Database Management System

Short notes

Ashutosh: DBMS

2017

**1 OVERVIEW**

A **database management system** stores data in such a way that it becomes easier to retrieve, manipulate, and produce information.

**Characteristics**

* **Real-world entity:** A modern DBMS is more realistic and uses real-world entities to design its architecture
* **Relation-based tables:** DBMS allows entities and relations among them to form tables.
* **Isolation of data and application:** Detail information of data.
* **Less redundancy:** DBMS follows the rules of normalization, which splits a relation when any of its attributes is having redundancy in values.
* **Consistency:** Consistency is a state where every relation in a database remains consistent.
* **Query Language:** DBMS is equipped with query language, which makes it more efficient to retrieve and manipulate data.
* **ACID Properties:** DBMS follows the concepts of Atomicity, Consistency, Isolation, and Durability (normally shortened as ACID).
* **Multiuser and Concurrent Access:** DBMS supports multi-user environment and allows them to access and manipulate data in parallel.
* **Multiple views:** DBMS offers multiple views for different users.
* requirements.
* **Security:** Features like multiple views offer security to some extent where users are unable to access data of other users and departments.

**Users**

* **Administrators:** Administrators maintain the DBMS and are responsible for administrating the database. They are responsible to look after its usage and by whom it should be used.
* **Designers:** Designers are the group of people who actually work on the designing part of the database.
* **End Users:** End users are those who actually reap the benefits of having a DBMS.

**2 ARCHITECTURE**

**3-tier Architecture**

**Database (Data) Tier:** the database resides along with its query processing languages. We also have the relations that define the data and their constraints at this level.

**Application (Middle) Tier:** Reside the application server and the programs that access the database. For a user, this application tier presents an abstracted view of the database. The application layer sits in the middle and acts as a mediator between the end-user and the database.

**User (Presentation) Tier:** End-users operate on this tier and they know nothing about any existence of the database beyond this layer. At this layer, multiple views of the database can be provided by the application. All views are generated by applications that reside in the application tier.

**3 DATA MODELS**

**Entity-Relationship Model**

ER Model is based on:

* **Entity:** An entity in an ER Model is a real-world entity having properties called attributes.
* **Relationship:** The logical association among entities is called relationship. Mapping cardinalities define the number of association between two entities.
  + one to one
  + one to many
  + many to one
  + many to many

**Relational Model**

The main highlights of this model are:

 Data is stored in tables called relations.

 Relations can be normalized.

 In normalized relations, values saved are atomic values.

 Each row in a relation contains a unique values

 Each column in a relation contains values from a same domain.

**4 DATA SCHEMAS**

**Database Schema:** A database schema is the skeleton structure that represents the logical view of the entire database. It defines how the data is organized and how the relations among them are associated.

A database schema can be divided into two categories:

 **Physical Database Schema:** This schema pertains to the actual storage of data and its form of storage like files, indices, etc. It defines how the data will be stored in a secondary storage.

 **Logical Database Schema:** This schema defines all the logical constraints that need to be applied on the data stored. It defines tables, views, and integrity constraints

**Database Instance:** A database instance is a state of operational database with data at any given time

**5 DATA INDEPENDENCE**

**Data Independence:** A database system normally contains a lot of data in addition to users’ data. For example, it stores data about data, known as metadata, to locate and retrieve data easily. It is rather difficult to modify or update a set of metadata once it is stored in the database

1. **Logical Data Independence:** Logical data is data about database, that is, it stores information about how data is managed inside. For example, a table (relation) stored in the database and all its constraints applied on that relation.
2. **Physical Data Independence:** Physical data independence is the power to change the physical data without impacting the schema or logical data.

**6 ER MODEL – BASIC CONCEPTS**

**Entity:** An entity set is a collection of similar types of entities. An entity set may contain entities with attribute sharing similar values.

**Attributes:** Entities are represented by means of their properties called attributes. All attributes have values. For example, a student entity may have name, class, and age as attributes.

Types of Attributes

* **Simple attribute:** Simple attributes are atomic values, which cannot be divided further. For example, a student's phone number is an atomic value of 10 digits.
* **Composite attribute:** Composite attributes are made of more than one simple attribute. For example, a student's complete name may have first\_name and last\_name.
* **Derived attribute:** Derived attributes are the attributes that do not exist in the physical database, but their values are derived from other attributes present in the database.
* **Single-value attribute:** Single-value attributes contain single value. For example: Social\_Security\_Number.
* **Multi-value attribute:** Multi-value attributes may contain more than one values. For example, a person can have more than one phone number, email\_address, etc.

**Entity-Set and Keys**

* **Super Key:** A set of attributes (one or more) that collectively identifies an entity in an entity set.
* **Candidate Key:** A minimal super key is called a candidate key. An entity set may have more than one candidate key.
* **Primary Key:** A primary key is one of the candidate keys chosen by the database designer to uniquely identify the entity set.

**Relationship**

**Relationship Set:** A set of relationships of similar type is called a relationship set. Like entities, a relationship too can have attributes. These attributes are called descriptive attributes.

**Degree of Relationship:** The number of participating entities in a relationship defines the degree of the relationship.

**Mapping Cardinalities:** Cardinality defines the number of entities in one entity set, which can be associated with the number of entities of other set via relationship set.

* **One-to-one:** One entity from entity set A can be associated with at most one entity of entity set B and vice versa.
* **One-to-many:** One entity from entity set A can be associated with more than one entities of entity set B, however an entity from entity set B can be associated with at most one entity.
* **Many-to-one:** More than one entities from entity set A can be associated with at most one entity of entity set B, however an entity from entity set B can be associated with more than one entity from entity set A.
* **Many-to-many:** One entity from A can be associated with more than one entity from B and vice versa.

