DBMS

(DATABASE MANAGEMENT SYSTEM)

LESSON 4

Normalisation: Divided into 4 modules

1. Functional dependency.
2. Candidate key.
3. Decomposition.
4. Normalisation.

Functional Dependency:

A functional dependency X->Y in a relation holds if two tuples having same value for X also have same value for Y i.e X uniquely determines Y.

A functional dependency A->B in a relation holds if two tuples having same value of attribute A also have same value for attribute B. For Example, in relation STUDENT shown in table ,

Functional Dependencies

STUD_NO->STUD_NAME, STUD_NO->STUD_PHONE hold

but

STUD_NAME->STUD_ADDR do not hold

STUDENT

STUD_NO	STUD_NAME	STUD_PHONE	STUD_STATE	STUD_COUNTRY	STUD_AGE
1	RAM	9716271721	Haryana	India	20
2	RAM	9898291281	Punjab	India	19
3	SUJIT	7898291981	Rajsthan	India	18
4	SURESH		Punjab	India	21

Table 1

How to find functional dependencies for a relation?

Functional Dependencies in a relation are dependent on the domain of the relation. Consider the STUDENT relation given in Table 1.

- We know that STUD_NO is unique for each student. So STUD_NO->STUD_NAME, STUD_NO->STUD_PHONE, STUD_NO->STUD_STATE, STUD_NO->STUD_COUNTRY and STUD_NO -> STUD_AGE all will be true.
- Similarly, STUD_STATE->STUD_COUNTRY will be true as if two records have same STUD_STATE, they will have same STUD_COUNTRY as well.
- For relation STUDENT_COURSE, COURSE_NO->COURSE_NAME will be true as two records with same COURSE_NO will have same COURSE_NAME.

Functional Dependency Set: Functional Dependency set or FD set of a relation is the set of all FDs present in the relation. For Example, FD set for relation STUDENT shown in table 1 is:

{ STUD_NO->STUD_NAME, STUD_NO->STUD_PHONE, STUD_NO->STUD_STATE, STUD_NO->STUD_COUNTRY,

STUD_NO -> STUD_AGE, STUD_STATE->STUD_COUNTRY }

Attribute Closure: Attribute closure of an attribute set can be defined as set of attributes which can be functionally determined from it.

How to find attribute closure of an attribute set?

To find attribute closure of an attribute set:

- Add elements of attribute set to the result set.
- Recursively add elements to the result set which can be functionally determined from the elements of the result set.

Using FD set of table , attribute closure can be determined as:

(STUD_NO)+ = {STUD_NO, STUD_NAME, STUD_PHONE, STUD_STATE, STUD_COUNTRY, STUD_AGE}

(STUD_STATE)+ = {STUD_STATE, STUD_COUNTRY}

How to find Candidate Keys and Super Keys using Attribute Closure?

- If attribute closure of an attribute set contains all attributes of relation, the attribute set will be super key of the relation.
- If no subset of this attribute set can functionally determine all attributes of the relation, the set will be candidate key as well. For Example, using FD set of table 1,

(STUD_NO, STUD_NAME)+ = {STUD_NO, STUD_NAME, STUD_PHONE, STUD_STATE, STUD_COUNTRY, STUD_AGE}

(STUD_NO)+ = {STUD_NO, STUD_NAME, STUD_PHONE, STUD_STATE, STUD_COUNTRY, STUD_AGE}

(STUD_NO, STUD_NAME) will be super key but not candidate key because its subset (STUD_NO)+ is equal to all attributes of the relation. So, STUD_NO will be a candidate key.

How to check whether an FD can be derived from a given FD set?

To check whether an FD A->B can be derived from an FD set F,

1. Find (A)+ using FD set F.
2. If B is subset of (A)+, then A->B is true else not true.

GATE Question: In a schema with attributes A, B, C, D and E following set of functional dependencies are given

{A -> B, A -> C, CD -> E, B -> D, E -> A}

Which of the following functional dependencies is NOT implied by the above set? (GATE IT 2005)

A. CD -> AC

B. BD -> CD

C. BC -> CD

D. AC -> BC

Answer: Using FD set given in question,

(CD)+ = {CDEAB} which means CD -> AC also holds true.

(BD)+ = {BD} which means BD -> CD can't hold true. So this FD is no implied in FD set. So (B) is the required option.

Others can be checked in the same way.

Prime and non-prime attributes:

Attributes which are parts of any candidate key of relation are called as prime attribute, others are non-prime attributes. For Example, STUD_NO in STUDENT relation is prime attribute, others are non-prime attribute.

Finding Attribute Closure and Candidate Keys using Functional Dependencies:

In EMPLOYEE relation given in Table 1,

- FD **E-ID->E-NAME** holds because for each E-ID, there is a unique value of E-NAME.
- FD **E-ID->E-CITY** and **E-CITY->E-STATE** also holds.
- FD E-NAME->E-ID **does not hold** because E-NAME 'John' is not uniquely determining E-ID. There are 2 E-IDs corresponding to John (E001 and E003).

