Write the condition of fourth normal from. Define a schema which is in BCNF but not in 4F.

Any relation is said to be in the fourth normal form when it satisfies the following conditions :

- It must be in Boyce Codd Normal Form (BCNF).
- It should have no multi-valued dependency.

Consider a relation schema $R(A,B,C)$ with functional dependencies:

A→B

A→C

This schema is in Boyce-Codd Normal Form (BCNF) because for every non-trivial functional dependency, the left-hand side is a superkey.

However, this schema is not in Fourth Normal Form (4NF) because it violates the condition for MVDs.

How does object relational model support user defined data type concept?

- **Data Abstraction:** ORM provides a layer of abstraction that allows users to define complex data structures beyond simple scalar types.
- **Custom Data Types:** ORM frameworks provide mechanisms for defining custom data types that map to underlying database types.
- **Object-Relational Mapping:** ORM bridges the gap between object-oriented programming languages and relational databases by mapping objects to database tables and vice versa.
- **Encapsulation and Reusability:** Users can define UDTs once and reuse them across multiple parts of the application, reducing redundancy and improving maintainability.
- **Data Integrity and Validation:** Users can define rules and validations for UDT attributes, ensuring that only valid data is stored in the database.

What is a nested transaction? How does it differ from conventional transaction?

A nested transaction is a transaction that contains one or more sub-transactions within its scope. Each sub-transaction operates as a separate transaction and can be committed or rolled back independently of the parent transaction.

Feature	Nested Transactions	Conventional Transactions
Scope	Contains one or more sub-transactions.	Des not contain one or more sub-transactions.
Independence	Sub-transactions are independent and can be committed or rolled back separately.	All operations are part of a single transaction that must be committed or rolled back as a whole.
Error Handling	Errors in sub-transactions can be rolled back independently.	Errors causes the entire transaction to roll back.

Feature	Nested Transactions	Conventional Transactions
Commit Control	May use different commit control mechanisms internally	Follows a single commit control mechanism.
Isolation Levels	Each sub-transaction can have its own isolation level.	Entire transaction has a single isolation level.
Complexity	Complex	Simple

Explain the nested loop join method of implementation for natural join operation.

```
for each tuple $t_r$ in r do begin
        for each tuple $t_s$ in s do begin
              test pair ($t_r$, $t_s$) to test if they satisfy the given join
condition ?
              if test satisfied
                 add $t_r$ . $t_s$ to the result;
      end inner loop
end outer loop
```

In the algorithm, t_r and t_s are the tuples of relations r and s, respectively. The notation

t_r. t_s is a tuple constructed by concatenating the attribute values of tuples t_r and t_s.

Cost Analysis of Nested-loop Join Algorithm

For analyzing the cost of the nested-loop join algorithm, consider a number of pairs of tuples as $n_r * n_s$. Here, n_r specifies the number of tuples in relation r and n_s specifies the number of tuples in relation s. For computing the cost, perform a complete scan on relation s. Thus,

Total number of block transfers in worst case = $n_r * b_s + b_r$

Total number of seeks required in worst case = $n_r + b_r$

Here, b_s and b_r are the number of blocks holding the tuples of relation r and s, respectively.

In the best case, both relations r and s have sufficient memory to fit in the memory simultaneously. So, each block will read only once. Thus,

Total number of block transfers in best case = $b_r + b_s$.

Total number of seeks required = $2(n_r + b_r)$

In case if any one of the relations given fits entirely into the memory, it is a must to use that relation as the inner relation. It is because we will read the inner relation only once. Thus,

