECTION

- Condition that occurs when two transactions wait for each other to unlock data

T1 needs data items X and Y; T needs Y and X

Each gets the its first piece of data but then waits to get the second (which the other transaction is already holding) – ***deadly embrace***

Possible only if one of the transactions wants to obtain an exclusive lock on a data item

No deadlock condition can exist among *shared* locks

Control through

Prevention

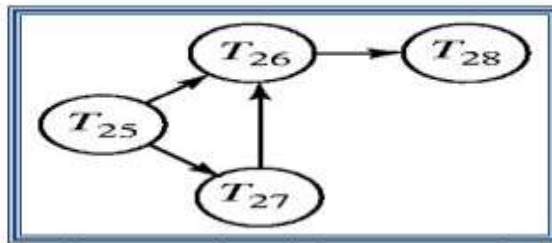
Detection

Avoidance

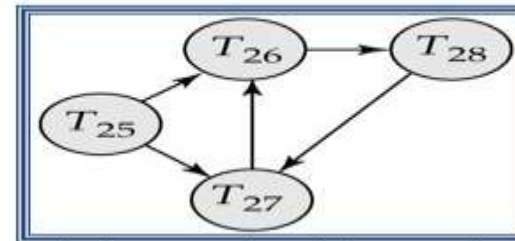
DEADLOCKS-DETECTION

Deadlock Detection

- Deadlocks can be described as a *wait-for graph*, which consists of a pair $G = (V, E)$,
 - V is a set of vertices (all the transactions in the system)
 - E is a set of edges; each element is an ordered pair $T_i \rightarrow T_j$.
- If $T_i \rightarrow T_j$ is in E , then there is a directed edge from T_i to T_j
 - implying that T_i is waiting for T_j to release a data item.
- When T_i requests a data item held by T_j , then $T_i \rightarrow T_j$ is inserted in the wait-for graph.
 - This edge is removed only when T_j is no longer holding a data item needed by T_i .
- The system is in a deadlock state if and only if the wait-for graph has a cycle.
- The system invokes a deadlock-detection algorithm periodically to look for cycles.



