DBMS Practical 08

Aim: To study lossy and lossless decomposition in DBMS

Theory

- What is decomposition ?
- Why do we need decomposition ?
- Explain Hierarchial Decomposition.
- Explain how lossy and lossless decomposition is done. Give examples.
- Explain the identification of lossy and lossless decomposition.
- Explain woolman's decomposition algorithm for multiple decompositions, and find out whether the decomposition is lossy or lossless.

Code: Not Applicable

Decomposition is the process of breaking down a complex database schema into smaller, more manageable components. The main goal of decomposition is to improve the efficiency, maintainability, and scalability of a database system by reducing redundancy and increasing modularity.

The need for decomposition arises from the fact that as a database system grows in size and complexity, it becomes increasingly difficult to manage and maintain. Decomposing the system into smaller components allows for better organization and easier maintenance. It also helps in minimizing the risk of data inconsistencies, as each component can be designed to manage a specific aspect of the data.

Hierarchical decomposition is a technique used in database design to break down a complex schema into a set of smaller, more manageable schemas that are organized in a hierarchical fashion. The goal of hierarchical decomposition is to reduce the complexity of the database schema, making it easier to understand, maintain and modify.

BCNF, or Boyce-Codd Normal Form, is a higher level of normalization that guarantees that a relation is free of anomalies and dependencies. It is a more stringent form of normalization than third normal form (3NF) and is considered to be one of the most desirable forms of normalization.

To achieve BCNF, hierarchical decomposition is often used. The process of hierarchical decomposition involves breaking down a complex schema into a set of smaller schemas, each of which satisfies BCNF. This is achieved by identifying functional dependencies and then creating separate schemas for each of the dependencies that do not meet the BCNF criteria.

For example, let's say we have a database schema with the following relation:

`R(A, B, C, D, E, F)`

We identify the following functional dependencies:

`A -> B`

`B -> C, D`

`D -> E, F`

The last two functional dependencies do not meet the BCNF criteria, so we create these schemas:

`R1(D, E, F) satisfying D -> E, F` where D is the key `(BCNF)`

`R2(B, C, D) satisfying B -> C, D` where B is the key `(BCNF)`

`R3(A, B) satisfying A -> B` where A is the key `(BCNF)`

By decomposing the original schema in this way, we have achieved BCNF for each of the smaller schemas, while also ensuring that the original schema is free of anomalies and dependencies.

Lossless decomposition is a decomposition technique where we ensure that we can always recover the original relation from the smaller relations produced by the decomposition. In other words, no information is lost during the decomposition process. This ensures that we maintain all of the functional dependencies present in the original schema.

Lossy decomposition is a technique where some information is lost during the decomposition process. This is done to reduce redundancy and improve the overall efficiency of the database. When performing a lossy decomposition, we accept that we may not be able to recover the original schema from the smaller relations produced by the decomposition.

Testing for lossless decomposition is done by checking if the closure of the intersection of the smaller relations produced by the decomposition is equal to either of the original relations. If it is, then the decomposition is lossless. If not, then the decomposition is lossy.

For Example:

If we have a schema `R(A, B, C, D)` with functional dependencies

`A -> B`,

`B -> C` and

`C -> D`

and we decompose it into

`R1(A, B)` and

`R2(B, C, D)`,

the common attribute `(B)` can derive the second relation in its entirety `(B⁺ = BCD)`. Therefore, this is a lossless decomposition.

If we have a schema `R(A, B, C, D)` with functional dependencies

`A -> B`,

`BC -> D`,

`C -> D`

and we decompose it into

`R1(A, B)` and

`R2(B, C, D)`,

the common attribute cannot derive any of the relations in their entirety `(B⁺ = B)`. Therefore, this is a lossy decomposition.

For more complicated decompositions, we use the Woolman Algorithm

The Woolman algorithm has 3 parts.

1. Initialization : `A table is created with the attributes of the original relation as the columns and the smaller relations as the rows. The table is then filled with the values of the smaller relations, with a(column) if attribute is present and b(row, column) if attribute is not present.`
2. Processing: `For every functional dependency in the original relation, we check if the smaller relations commonly contain the attributes on the left hand side of the functional dependency. If they do, we mark the right hand side with the common value in the table, preferring a over b. If they do not, we leave it as it is.`
3. Termination: `The algorithm terminates when either one of the rows is completely filled with a or, when the table is not changed in a pass. If there is a row that contains all a, then the decomposition is lossless. If not, then the decomposition is lossy.`

Conclusion

In this practical we learnt about decomposition and how to identify lossy and lossless decomposition for simple decompositions with closure checking and for more complicated decompositions with the Woolman algorithm.