**7 ER DIAGRAM REPRESENTATION**

|  |  |  |
| --- | --- | --- |
| **Symbol** | **Shape Name** | **Symbol Description** |
| **Entities** | | |
| ERD Symbols and Meaning - Entity | Entity | An entity is represented by a rectangle which contains the entity’s name. |
| ERD Symbols and Meaning - Weak Entity | Weak Entity | An entity that cannot be uniquely identified by its attributes alone. The existence of a weak entity is dependent upon another entity called the owner entity. The weak entity’s identifier is a combination of the identifier of the owner entity and the partial key of the weak entity. |
| ERD Symbols and Meaning - Associative Entity | Associative Entity | An entity used in a many-to-many relationship (represents an extra table). All relationships for the associative entity should be many |
| **Attributes** | | |
| ERD Symbols and Meaning - Attribute | Attribute | In the Chen notation, each attribute is represented by an oval containing atributte’s name |
| ERD Symbols and Meaning - Key attribute | Key attribute | An attribute that uniquely identifies a particular entity. The name of a key attribute is underscored. |
| ERD Symbols and Meaning - Multivalue attribute | Multivalued attribute | An attribute that can have many values (there are many distinct values entered for it in the same column of the table). Multivalued attribute is depicted by a dual oval. |
| ERD Symbols and Meaning - Derived attribute | Derived attribute | An attribute whose value is calculated (derived) from other attributes. The derived attribute may or may not be physically stored in the database. In the Chen notation, this attribute is represented by dashed oval. |
| **Relationships** | | |
| ERD Symbols and Meaning - Relationship | Strong relationship | A relationship where entity is existence-independent of other entities, and PK of Child doesn’t contain PK component of Parent Entity. A strong relationship is represented by a single rhombus |
| ERD Symbols and Meaning - Identifying Relationship | Weak (identifying) relationship | A relationship where Child entity is existence-dependent on parent, and PK of Child Entity contains PK component of Parent Entity. This relationship is represented by a double rhombus. |

**Relationship**

**Binary Relationship and Cardinality:** A relationship where two entities are participating is called a binary relationship.

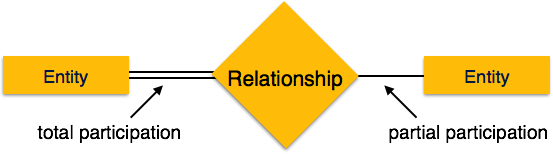
 One-to-one:

 One-to-many:

 Many-to-one:

 Many-to-many:

**Participation Constraints:**

* **Total Participation:** Each entity is involved in the relationship. Total participation is represented by double lines.
* **Partial participation:** Not all entities are involved in the relationship. Partial participation is represented by single lines.

**8 GENERALIZATION & SPECIALIZATION**

**Generalization:** where the generalized entities contain the properties of all the generalized entities, is called generalization

**Specialization:** Specialization is the opposite of generalization. In specialization, a group of entities is divided into sub-groups based on their characteristics

**Inheritance:** We use all the above features of ER-Model in order to create classes of objects in object-oriented programming. The details of entities are generally hidden from the user; this process known as **abstraction**. Inheritance is an important feature of Generalization and Specialization. It allows lower-level entities to inherit the attributes of higher-level entities.

**9 CODD’S 12 RULES**

**Rule 1:** **Information Rule:** Everything in a database must be stored in a table format

**Rule 2: Guaranteed Access Rule:** Every single data element (value) is guaranteed to be accessible logically with a combination of table-name, primary-key (row value), and attribute-name (column value). No other means, such as pointers, can be used to access data.

**Rule 3: Systematic Treatment of NULL Values:** The NULL values in a database must be given a systematic and uniform treatment. This is a very important rule because a NULL can be interpreted as one the following: data is missing, data is not known, or data is not applicable.

**Rule 4: Active Online Catalog:** The structure description of the entire database must be stored in an online catalog, known as data dictionary

**Rule 5: Comprehensive Data Sub-Language Rule:** A database can only be accessed using a language having linear syntax that supports data definition, data manipulation, and transaction management operations.

**Rule 6: View Updating Rule:** All the views of a database, which can theoretically be updated, must also be updatable by the system.

**Rule 7: High-Level Insert, Update, and Delete Rule:** A database must support high-level insertion, updation, and deletion. This must not be limited to a single row, that is, it must also support union, intersection and minus operations to yield sets of data records.

**Rule 8: Physical Data Independence:** The data stored in a database must be independent of the applications that access the database. Any change in the physical structure of a database must not have any impact on how the data is being accessed by external applications.

**Rule 9: Logical Data Independence:** The logical data in a database must be independent of its user’s view (application). Any change in logical data must not affect the applications using it.

**Rule 10: Integrity Independence:** All its integrity constraints can be independently modified without the need of any change in the application.

**Rule 11: Distribution Independence:** The end-user must not be able to see that the data is distributed over various locations. Users should always get the impression that the data is located at one site only.

**Rule 12: Non-Subversion Rule:**

**10 RELATIONAL DATA MODEL**

**Concepts:**

* **Tables:** In relational data model, relations are saved in the format of Tables. This format stores the relation among entities. A table has rows and columns, where rows represent records and columns represent the attributes.
* **Tuple:** A single row of a table, which contains a single record for that relation is called a tuple.
* **Relation instance:** A finite set of tuples in the relational database system represents relation instance. Relation instances do not have duplicate tuples.
* **Relation schema:** A relation schema describes the relation name (table name), attributes, and their names.
* **Relation key:** Each row has one or more attributes, known as relation key, which can identify the row in the relation (table) uniquely.
* **Attribute domain:** Every attribute has some predefined value scope, known as attribute domain.

**Constraints:**

Every relation has some conditions that must hold for it to be a valid relation. These conditions are called Relational Integrity Constraints. There are three main integrity constraints:

* **Key Constraints:** There must be at least one minimal subset of attributes in the relation, which can identify a tuple uniquely. This minimal subset of attributes is called key for that relation. If there are more than one such minimal subsets, these are called candidate keys.
* **Domain Constraints:** Attributes have specific values in real-world scenario. For example, age can only be a positive integer. The same constraints have been tried to employ on the attributes of a relation. Every attribute is bound to have a specific range of values. For example, age cannot be less than zero and telephone numbers cannot contain a digit outside 0-9.
* **Referential Integrity Constraints:** Referential integrity constraints work on the concept of Foreign Keys. A foreign key is a key attribute of a relation that can be referred in other relation.