Total number of block transfers in such case = $b_r + b_s$

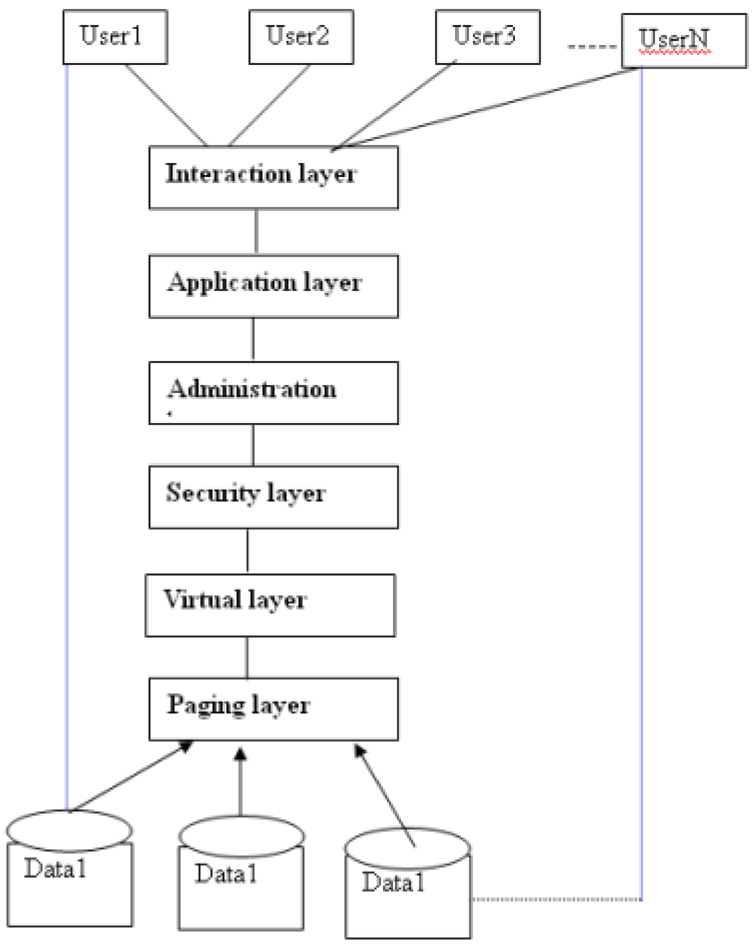
Total number of seeks required = $2(n_r + b_r)$

Explain the role of persistent objects.

Persistent objects are stored in the database and persist after program termination. The typical mechanisms for making an object persistent are naming and reachability.

- **Naming:** The naming mechanism involves giving an object a unique persistent name within a particular database. This persistent object name can be given via a specific statement or operation in the program. The named persistent objects are used as entry points to the database through which users and applications can start their database access.
- **Reachability:** The reachability mechanism works by making the object reachable from some other persistent object. An object B is said to be reachable from an object A if a sequence of references in the database lead from object A to object B.

Draw a neat diagram of the architecture of object oriented data base and explain.



The six-layer architecture model for object oriented database:

1. **Interaction Layer**- It is the first layer of this model. In this layer, user can interact with the databases. The user can send the data to databases as well as data can be retrieved from database to user.

2. **Application Layer**- It is the second layer of this model. If an early adopter of object-oriented database (ODB) technology selects an ODB system whose architecture is not well-suited for their specific application requirements, they may mistakenly conclude that no ODB system is suitable for addressing their needs.
3. **Administration Layer** -This layer is responsible for management of administrative information. This layer can change responsibility as per requirement.
4. **Security Layer** -The security layer plays important role in this model. The security layer is responsible to provide the full security to data and also provide to the security of application used to manage the data also.
5. **Virtual Layer** - The virtual layer manages the data virtually. Large volume of data is managed by putting the data outside the memory.
6. **Paging Layer**- The paging layer divides data into fixed-size pages and manages these pages within page frames in memory. This allows efficient data management, memory optimization and simplified data manipulation.

Differentiate between RDBMS, ORDBMS and OODBMS

Feature	**RDBMS**	**ORDBMS**	**OODBMS**
Data Model	Relational	Extends relational with object-oriented features	Object-Oriented
Data Persistence	Tables with predefined schemas	Supports both relational and object-oriented data storage	Stores objects directly with complex structures
Query Language	SQL (Structured Query Language)	Extended SQL with object-oriented features	Proprietary or extended query language
Schema Evolution	Complex schema changes, may require downtime	Supports schema evolution without disruption	Flexible schema evolution
Application Domain	Data-centric applications	Applications requiring flexibility	Complex data structures

What is the requirements of Data partitioning? Discuss the problems and challenges associated with data partitioning. Enlist advantages of the partitioning in brief.

Requirements of Data Partitioning:

- **Scalability:** Data partitioning should allow the database to scale horizontally by distributing data across multiple nodes or servers.
- **Performance:** It should enable parallel processing of queries across multiple partitions.
- **Availability:** Partitioning should enhance system availability by reducing the impact of failures on the overall system.

- **Load Balancing:** Partitioning should distribute the workload evenly across nodes or servers to avoid hotspots and prevent overloading of individual resources.

