Database Management System (DBMS)

Lecture-32

Dharmendra Kumar

November 4, 2020

Decomposition

Let R be a relation schema. A set of relation schemas $\{R_1, R_2, ..., R_n\}$ is a decomposition of R if

$$R = R_1 \cup R_2 \cup .... \cup R_n$$

That is, $\{R_1, R_2, ..., R_n\}$ is a decomposition of R if, for i = 1, 2,..., n, each R_i is a subset of R, and every attribute in R appears in at least one R_i. Let r be a relation on schema R, and let $r_i = \Pi_{R_i}(r)$ for i = 1, 2,... , n. That is, $\{r_1, r_2, ..., r_n\}$ is the database that results from decomposing R into $\{R_1, R_2, ..., R_n\}$. It is always the case that

$$r \subseteq r_1 \bowtie r_2 \bowtie .... \bowtie r_n.$$

A decomposition $\{R_1, R_2, ..., R_n\}$ of R is a lossless-join decomposition if, for all relations r on schema R,

$$r = \Pi_{R_1}(r) \bowtie \Pi_{R_2}(r) \bowtie .... \bowtie \Pi_{R_n}(r) ....(1)$$

If equation (1) is not satisfied, then this decomposition is said to be lossy decomposition.

Desirable Properties of Decomposition

Lossless-Join Decomposition

Let R be a relation schema, and let F be a set of functional dependencies on R. Let R_1 and R_2 form a decomposition of R. This decomposition is a lossless-join decomposition of R if at least one of the following functional dependencies is in F+:

- $R_1 \cap R_2 \rightarrow R_1$
- $R_1 \cap R_2 \rightarrow R_2$

In other words, if $R_1 \cap R_2$ forms a superkey of either R_1 or R_2, then the decomposition of R is a lossless-join decomposition.

Relational Database Design

Eample: Consider the following schema R and F.
R = (branch-name, branch-city, assets, customer-name, loan-number, amount)
F is the following :-
branch-name → branch-city assets
loan-number → amount branch-name
R is decomposed into following relation schemas:-
Branch-schema = (branch-name, branch-city, assets)
Loan-info-schema = (branch-name, customer-name, loan-number, amount)
Is this decomposition lossless or lossy?

Relational Database Design

Solution:

Branch-schema $\cap$ Loan-info-schema $=$ $\{$branch-name$\}$

Clearly, branch-name is a super key of Branch-schema. Therefore this decomposition is lossless.

Dependency Preservation

Let F be a set of functional dependencies on a schema R, and let $R_1, R_2, ..., R_n$ be a decomposition of R. Let $F_1, F_2, ..., F_n$ is the set of dependencies corresponding to $R_1, R_2, ..., R_n$.

Let F' = $F_1 \cup F_2 \cup ... \cup F_n$

If F' = F, then the decomposition will be functionally dependency preserve.

If F' $\neq$ F , then the decomposition may or may not be functionally dependency preserve.

In this case, compute F^+ and F'^+. If $F^+ = F'^+$, then the decomposition will be functionally dependency preserve otherwise not.

Relational Database Design

Eample: Consider the following schema R and F.
R = (branch-name, branch-city, assets, customer-name, loan-number, amount)
F is the following :-
branch-name $\rightarrow$ branch-city assets
loan-number $\rightarrow$ amount branch-name
R is decomposed into following relation schemas:-
Branch-schema = (branch-name, branch-city, assets)
Loan-info-schema = (branch-name, customer-name, loan-number, amount)
Is this decomposition functionally dependency preserve?

Relational Database Design

Solution:

Let F_1 and F_2 are the set of functional dependencies corresponding to Branch-schema and Loan-info-schema respectively. Therefore
$F_1 = \{$ branch-name $\rightarrow$ branch-city assets$\}$ and
$F_2 = \{$ loan-number $\rightarrow$ amount branch-name$\}$
Now, F' $= F_1 \cup F_2 = \{$branch-name $\rightarrow$ branch-city assets, loan-number $\rightarrow$ amount branch-name$\}$
Clearly, F' $=$ F. Therefore this decomposition is functionally dependency preserve.

Normalization

Normalization is a database design technique that reduces data redundancy and eliminates undesirable characteristics like Insertion, Update and Deletion Anomalies. Normalization rules divides larger tables into smaller tables and links them using relationships. The purpose of Normalization is to eliminate redundant (repetitive) data and ensure data is stored logically.

Anomalies in DBMS

Anomalies are problems that can occur in poorly planned, unnormalized databases where all the data is stored in one table (a flatfile database).
There are three types of anomalies that occur when the database is not normalized. These are – **insertion, update and deletion** anomaly. Let's take an example to understand this. Consider a relation Emp-Dept.

E#	Ename	Address	D#	Dname	Dmgr#
123456789	Akhilesh	Ghaziabad	5	Research	333445555
333445555	Ajay	Kanpur	5	Research	333445555
999887777	Shreya	Lucknow	4	Administration	987654321
987654321	Sanjay	Mirjapur	4	Administration	987654321
666884444	Om Prakash	Lucknow	5	Research	333445555
453453453	Manish	Delhi	5	Research	333445555
987987987	Ishani	Prayagraj	4	Administration	987654321
888665555	Garvita	Prayagraj	1	Headquarters	888665555

Table 1: Emp-Dept

Relational Database Design

Insertion anomaly: Let us assume that a new department has been started by the organization but initially there is no employee appointed for that department, then the tuple for this department cannot be inserted into this table as the E# will have NULL, which is not allowed as E# is primary key.
This kind of a problem in the relation where some tuple cannot be inserted is known as insertion anomaly.
Deletion anomaly:
Now consider there is only one employee in some department and that employee leaves the organization, then the tuple of that employee has to be deleted from the table, but in addition to that the information about the department also will get deleted.
This kind of a problem in the relation where deletion of some tuples can lead to loss of some other data not intended to be removed is known as deletion anomaly.

Relational Database Design

Modification/update anomaly:
Suppose the manager of a department has changed, this requires that the Dmgr# in all the tuples corresponding to that department must be changed to reflect the new status. If we fail to update all the tuples of the given department, then two different records of employee working in the same department might show different Dmgr# leading to inconsistency in the database.
This is known as modification/update anomaly.