**11 RELATIONAL ALGEBRA**

**Relational Algebra:** Relational algebra is a procedural query language, which takes instances of relations as input and yields instances of relations as output. It uses operators to perform queries. An operator can be either unary or binary. Operations of relational algebra are as follows:

* Select (σ)
* Project (∏)
* Union (∪)
* Set different (−)
* Cartesian product (Χ)
* Rename (ρ)

**Relational Calculus:**

In contrast to Relational Algebra, Relational Calculus is a non-procedural query language, that is, it tells what to do but never explains how to do it.

Relational calculus exists in two forms:

1. **Tuple Relational Calculus (TRC):**

Filtering variable ranges over tuples

**Notation:** {T | Condition}

Returns all tuples T that satisfies a condition.

For example:

{ T.name | Author(T) AND T.article = 'database' }

Output: Returns tuples with 'name' from Author who has written article on 'database'

1. **Domain Relational Calculus (DRC):**

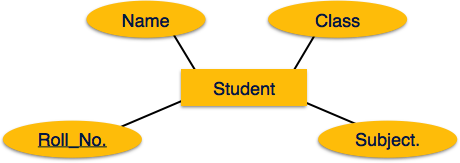
In DRC, the filtering variable uses the domain of attributes instead of entire tuple values (as done in TRC, mentioned above).

**Notation:** { a1, a2, a3, ..., an | P (a1, a2, a3, ... ,an)}

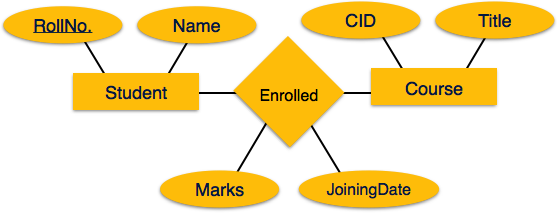
Where a1, a2 are attributes and P stands for formulae built by inner attributes.

**12 ER MODEL TO RELATIONAL MODEL**

**1. Mapping Entity:** An entity is a real-world object with some attributes.

**Mapping Process (Algorithm)**

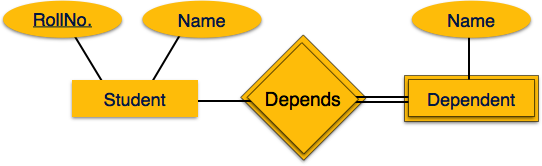
* Create table for each entity.
* Entity's attributes should become fields of tables with their respective data types.
* Declare primary key.

**2. Mapping Relationship:** A relationship is an association among entities.

**Mapping Process**

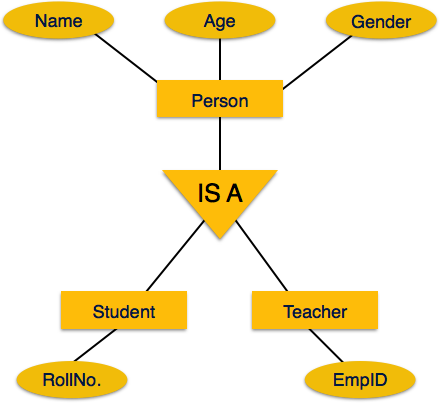
* Create table for a relationship.
* Add the primary keys of all participating Entities as fields of table with their respective data types.
* If relationship has any attribute, add each attribute as field of table.
* Declare a primary key composing all the primary keys of participating entities.
* Declare all foreign key constraints

**3. Mapping Weak Entity Sets:** A weak entity set is one which does not have any primary key associated with it.

**Mapping Process**

* Create table for weak entity set.
* Add all its attributes to table as field.
* Add the primary key of identifying entity set.
* Declare all foreign key constraints.

**4. Mapping Hierarchical Entities:** ER specialization or generalization comes in the form of hierarchical entity sets.

**Mapping Process**

* Create tables for all higher-level entities.
* Create tables for lower-level entities.
* Add primary keys of higher-level entities in the table of lower-level entities.
* In lower-level tables, add all other attributes of lower-level entities.
* Declare primary key of higher-level table and the primary key for lower-level table.
* Declare foreign key constraints.

**13SQL OVERVIEW**

**Data Definition Language:**

1. **CREATE**: Creates new databases, tables, and views from RDBMS. For example:
   1. Create database tutorialspoint;
   2. Create table article;
   3. Create view for\_students;
2. **DROP**: Drops commands, views, tables, and databases from RDBMS. For example:
   1. Drop object\_type bject\_name;
   2. Drop database tutorialspoint;
   3. Drop table article;
   4. Drop view for\_students;
3. **ALTER**: Modifies database schema. Alter object\_type object\_name parameters; For example:
   1. Alter table article add subject varchar;

This command adds an attribute in the relation article with the name subject of string type.

**Data Manipulation Language:**

* **SELECT/FROM/WHHERE:** 
  + SELECT: This is one of the fundamental query command of SQL. It is similar to the projection operation of relational algebra. It selects the attributes based on the condition described by WHERE clause.
  + FROM: This clause takes a relation name as an argument from which attributes are to be selected/projected. In case more than one relation names are given, this clause corresponds to Cartesian product.
  + WHERE: This clause defines predicate or conditions, which must match in order to qualify the attributes to be projected.

For example: Select author\_name From book\_author Where age > 50;

* **INSERT INTO/VALUES:** This command is used for inserting values into the rows of a table (relation).

For example: INSERT INTO tutorialspoint (Author, Subject) VALUES ("anonymous", "computers");

* **UPDATE/SET/WHERE:** This command is used for updating or modifying the values of columns in a table (relation).

For example: UPDATE tutorialspoint SET Author="webmaster" WHERE Author="anonymous";

* **DELETE/FROM/WHERE:** This command is used for removing one or more rows from a table (relation).

For example: DELETE FROM tutorialspoint WHERE Author="unknown";

**14 NORMALIZATION**

**Functional Dependency:**

Functional dependency (FD) is a set of constraints between two attributes in a relation. Functional dependency says that if two tuples have same values for attributes A1, A2,..., An, then those two tuples must have to have same values for attributes B1, B2, ..., Bn. Functional dependency is represented by an arrow sign (→) that is, X→Y, where X functionally determines Y. The left-hand side attributes determine the values of attributes on the right-hand side.

