Ganesar College of Arts and Science

Department of Computer Science

Database System

Unit -V

Chapter : Relational Database design:

1. Features of Good Relational Designs

There are two types of good relational designs

i. Design alternative: Larger Schemas
ii. Design alternative: Smaller Schemas

Larger Schemas:

bor-loan = (customer-id,loan-no,amount)

- ✓ This represent the result of a natural join on the relations corresponding to borrower and loan. Notice that the borrower relationship set is many-to-many.
- ✓ This allows a customer to have several loans and also allows a loan to be made to several customers.
- ✓ It is important that all these tuples agree as to the amount of loan L-100 since otherwise our database would be inconsistent. In our original design using loan and borrower, we stored the amount of each loan exactly once.

Loan no	amount
-	-
L-100	10000
-	-

Loan

Customer id	Loan no
-	-
123-763	L-100
124-738	L-100

borrower

Customer id	Loan no	Amount
-	-	-
123-763	L-100	10000
124-738	L-100	10000
-	-	-
-	-	-

bor-loan

Smaller Schemas:

- ✓ A real world database has a large number of schemas and even larger number of attributes.
- ✓ Discovering repetition would be costly.
- ✓ There is an even more fundamental problem with this approach.
- ✓ We need to write a rule that says if there were a schema(loan no,amount),then loan no is able to serve as the primary key. This rule is specified as a functional dependency.

  Loan no→ amount

2. Atomic Domains and First Normal Form

- ✓ In the relational model, we formalize this idea that attribute do not have any substructure.
- ✓ A domain is atomic if elementsof the domain are considered to be invisible units.
- ✓ We say a relation schema R is in first normal form(1NF)

  If the domains of all attributes of R are atomic.

A set of names is an example of a nonatomic value. For example,if the schema of a relation employee included an attribute children whose domain elements are set of names,the schema would not be in first normal form.

3. Decomposition using Functional Dependencies

Define: Functional dependency is very important concept which is used in normalization. It is the pure decision and common sense of user to specify the functional dependency. Let A and B be two attributes of relation R.

$$\begin{tikzcd} & \mathbf{R} \arrow[dl] \arrow[dr] & \\ \mathbf{A} & & \mathbf{B} \end{tikzcd}$$

Given the value of A,if there is only one value of B analogous to it, then B is thought to be functionally dependent on A.

1) Keys and Functional Dependencies
2) Boyce-Codd Normal Form
3) BCNF and Dependency Preservation
4) Third Normal Form
5) Higher Normal Form

Keys and Functional Dependencies:

A subset K of R is a superkeny of R if, in any legal relation r(R), for all pairs t1 and t2 of tuples in r such that t1 ≠ t2, then t1[K] ≠ t2[K]. That is, no two tuples in any legal relation r(R) may have the same attribute set K.Consider a relation schema R, and let α ≤ R and β ≤ R.

bor-loan = (customer id, loan no, amount)

Functional Dependencies in Two ways:

1. To test relations to see whether they are legal under a given set of functional dependencies. If a relation r is legal under a set F of functional dependencies, we say that r satisfies F.
2. To specify constraints on the set of legal relation. We shall thus concern ourselves with only those relations that satisfy a given set of functional dependencies. If we wish to constrain ourselves to relation on schema R that satisfy a set F of functional dependencies, we say that F holds on R.

   Given that a set of functional dependencies F holds on a relation r, it may be possible to infer that certain other functional dependencies must also hold on the relation.
   Sample Example:

A	B	C	D
A1	B1	C1	D1
A1	B2	C1	D2
A2	B2	C2	D2
A2	B3	C2	D3
A3	B3	C2	D4

Boyce- Codd Normal Form

It eliminates all redundancy that can be discovered based on functional dependencies, though, there may be other types of redundancy remaining.

A relation schema R is in BCNF with respect to a set F of functional dependencies if, for all functional dependencies in F^+ of the form $\alpha \rightarrow \beta$, where $\alpha \leq R$ and $\beta \leq R$,at least one of the following holds.

- $\alpha \rightarrow \beta$ is a trivial functional dependency (that is, $\beta \leq \alpha$)
- α is a superkey for schema R.