EMPLOYEE

E-ID	**E-NAME**	**E-CITY**	**E-STATE**
E001	**John**	**Delhi**	**Delhi**

E002	**Mary**	**Delhi**	**Delhi**
E003	**John**	**Noida**	**U.P.**

The FD set for EMPLOYEE relation given in Table 1 are:

{E-ID->E-NAME, E-ID->E-CITY, E-ID->E-STATE, E-CITY->E-STATE}

Trivial versus Non-Trivial Functional Dependency: A trivial functional dependency is the one which will always hold in a relation.

X->Y will always hold if X ⊇ Y

In the example given above, **E-ID, E-NAME->E-ID** is a trivial functional dependency and will always hold because {E-ID,E-NAME} ⊃ {E-ID}. You can also see from the table that for each value of {E-ID, E-NAME}, value of E-ID is unique, so {E-ID, E-NAME} functionally determines E-ID.

If a functional dependency is not trivial, it is called **Non-Trivial Functional Dependency**. Non-Trivial functional dependency may or may not hold in a relation. e.g; **E-ID->E-NAME** is a non-trivial functional dependency which holds in the above relation.

Properties of Functional Dependencies

Let *X*, *Y*, and *Z* are sets of attributes in a relation *R*. There are several properties of functional dependencies which always hold in R also known as Armstrong Axioms.

1. **Reflexivity**: If *Y* is a subset of *X*, then $X \rightarrow Y$. e.g.; Let X represents {E-ID, E-NAME} and Y represents {E-ID}. {E-ID, E-NAME}->E-ID is true for the relation.
2. **Augmentation**: If $X \rightarrow Y$, then $XZ \rightarrow YZ$. e.g.; Let X represents {E-ID}, Y represents {E-NAME} and Z represents {E-CITY}. As {E-ID}->E-NAME is true for the relation, so { E-ID,E-CITY}->{E-NAME,E-CITY} will also be true.

3. **Transitivity**: If $X \rightarrow Y$ and $Y \rightarrow Z$, then $X \rightarrow Z$. e.g.; Let X represents {E-ID}, Y represents {E-CITY} and Z represents {E-STATE}. As {E-ID} ->{E-CITY} and {E-CITY}->{E-STATE} is true for the relation, so { E-ID }->{E-STATE} will also be true.
4. **Attribute Closure:** The set of attributes that are functionally dependent on the attribute A is called Attribute Closure of A and it can be represented as A^+.

Steps to Find the Attribute Closure of A

Q. Given FD set of a Relation R, The attribute closure set S be the set of A

1. Add A to S.
2. Recursively add attributes which can be functionally determined from attributes of the set S until done.

From Table 1, FDs are

Given R(E-ID, E-NAME, E-CITY, E-STATE)

FDs = {E-ID->E-NAME, E-ID->E-CITY, E-ID->E-STATE,
E-CITY->E-STATE }

The attribute closure of E-ID can be calculated as:

1. Add E-ID to the set {E-ID}
2. Add Attributes which can be derived from any attribute of set. In this case, E-NAME and E-CITY, E-STATE can be derived from E-ID. So these are also a part of closure.
3. As there is one other attribute remaining in relation to be derived from E-ID. So result is:

$(\text{E-ID})_+$ = {E-ID, E-NAME, E-CITY, E-STATE }

Similarly,

$(\text{E-NAME})_+$ = {E-NAME}

$(\text{E-CITY})_+$ = {E-CITY, E_STATE}

Q. Find the attribute closures of given FDs R(ABCDE) = {AB->C, B->D, C->E, D->A} To find $(B)^+$,we will add attribute in set using various FD which has been shown in table below.

Attributes Added in Closure	FD used
{B}	Triviality
{B,D}	B->D
{B,D,A}	D->A

{B,D,A,C}	AB->C
{B,D,A,C,E}	C->E

- We can find $(C, D)_+$ by adding C and D into the set (triviality) and then E using(C->E) and then A using (D->A) and set becomes.

  $(C,D)_+$ = {C,D,E,A}
- Similarly we can find $(B,C)_+$ by adding B and C into the set (triviality) and then D using (B->D) and then E using (C->E) and then A using (D->A) and set becomes

  $(B,C)_+$ ={B,C,D,E,A}

Candidate Key: Candidate Key is **minimal set of attributes** of a relation which can be used to identify a tuple uniquely. For Example, each tuple of EMPLOYEE relation given in Table 1 can be uniquely identified by **E-ID** and it is minimal as well. So it will be Candidate key of the relation.

A candidate key may or may not be a primary key.

Super Key: Super Key is **set of attributes** of a relation which can be used to identify a tuple uniquely.For Example, each tuple of EMPLOYEE relation given in Table 1 can be uniquely identified by **E-ID or (E-ID, E-NAME) or (E-ID, E-CITY) or (E-ID, E-STATE) or (E_ID, E-NAME, E-STATE)** etc. So all of these are super keys of EMPLOYEE relation.

Note: A candidate key is always a super key but vice versa is not true.

