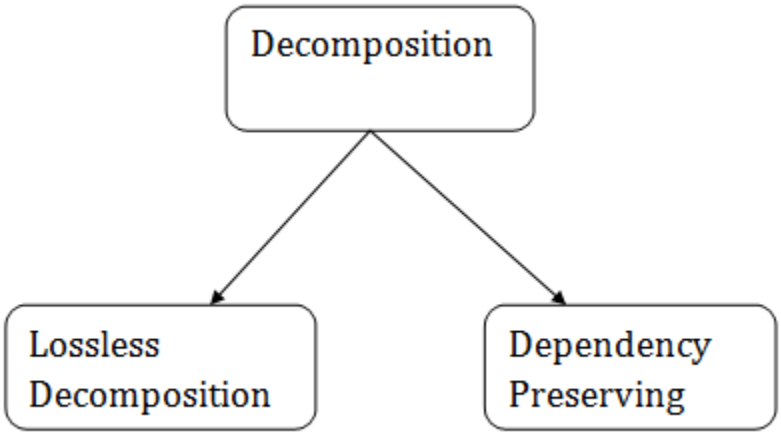
Relational Decomposition

- 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_DEPARTMENT table:

EMP_ID	EMP_NAME	EMP_AGE	EMP_CITY	DEPT_ID	DEPT_NAME
22	Denim	28	Mumbai	827	Sales
33	Alina	25	Delhi	438	Marketing
46	Stephan	30	Bangalore	869	Finance
52	Katherine	36	Mumbai	575	Production
60	Jack	40	Noida	678	Testing

The above relation is decomposed into two relations EMPLOYEE and DEPARTMENT

EMPLOYEE table:

EMP_ID	EMP_NAME	EMP_AGE	EMP_CITY
22	Denim	28	Mumbai
33	Alina	25	Delhi
46	Stephan	30	Bangalore
52	Katherine	36	Mumbai
60	Jack	40	Noida

DEPARTMENT table

DEPT_ID	EMP_ID	DEPT_NAME
827	22	Sales
438	33	Marketing
869	46	Finance
575	52	Production
678	60	Testing

Now, when these two relations are joined on the common column "EMP_ID", then the resultant relation will look like:

Employee ⋈ Department

EMP_ID	EMP_NAME	EMP_AGE	EMP_CITY	DEPT_ID	DEPT_NAME
22	Denim	28	Mumbai	827	Sales
33	Alina	25	Delhi	438	Marketing
46	Stephan	30	Bangalore	869	Finance
52	Katherine	36	Mumbai	575	Production
60	Jack	40	Noida	678	Testing

Hence, the decomposition is Lossless join decomposition.

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.
- For 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).

Properties of 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, breaking down the table into multiple tables termed as decomposition.

The properties of a relational decomposition are listed below :

1. **Attribute Preservation:**
   Using functional dependencies the algorithms decompose the universal

relation schema R in a set of relation schemas D = { R1, R2, ….. Rn } relational database schema, where ‘D’ is called the Decomposition of R. The attributes in R will appear in at least one relation schema Ri in the decomposition, i.e., no attribute is lost. This is called the *Attribute Preservation* condition of decomposition.

2. Dependency Preservation**:**
   If each functional dependency X->Y specified in F appears directly in one of the relation schemas Ri in the decomposition D or could be inferred from the dependencies that appear in some Ri. This is the *Dependency Preservation*.
   If a decomposition is not dependency preserving some dependency is lost in decomposition. To check this condition, take the JOIN of 2 or more relations in the decomposition.

   For example:

   ```
   R = (A, B, C)

   F = {A ->B, B->C}

   Key = {A}


   R is not in BCNF.

   Decomposition R1 = (A, B), R2 = (B, C)
   ```

   R1 and R2 are in BCNF, Lossless-join decomposition, Dependency preserving.
   Each Functional Dependency specified in F either appears directly in one of the relations in the decomposition.
   It is not necessary that all dependencies from the relation R appear in some relation Ri.
   It is sufficient that the union of the dependencies on all the relations Ri be equivalent to the dependencies on R.

3. **Non Additive Join Property:**
   Another property of decomposition is that D should possess is the *Non Additive Join Property*, which ensures that no spurious tuples are generated when a NATURAL JOIN operation is applied to the relations resulting from the decomposition.

4. **No redundancy:**
   Decomposition is used to eliminate some of the problems of bad design like anomalies, inconsistencies, and redundancy.If the relation has no proper decomposition, then it may lead to problems like loss of information.

5. Lossless Join**:**
   Lossless join property is a feature of decomposition supported by normalization. It is the ability to ensure that any instance of the original relation can be identified from corresponding instances in the smaller relations.
   For example:
   R : relation, F : set of functional dependencies on R,
   X, Y : decomposition of R,
   A decomposition {R1, R2, …, Rn} of a relation R is called a lossless decomposition for R if the natural join of R1, R2, …, Rn produces exactly the relation R.

