Functional Dependencies, Superkeys and Candidate Keys

Functional Dependencies

Definition: X -> A

A is functionally dependent upon X

Whenever two tuples agree on all the attributes of X, then they must also agree on attribute A.
A *many to 1* or *1 to 1* mapping must exist as in function theory (not *1 to many*)

Typically some relation scheme will have various dependencies between the attributes. By this we mean that as one attribute varies so does the other.

Consider the following relation:

Name	Location	Part
Smith	London	P1
Smith	London	P2
Smith	London	P3
Jones	York	P1

Assume that this relation shows names of suppliers with their location and the name of the part they supply.

As each Supplier seems to have only has one Location, we would say that Location is functionally dependent upon Name.

Name->Location

Are there other functional dependencies in this relation?

Not Name->Part
Not Location -> Part
Not Name,Location -> Part

Name,Part ->Location is also true.

Functional Dependencies are important as they give us a mathematical tool for explaining the process of Normalization, which is vital for redesigning database schemas when the original design has certain flaws.

Consider the following example

Drinkers (name, address, beersLiked, manufacturer, favoriteBeer)

Reasonable functional dependencies to assert:

name -> address
name -> favoriteBeer
beersLiked -> manufacturer

These happen to imply the underlined key, but the FD's give more detail than the mere assertion of a key

Note that we could combine the first two functional dependencies to give

name ->address, favoriteBeer

Shorthand: combine FD's with common left side by concatenating their right sides.

Superkeys and Candidate Keys

A key functionally determines all the attributes in the relation – in the previous example

name, beersLiked -> address, favoriteBeer, manufacturer

When FD's are not of the form Key -> other attribute(s), then there is typically an attempt to “cram" too much into one relation.

Sometimes, several attributes jointly determine another attribute, although neither does by itself. Consider

beer, bar -> price

Definition: K is a Candidate key for relation R if:

1. K -> all attributes of R.
2. For no proper subset of K is (1) true.

If K satisfies only (1), then K is a superkey.

Consider the previous relation

Drinkers (name, address, beersLiked, manufacturer, favoriteBeer)

{name, beersLiked} FD's all attributes which shows that {name, beersLiked} is a superkey.

To see if this is a candidate key we must determine if a subset of the superkey FD’s all the attributes. There are only two possible alternatives name and beersLiked:

name -> beersLiked is false, so name not a superkey

beersLiked ->name is also false, so beersLiked not a superkey

Thus, {name, beersLiked} is a Superkey and a Candidate key.

Neither name nor beersLiked is on the right of any observed FD, so they must be part of any superkey.

We could define a relation schema by simply giving a single key K. Then the only FD's asserted are that K -> A for every attribute A. K is then the only key for those FD's, according to the formal definition of a “key.”

More frequently we assert some FD's and deduce one or more keys by the formal definition. Often there can be several candidate keys – consider an employee relation with National Insurance# and employee ID attributes.

FD's often come from some physical characteristic of the attributes involved e.g. “no two courses can meet in the same room at the same time" yields room, time -> course.

When we talk about improving relational designs, we often need to ask, “Does this FD hold in this relation?”

Proving a Key holds

Given a relation and a set of functional dependencies, how can we prove that a certain combination of attributes is a Key? This can be achieved be using *Armstrong’s Axioms* to calculate the *Closure* of an attribute set.

Armstrong’s Axioms allow new inferences to be deduced from a set of FD’s

Primitive set

A1: If Y is a subset of X then X->Y	*Reflexivity*
A2: If X->Y then XZ->YZ	*Augmentation*
A3: If X->Y and YW->Z then XW->Z	*Pseudo-Transitivity*

Consequences of the primitive set

A4: If X->Y and Y->Z then X->Z	*Transitivity*
A5: If X->Y and X->Z then X->YZ	*Union*
A6: If X->YZ then X->Y and X->Z	*Decomposition*

Example
If name->town and town->postcode then A4 suggests name->postcode

The *Closure* of an attribute set Y is designated Y^+ and is the entire set of functional dependencies that Y implies. If we determine the Closure of an attribute, and find that it functionally determines the entire set of attributes within a relation, then we have a Superkey and if minimal a Candidate key.

Example
Find A^+ under S= {A->B, B->C, AC->D} on R=(ABCD)

Start with	A^+={A}	as A->A must hold
Then	A^+={A} U {B}= {AB}	from A->B
Then	A^+={AB} U {C} = {ABC}	from B->C
Then	A^+={ABC} U {D} = {ABCD}	from AC->D

Hence A^+={ABCD}, so A is a superkey and is also a candidate key (assuming A is minimal)

Example
Determine if AC is a superkey or candidate key under S=(A -> B, BC -> D} on R=(ABCD)

Start with	AC^+={AC}	
Then	AC^+={AC} U {B} = {ABC}	from A->B
Then	AC^+={ABC} U {D} = {ABCD}	from BC->D

Hence AC^+ is a superkey.

Is it minimal? We can test for this by calculating A^+ and C^+

$A^+=\{A\}$
$A^+=\{A\} \cup \{B\} = \{AB\}$

At this point saturation is reached – can't derive C or D

$C^+ = \{C\}$

Hence AC is a Superkey and a Candidate key.

Exercises

One
Determine if { SSNo, ProjNo } is a Superkey or Candidate key for R, where
R=(SSNo, ProjNo, Hours, Ename, ProjName, ProjLoc) under
S={SSNo->Ename, ProjNo->ProjName,ProjLoc, SSNo,ProjNo->Hours}

Two
Given R=(ABCDEF) and S={A->BC, E->CF, B->E, CD->EF}
Determine if AB is a Superkey or Candidate key.