Decomposition Using Functional Dependencies

legal instance

There are usually a variety of constraints (rules) on the data in the real world. An instance of a relation that satisfies all such real-world constraints is called a legal instance of the relation

superkey

a set of one or more attributes that, taken collectively, allows us to identify uniquely a tuple in the relation.Given r(R), a subset K of R is a superkey of r(R) if, in any legal instance of r(R), for all pairs t 1 and t 2 of tuples in the instance of r if t 1 ≠ t 2 ,then t 1 [K] ≠ t 2 [K]. That is, no two tuples in any legal instance of relation r(R) may have the same value on attribute set K. 3 If no two tuples in r have the same value on K, then a K-value uniquely identifies a tuple in r.

functional dependency

- Given an instance of $r(R)$, we say that the instance **satisfies** the **functional dependency** $\alpha \rightarrow \beta$ if for all pairs of tuples t_1 and t_2 in the instance such that $t_1[\alpha] = t_2[\alpha]$, it is also the case that $t_1[\beta] = t_2[\beta]$.
- We say that the functional dependency $\alpha \rightarrow \beta$ **holds** on schema $r(R)$ if, every legal instance of $r(R)$ satisfies the functional dependency.

We shall use functional dependencies in two ways:

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

A	B	C	D
a_1	b_1	c_1	d_1
a_1	b_2	c_1	d_2
a_2	b_2	c_2	d_2
a_2	b_3	c_2	d_3
a_3	b_3	c_2	d_4

Observe that A → C is satisfied
The functional dependency C → A is not satisfied,

trivial Functional Dependancy

In general, a functional dependency of the form α → β is trivial if β ⊆ α.

1)A → A
2)AB → A
3)AB-->B
ALL ARE CALLED TRIVIAL

1)A-->C
C IS NOT A SUBSET OF A,SO THIS IS NONTRIVIAL

....

It is important to realize that an instance of a relation may satisfy some functional dependencies that are not required to hold on the relation's schema. Eg room number → capacity

building	*room_number*	*capacity*
Packard	101	500
Painter	514	10
Taylor	3128	70
Watson	100	30
Watson	120	50

However, we would expect the functional dependency
building, room number → capacity   to hold on the classroom schema

inferences in FD

r(A, B, C)
if A → B and B → C hold on r
we can infer the functional dependency A → C must also hold on r

closure F +

Given $F = \{A \rightarrow B, B \rightarrow C, C \rightarrow D\}$, which of the following represents F^+ ?

- {A → B, B → C, C → D, A → BCD}
- {A → B, B → C, C → D, A → C, B → D, A → D, B → CD, A → BCD}
- {A → C, B → D, A → D}
- {A → C, B → D, A → D, A → BCD}

the set of all functional dependencies that can be inferred given the set F

Lossless Decomposition and Functional Dependencies/to check whether functional dependencies are lossless with the help of functional dependencies

use functional dependencies to show when certain decompositions are lossless

R 1 and R 2 form a lossless decomposition of R if at least one of the following functional dependencies is in F + :

- $R_1 \cap R_2 \rightarrow R_1$
- $R_1 \cap R_2 \rightarrow R_2$

if $R_1 \cap R_2$ forms a superkey for either R 1 or R 2

in dep(ID , name, salary, dept name, building, budget)

instructor (ID , name, dept name, salary)
department (dept name, building, budget)

the intersection of these two schemas, which is dept name

dept name→ dept name, building, budget, the lossless-decomposition rule is satisfied.

relation schema r(R) into $r_1(R_1)$ and $r_2(R_2)$, where $R_1 \cap R_2 \rightarrow R_1$

following SQL constraints must be imposed on the decomposed schema

- $R_1 \cap R_2$ is the primary key of r_1.
  This constraint enforces the functional dependency.
- $R_1 \cap R_2$ is a foreign key from r_2 referencing r_1.
  This constraint ensures that each tuple in r_2 has a matching tuple in r_1, without which it would not appear in the natural join of r_1 and r_2.