Wait-for graph without a cycle

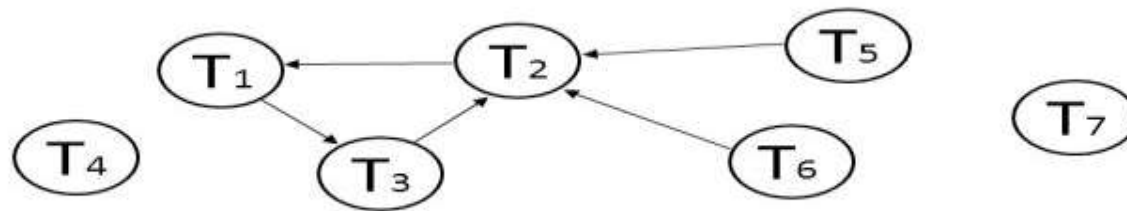


Wait-for graph with a cycle

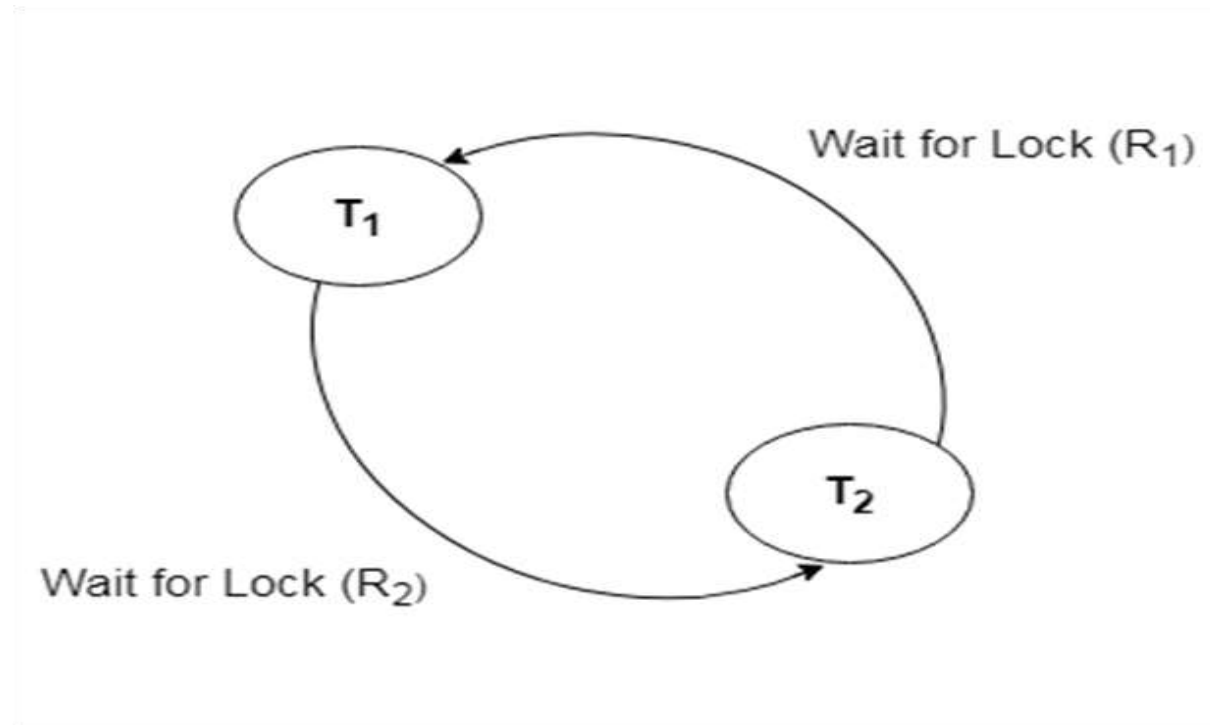
DEADLOCKS-DETECTION

Deadlock Detection

- Build Wait-For graph
- Use lock table structures
- Build incrementally or periodically
- When cycle found, rollback victim



DEADLOCK-DETECTION



CONTROLLING DEADLOCKS

- Prevention – A transaction requesting a new lock is aborted if there is the possibility of a deadlock – Transaction is rolled back, Locks are released, Transaction is rescheduled
- Detection – Periodically test the database for deadlocks. If a deadlock is found, abort / rollback one of the transactions
- Avoidance – Requires a transaction to obtain all locks needed before it can execute – requires locks to be obtained in succession

DATABASE RECOVERY

- Restore a database from a given state to a previous consistent state
- Atomic Transaction Property (All or None)
- Backup Levels:
 - Full Backup
 - Differential Backup
 - Transaction Log Backup
- Database / System Failures:
 - Software (O.S., DBMS, Application Programs, Viruses)
 - Hardware (Memory Chips, Disk Crashes, Bad Sectors)
 - Programming Exemption (Application Program rollbacks)
 - Transaction (Aborting transactions due to deadlock detection)
 - External (Fire, Flood, etc)

RECOVERY MANAGEMENT

- Restores database from a given state, usually inconsistent, to a previously consistent state
 - Based on the atomic transaction property
 - All portions of the transaction must be treated as a single logical unit of work, in which all operations must be applied and completed to produce a consistent database
 - If transaction operation cannot be completed, transaction must be aborted, and any changes to the database must be rolled back (undone)
-
- <https://www.geeksforgeeks.org/database-recovery-techniques-in-dbms/>