**Armstrong's Axioms:**

* **Reflexive rule:** If alpha is a set of attributes and beta is subset of alpha, then alpha holds beta.
* **Augmentation rule:** If a → b holds and y is attribute set, then ay → by also holds. That is adding attributes in dependencies, does not change the basic dependencies.
* **Transitivity rule:** Same as transitive rule in algebra, if a → b holds and b → c holds, then a → c also holds. a → b is called as a functionally that determines b.

**Trivial Functional Dependency:**

* **Trivial:** If a functional dependency (FD) X → Y holds, where Y is a subset of X, then it is called a trivial FD. Trivial FDs always hold.
* **Non-trivial:** If an FD X → Y holds, where Y is not a subset of X, then it is called a non-trivial FD.
* **Completely non-trivial:** If an FD X → Y holds, where x intersect Y = Φ, it is said to be a completely non-trivial FD.

**Normalization:**

Used for Reduce Redundancy & functional Dependency.

* Update anomalies
* Deletion anomalies
* Insert anomalies

Normalization is a method to remove all these anomalies and bring the database to a consistent state.

**First Normal Form(1NF): -**

First Normal Form is defined in the definition of relations (tables) itself. This rule defines that all the attributes in a relation must have atomic domains.

**Second Normal Form(2NF): -**

* Prime attribute: An attribute, which is a part of the prime-key, is known as a prime attribute.
* Non-prime attribute: An attribute, which is not a part of the prime-key, is said to be a non-prime attribute.

If we follow second normal form, then every non-prime attribute should be fully functionally dependent on prime key attribute.

**Third Normal Form(3NF): -**

For a relation to be in Third Normal Form, it must be in Second Normal form and the following must satisfy:

* No non-prime attribute is transitively dependent on prime key attribute.

For any non-trivial functional dependency, X → A, then either:

X is a superkey or,

A is prime attribute.

**Boyce-Codd Normal Form(BCNF): -**

Boyce-Codd Normal Form (BCNF) is an extension of Third Normal Form on strict terms. BCNF states that –

* For any non-trivial functional dependency, X → A, X must be a super-key.

**15 JOINS**

**Join** is a combination of a Cartesian product followed by a selection process. A Join operation pairs two tuples from different relations, if and only if a given join condition is satisfied.

1. **Theta (θ) Join: -** Theta join combines tuples from different relations provided they satisfy the theta condition. The join condition is denoted by the symbol θ. **Notation:** R1 ⋈θ R2 R1 and R2 are relations having attributes (A1, A2, .., An) and (B1, B2,.. ,Bn) such that the attributes don’t have anything in common, that is, R1 ∩ R2 = Φ.
2. **Equijoin: -**When Theta join uses only equality comparison operator, it is said to be equijoin. The above example corresponds to equijoin.
3. **Natural Join (⋈): -**Natural join does not use any comparison operator. It does not concatenate the way a Cartesian product does. We can perform a Natural Join only if there is at least one common attribute that exists between two relations. In addition, the attributes must have the same name and domain. Natural join acts on those matching attributes where the values of attributes in both the relations are same.
4. **Outer Joins: -**Theta Join, Equijoin, and Natural Join are called inner joins. An inner join includes only those tuples with matching attributes and the rest are discarded in the resulting relation. Therefore, we need to use outer joins to include all the tuples from the participating relations in the resulting relation. There are three kinds of outer joins: left outer join, right outer join, and full outer join.
   1. **Left Outer Join: -** All the tuples from the Left relation, R, are included in the resulting relation. If there are tuples in R without any matching tuple in the Right relation S, then the S-attributes of the resulting relation are made NULL.
   2. **Right Outer Join: -** All the tuples from the Right relation, S, are included in the resulting relation. If there are tuples in S without any matching tuple in R, then the R-attributes of resulting relation are made NULL.
   3. **Full Outer Join: -** All the tuples from both participating relations are included in the resulting relation. If there are no matching tuples for both relations, their respective unmatched attributes are made NULL.

**16 STORAGE SYSTEM**

* **Primary Storage:** The memory storage that is directly accessible to the CPU comes under this category. CPU's internal memory (registers), fast memory (cache), and main memory (RAM) are directly accessible to the CPU, as they are all placed on the motherboard or CPU chipset. This storage is typically very small, ultra-fast, and volatile. Primary storage requires continuous power supply in order to maintain its state. In case of a power failure, all its data is lost.
* **Secondary Storage:** Secondary storage devices are used to store data for future use or as backup. Secondary storage includes memory devices that are not a part of the CPU chipset or motherboard, for example, magnetic disks, optical disks (DVD, CD, etc.), hard disks, flash drives, and magnetic tapes.
* **Tertiary Storage:** Tertiary storage is used to store huge volumes of data. Since such storage devices are external to the computer system, they are the slowest in speed. These storage devices are mostly used to take the back up of an entire system. Optical disks and magnetic tapes are widely used as tertiary storage.

**Memory Hierarch**

**Magnetic Disks: -** Hard disk drives are the most common secondary storage devices in present computer systems. These are called magnetic disks because they use the concept of magnetization to store information. Hard disks consist of metal disks coated with magnetizable material. These disks are placed vertically on a spindle. A read/write head moves in between the disks and is used to magnetize or de-magnetize the spot under it. A magnetized spot can be recognized as 0 (zero) or 1 (one). Hard disks are formatted in a well-defined order to store data efficiently. A hard disk plate has many concentric circles on it, called **tracks**. Every track is further divided into **sectors**. A sector on a hard disk typically stores 512 bytes of data.

**RAID: -** RAID stands for Redundant Array of Independent Disks, which is a technology to connect multiple secondary storage devices and use them as a single storage media.

RAID consists of an array of disks in which multiple disks are connected together to achieve different goals. RAID levels define the use of disk arrays.

 RAID 0:

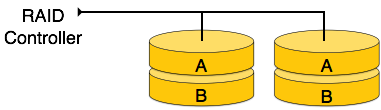
In this level, a striped array of disks is implemented. The data is broken down into blocks and the blocks are distributed among disks. Each disk receives a block of data to write/read in parallel. It enhances the speed and performance of the storage device. There is no parity and backup in Level 0.



 RAID 1:

RAID 1 uses mirroring techniques. When data is sent to a RAID controller, it sends a copy of data to all the disks in the array. RAID level

1 is also called mirroring and provides 100% redundancy in case of a failure.