   A decomposition is lossless if we can recover:
   R(A, B, C) -> Decompose -> R1(A, B) R2(A, C) -> Recover -> R'(A, B, C)
   Thus, R' = R
   Decomposition is lossless if:
   X intersection Y -> X, that is: all attributes common to both X and Y functionally determine ALL the attributes in X.
   X intersection Y -> Y, that is: all attributes common to both X and Y functionally determine ALL the attributes in Y
   If X intersection Y forms a superkey of either X or Y, the decomposition of R is a lossless decomposition.

Practice Questions on Decomposition in DBMS

1. Apply Natural Join decomposition on the below two tables:

Cust_ID	Cust_Name	Cust_Age	Cust_Location
C001	Monica	22	Texas
C002	Rachel	33	Toronto
C003	Phoebe	44	Minnesota

Sec_ID	Cust_ID	Sec_Name

Sec1	S001	Accounts
Sec2	S002	Marketing
Sec3	S003	Telecom

Answer: The result will be:

Cust_ID	Cust_Name	Cust_Age	Cust_Location	Sec_ID	Sec_Name
S001	Monica	22	Texas	Sec1	Accounts
S002	Rachel	33	Toronto	Sec2	Marketing
S003	Phoebe	44	Minnesota	Sec3	Telecom

Thus, the relation mentioned above had lossless decomposition, which means there was no loss of data/information here.

2. Check whether this decomposition given is a lossy join decomposition.

Relational Schema = A (X, Y, Z) Decompositions, A1 (X, Y) A2 (X, Z)

Relational Schema

X	Y	Z
X1	Y1	Z1
X2	Y1	Z1
X1	Y2	Z2
X1	Y3	Z3

Decompositions

X	Y
X1	Y1
X2	Y1
X1	Y2
X1	Y3

X	Z
X1	Z1
X2	Z1
X1	Z2
X1	Z3

Answer: Lossy Decomposition

Solution: Now, if we want this decomposition to be lossy, then

$A \subset A1 \bowtie A2$ Thus, $A1 \bowtie A2$ will be equal to

X	Y	Z
X1	Y1	Z1
X1	Y1	Z2
X2	Y1	Z1
X1	Y2	Z2
X1	Y2	Z1
X1	Y3	Z3
X1	Y3	Z1

Here, since $A \subset A1 \bowtie A2$,

Thus, this is a lossy join decomposition.

3. Apply Lossless Join decomposition on the below two tables.

Student Table:

Name	Class
Nikita	8th
Aditya	9th
Ayush	10th

Detail Table:

Name	Roll Number
Nikita	21
Aditya	22
Ayush	23

Answer: If we join both of these relations, then the resultant relation would look like the table given below:

Student_Detail Table

Name	Class	Roll Number
Nikita	8th	21
Aditya	9th	22
Ayush	10th	23

FAQs

What are the advantages of decomposition?

Decomposition is a process that saves a lot of time. For instance, the code that is there for a complex program could be easily run to multiple lines of code. In case we make a mistake, it would take a very prolonged time to discover. Another advantage of decomposition is that it lets programmers copy useful chunks of code and then reuse them easily for various other programs.

Why do we use decomposition in DBMS?

The process of decomposition in DBMS helps us remove redundancy, inconsistencies and anomalies from a database when we divide the table into numerous tables.

What is lossy decomposition in DBMS?

In the case of lossy decomposition, whenever we decompose a relation into multiple relational schemas, then the loss of data/information occurs whenever we try to retrieve the original relation.

What is lossless decomposition in DBMS?

A decomposition is said to be lossless when it is feasible to reconstruct the original relation R using joins from the decomposed tables. It is the most preferred choice.

What is the difference between lossy and lossless decomposition?

A decomposition is said to be lossless when it is feasible to reconstruct the original relation R using Joins from the decomposed tables. It is the most preferred choice. In the case of lossy decomposition, whenever we decompose a relation into multiple relational schemas, then the loss of data/information occurs whenever we try to retrieve the original relation.

Decomposition Algorithms

In the previous section, we discussed decomposition and its types with the help of small examples. In the actual world, a database schema is too wide to handle. Thus, it requires algorithms that may generate appropriate databases.

2NF Decomposition

- Identify each partial FD.
- Remove the attributes that depend on each of the determinants so identified.
- Place these determinants in separate relations along with their dependent attributes.
- In original relation keep the composite key and any attributes that are fully functionally dependent on all of it.

Here, we will get to know the decomposition algorithms using functional dependencies for two different normal forms, which are:

- Decomposition to BCNF
- Decomposition to 3NF

Decomposition using functional dependencies aims at dependency preservation and lossless decomposition.

