Chapter 8 – Relational Database Design Algorithms and Further Dependencies

There are two main approaches for relational database design.

Top - down design – this involves designing a conceptual schema in a high-level data model, such as ER model, and then mapping the conceptual schema into a set of relations using mapping procedures. Following this each of the relations is analyzed based on the FDs and assigned primary keys, by applying normalization.

Bottom – up design – After the database designer specifies the dependencies, a normalization algorithm is applied to synthesize the relation schemas. Each individual relation schema resulted should be in **3NF** or **BCNF** or some higher normal form.

The normalization algorithms typically start by synthesizing the **universal relation** R = {A1, A2... An} that includes all the attributes of the database. Using the functional dependencies the algorithms decompose the universal relation schema R into a set of relation schemas D = {R1, R2, ..., Rm}. D is called the decomposition of R.

A decomposition must possess the following properties:

Attribute Preservation Condition : each attribute in R will appear in at least one relation schema in the decomposition so that no attributes are "lost"**.**

$$\bigcup_{i=1}^{m} R_i = R$$

Dependency Preservation: Each functional dependency *f* specified in *F* either appears directly in one of the relation schemas R_i in the decomposition *D* or could be inferred from the dependencies that appear in some R_i.

Lossless (Nonadditive) Joins: *Nonadditive join property* ensures that no spurious tuples are generated when a NATURAL JOIN operation is applied to the relations in the decomposition.

Example 1: A Lossy Decomposition

AB JOIN BC

A	B	C
a1	100	c1
a2	200	c2
a2	200	c4
a3	300	c3
a4	200	c2
a4	200	c4

Algorithm: Relational synthesis algorithm with dependency preservation

Input: A universal relation R and a set of functional dependencies F on the attributes of R

1. Find a minimal cover G for F;
2. For each left-hand-side X of a functional dependency that appears in G, create a relation schema in D with attributes $\{X \cup \{A_1\} \cup \{A_2\} \ldots \cup \{A_k\}\}$, where $X \rightarrow A_1$, $X \rightarrow A_2$, …, $X \rightarrow A_k$ are the only dependencies in G with X as left-hand-side (X is the key of this relation);
3. Place any remaining attributes (that have not been placed in any relation) in a single relation schema to ensure the attribute preservation property.

The above algorithm guarantees the dependency-preserving property; it does not guarantee the lossless join property.

Algorithm: Testing for the lossless (nonadditive) join property

1. Create an initial matrix S with 1 row I for each relation Ri in D, and 1 column j for each attribute Aj in R
2. Set S(i,j) := bij for all matrix entries.
3. For each row i representing relation schema Ri

   { for each column j representing attribute Aj

{ if (relation Ri includes attribute Aj) then set S(i,j) := aj; }; };

4. Repeat the following loop until a complete loop execution results in no changes to S

{ for each FD X→Y in F

{ for all rows in S which have the same symbols in the columns corresponding to attributes in X

{ make the symbols in each column that correspond to an attribute in Y be the same in all these rows as follows:

if any of the rows has an "a" symbol for the column, set the other rows to that same "a" symbol in the column. If no "a" symbol exists for the attribute in any of the rows, choose one of the "b" symbols that appear in one of the rows for the attribute and set the other rows to that same "b" symbol in the column ; } ;

} ; };

5. If a row is made up entirely of "a" symbols, then the decomposition has the lossless join property ;otherwise, it does not

Algorithm: Relational decomposition into BCNF relations with lossless join property

1. Set D := { R };
2. While there is a relation schema Q in D that is not in BCNF do:

{

choose a relation schema Q in D that is not in BCNF;

find a FD X → Y in Q that violates BCNF;

replace Q in D by 2 relation schemas (Q-Y) and (X U Y);

};

Algorithm: Relational synthesis algorithm with dependency preservation and lossless join property

1. Find a minimal cover G for F;
2. For each LHS, X of a FD that appears in G, create a relation schema in D with attributes {X U {A1} U {A2} U …{Ak}}, where X→A1, X→A2,…,X→Ak are the only dependencies in G with X as LHS
3. If none of the relation schemas in D contains a key of R, then create one or more relation schemas in D that contains attributes that form a key of R.

The above algorithm

- Preserves dependencies.
- Has the lossless join property.
- Is such that each resulting relation schema in the decomposition is in 3NF.

Multi-valued Dependencies

Multi-valued dependencies are a consequence of first normal form (1NF), which disallowed an attribute in a tuple to have a set of values. If we have two or more multi-valued independent attributes in the same relation schema, we get into a problem of having to repeat every value of one of the attributes with every value of the other attribute to keep the relation state consistent and to maintain the independence among the attributes involved. This constraint is specified by a multi-valued dependency.

