Armstrong Axioms

The term Armstrong Axioms refers to the sound and complete set of inference rules or axioms, introduced by William W. Armstrong, that is used to test the logical implication of functional dependencies. If F is a set of functional dependencies then the closure of F, denoted as F+, is the set of all functional dependencies logically implied by F. Armstrong's Axioms are a set of rules, that when applied repeatedly, generates a closure of functional dependencies.

Axioms

- Axiom of Reflexivity: If A is a set of attributes and B is a subset of A, then A holds B. If B⊆A then A→B. This property is trivial property.
- Axiom of Augmentation: If A→B holds and Y is the attribute set, then AY→BY also holds. That is adding attributes to dependencies, does not change the basic dependencies. If A→B, then AC→BC for any C.
- Axiom of Transitivity**:** Same as the transitive rule in algebra, if A→B holds and B→C holds, then A→C also holds. A→B is called A functionally which determines B. If X→Y and Y→Z, then X→Z.

Secondary Rules

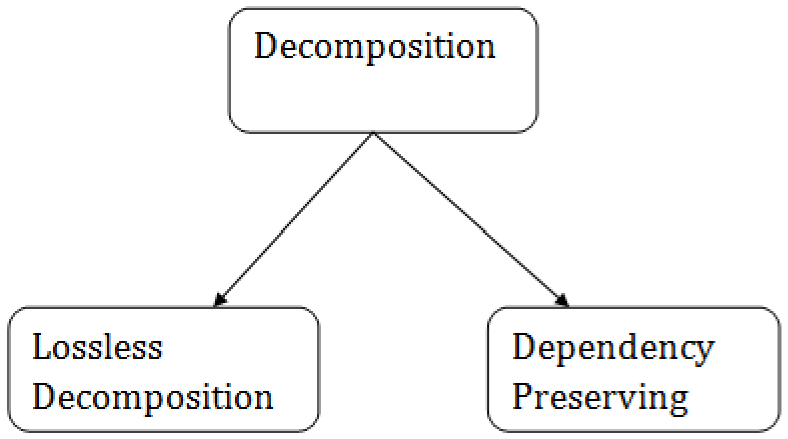
These rules can be derived from the above axioms.
Union: If A→B holds and A→C holds, then A→BC holds.
If X→Y and X→Z then X→YZ.
Composition: If A→B and X→Y hold, then AX→BY holds.
Decomposition: If A→BC holds then A→B and A→C hold.
If X→YZ then X→Y and X→Z.
Pseudo Transitivity: If A→B holds and BC→D holds, then AC→D holds.
If X→Y and YZ→W then XZ→W.
Self Determination: It is similar to the Axiom of Reflexivity, i.e. A→A for any A.

Extensivity: Extensivity is a case of augmentation. If AC→A, and A→B, then AC→B. Similarly, AC→ABC and ABC→BC. This leads to AC→BC.

Relational Decomposition

- When a relation in the relational model is not in appropriate normal form then the decomposition of a relation is required.
- In a database, it breaks the table into multiple tables.
- If the relation has no proper decomposition, then it may lead to problems like loss of information.
- Decomposition is used to eliminate some of the problems of bad design like anomalies, inconsistencies, and redundancy.

Types of Decomposition



Lossless Decomposition

- If the information is not lost from the relation that is decomposed, then the decomposition will be lossless.
- The lossless decomposition guarantees that the join of relations will result in the same relation as it was decomposed.
- The relation is said to be lossless decomposition if natural joins of all the decomposition give the original relation.

Example

— Employee (Employee_Id, Ename, Salary, Department_Id, Dname)
Can be decomposed using **lossless decomposition** as,
—Employee_desc(Employee_Id,Ename,Salary,Department_Id)
—Department_desc(Department_Id,Dname)
Alternatively the **lossy decomposition** would be as joining these tables is not possible so not possible to get back original data.
–Employee_desc(Employee_Id,Ename,Salary)
– Department_desc (Department_Id, Dname)

Advantages of Lossless Decomposition

Reduced Data Redundancy: Lossless decomposition helps in reducing the data redundancy that exists in the original relation. This helps in improving the efficiency of the database system by reducing storage requirements and improving query performance.

Maintenance and Updates: Lossless decomposition makes it easier to maintain and update the database since it allows for more granular control over the data.

Improved Data Integrity: Decomposing a relation into smaller relations can help to improve data integrity by ensuring that each relation contains only data that is relevant to that relation. This can help to reduce data inconsistencies and errors.

Improved Flexibility: Lossless decomposition can improve the flexibility of the database system by allowing for easier modification of the schema

Disadvantages of Lossless Decomposition

Increased Complexity: Lossless decomposition can increase the complexity of the database system, making it harder to understand and manage.

Increased Processing Overhead: The process of decomposing a relation into smaller relations can result in increased processing overhead. This can lead to slower query performance and reduced efficiency.

Join Operations: Lossless decomposition may require additional join operations to retrieve data from the decomposed relations. This can also result in slower query performance.

Costly: Decomposing relations can be costly, especially if the database is large and complex. This can require additional resources, such as hardware and personnel.

Dependency Preserving

- It is an important constraint of the database.
- In the dependency preservation, at least one decomposed table must satisfy every dependency.
- If a relation R is decomposed into relation R1 and R2, then the dependencies of R either must be a part of R1 or R2 or must be

derivable from the combination of functional dependencies of R1 and R2.

Example

suppose there is a relation R (A, B, C, D) with functional dependency set (A->BC). The relational R is decomposed into R1(ABC) and R2(AD) which is dependency preserving because FD A->BC is a part of relation R1(ABC).