Example of a schema that is not in BCNF, bor-loan=(<u>cid,lno</u>,amount). The functional dependency lno→ amount holds on bor-loan, but lno is not superkey. The borrower schema is in BCNF because no nontrivial functional dependencies hold on it.

Then there is at least one nontrivial functional dependency $\alpha \rightarrow \beta$ such that α is not a superkey for R. We replace R in our design with two schemas.

- $(\alpha \cup \beta)$
- $(R - (\beta - \alpha))$

In the case of bor-loan above, α = lno, β = amount, and bor-loan is replaced by

- (α U β) = (lno,amount)
- (R- (β - α)) = (cid,lno)

BCNF and Dependency Preservation

Several ways of expressing database consistency constraints primarykey,functional dependency,check constraints,assertions, and triggers. Testing these constraints each time the database is updated can be costly and, therefore, it os useful to design the database in a way that constraints can be tested efficiently.

In particular, if testing a functional dependency can be done by considering just one relation, then the cost of testing this constraint is low. Decomposition into BCNF can prevent efficient testing of certain functional dependencies.

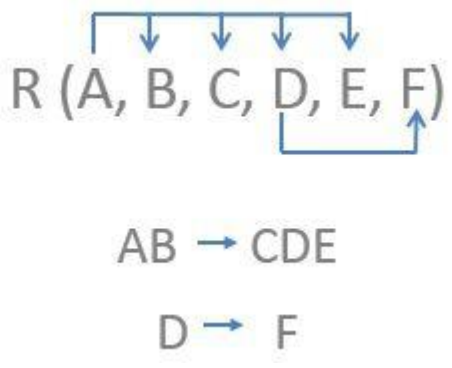
Third Normal Form

A table or a relation is considered to be in **Third Normal Form** only if the table is already in **2NF** and there is no **non-prime** attribute **transitively** dependent on the **candidate key** of a relation.

So, before I address the process of normalizing a table in 3NF, allow me to discuss the candidate key. A **Candidate Key** is **minimal super key** i.e. a super key with minimum attributes that can define all attributes of a relation.

So, in the process of normalizing your table, first, you recognise the candidate key of a given relation. The attributes that are part of candidate key are **prime attributes**, and the attributes that are not the part of candidate key are **non-prime attributes**.

Now if we a relation R(A, B, C, D, E, F) and we have following function dependencies for the relation R.



BCNF requires that all nontrivial dependencies be of the form α →β, where α is a superkey. Third normal form (3NF) relaxes this constraint slightly by allowing nontrivial functional dependencies whose left side is not a superkey. Before we define 3NF, we recall that a candidate key is a minimal superkey that is , a superkey no proper subset of which is also a superkey.

A relation schema R is in third normal form with respect to a set F of functional dependencies if, for all functional dependencies in F^+ of the form $\alpha \rightarrow \beta$, where $\alpha \leq R$ and $\beta \leq R$, at least one of the following holds,

- $\alpha \rightarrow \beta$ is a trivial functional dependency.
- α is a superkey for R.
- Each attribute A in $\beta - \alpha$ is contained in a candidate key for R.

Higher Normal Form

Using functional dependencies to decompose schemas may not be sufficient to avoid unnecessary repetition of information in certain cases. The employee entity set definition in which we allow employee to have several phone no, some of which may be shared by multiple employees .

Then telephone no would be a multivalued attribute and following our rules for generating schemas from an E-R design,we would have two schemas, one for each of the multivalued attributes, telephone no and dname.

(employee id,dname)

(employee id,telephone no)

If we were to combine these schemas to get (employee id, dname,telephone no).

Example :

(555-665, Dhaanya, 35985-33793)

(555-664, Dashwin, 34444-50696)

(555-666, David, 24555-76859)

4. Functional – Dependency Theory

1. <u>Closure of a set of functional dependencies</u>

Let F be the set of functional dependencies and the closure of F (denoted by F^+) is the set of all functional dependencies logically implied by F

With the help of armstrong's axioms we can find logically implied functional dependencies. We can find all of F^+ with the help of applying these rules repeatedly.

<u>Rules are given below:</u>

(R1) **Reflexivity rule :** If A is set of attributes and $B \leq A$, then $A \rightarrow B$ holds.