Problems and Challenges with Data Partitioning:

- **Data Skew:** Uneven distribution of data across partitions can lead to data skew.
- **Data Consistency:** Ensuring data consistency across partitions can be challenging, especially in distributed environments.
- **Joins and Aggregations:** Performing joins and aggregations across partitioned data can be complex and resource-intensive.
- **Partition Maintenance:** Partition maintenance requires careful planning.

Advantages of Data Partitioning:

- **Scalability:** Data partitioning enables horizontal scalability, allowing the database to handle larger data volumes and user loads by adding more nodes or servers.
- **Performance:** Partitioning improves query performance by reducing the data size for each query and enabling parallel processing across partitions.
- **Availability:** Partitioning enhances system availability by isolating failures to individual partitions, minimizing the impact on the overall system.
- **Fault Isolation:** Partitioning isolates faults to individual partitions, preventing them from affecting the entire system and improving fault tolerance.

Differentiate between Round-robin partitioning, Range index partitioning, Hash partitioning using suitable block diagram.

	Round-robin Partitioning	**Range Partitioning**	**Hash-based Partitioning**
Description	Divides dataset in a cyclic manner	Divides dataset based on specific range	Divides dataset based on hash function
Suitable Data	Equal-sized datasets	Ordered datasets	Unordered datasets
Query Performance	Basic load balancing	Efficient range queries	Even distribution
Data Distribution	Even distribution	Even distribution by range	Random distribution
Complexity	Limited query optimization	Joins and range queries	Inefficient key-based queries

What do you mean by a query tree and how is it helpful in query optimization?

A Query Tree is a data structure used for the internal representation of a query in RDBMS. The leaf nodes of the query tree represent the relations, and the internal nodes are the relational algebra operators like SELECT (**σ**), JOIN (**⋈**), etc. The root node gives the output of the query on execution.

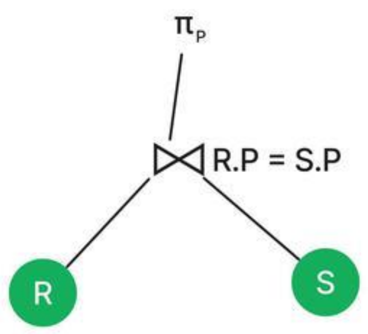
Steps To Make a Query Tree

Step 1: Execute the leaf nodes with their corresponding internal nodes having the relational algebra operator with the specified conditions to get the resulting tuples that we use for the execution of the next operation.

Step 2: This process continues until we reach the root node, where we PROJECT (**π**) the required tuples as the output based on the given conditions.

Example 1: Consider a relational algebra expression:

```
πp (R ⋈ R.P = S.P S)
```



Query trees allow database systems to apply optimization techniques such as reordering operations or using alternative access paths to improve query performance.

Consider three relations R (A, B, C), S(C, D, E) and T (E, F) where attributes A, C and E form the primary keys of relations R, S and T respectively. If R has 500 tuples, S has 1000 tuples and T has 900 tuples then compute the size of (R natural join S natural join T) and give an efficient way for computing the join.

Given:

Relation R has 500 tuples

Relation S has 1000 tuples

Relation T has 900 tuples

To find the number of tuples in the natural join (R ⋈ S ⋈ T), we need to consider the common attributes between pairs of relations. Since A, C, and E are the primary keys of R, S, and T respectively, the join conditions are:

R ⋈ S: Join on attribute C

(R ⋈ S) ⋈ T: Join on attribute E

Now, let's calculate the number of tuples in each step:

R ⋈ S:

Since attribute C is common, we perform a join on attribute C.

The number of tuples in the join result will be the minimum of the number of tuples in R and S, which is 500 (since R has 500 tuples).

(R ⋈ S) ⋈ T:

Since attribute E is common, we perform a join on attribute E.

The number of tuples in the join result will be the minimum of the number of tuples obtained in the previous step (500 tuples) and the number of tuples in T (900 tuples), which is 500 (since the previous result has 500 tuples).