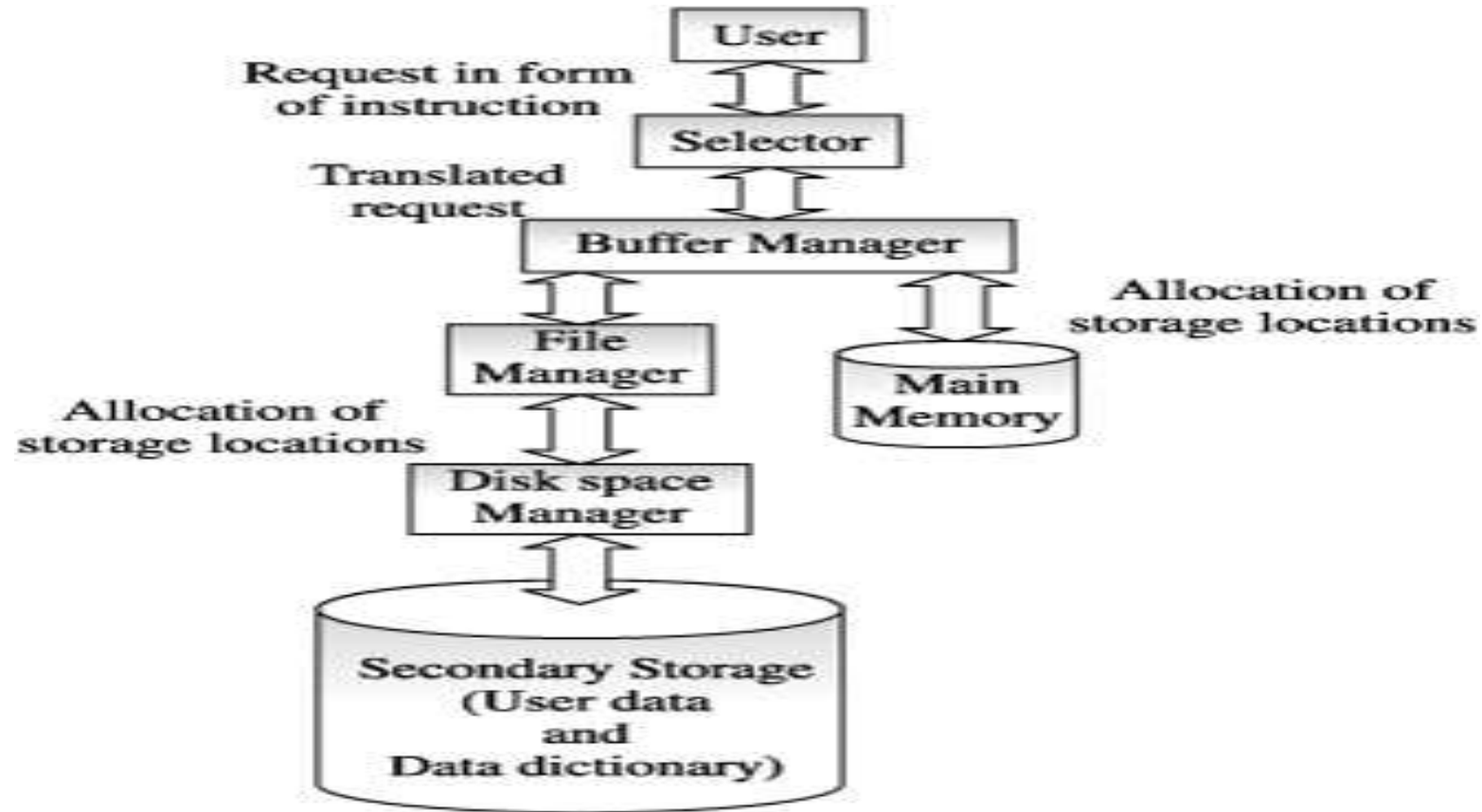
RECOVERY SYSTEM

- Recover Database by using data in the Transaction Log
- Write-Ahead-Log – Transaction logs need to be written before any database data is updated
- Redundant Transaction Logs – Several copies of log on different devices
- Database Buffers – Buffers are used to increase processing time on updates instead of accessing data on disk
- Database Checkpoints – Process of writing all updated buffers to disk ☐ While this is taking place, all other requests are not executes
 - Scheduled several times per hour
 - Checkpoints are registered in the transaction log