(R2) **Augmentation rule:** If $A \rightarrow B$ holds C is a set of attributes then $CA \rightarrow CB$ holds.

(R3) **Transitivity rule:** If $A \rightarrow B$ holds and $B \rightarrow C$ holds then $A \rightarrow C$ holds.

(R4) **Union rule:** If A → B holds and A→C holds then A→BC holds.

(R5) **Decomposition rule:** If A→BC holds then A→B holds and A→C holds.

(R6) **Pseudotransitivity rule:** If A→B holds and CB→D holds then AC→D holds.

Q. Let us apply rules to the scheme R = (A,B,C,G,H,I) and the set F of functional dependencies { A→B, A→C,CG→H, CG→I, B→H} List several members of F^+

Solution: A→H. Since A→B and B→H holds (transitivity rule)
CG→HI. Since CG→H and CG→I(union rule)
Implies that CG→ HI
AG→I. Since A→ C and CG → I (pseudotransitivity rule)
Implies that AG→I.

2. **Closure of Attribute sets**

An algorithm to compute closure of A is given below

```
result:= A
while( changes to result) do
for each function dependency B→C in F do
begin
if B ≤ result then result:=result U C
end
```

Example:

Given R=(A,B,C,G,H,I) with set

{A→B,A→C,CG→H,CG→I,B→H}

Find the closure of AG i.e. $(AG)^+$

Initialize X=AG

We start with result:=AG. First time when we execute the while loop to test each functional dependency we find that:

- ❖ A→ B cause us to include B in result. To check this fact we find that A→B is in F, A≤ result (which is AG) ,so result:=result U B.
- ❖ A→ C causes result become ABCG.
- ❖ CG → H causes result to become ABCGH.
- ❖ CG→I causes result to become ABCGHI.

3. Canonical Cover

An attributes of a functional dependency is said to be extraneous if we can remove it without changing the closure of the set of functional dependencies.

The formal definition of extraneous attributes is as follows.consider a set F of functional dependencies and the functional dependency $\alpha \rightarrow \beta$ in F.

A canonical cover F_c for F is set of dependencies such that F logically implies all dependencies in F_c and F_c logically implies all dependencies in F. Furthermore, F_c must have the following properties.

- No functional dependency in F_c contains an extraneous attribute.
- Each left side of a functional dependency in F_c is unique. That is there are no two dependencies $\alpha 1 \rightarrow \beta 1$ and $\alpha 2 \rightarrow \beta 2$ in F_c such that $\alpha 1 = \alpha 2$.

4. Lossless Decomposition

The decomposition is a lossless decomposition if for all legal database instances.

A decomposition that is not a lossless decomposition is called a lossy decomposition. The terms lossless-join decomposition and lossy-join decomposition are sometimes used in place of lossless decomposition and lossy decomposition.

Let R,R1,R2, and F be as above R1 and R2 form a lossless decomposition of R if at least one of the following functional dependencies is in F^+

$R1 \cap R2 \rightarrow R1$

$R1 \cap R2 \rightarrow R2$

5. Dependency Preservation

The restriction of F to R_i is the set F_i of all functional dependencies in F^+ that include only attributes of R_i . A decomposition having the property $F'^+ = F^+$ is a dependency –preserving decomposition.

```
compute $F^{+}$;
for each schema $R_i$ in D do
begin
$F_i$:= the restriction of $F^+$ to $R_i$;
end
F':=Ø
for each restriction $F_i$ do
begin
F'= F' U $F_i$
end
compute $F'^+$;
if($F'^+ = F^+$) then return (true)
            else return (false);
```

5. Decomposition using Functional Dependencies

BCNF Decomposition

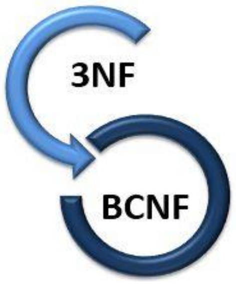
- Testing for BCNF
- BCNF Decomposition Algorithm

3NF Decomposition

Decomposition Given: relation R, set F of functional dependencies Find: decomposition of R into a set of 3NF relation Ri Algorithm: (1) Eliminate redundant FDs, resulting in a canonical cover Fc of F (2) Create a relation Ri = XY for each FD $X \rightarrow Y$ in Fc (3) If the key K of R does not occur in any relation Ri, create one more relation Ri=K.