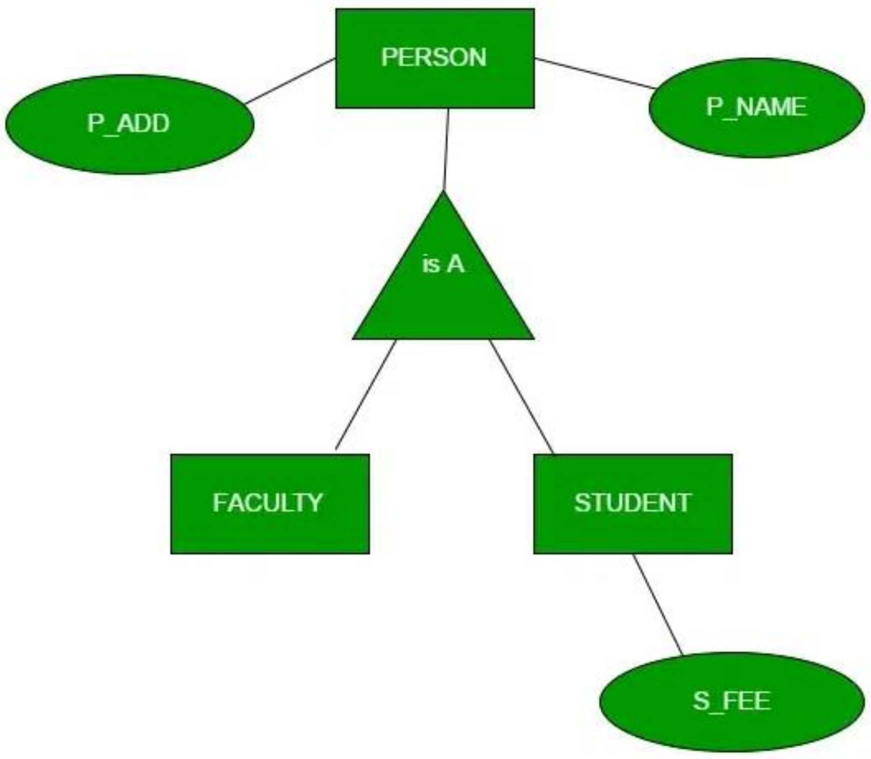
Therefore, the size of the natural join (R ⋈ S ⋈ T) will be the same as the number of tuples in the result of the last join, which is 500 tuples.

To efficiently compute the join, we can use indexing on the join attributes (C and E). Indexing helps reduce the time complexity of join operations by enabling faster lookup and retrieval of matching tuples. By indexing attributes C and E in relations S and T respectively, the database system can efficiently perform the join operations using index-based lookup methods such as hash join or merge join, resulting in faster query execution. Additionally, optimizing the join order (e.g., performing the join with the smallest relation first) and using appropriate join algorithms based on the size and characteristics of the relations can further improve the efficiency of the join operation.

How does generalization concept extend data base design part? Explain types of generalizations.

Generalization is the process of extracting common properties from a set of entities and creating a generalized entity from it. It is a bottom-up approach in which two or more entities can be generalized to a higher-level entity if they have some attributes in common.

For Example, STUDENT and FACULTY can be generalized to a higher-level entity called PERSON as shown in Figure 1. In this case, common attributes like P_NAME, and P_ADD become part of a higher entity (PERSON), and specialized attributes like S_FEE become part of a specialized entity (STUDENT).



Types of generalization:

- In a **total disjoint** generalization, every instance of the superclass must be a member of one and only one subclass.
- In a **partial disjoint** generalization, some instances of the superclass might not belong to any subclass.
- In a **total overlapping** generalization, every instance of the superclass must belong to at least one subclass, and it can belong to multiple subclasses.
- In a **partial overlapping** generalization, instances of the superclass may belong to one or more subclasses, but it's not mandatory for every instance to belong to any subclass.

Explain two phase commit protocol in distributed database. Also explain how does database recovery is performed in this protocol.

Two phase commit protocol:

1. **Prepare Phase**

- Each slave sends a ‘DONE’ message to the controlling site after each slave has completed its transaction.
- After getting ‘DONE’ message from all the slaves, it sends a “prepare” message to all the slaves.
- Then the slaves share their vote or opinion whether they want to commit or not. If a slave wants to commit, it sends a message which is “Ready”.
- If the slaves don’t want to commit, it then sends a “Not Ready” message.

2. **Commit/Abort Phase**

Controlling Site, after receiving "Ready" message from all the slaves

- The controlling site sends a message "Global Commit" to all the slaves. The message contains the details of the transaction which needs to be stored in the databases.
- Then each slave completes the transaction and returns an acknowledgement message back to the controlling site.
- The controlling site after receiving acknowledgement from all the slaves, which means the transaction is completed.

When the controlling site receives "Not Ready" message from the slaves

- The controlling site sends a message "Global Abort" to all the slaves.
- After receiving the "Global Abort" message, the transaction is aborted by the slaves. Then the slaves sends back an acknowledgement message back to the controlling site.
- When the controlling site receives Abort Acknowledgement from all the slaves, it means the transaction is aborted.