Let's discuss this in detail.

Decomposition to BCNF

Before applying the BCNF decomposition algorithm to the given relation, it is necessary to test if the relation is in Boyce-Codd Normal Form. After the test, if it is found that the given relation is not in BCNF, we can decompose it further to create relations in BCNF.

There are following cases which require to be tested if the given relation schema R satisfies the BCNF rule:

Case 1: Check and test, if a nontrivial dependency **α -> β** violate the BCNF rule, evaluate and compute α^+ , i.e., the attribute closure of α. Also, verify that α+ includes all the attributes of the given relation R. It means it should be the superkey of relation R.

Case 2: If the given relation R is in BCNF, it is not required to test all the dependencies in F^+. It only requires determining and checking the dependencies in the provided dependency set F for the BCNF test. It is because if no dependency in F causes a violation of BCNF, consequently, none of the F^+ dependency will cause any violation of BCNF.

> *Note: Case2 does not work if the relation gets decomposed. It means during the testing of the given relation R, we cannot check the dependency of F for the cause of violation of BCNF.*

BCNF Decomposition Algorithm

This algorithm is used if the given relation R is decomposed in several relations R_1, R_2,..., R_n because it was not present in the BCNF. Thus,

For every subset α of attributes in the relation R_i, we need to check that α^+ (an attribute closure of α under F) either includes all the attributes of the relation R_i or no attribute of R_i-α.

```
result={R};
done=false;
compute F+;
while (not done) do
       if (there is a schema Ri in result that is not in BCNF)
              then begin
                     let α->β be a nontrivial functional dependency that
holds
                     on Ri such that α->Ri is not in F+, and α □ β= ø;
                     result=(result-Ri) U (Ri-β) U (α,β);
              end
              else done=true;
```

Note: If some set of attributes α in Ri violates the specified condition in the algorithm, in such case consider the functional dependency α->(α+ - α) □ R_i. Such dependency can be present in the F^+ dependency.

This algorithm is used for decomposing the given relation R into its several decomposers. This algorithm uses dependencies that show the violation of BCNF for performing the decomposition of the relation R. Thus, such an algorithm not only generates the decomposers of relation R in BCNF but is also a lossless decomposition. It means there occurs no data loss while decomposing the given relation R into R_1, R_2, and so on...

The BCNF decomposition algorithm takes time exponential in the size of the initial relation schema R. With this, a drawback of this algorithm is that it may unnecessarily decompose the given relation R, i.e., over-normalizing the relation. Although decomposing algorithms for BCNF and 4NF are similar, except for a difference. The fourth normal form works on **multivalued dependencies,** whereas BCNF focuses on the **functional dependencies**. The multivalued dependencies help to reduce some form of repetition of the data, which is not understandable in terms of functional dependencies.

Difference between Multivalued Dependency and Functional Dependency

The difference between both dependencies is that a functional dependency expels certain tuples from being in a relation, but a multivalued dependency does not do so. It means a multivalued dependency does not expel or rule out certain tuples. Rather it requires other tuples of certain forms to exist in relation. Due to such a difference, the multivalued dependency is also referred to as **tuple-generating dependency,** and the functional dependency is referred to as **equality-generating dependency.**

Decomposition to 3NF

The decomposition algorithm for 3NF ensures the preservation of dependencies by explicitly building a schema for each dependency in the canonical cover. It guarantees that at least one schema must hold a candidate key for the one being decomposed, which in turn ensures the decomposition generated to be a lossless decomposition.

3NF Decomposition Algorithm

```
let Fc be a canonical cover for F;
i=0;
for each functional dependency α->β in Fc
      i = i+1;
R = αβ;
If none of the schemas Rj, j=1,2,…I holds a candidate key for R
Then
      i = i+1;
      Ri= any candidate key for R;
/* Optionally, remove the repetitive relations*/
Repeat
      If any schema Rj is contained in another schema Rk
 Then
/* Delete Rj */
Rj = Ri;
i = i-1;
until no more Rjs can be deleted
return (R1, R2, . . .  ,Ri)
```

Here, R is the given relation, and F is the given set of functional dependency for which F_c maintains the canonical cover. R_1, R_2, . . . , R_i are the decomposed parts of the given relation R. Thus, this algorithm preserves the dependency as well as generates the lossless decomposition of relation R.

A 3NF algorithm is also known as a **3NF synthesis algorithm.** It is called so because the normal form works on a dependency set, and instead of repeatedly decomposing the initial schema, it adds one schema at a time.

Drawbacks of 3NF Decomposing Algorithm

- The result of the decomposing algorithm is not uniquely defined because a set of functional dependencies can hold more than one canonical cover.
- In some cases, the result of the algorithm depends on the order in which it considers the dependencies in F_c.
- If the given relation is already present in the third normal form, then also it may decompose a relation.