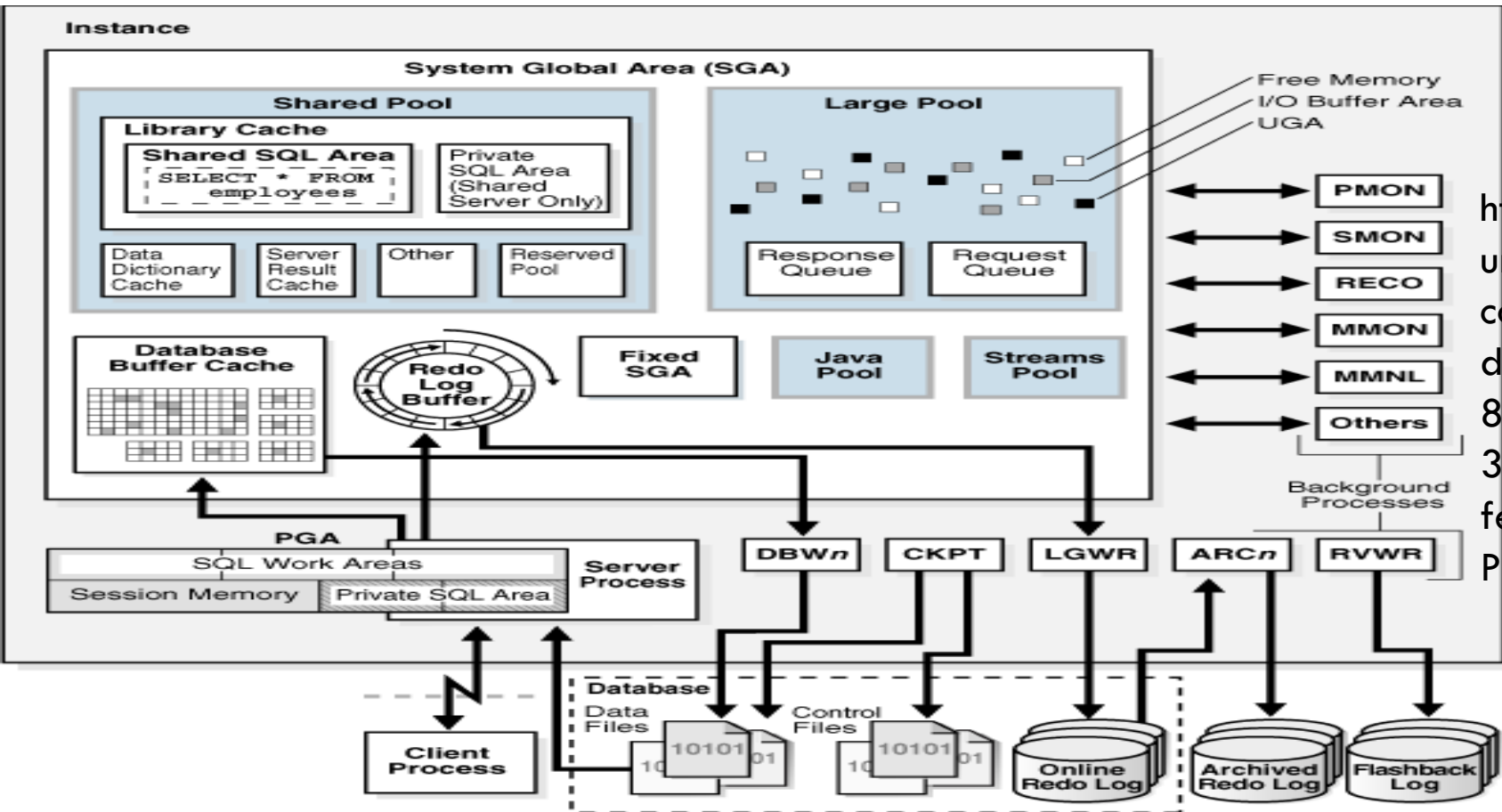
PHYSICAL STORAGE AND DATABASE CONCEPTS

- Several types of data storage exist in most computer systems. They vary in speed of access, cost per unit of data, and reliability.
 - **Cache:** most costly and fastest form of storage. Usually very small, and managed by the operating system.
 - **Main Memory (MM):** the storage area for data available to be operated on.
 - General-purpose machine instructions operate on main memory.
 - Contents of main memory are usually lost in a power failure or ``crash".
 - Usually too small (even with megabytes) and too expensive to store the entire database.
 - **Flash memory:** EEPROM (*electrically erasable programmable read-only memory*).
 - Data in flash memory survive from power failure.
 - Reading data from flash memory takes about 10 nano-secs (roughly as fast as from main memory), and writing data into flash memory is more complicated: write-once takes about 4-10 microsecs.