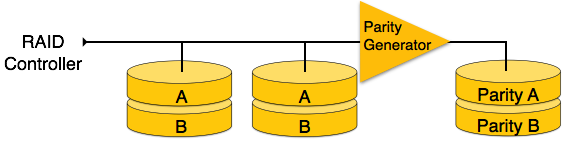
 RAID 2:

RAID 2 records Error Correction Code using Hamming distance for its data, striped on different disks. Like level 0, each data bit in a word is recorded on a separate disk and ECC codes of the data words are stored on a different set disks. Due to its complex structure and high cost, RAID 2 is not commercially available.



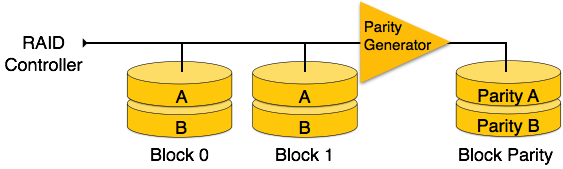
 RAID 3:

RAID 3 stripes the data onto multiple disks. The parity bit generated for data word is stored on a different disk. This technique makes it to overcome single disk failures.



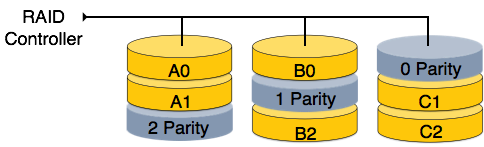
 RAID 4:

In this level, an entire block of data is written onto data disks and then the parity is generated and stored on a different disk. Note that level 3 uses byte-level striping, whereas level 4 uses block-level striping. Both level 3 and level 4 require at least three disks to implement RAID.



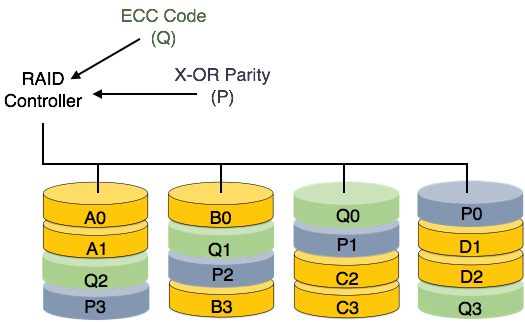
 RAID 5:

RAID 5 writes whole data blocks onto different disks, but the parity bits generated for data block stripe are distributed among all the data disks rather than storing them on a different dedicated disk.



 RAID 6:

RAID 6 is an extension of level 5. In this level, two independent parities are generated and stored in distributed fashion among multiple disks. Two parities provide additional fault tolerance. This level requires at least four disk drives to implement RAID.



**17 FILE STRUCTURE**

**File Organization: -**

* **Heap File Organization: -** When a file is created using Heap File Organization, the Operating System allocates memory area to that file without any further accounting details. File records can be placed anywhere in that memory area. It is the responsibility of the software to manage the records. Heap File does not support any ordering, sequencing, or indexing on its own.
* **Sequential File Organization: -** Every file record contains a data field (attribute) to uniquely identify that record. In sequential file organization, records are placed in the file in some sequential order based on the unique key field or search key. Practically, it is not possible to store all the records sequentially in physical form.
* **Hash File Organization: -** Hash File Organization uses Hash function computation on some fields of the records. The output of the hash function determines the location of disk block where the records are to be placed.
* **Clustered File Organization: -** is not considered good for large databases. In this mechanism, related records from one or more relations are kept in the same disk block, that is, the ordering of records is not based on primary key or search key.

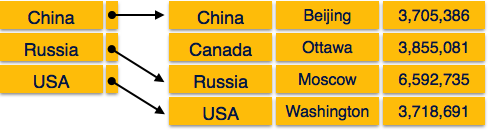
**File Operations: -**

* **Open:** A file can be opened in one of the two modes, read mode or write mode.
* **Locate:** Every file has a file pointer, which tells the current position where the data is to be read or written. This pointer can be adjusted accordingly. Using find (seek) operation, it can be moved forward or backward.
* **Read:** By default, when files are opened in read mode, the file pointer points to the beginning of the file.
* **Write:** User can select to open a file in write mode, which enables them to edit its contents. It can be deletion, insertion, or modification.
* **Close:** This is the most important operation from the operating system’s point of view. When a request to close a file is generated, the operating system
  + removes all the locks (if in shared mode),
  + saves the data (if altered) to the secondary storage media, and
  + releases all the buffers and file handlers associated with the file.

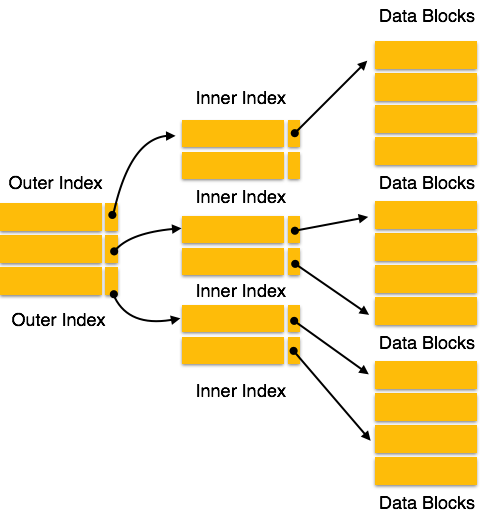
**18 INDEXING**

**Dense Index: -** there is an index record for every search key value in the database. This makes searching faster but requires more space to store index records itself. Index records contain search key value and a pointer to the actual record on the disk.

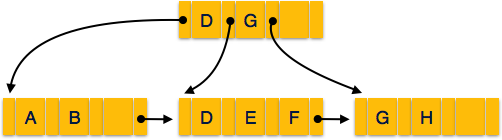
**Sparse Index: -**

In sparse index, index records are not created for every search key. An index record here contains a search key and an actual pointer to the data on the disk. To search a record, we first proceed by index record and reach at the actual location of the data. If the data we are looking for is not where we directly reach by following the index, then the system starts sequential search until the desired data is found.

**Multilevel Index: -**

Index records comprise search-key values and data pointers. Multilevel index is stored on the disk along with the actual database files. As the size of the database grows, so does the size of the indices. There is an immense need to keep the index records in the main memory so as to speed up the search operations. If single-level index is used, then a large size index cannot be kept in memory which leads to multiple disk accesses.