EMP

ENAME	PNAME	DNAME
Smith	X	John
Smith	Y	Anna
Smith	X	Anna
Smith	Y	John

For example, consider the relation EMP shown in the above Figure. A tuple in this EMP relation represents the fact that an employee whose name is ENAME works on the project whose name is PNAME and has a dependent whose name is DNAME. An employee may work on several projects and may have several dependents, and the employee's projects and dependents are independent of one another. To keep the relation state consistent, we must have a separate tuple to represent every combination of an employee's dependent and an employee's project. This constraint is specified as a multi-valued dependency on the EMP relation.

Formal Definition of Multi-valued Dependency

Formally, a multi-valued dependency (MVD) X Y specified on relation schema R, where X and Y are both subsets of R, specifies the following constraint on any relation state r of R: If two tuples t1and t2 exist in r such that t1[X] = t2[X], then two tuples t3 and t4 should also exist in r with the following properties, where Z is used to denote (R - (X U Y))

- t3[X] = t4[X] = t1[X] = t2[X]
- t3[y] = t1[Y] and t4[Y] = t2[Y]
- t3[Z] = t2[Z] and t4[Z] = t1[Z]

An MVD X Y in R is called a **trivial MVD** if
(a) Y is a subset of X, or
(b) X U Y = R.

Fourth Normal Form

A relation schema R is in 4NF with respect to a set of dependencies F (that includes functional dependencies and multi-valued dependencies) if, for every nontrivial multi-valued dependency X Y in R , X is a super key for R.

The EMP relation in the above figure is not in 4NF because in the nontrivial MVDs
ENAME PNAME and ENAME DNAME,
ENAME is not a super key of EMP. We decompose EMP into EMP_PROJECTS and EMP_DEPENDENTS, shown in the following figure. Both EMP_PROJECTS and EMP_ DEPENDENTS are in 4NF, because the MVDs ENAME PNAME in EMP_PROJECTS and ENAME DNAME in EMP_DEPENDENTS are trivial MVDs.

EMP_PROJECTS

ENAME	PNAME
Smith	X
Smith	Y

EMP_DEPENDENTS

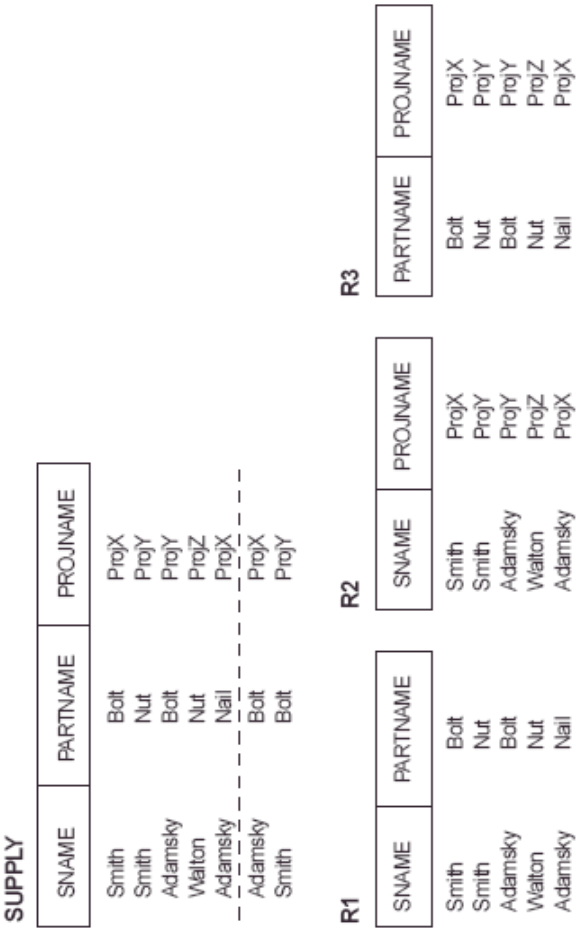
ENAME	DNAME
Smith	Anna
Smith	John

Join Dependencies

A join dependency (JD), denoted by JD, specified on relation schema R, specifies a constraint on the states r of R. The constraint states that every legal state r of R should have a lossless join decomposition into R1,R2, … Rn; that is, for every such r we have

$*(\pi_{R1}(r), \pi_{R2}(r) \ldots \pi_{Rn}(r)) = r$

For an example of a JD, consider once again the SUPPLY all-key relation of following figure. Figure shows how the SUPPLY relation with the join dependency is decomposed into three relations R1, R2, and R3. Applying NATURAL JOIN to any two of these relations produces spurious tuples, but applying NATURAL JOIN to all three together does not.



A join dependency JD, specified on relation schema R, is a trivial JD if one of the relation schemas in JD is equal to R.

Fifth Normal Form(Project Join Normal Form)

A relation schema R is in fifth normal form (5NF) (or project-join normal form (PJNF)) with respect to a set F of functional, multi-valued, and join dependencies if, for every nontrivial join dependency JD(R1, R2, …, Rn) in F^+(that is, implied by F), every Ri is a super key of R.