Database Recovery in Two-Phase Commit Protocol:

Coordinator Recovery:

- **Analyze Log:** Check for <prepare T>, <commit T>, or <abort T>.
- **Decision:**
    - If <commit T> or <abort T> present, resend final decision to participants.
    - If only <prepare T>, consult participants and decide.
- **Force Log:** Log and force the final decision.
- **Resend Decision:** Ensure participants execute the final decision.

Participant Recovery:

- **Analyze Log:** Check for <ready T>, <no T>, <commit T>, or <abort T>.
- **State-Based Action:**
    - If <commit T> or <abort T>, execute the logged decision.
    - If <ready T> without a final decision, contact coordinator.
- **Execute Decision:** Perform commit or abort based on coordinator's response.
- **Log Final Action:** Log the final action.

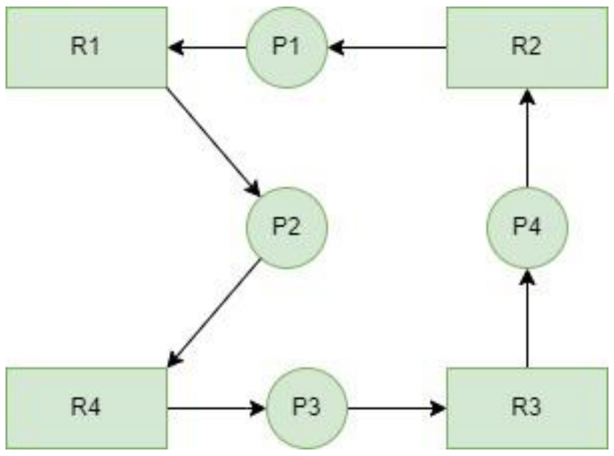
Explain graph based method of distributed deadlock handling.

Wait for Graph Algorithm

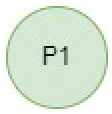
Step 1: Take the first process (Pi) from the resource allocation graph and check the path in which it is acquiring resource (R_i), and start a wait-for-graph with that particular process.
Step 2: Make a path for the Wait-for-Graph in which there will be no Resource included from the current process (P_i) to next process (P_j), from that next process (P_j) find a resource (R_j) that will be acquired by next Process (P_k) which is released from Process (P_j).
Step 3: Repeat Step 2 for all the processes.

Step 4: After completion of all processes, if we find a closed-loop cycle then the system is in a deadlock state, and deadlock is detected.

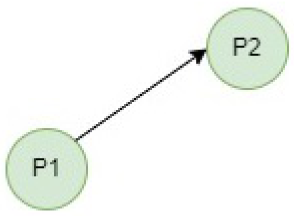
Now we will see the working of this Algorithm with an Example. Consider a Resource Allocation Graph with 4 Processes P1, P2, P3, P4, and 4 Resources R1, R2, R3, R4. Find if there is a deadlock in the Graph using the Wait for Graph-based deadlock detection algorithm.



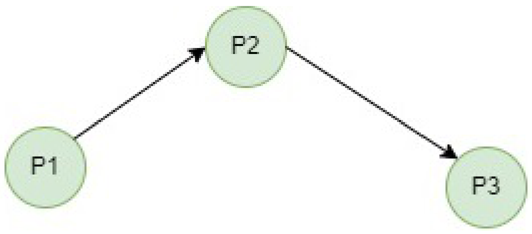
Step 1: First take Process P1 which is waiting for Resource R1, resource R1 is acquired by Process P2, Start a Wait-for-Graph for the above Resource Allocation Graph.



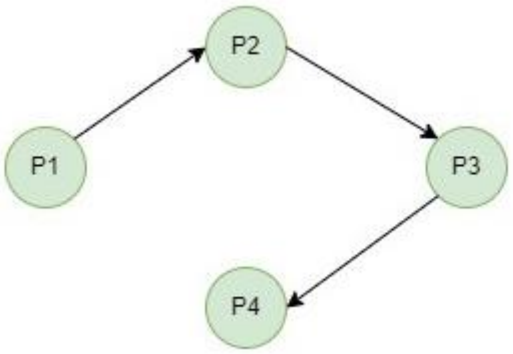
Step 2: Now we can observe that there is a path from P1 to P2 as P1 is waiting for R1 which is been acquired by P2. Now the Graph would be after removing resource R1 looks like.