Q. Finding Candidate Keys and Super Keys of a Relation using FD set

The **set of attributes** whose attribute closure is set of all attributes of relation is called super key of relation. For Example, the EMPLOYEE relation shown in Table 1 has following FD set. **{E-ID->E-NAME, E-ID->E-CITY, E-ID->E-STATE, E-CITY->E-STATE}** Let us calculate attribute closure of different set of attributes:

(E-ID)$_+$ = {E-ID, E-NAME,E-CITY,E-STATE}
(E-ID,E-NAME)$_+$ = {E-ID, E-NAME,E-CITY,E-STATE}
(E-ID,E-CITY)$_+$ = {E-ID, E-NAME,E-CITY,E-STATE}
(E-ID,E-STATE)$_+$ = {E-ID, E-NAME,E-CITY,E-STATE}
(E-ID,E-CITY,E-STATE)$_+$ = {E-ID, E-NAME,E-CITY,E-STATE}
(E-NAME)$_+$ = {E-NAME}

(E-CITY)$_+$ = {E-CITY,E-STATE}

As (E-ID)$_+$, (E-ID, E-NAME)$_+$, (E-ID, E-CITY)$_+$, (E-ID, E-STATE)$_+$, (E-ID, E-CITY, E-STATE)$_+$ give set of all attributes of relation EMPLOYEE. So all of these are super keys of relation.

The **minimal set of attributes** whose attribute closure is set of all attributes of relation is called candidate key of relation. As shown above, (E-ID)$_+$ is set of all attributes of relation and it is minimal. So E-ID will be candidate key. On the other hand (E-ID, E-NAME)$_+$ also is set of all attributes but it is not minimal because its subset (E-ID)$_+$ is equal to set of all attributes. So (E-ID, E-NAME) is not a candidate key.

Equivalence of Functional Dependencies:

Given a Relation with different FD sets for that relation, we have to find out whether one FD set is subset of other or both are equal.

How to find relationship between two FD sets?

Let FD1 and FD2 are two FD sets for a relation R.

1. If all FDs of FD1 can be derived from FDs present in FD2, we can say that FD2 ⊃ FD1.
2. If all FDs of FD2 can be derived from FDs present in FD1, we can say that FD1 ⊃ FD2.
3. If 1 and 2 both are true, FD1=FD2.

DECOMPOSITION:

Dependency Preservation

A Decomposition D = { R1, R2, R3….Rn } of R is dependency preserving wrt a set F of Functional dependency if

(F1 ∪ F2 ∪ … ∪ Fm)+ = F+.
Consider a relation R
R ---> F{...with some functional dependency(FD)....}

R is decomposed or divided into R1 with FD { f1 } and R2 with { f2 }, then
there can be three cases:

f1 U f2 = F -----> Decomposition is dependency preserving.
f1 U f2 is a subset of F -----> Not Dependency preserving.
f1 U f2 is a super set of F -----> This case is not possible.

Problem: Let a relation R (A, B, C, D) and functional dependency {AB –> C, C –> D, D –> A}. Relation R is decomposed into R1(A, B, C) and R2(C, D). Check whether decomposition is dependency preserving or not.

Solution:

R1(A, B, C) and R2(C, D)

Let us find closure of F1 and F2
To find closure of F1, consider all combination of
ABC. i.e., find closure of A, B, C, AB, BC and AC
Note ABC is not considered as it is always ABC

closure(A) = { A } // Trivial

closure(B) = { B } // Trivial
closure(C) = {C, A, D} but D can't be in closure as D is not present R1.
= {C, A}
C--> A   // Removing C from right side as it is trivial attribute

closure(AB) = {A, B, C, D}
= {A, B, C}
AB --> C // Removing AB from right side as these are trivial attributes

closure(BC) = {B, C, D, A}
= {A, B, C}
BC --> A // Removing BC from right side as these are trivial attributes

closure(AC) = {A, C, D}
AC --> D // Removing AC from right side as these are trivial attributes

F1 {C--> A, AB --> C, BC --> A}.
Similarly F2 { C--> D }

In the original Relation Dependency { AB --> C , C --> D , D --> A}.
AB --> C is present in F1.
C --> D is present in F2.
D --> A is not preserved.

F1 U F2 is a subset of F. So **given decomposition is not dependency preserving**

Lossless Decomposition in DBMS:

Lossless join decomposition is a decomposition of a relation R into relations R1,R2 such that if we perform natural join of two smaller relations it will return the original relation. This is effective in removing redundancy from databases while preserving the original data..

In other words by lossless decomposition it becomes feasible to reconstruct the relation R from decomposed tables R1 and R2 by using Joins.

In Lossless Decomposition we select the common element and the criteria for selecting common element is that the common element must be a candidate key or super key in either of relation R1,R2 or both.

Decomposition of a relation R into R1 and R2 is a lossless-join decomposition if at least one of the following functional dependencies are in F+ (Closure of functional dependencies)

$R1 \cap R2 \rightarrow R1$

OR

$R1 \cap R2 \rightarrow R2$