PHYSICAL STORAGE AND DATABASE CONCEPTS

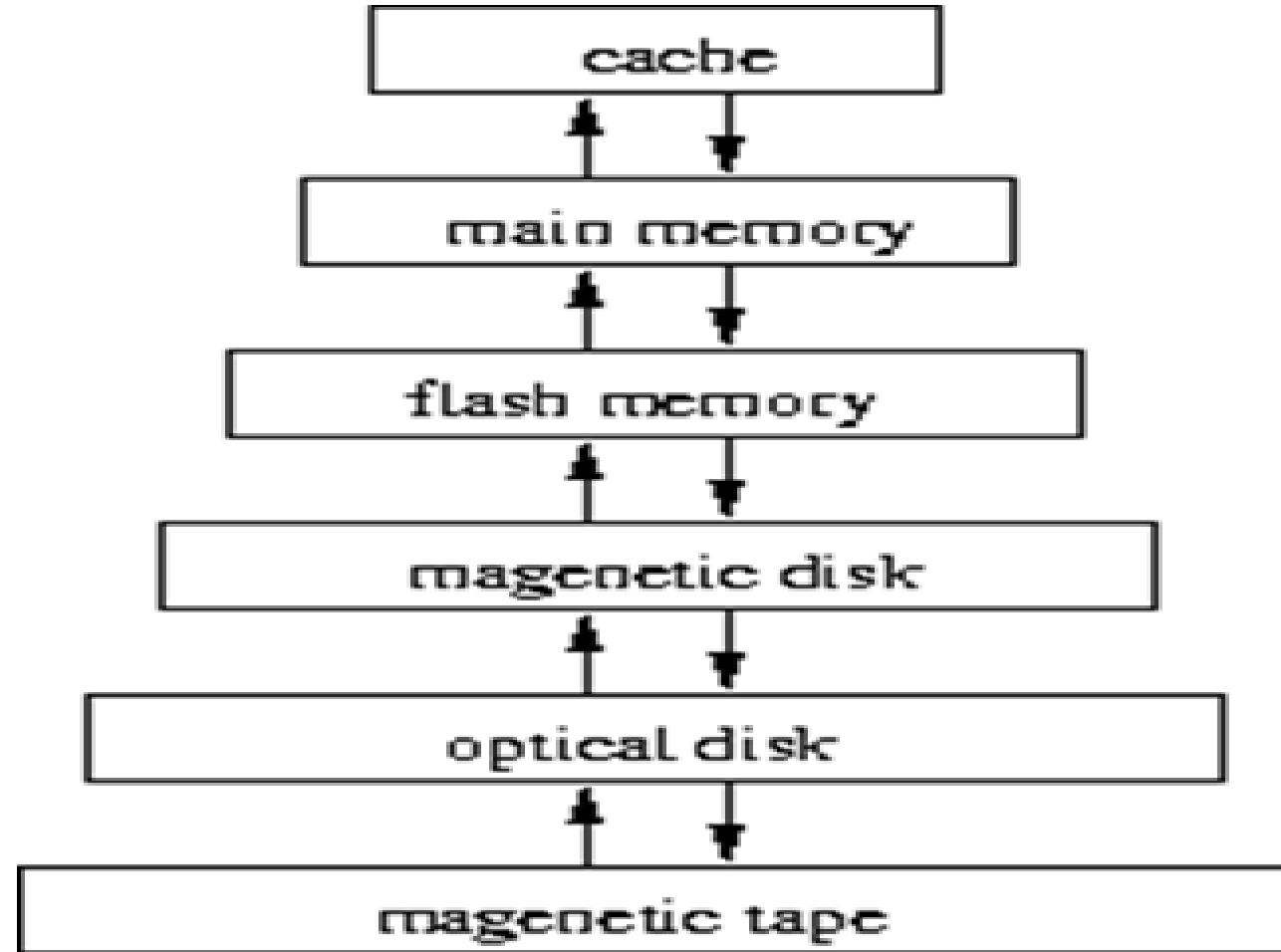


PHYSICAL STORAGE AND DATABASE CONCEPTS

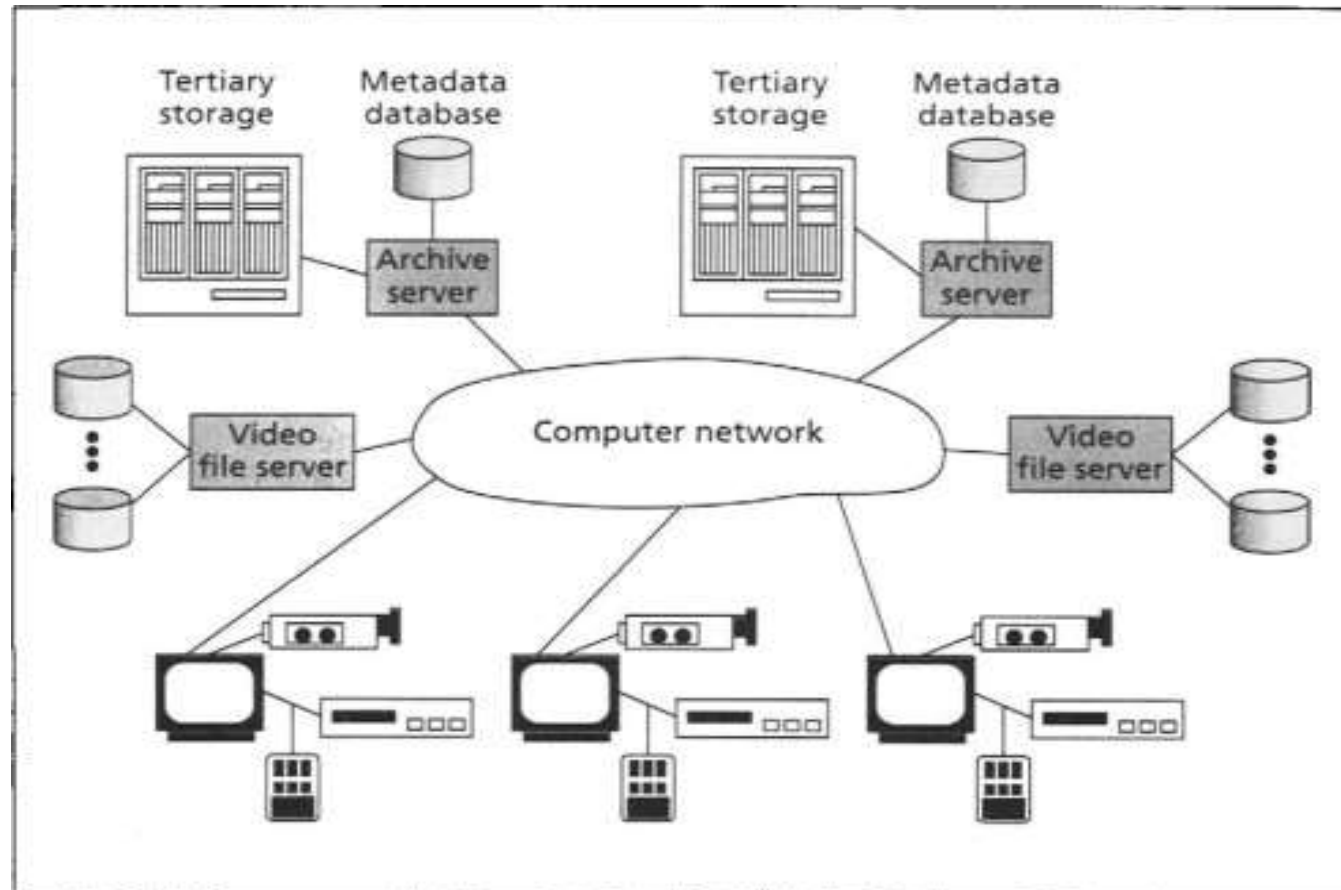


<https://www.google.com/url?sa=i&url=https%3A%2F%2Fwww.educba.com%2Fdeadlock-in-dbms%2F&psig=AOvVaw15JIM8Sb8CDM8nuFcWmU3h&ust=1644246334710000&source=images&cd=vfe&ved=0CAsQjRxqFwoTcli57afH7PUCFQAAAAAdAAAAABAD>

