Design by synthesis

Colm O'Riordan

School of Computer Science

Design by Synthesis - Background

Typically, we have the relation R and a set of functional dependencies F.

We wish to create a decomposition $D = R_1, R_2, ...R_m$.

Clearly, all attributes of R must occur in a least one schema R_i, i.e.,

$$U_{i=1}^{m} R_i = R$$

This is known as the **attribute preservation** constraint.

Functional dependencies

A functional dependency is a constraint between two sets of attributes. A functional dependency $X \rightarrow Y$ exists if for all tuples t_1 and t_2, if $t_1[X] = t_2[X]$, then $t_1[Y] = t_2[Y]$.

Usually only specify the obvious functional dependencies. There may exist many more.

Given a set of functional dependencies F, the closure of F (denoted F^+) refers to all dependencies that can be derived from F.

A set of inference rules exist, that allow us to deduce or infer all functional dependencies from a given initial set.

Known as Armstrong's Axioms

Armstrong's Axioms

- Reflexivity: if $X \supseteq Y$, then $X \rightarrow Y$
- Augmentation: if $X \rightarrow Y$, then $XZ \rightarrow YZ$
- Transitivity: if $X \rightarrow Y$, $Y \rightarrow Z$, then $X \rightarrow Z$
- Projectivity: if $X \rightarrow YZ$, then $X \rightarrow Z$
- Additivity: if $X \rightarrow Y$, $X \rightarrow Z$, then $X \rightarrow YZ$
- Pseudo-transitivity: if $X \rightarrow Y$, $WY \rightarrow Z$, then $WX \rightarrow Z$

The first three rules have be shown to be *sound* and *complete*.

Sound

Given a set F specified on a relation R, any dependency we can infer from F using the first three rules, holds for every state r of R that satisfies the dependencies in F.

Complete

We can use the first three rules repeatedly to infer all possible dependencies that be can be inferred from F.

For any set of attributes A, we can infer A^+, the set of attributes that are functionally determined by A given a set of functional dependencies.

Algorithm to determine the closure of A under F

A^+ := A;
repeat
$oldA+$:= A^+
for each functional dependency $Y \rightarrow Z \in F$ do
        if $A^+ \supseteq Y$, then
                A^+ := $A^+ \cup Z$
until (A^+ == $oldA^+$)

Cover Sets

A set of functional dependencies, F, *covers* a set of functional dependencies E, if every functional dependency in E is in F^+

Equivalence

Two set of functional dependencies, E and F are equivalent is $E^+ = F^+$

We can check if F covers E by calculating A^+ with respect to F for each functional dependency $A \rightarrow B$ and then checking that A^+ includes the attributes of B

Minimal Cover Sets

A set of functional dependencies, F, is minimal if:

- Every functional dependency in F has a single attribute for its right hand side.
- We cannot remove any dependency from F and maintain a set of dependencies equivalent to F.
- We cannot replace any dependency $X \rightarrow A$ with a dependency $Y \rightarrow A$ where $Y \subset X$, and still maintain a set of dependencies equivalent to F.

All functional dependencies $X \rightarrow Y$, specified in F, should exist in one of the schema R_i, or should be inferrable from the dependencies in R_i.

This is known as the **dependency preservation** constraint.

Each functional dependency specifies some constraint; if the dependency is absent then some desired constraint is also absent.

If a functional dependency is absent then we must enforce the constraint in some other manner. This can be inefficient.

Given F and R, the *projection* of F on R_i, denoted $\pi_{R_i}(F)$ where R_i is a subset of R, is the set $X \rightarrow Y$ in F^+ such that attributes $X \cup Y \in R_i$.

A decomposition of R is dependency-preserving if $((\pi_{R_1}(F)) \cup \ldots \cup (\pi_{R_m}(F)))^+ = F^+$.

Theorem:

It is always possible to find a decomposition D with respect to F such that:

1. the decomposition is dependency-preserving
2. all R_i in D are in 3NF

We can always guarantee a dependency-preserving decomposition to 3NF.
Algorithm:

1. Find a minimal cover set G for F.
2. for each left hand side X of a functional dependency in G, create a relation $X \cup A_1 \cup A_2 \ldots A_m$ in D, where $X \rightarrow A_1 X \rightarrow A_2 \ldots$ are the only dependencies in G with X as a left hand side.
3. Group any remaining attributes into a single relation.

Lossless joins

Consider the following relation:
EMPPROJ: ssn, pnumber, hours, ename, pname, plocation

and its decomposition to:

EMPPROJ1: ename, plocation
EMPLOCAN: ssn, pno, hrs, pname, plocation

If we perform a natural join on these relations, we may generate spurious tuples.

Lossless Joins

Also known as *non-additive joins*.

When a natural join is issued against relations, no spurious tuples should be generated.

A decomposition $D = \{R_1, R_2, \ldots R_n\}$ of R has the lossless join property wrt to F on R if for every instance r the following holds:

$$\bowtie (\pi_{R_1}(r), \ldots \pi_{R_m}(r)) = r)$$

We can automate procedure for testing for lossless property.

Can also automate the decomposition of R into $R_1, \ldots R_m$ such that it possesses the lossless join property.

A decomposition $D = \{R_1, R_2\}$ has the lossless property iff:

- functional dependency $(R1 \cap R_2) \rightarrow \{R_1 - R_2\}$ is in F^+
- or functional dependency $(R1 \cap R_2) \rightarrow \{R_2 - R_1\}$ is in F^+

Furthermore, if a decomposition has the lossless property, and we decompose one of R_i such that this also is a lossless decomposition, then replacing that decomposition of R_i in the original decomposition will result in a lossless decomposition.

Algorithm to decompose to BCNF

Let $D = R$
while there is a schema B in D that violates BCNF do
  choose B
  find functional dependency $(X \rightarrow Y)$ that violates BCNF
  replace B with
   $(B - Y)$ and $(X \cup Y)$

So, we guarantee a decomposition such that:

- all attributes are preserved
- lossless join property is enforced
- all R_i are in BCNF

It is not always possible to decompose R into a set of R_i such that all R_i satisfy BCNF and properties of lossless joins and dependency preservation are maintained.

We can guarantee a decomposition such that:

- all attributes are preserved
- all relations are in 3NF
- all functional dependencies are maintained
- the lossless join property is maintained

Algorithm: Finding a key for relation schema R

set K := R.
For each attribute A $\in$ K.
compute $(K - A)^+$ wrt to set of functional dependencies.
if e $(K - A)^+$ contains all the attributes in R, the set K := K - {A}.

Summary

Given a set of functional dependencies F, we can develop a minimal cover set.
Using this we can decompose R into a set of relations such that all attributes are preserved, all functional dependencies are preserved, the decomposition has the lossless join property and all relations are in 3NF.

Advantages

- Provides a good database design.
- Can be automated.

Disadvantages

- Oftentimes, numerous good designs are possible.