**B+ Tree: -**

A B+ tree is a balanced binary search tree that follows a multi-level index format. The leaf nodes of a B+ tree denote actual data pointers. B+ tree ensures that all leaf nodes remain at the same height, thus balanced. Additionally, the leaf nodes are linked using a link list; therefore, a B+ tree can support random access as well as sequential access

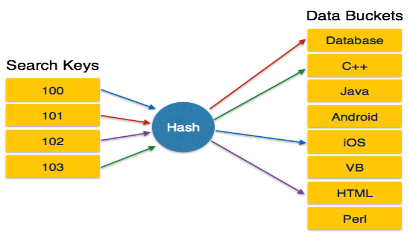
**19 HASHING**

Hashing uses hash functions with search keys as parameters to generate the address of a data record.

**Hash Organization: -**

* **Bucket:** A hash file stores data in bucket format. Bucket is considered a unit of storage. A bucket typically stores one complete disk block, which in turn can store one or more records.
* **Hash Function:** A hash function, h, is a mapping function that maps all the set of search-keys K to the address where actual records are placed. It is a function from search keys to bucket addresses.

**Static Hashing: -**

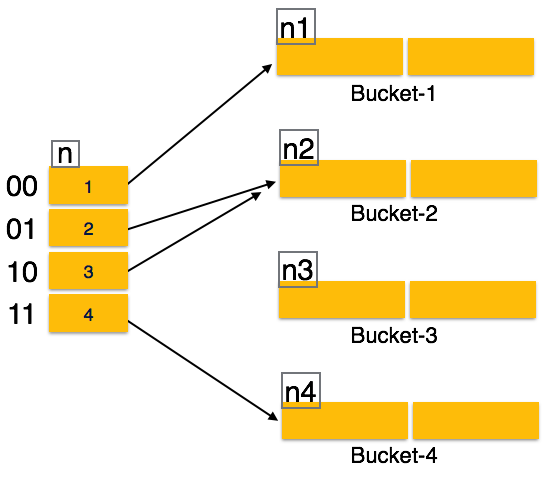
In static hashing, when a search-key value is provided, the hash function always computes the same address.

**Bucket Overflow: -**

he condition of bucket-overflow is known as **collision**. This is a fatal state for any static hash function. In this case, overflow chaining can be used.

|  |  |
| --- | --- |
| **Overflow Chaining** − When buckets are full, a new bucket is allocated for the same hash result and is linked after the previous one. This mechanism is called **Closed Hashing**. | Overflow chaining |
| Linear Probing: When a hash function generates an address at which data is already stored, the next free bucket is allocated to it. This mechanism is called Open Hashing. | Linear Probing |

**Dynamic Hashing: -**

The problem with static hashing is that it does not expand or shrink dynamically as the size of the database grows or shrinks. Dynamic hashing provides a mechanism in which data buckets are added and removed dynamically and on-demand. Dynamic hashing is also known as **extended hashing.**

Operation: -

* **Querying:** Look at the depth value of the hash index and use those bits to compute the bucket address.
* **Update:** Perform a query as above and update the data.
* **Deletion:** Perform a query to locate the desired data and delete the same.
* **Insertion:** Compute the address of the bucket.
  + If the bucket is already full,
    - Add more buckets.
    - Add additional bits to the hash value.
    - Re-compute the hash function.
  + Else,
    - Add data to the bucket,
  + If all the buckets are full, perform the remedies of static hashing.

**20 TRANSACTION**

**ACID Properties: -**

* **Atomicity:** This property states that a transaction must be treated as an atomic unit, that is, either all of its operations are executed or none. There must be no state in a database where a transaction is left partially completed. States should be defined either before the execution of the transaction or after the execution/abortion/failure of the transaction.
* **Consistency:** The database must remain in a consistent state after any transaction. No transaction should have any adverse effect on the data residing in the database. If the database was in a consistent state before the execution of a transaction, it must remain consistent after the execution of the transaction as well.
* **Durability:** The database should be durable enough to hold all its latest updates even if the system fails or restarts. If a transaction updates a chunk of data in a database and commits, then the database will hold the modified data. If a transaction commits but the system fails before the data could be written on to the disk, then that data will be updated once the system springs back into action.
* **Isolation:** In a database system where more than one transaction are being executed simultaneously and in parallel, the property of isolation states that all the transactions will be carried out and executed as if it is the only transaction in the system. No transaction will affect the existence of any other transaction.

**Serializability: -**

* **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.

two transactions are working on the same data, then the results may vary. This ever-varying result may bring the database to an inconsistent state. 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.

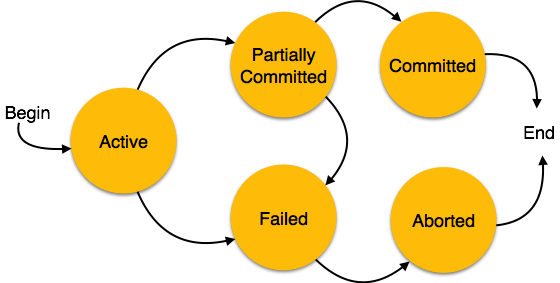
**Equivalence Schedules: -**

* **Result Equivalence: -** 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.
* **View Equivalence**: - Two schedules would be view equivalence if the transactions in both the schedules perform similar actions in a similar manner.
* **Conflict Equivalence: -** Two schedules would be conflicting if they have the following properties:
  + Both belong to separate transactions.
  + Both accesses the same data item.
  + At least one of them is "write" operation.

Two schedules having multiple transactions with conflicting operations are said to be conflict equivalent if and only if:

* + Both the schedules contain the same set of Transactions.
  + The order of conflicting pairs of operation is maintained in both the schedules.

**States of Transactions: -**

* **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:
  + Re-start the transaction
  + 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.

**21 Concurrency control**

Concurrency control protocols can be broadly divided into two categories:

**Lock-based Protocols: -** Database systems equipped with lock-based protocols use a mechanism by which any transaction cannot read or write data until it acquires an appropriate lock on it. Locks are of two kinds:

* **Binary Locks:** A lock on a data item can be in two states; it is either locked or unlocked.
* **Shared/exclusive Locks:** This type of locking mechanism differentiates the locks based on their uses. If a lock is acquired on a data item to perform a write operation, it is an exclusive lock. Allowing more than one transaction to write on the same data item would lead the database into an inconsistent state. Read locks are shared because no data value is being changed.