OVERVIEW OF PHYSICAL STORAGE MEDIA AND RAID



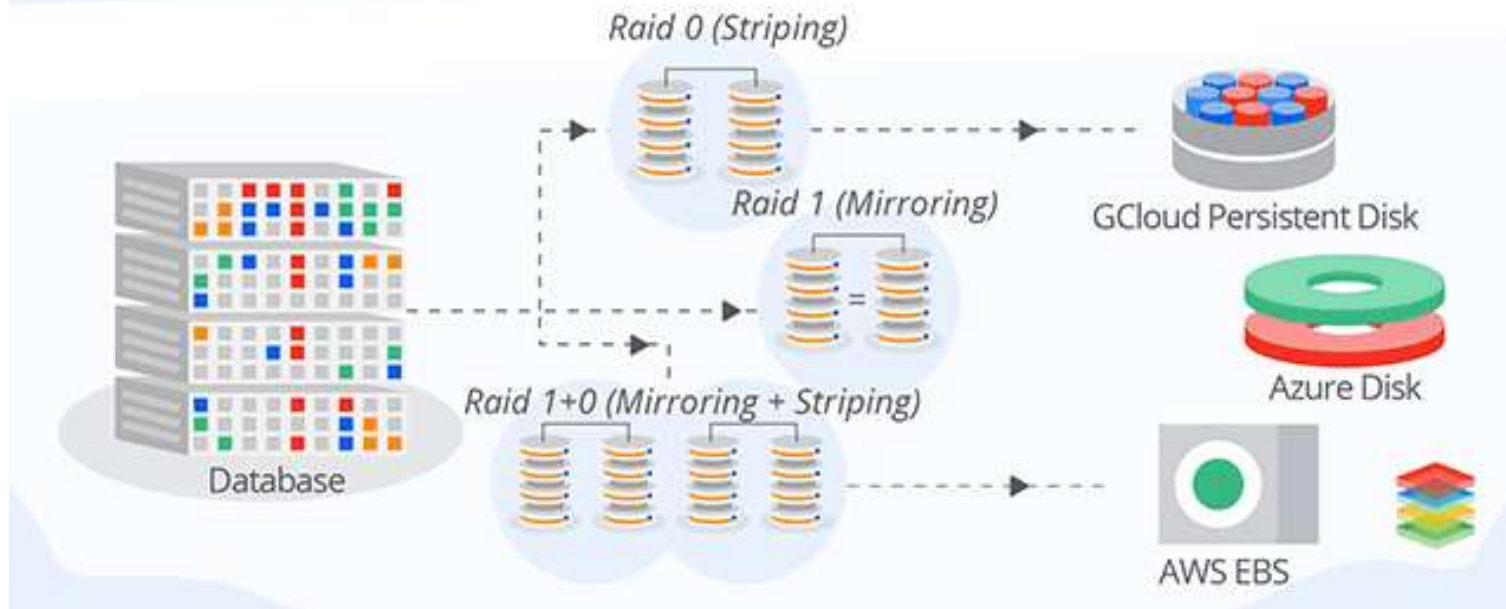
OVERVIEW OF PHYSICAL STORAGE MEDIA AND RAID