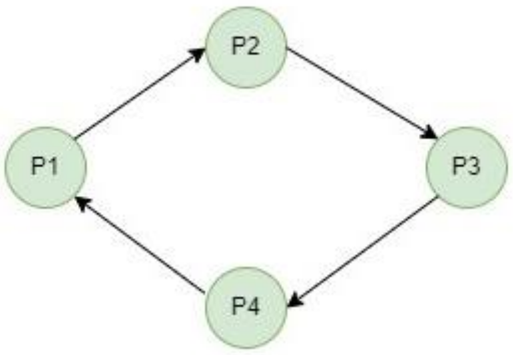
Step 3: From P2 we can observe a path from P2 to P3 as P2 is waiting for R4 which is acquired by P3. So make a path from P2 to P3 after removing resource R4 looks like.



Step 4: From P3 we find a path to P4 as it is waiting for P3 which is acquired by P4. After removing R3 the graph looks like this.



Step 5: Here we can find Process P4 is waiting for R2 which is acquired by P1. So finally the Wait-for-Graph is as follows:



Step 6: Finally, in this Graph, we found a cycle as the Process P4 again came back to the Process P1 which is the starting point (i.e., it's a closed-loop). So, according to the Algorithm if we found a closed loop, then the system is in a deadlock state. So here we can say the system is in a deadlock state.

What do you mean by false deadlock? Explain with suitable example.

Same as Phantom Deadlock

Explain multidimensional data model in data warehousing? What are its applications?
Multidimensional data model is a conceptual representation of data that organizes information into multiple dimensions. They allow users to rapidly receive answers to the requests which they made by creating and examining the data comparatively fast.

Features of multidimensional data models:

- **Measures:** Measures are numerical data that can be analyzed and compared, such as sales or revenue.
- **Dimensions:** Dimensions are attributes that describe the measures, such as time, location, or product.
- **Cubes:** Cubes are structures that represent the multidimensional relationships between measures and dimensions in a data model.
- **Aggregation:** Aggregation is the process of summarizing data across dimensions and levels of detail.
- **Drill-down and roll-up:** Drill-down is the process of moving from a higher-level summary of data to a lower level of detail, while roll-up is the opposite process of moving from a lower-level detail to a higher-level summary.
- **Hierarchies:** Hierarchies provide a way to navigate the data and perform drill-down and roll-up operations.

Applications of Multidimensional Data Model:

- **Data Analysis and Reporting:** Multidimensional data models are widely used for data analysis and reporting in business intelligence (BI).
- **OLAP (Online Analytical Processing):** OLAP tools uses multidimensional data models to provide interactive querying capabilities for exploring large volumes of data.
- **Data Mining and Predictive Analytics:** Multidimensional data models serve as the foundation for data mining and predictive analytics applications.
- **Budgeting and Forecasting:** Multidimensional data models are used in financial planning, budgeting, and forecasting processes.
- **Customer Segmentation and Personalization:** Multidimensional data models are used to segment customers based on demographic and behavioural data.

Association and Classification rule in Data Mining

Association Rule: An association rule is an implication of the form A→B, where A and B are disjoint itemsets. The rule means that the presence of A in a transaction implies the presence of B.

Key Metrics:

1. **Support**:

   - Indicates how frequently an itemset appears in the database.
   - Formula: $\text{Support}(A \rightarrow B) = \frac{\text{Number of transactions containing } A \text{ and } B}{\text{Total number of transactions}}$

2. **Confidence**:

   - Measures how often items in B appear in transactions that contain A.
   - Formula: $\text{Confidence}(A \rightarrow B) = \frac{\text{Number of transactions containing } A \text{ and } B}{\text{Number of transactions containing } A}$

3. **Lift**:

   - Indicates the strength of a rule over the random occurrence of A and B.
   - Formula: $\text{Lift}(A \rightarrow B) = \frac{\text{Support}(A \rightarrow B)}{\text{Support}(A) \times \text{Support}(B)}$

Classification Rule: A classification rule assigns items to predefined categories or classes.

Key Metrics:

1. **Accuracy**:

   - The proportion of correctly predicted instances out of the total instances.
   - Formula: $\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}}$

2. **Precision**:

   - The proportion of true positive predictions out of all positive predictions.
   - Formula: $\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$

3. **Recall**:

- The proportion of true positive predictions out of all actual positive instances.
- Formula: $\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$