Comparison of BCNF and 3NF

Normalization is a method that removes **redundancy** from a relation thereby minimizing the insertion, deletion and update anomalies that degrade the performance of databases. In this article, we will differentiate among two higher normal forms i.e. 3NF and BCNF.



BASIS FOR COMPARISON	**3NF**	**BCNF**
Concept	No non-prime attribute must be transitively dependent on the Candidate key.	For any trivial dependency in a relation R say X->Y, X should be a super key of relation R.
Dependency	3NF can be obtained without sacrificing all dependencies.	Dependencies may not be preserved in BCNF.
Decomposition	Lossless decomposition can be achieved in 3NF.	Lossless decomposition is hard to achieve in BCNF.

6. Decomposition using Multivalued Dependencies

➢ <u>**Multivalued Dependencies**</u>

A **Multivalued Dependency** is a full constraint between two sets of attributes in a relation. With an aim to provide the best possible help for the students in Multivalued Dependencies Assignment help, we have implemented the latest technology and modern method of teaching tools, so that the students will not only be helped in completing their projects but will also gain an in-depth knowledge of the subject which can ultimately help them in their academics. Multivalued dependencies occur when the presence of one or more rows in a table implies the presence of one or more other rows in that same table.

In contrast to the functional dependency, the **multivalued dependency** requires that certain tuples be present in a relation. Therefore, a multivalued dependency is a special case of tuple-generating dependency. The multivalued dependency plays a role in the 4NF database normalization.

A multivalued dependency is a special case of a join dependency, with only two sets of values involved, i.e. it is a binary join dependency.

	A	**B**	**R- α – β**
t1	$a_1 \ldots a_i$	$a_{i+1} \ldots a_j$	$a_{j+1} \ldots a_n$
t2	$a_1 \ldots a_i$	$b_{i+1} \ldots b_j$	$b_{j+1} \ldots b_n$
t3	$a_1 \ldots a_i$	$a_{i+1} .. a_j$	$b_{j+1} \ldots b_n$
t4	$a_1 \ldots a_i$	$b_{i+1} \ldots b_j$	$a_{j+1} \ldots a_n$

Tabular representation of **$\alpha \rightarrow \beta$.**

t1[α] = t2 [α] =t3 [α] = t4=[α]
t3 [β]= t1[β]
t3[R- β]= t2[R- β]
t4[β]=t2[β]
t4[R- β] =t1[R- β]

➢ Fourth Normal Form

Fourth normal form (4NF) is a level of database normalization where there are no non-trivial multivalued dependencies other than a candidate key. It builds on the first three normal forms (1NF, 2NF and 3NF) and the Boyce-Codd Normal Form (BCNF). It states that, in addition to a database meeting the requirements of BCNF, it must not contain more than one multivalued dependency.

Rules for 4th Normal Form

For a table to satisfy the Fourth Normal Form, it should satisfy the following two conditions:

1. It should be in the **Boyce-Codd Normal Form**.
2. And, the table should not have any **Multi-valued Dependency**.

A table with a multivalued dependency violates the normalization standard of Fourth Normal Form (4NK) because it creates unnecessary redundancies and can contribute to inconsistent data. To bring this up to 4NF, it is necessary to break this information into two tables.

Example – Consider the database table of a class whaich has two relations R1 contains student ID(SID) and student name (SNAME) and R2 contains course id(CID) and course name (CNAME).

Table – R1(SID, SNAME)

SID	SNAME
S1	A
S2	B

➢ **4NF Decomposition**

Consider relation R(A,B,C,D,E) with multivalued dependencies:
A -» B, B -» D and no functional dependencies. Suppose we decompose R into 4th Normal Form. Depending on the order in which we deal with 4NF violations, we can get different final decompositions. Which one of the following relation schemas could be in the final 4NF decomposition.

7. More Normal Forms

The fourth Normal form is by means the “ultimate’ normal form. As we saw earlier, multivalued dependencies help us understand and tackle some forms of repetition. There are types of constraints called join dependencies that generalize multivalued dependencies, and lead to another normal form called project-join normal form(PJNF).

There is a class of even more general constraints that leads to a normal form called domain-key normal form (DKNF).