ST. XAVIER’S COLLEGE

(Affiliated to Tribhuvan University)

Maitighar, Kathmandu

****

**Database Management System**

Theory Assignment # 10

**SUBMITTED BY:**

Dwarika Shiwakoti

013BSCCSIT020

**SUBMITTED TO:**

|  |  |
| --- | --- |
| **Er. Sanjay Kr. Yadav**  **Lecturer** |  |

Submission Date: 8th October 2015

1. FUNCTIONAL DEPENDENCIES

1.1 BASIC CONCEPTS

Functional dependency is a relationship that exists when one attribute uniquely determines another attribute.

Functional dependency in a database serves as a constraint between two sets of attributes. Defining functional dependency is an important part of relational database design and contributes to aspect normalization.

Functional dependency (FD) is a set of constraints between two attributes in a relation. Functional dependency says that if two tuples have same values for attributes A1, A2,..., An, then those two tuples must have to have same values for attributes B1, B2, ..., Bn.

Functional dependency is represented by an arrow sign (→) that is, X→Y, where X functionally determines Y. The left-hand side attributes determine the values of attributes on the right-hand side.

* Functional dependencies are a constraint on the set of legal relations in a database.
* They allow us to express facts about the real world we are modeling.
* The notion generalizes the idea of a superkey.
* Let and .
* Then the functional dependency holds on R if in any legal relation r(R), for all pairs of tuples and in r such that , it is also the case that .
* Using this notation, we say K is a superkey of R if .
* In other words, K is a superkey of R if, whenever , then (and thus ).

1.2 CLOSURE OF A SET OF FUNCTIONAL DEPENDENCIES

* We need to consider all functional dependencies that hold. Given a set F of functional dependencies, we can prove that certain other ones also hold. We say these ones are logically implied by F.
* Suppose we are given a relation scheme R=(A,B,C,G,H,I), and the set of functional dependencies:

A tex2html_wrap_inline1090 B

A tex2html_wrap_inline1090 C

CG tex2html_wrap_inline1090 H

CG tex2html_wrap_inline1090 I

B tex2html_wrap_inline1090 H

Then the functional dependency tex2html_wrap_inline1194 is logically implied.

To see why, let tex2html_wrap_inline940 and tex2html_wrap_inline946 be tuples such that

tex2html_wrap_inline1200

As we are given A tex2html_wrap_inline1090 B , it follows that we must also have

tex2html_wrap_inline1204

Further, since we also have B tex2html_wrap_inline1090 H , we must also have

tex2html_wrap_inline1208

Thus, whenever two tuples have the same value on A, they must also have the same value on H, and we can say that A tex2html_wrap_inline1090 H .

* The closure of a set F of functional dependencies is the set of all functional dependencies logically implied by F.
* We denote the closure of F by tex2html_wrap_inline1222 .
* To compute tex2html_wrap_inline1222 , we can use some rules of inference called Armstrong's Axioms:
* Reflexivity rule: if tex2html_wrap_inline958 is a set of attributes and tex2html_wrap_inline1158 , then tex2html_wrap_inline1058 holds.
* Augmentation rule: if tex2html_wrap_inline1058 holds, and tex2html_wrap_inline1234 is a set of attributes, then tex2html_wrap_inline1236 holds.
* Transitivity rule: if tex2html_wrap_inline1058 holds, and tex2html_wrap_inline1240 holds, then tex2html_wrap_inline1242 holds.
* These rules are sound because they do not generate any incorrect functional dependencies. They are also complete as they generate all of tex2html_wrap_inline1222 .
* To make life easier we can use some additional rules, derivable from Armstrong's Axioms:
* Union rule: if tex2html_wrap_inline1058 and tex2html_wrap_inline1242 , then tex2html_wrap_inline1250 holds.
* Decomposition rule: if tex2html_wrap_inline1250 holds, then tex2html_wrap_inline1058 and tex2html_wrap_inline1242 both hold.
* Pseudo transitivity rule: if tex2html_wrap_inline1058 holds, and tex2html_wrap_inline1260 holds, then tex2html_wrap_inline1262 holds.

Applying these rules to the scheme and set F mentioned above, we can derive the following:

* A tex2html_wrap_inline1090 H, as we saw by the transitivity rule.
* CG tex2html_wrap_inline1090 HI by the union rule.
* AG tex2html_wrap_inline1090 I by several steps:
  + Note that A tex2html_wrap_inline1090 C holds.
  + Then AG tex2html_wrap_inline1090 CG , by the augmentation rule.
  + Now by transitivity, AG tex2html_wrap_inline1090 I .

**2. DECOMPOSITION**

Decomposition means replacing a relation with a set of smaller relations.

**2.1. Lossless – Join Decomposition**

Let R be a relation schema.

Let F be a set of functional dependencies on R.

Let and form a decomposition of R.

Why is this true? Simply put, it ensures that the attributes involved in the natural join ( ) are a candidate key for at least one of the two relations.

This ensures that we can never get the situation where spurious tuples are generated, as for any value on the join attributes there will be a unique tuple in one of the relations.

**2.2 DEPENDENCY PRESERVATION**

A desirable property in database design is dependency preservation. We would like to check easily that updates to the database do not result in illegal relations being created. It would be nice if our design allowed us to check updates without having to compute natural joins

Let Fibe the set of dependencies F+that includes only attributes in Ri

• A decomposition is dependency preserving, if (F1∪F2∪…∪Fn)+= F+

• If it is not, then checking updates for violation of functional dependencies may require computing joins, which is expensive

Testing Dependency Preservation

• To check if a dependency α→βis preserved in a decomposition of Rinto R1, R2, …, Rn

result = α

repeat

for each Ri in the decomposition t= (result ∩Ri)+ ∩Ri, result = result ∪t

until result does not change

• If resultcontains all attributes in β, then the functional dependency

α→βis preserved

• We apply the test on all dependencies in Fto check if a decomposition is dependency preserving

• This procedure takes polynomial time

ComputeF+and(F1∪F2∪…∪Fn)+requires exponential time

Example

• R = (A, B, C )

F = {A →B, B →C}

Key = {A}

• R is not in BCNF

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