**Four types of lock protocols available:**

1. **Simplistic Lock Protocol:** Simplistic lock-based protocols allow transactions to obtain a lock on every object before a 'write' operation is performed. Transactions may unlock the data item after completing the ‘write’ operation.
2. **Pre-claiming Lock Protocol:** It is evaluate their operations and create a list of data items on which they need locks. Before initiating an execution, the transaction requests the system for all the locks it needs beforehand. If all the locks are granted, the transaction executes and releases all the locks when all its operations are over. If all the locks are not granted, the transaction rolls back and waits until all the locks are granted.
3. **Two-Phase Locking – 2PL:** This locking protocol divides the execution phase of a transaction into three parts. In the first part, when the transaction starts executing, it seeks permission for the locks it requires. The second part is where the transaction acquires all the locks. As soon as the transaction releases its first lock, the third phase starts. In this phase, the transaction cannot demand any new locks; it only releases the acquired locks.
4. **Strict Two-Phase Locking**: The first phase of Strict-2PL is same as 2PL. After acquiring all the locks in the first phase, the transaction continues to execute normally. But in contrast to 2PL, Strict-2PL does not release a lock after using it. Strict-2PL holds all the locks until the commit point and releases all the locks at a time.

**Timestamp-based Protocols: -**

This protocol uses either system time or logical counter as a timestamp. Timestamp-based protocols start working as soon as a transaction is created. Every transaction has a timestamp associated with it, and the ordering is determined by the age of the transaction.

**Timestamp Ordering Protocol**

The timestamp-ordering protocol ensures serializability among transactions in their conflicting read and write operations. This is the responsibility of the protocol system that the conflicting pair of tasks should be executed according to the timestamp values of the transactions.

* The timestamp of transaction Ti is denoted as TS(Ti).
* Read timestamp of data-item X is denoted by R-timestamp(X).
* Write timestamp of data-item X is denoted by W-timestamp(X).

Timestamp ordering protocol works as follows:

* If a transaction Ti issues a read(X) operation:
  + If TS(Ti) < W-timestamp(X)
    - Operation rejected.
  + If TS(Ti) >= W-timestamp(X)
    - Operation executed.
  + All data-item timestamps updated.
* If a transaction Ti issues a write(X) operation:
  + If TS(Ti) < R-timestamp(X)
    - Operation rejected.
  + If TS(Ti) < W-timestamp(X)
    - Operation rejected and Ti rolled back.
  + Otherwise, operation executed.

**Thomas' Write Rule:** This rule states if TS(Ti) < W-timestamp(X), then the operation is rejected and Ti is rolled back. Timestamp ordering rules can be modified to make the schedule view serializable. Instead of making Ti rolled back, the 'write' operation itself is ignored.

**22 DEADLOCK**

deadlock is an unwanted situation that arises in a shared resource environment, where a process indefinitely waits for a resource that is held by another process.

**Deadlock Prevention:-**

To prevent any deadlock situation in the system, the DBMS aggressively inspects all the operations, where transactions are about to execute. The DBMS inspects the operations and analyzes if they can create a deadlock situation.

**Wait-Die Scheme: -** In this scheme, if a transaction requests to lock a resource (data item), which is already held with a conflicting lock by another transaction, then one of the two possibilities may occur:

* If TS(Ti) < TS(Tj) — that is Ti, which is requesting a conflicting lock, is older than Tj — then Ti is allowed to wait until the data-item is available.
* If TS(Ti) > TS(tj) — that is Ti is younger than Tj — then Ti dies. Ti is restarted later with a random delay but with the same timestamp.

This scheme allows the older transaction to wait but kills the younger one.

**Wound-Wait Scheme: -** In this scheme, if a transaction requests to lock a resource (data item), which is already held with conflicting lock by another transaction, one of the two possibilities may occur:

* If TS(Ti) < TS(Tj), then Ti forces Tj to be rolled back — that is Ti wounds Tj. Tj is restarted later with a random delay but with the same timestamp.
* If TS(Ti) > TS(Tj), then Ti is forced to wait until the resource is available.

This scheme allows the younger transaction to wait; but when an older transaction requests an item held by a younger one, the older transaction forces the younger one to abort and release the item. In both the cases, the transaction that enters the system at a later stage is aborted.

**Deadlock Avoidance**

Aborting a transaction is not always a practical approach. Instead, deadlock avoidance mechanisms can be used to detect any deadlock situation in advance. Methods like "wait-for graph" are available but they are suitable for only those systems where transactions are lightweight having fewer instances of resource. In a bulky system, deadlock prevention techniques may work well.

**Wait-for Graph:** This is a simple method available to track if any deadlock situation may arise. For each transaction entering into the system, a node is created. When a transaction Ti requests for a lock on an item, say X, which is held by some other transaction Tj, a directed edge is created from Ti to Tj. If Tj releases item X, the edge between them is dropped and Ti locks the data item. The system maintains this wait-for graph for every transaction waiting for some data items held by others. The system keeps checking if there's any cycle in the graph.

Here, we can use any of the two following approaches:

* First, do not allow any request for an item, which is already locked by another transaction. This is not always feasible and may cause starvation, where a transaction indefinitely waits for a data item and can never acquire it.
* The second option is to roll back one of the transactions. It is not always feasible to roll back the younger transaction, as it may be important than the older one. With the help of some relative algorithm, a transaction is chosen, which is to be aborted. This transaction is known as the victim and the process is known as victim selection.

**23 DATA BACKUP**

**Loss of Volatile Storage**

A volatile storage like RAM stores all the active logs, disk buffers, and related data. In addition, it stores all the transactions that are being currently executed. What happens if such a volatile storage crashes abruptly? It would obviously take away all the logs and active copies of the database. It makes recovery almost impossible, as everything that is required to recover the data is lost. Following techniques may be adopted in case of loss of volatile storage:

* We can have checkpoints at multiple stages so as to save the contents of the database periodically.
* A state of active database in the volatile memory can be periodically dumped onto a stable storage, which may also contain logs and active transactions and buffer blocks.
* <dump> can be marked on a log file, whenever the database contents are dumped from a non-volatile memory to a stable one.