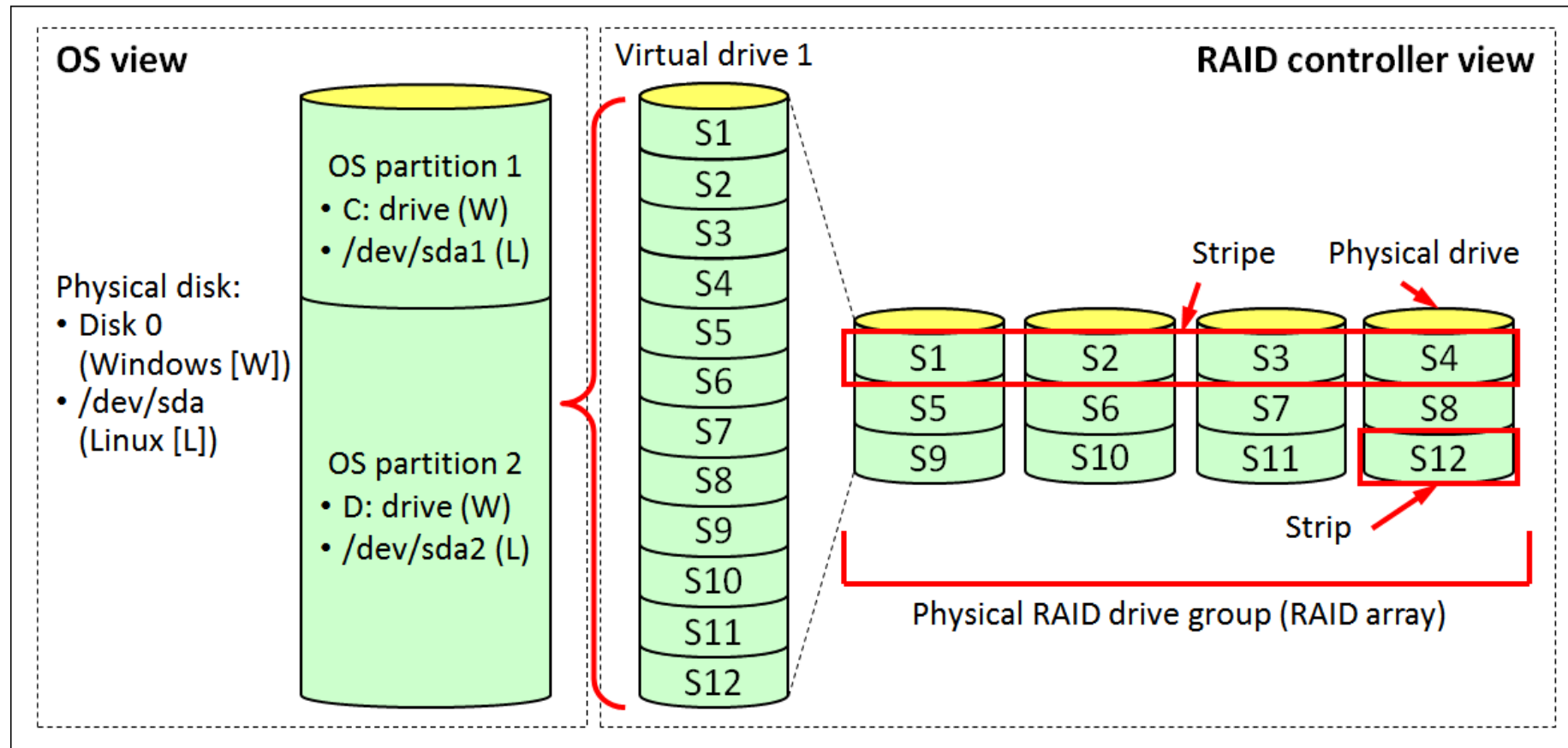
https://www.tutorialspoint.com/dbms/dbms_indexing.htm

OVERVIEW OF PHYSICAL STORAGE MEDIA AND RAID

Types of RAID Storage for Databases in Public Cloud



OVERVIEW OF PHYSICAL STORAGE MEDIA AND RAID



WEB LINKS

- DBMS NPTEL ON LINE COURSE

https://onlinecourses.nptel.ac.in/noc21_cs04/preview

- DBMS JAVAPPOINT TUTORIAL

<https://www.javatpoint.com/dbms-tutorial>

- DBMS GEEKSFORGEEKS TUTORIAL

<https://www.geeksforgeeks.org/dbms/>