**Recovery**:

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

**Database Backup & Recovery from Catastrophic Failure**

A catastrophic failure is one where a stable, secondary storage device gets corrupt. With the storage device, all the valuable data that is stored inside is lost. We have two different strategies to recover data from such a catastrophic failure:

* **Remote backup –** Here a backup copy of the database is stored at a remote location from where it can be restored in case of a catastrophe.
* Alternatively, database backups can be taken on magnetic tapes and stored at a safer place. This backup can later be transferred onto a freshly installed database to bring it to the point of backup.

Grown-up databases are too bulky to be frequently backed up. In such cases, we have techniques where we can restore a database just by looking at its logs. So, all that we need to do here is to take a backup of all the logs at frequent intervals of time. The database can be backed up once a week, and the logs being very small can be backed up every day or as frequently as possible.

**Remote Backup**

Remote backup provides a sense of security in case the primary location where the database is located gets destroyed. Remote backup can be offline or realtime or online. In case it is offline, it is maintained manually.

Online backup systems are more real-time and lifesavers for database administrators and investors. An online backup system is a mechanism where every bit of the real-time data is backed up simultaneously at two distant places. One of them is directly connected to the system and the other one is kept at a remote place as backup.

As soon as the primary database storage fails, the backup system senses the failure and switches the user system to the remote storage. Sometimes this is so instant that the users can’t even realize a failure.

**24 DATA RECOVERY**

**Crash Recovery**

DBMS is a highly complex system with hundreds of transactions being executed every second. The durability and robustness of a DBMS depends on its complex architecture and its underlying hardware and system software. If it fails or crashes amid transactions, it is expected that the system would follow some sort of algorithm or techniques to recover lost data.

**Failure Classification**

To see where the problem has occurred, we generalize a failure into various categories, as follows:

* **Transaction Failure: -** A transaction has to abort when it fails to execute or when it reaches a point from where it can’t go any further. This is called transaction failure where only a few transactions or processes are hurt. Reasons for a transaction failure could be:
  + **Logical errors:** Where a transaction cannot complete because it has some code error or any internal error condition.
  + **System errors:** Where the database system itself terminates an active transaction because the DBMS is not able to execute it, or it has to stop because of some system condition. For example, in case of deadlock or resource unavailability, the system aborts an active transaction.
* **System Crash** There are problems – external to the system – that may cause the system to stop abruptly and cause the system to crash. For example, interruptions in power supply may cause the failure of underlying hardware or software failure. Examples may include operating system errors.
* **Disk Failure** In early days of technology evolution, it was a common problem where hard-disk drives or storage drives used to fail frequently. 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.

**Storage Structure**

We have already described the storage system. In brief, the storage structure can be divided into two categories:

* **Volatile storage:** As the name suggests, a volatile storage cannot survive system crashes. Volatile storage devices are placed very close to the CPU; normally they are embedded onto the chipset itself. For example, main memory and cache memory are examples of volatile storage. They are fast but can store only a small amount of information.
* **Non-volatile storage:** These memories are made to survive system crashes. They are huge in data storage capacity, but slower in accessibility. Examples may include hard-disks, magnetic tapes, flash memory, and non-volatile (battery backed up) RAM.

**Recovery and Atomicity**

When a system crashes, it may have several transactions being executed and various files opened for them to modify the data items. Transactions are made of various operations, which are atomic in nature. But according to ACID properties of DBMS, atomicity of transactions as a whole must be maintained, that is, either all the operations are executed or none.

When a DBMS recovers from a crash, it should maintain the following:

* It should check the states of all the transactions, which were being executed.
* A transaction may be in the middle of some operation; the DBMS must ensure the atomicity of the transaction in this case.
* It should check whether the transaction can be completed now or it needs to be rolled back.
* No transactions would be allowed to leave the DBMS in an inconsistent state.

There are two types of techniques, which can help a DBMS in recovering as well as maintaining the atomicity of a transaction:

* Maintaining the logs of each transaction, and writing them onto some stable storage before actually modifying the database.
* Maintaining shadow paging, where the changes are done on a volatile memory, and later, the actual database is updated.

**Log-based Recovery**

Log is a sequence of records, which maintains the records of actions performed by a transaction. It is important that the logs are written prior to the actual modification and stored on a stable storage media, which is failsafe. Log-based recovery works as follows:

* The log file is kept on a stable storage media.
* When a transaction enters the system and starts execution, it writes a log about it.

<Tn, Start>

* When the transaction modifies an item X, it write logs as follows:

<Tn, X, V1, V2>

It reads Tn has changed the value of X, from V1 to V2.

* When the transaction finishes, it logs:

<Tn, commit>

The database can be modified using two approaches:

* **Deferred database modification:** All logs are written on to the stable storage and the database is updated when a transaction commits.
* **Immediate database modification:** Each log follows an actual database modification. That is, the database is modified immediately after every operation.

**Recovery with Concurrent Transactions**

When more than one transaction are being executed in parallel, the logs are interleaved. At the time of recovery, it would become hard for the recovery system to backtrack all logs, and then start recovering. To ease this situation, most modern DBMS use the concept of 'checkpoints'.

**Checkpoint**: -

Keeping and maintaining logs in real time and in real environment may fill out all the memory space available in the system. As time passes, the log file may grow too big to be handled at all. Checkpoint is a mechanism where all the previous logs are removed from the system and stored permanently in a storage disk. Checkpoint declares a point before which the DBMS was in consistent state, and all the transactions were committed.

**Recovery**: -

When a system with concurrent transactions crashes and recovers, it behaves in the following manner:

* The recovery system reads the logs backwards from the end to the last checkpoint.
* It maintains two lists, an undo-list and a redo-list.
* If the recovery system sees a log with <Tn, Start> and <Tn, Commit> or just <Tn, Commit>, it puts the transaction in the redo-list.
* If the recovery system sees a log with <Tn, Start> but no commit or abort log found, it puts the transaction in the undo-list.

All the transactions in the undo-list are then undone and their logs are removed. All the transactions in the redo-list and their previous logs are removed and then redone before saving their logs.