CS252:HACD Fundamentals of Relational Databases
Notes for Section 8: Database Design Issues, Part I

1. Cover slide

We will discover that there is more than one way of telling the same story. Certain "normal forms" have been proposed in the never-ending search for the optimal design. The study of these normal forms requires an understanding of the concepts of keys and superkeys, which we have already encountered, and two new kinds of constraint: *functional dependency* and *join dependency*.

2. A "Reducible" Relation

A predicate for WIFE_OF_HENRY_VIII:

"The first name of the *Wife#*-th wife of Henry VIII is *FirstName* and her last name is *Lastname* and *Fate* is what happened to her."

The appearances of the word "and" in this predicate indicate that it is "decomposible" into two or more simpler predicates:

1. "The first name of the *Wife#*-th wife of Henry VIII is *FirstName*."
2. "The last name of the *Wife#*-th wife of Henry VIII is *Lastname* and *Fate* is what happened to her."

The relations corresponding to predicates 1 and 2 are shown on the next slide.

3. "Decomposing" H8's Wives

Notice:

- W_FN = WIFE_OF_HENRY_VIII { Wife#, FirstName }
- W_LN_F = WIFE_OF_HENRY_VIII { Wife#, LastName, Fate }
- WIFE_OF_HENRY_VIII = W_FN **JOIN** W_LN_F

In other words, we use projection to perform the decomposition and JOIN to undo the decomposition and revert to the original.

The property by which WIFE_OF_HENRY_VIII can be decomposed this way, without any loss or gain of tuples, is known as a **join dependency**.

You've probably noticed that W_LN_F can be further decomposed. I'll come back to this point on a later slide. W_FN, however, cannot be further decomposed. We say that W_FN is an *irreducible relation.* Equivalently, we say that the relvar W_FN is in *sixth normal form* (6NF). The significance of the number 6 here will become apparent eventually.

4. Join Dependency

So a JD is a kind of constraint. The notation shown, however, is merely that used in academic treatise on the subject and is not found in any database languages.

5. A Join Dependency That Does Not Hold

Exercise: You might like to write down the result of (W_FN { Wife# }) JOIN (W_FN { FirstName }) if you need to convince yourself that it is not equal to W_FN.

6. Decomposition of W_LN_F

We have decomposed predicate 2 ("The last name of the *Wife#*-th wife of Henry VIII is *Lastname* and *Fate* is what happened to her") into:

3. "The last name of the *Wife#*-th wife of Henry VIII is *Lastname*."

4. "*Fate* is what happened to the *Wife#*-th wife of Henry VIII."

Notice that in each of W_FN, W_LN and W_F, we have preserved the key of our original relvar, WIFE_OF_HENRY_VIII. A JD always exists when a relvar has two or more nonkey attributes with respect to a given key.

7. 3-way Join Dependency

No notes.

8. Design Comparison

No notes.

9. Conclusion

Of course, if the database containing WIFE_OF_HENRY_VIII had existed during King Henry's reign in the 16th century, the fate of the current marriage would not be known. For that reason, the relvar would have had to be decomposed somehow perhaps into W_FN_LN (for first names and last names) and W_F (for fates, where known).

We note in passing that WIFE_OF_HENRY_VIII has another key, {LastName}, if we can assume that, being dead, Henry cannot possibly have any more wives. So the following JD also holds in WIFE_OF_HENRY_VIII:

* { { Lastname, Wife# }, { LastName, FirstName }, { LastName, Fate } }

10. A Not-So-Special JD

The JD here is * { { StudentId, Name }, { StudentId, CourseId } }

Note that { StudentId, CourseId } is the only key of ENROLMENT, but is not preserved in the first projection shown in the JD.

11. Splitting ENROLMENT

Note that the key {StudentId, CourseId} of ENROLMENT is retained as the key of IS_ENROLLED_ON, but is not a key of IS_CALLED. The special thing about our WIFE_OF_HENRY_VIII example was that a key of WIFE_OF_HENRY_VIII was retained as a key of each of the three relvars resulting from its decomposition.

Although {StudentId} is not a key of ENROLMENT, it *is* a key of ENROLMENT { StudentId, Name } (and therefore of the relvar IS_CALLED).

12. Functional Dependency

Here is a list of all the FDs that hold in ENROLMENT:

{ StudentId } → { StudentId} (this is a trivial FD: everything determines itself)

{ Name } → { Name }

{ CourseId } → { CourseId }

{ StudentId, CourseId } → { StudentId, CourseId }

{ StudentId, CourseId } → { StudentId }

{ StudentId, CourseId } → { CourseId }

{ StudentId, Name } → { StudentId, Name }

{ StudentId, Name } → { StudentId }

{ StudentId, Name } → { Name }

{ Name, CourseId } → { Name, CourseId }

{ Name, CourseId } → { Name }

{ Name, CourseId } → { CourseId }

{ StudentId, Name, CourseId } → { StudentId, Name, CourseId }

{ StudentId, Name, CourseId } → { StudentId, Name }

{ StudentId, Name, CourseId } → { StudentId, CourseId }

{ StudentId, Name, CourseId } → { Name, CourseId }

{ StudentId, Name, CourseId } → { StudentId }

{ StudentId, Name, CourseId } → { Name }

{ StudentId, Name, CourseId } → { CourseId }

{ StudentId } → { } (this is trivial too: everything determines nothing, so to speak)

{ Name } → { }

{ CourseId } → { }

{ StudentId, CourseId } → { }

{ StudentId, Name } → { }

{ Name, CourseId } → { }

{ StudentId, Name, CourseId } → { }

{ StudentId } → { Name } (this is the only nontrivial FD that holds in ENROLMENT)

{ StudentId, CourseId } → { Name } (this is a trivial consequence of our only nontrivial FD)

13. Anatomy of An FD

No notes.

14. FDs That Do Not Hold in ENROLMENT

No notes.

15. Theorems About FDs

That final theorem (its proof is left as an exercise for those who are into such things) is sometimes referred to in the literature as “pseudotransitivity”.

The theorems have many interesting applications. Relevant to database design is the determination of keys from a given set of FDs. Consider, for example, a relvar *R,* with attributes {a,b,c,d,e}, in

which the following FDs hold: {a,b} → {c} and {d → e}. From the general theorem we can conclude that {a,b,d} → {c,e}, from which it immediately follows that {a,b,d} → {a,b,c,d,e} (by self-determination). Therefore {a,b,d} is a superkey of *R*. As we cannot conclude that any proper subset of {a,b,d} determines both {c} and {e}, {a,b,d} must actually be a key of *R*.

16. Left Irreducibility

No notes.

17. FDs and Keys

E.F. Codd introduced the term *candidate key* to indicate candidacy (of a set of attributes) for being the primary key. We have since decided that although the concept of a primary key is useful, it is not fundamentally important. For example, SQL allows several keys to be declared for a table without any one of them being singled out as primary. In that case, any foreign key referencing such a table must include an explicit specification of the referenced columns (which in SQL default to being the columns of the primary key). But mere matters of syntax and convenience are not fundamental!

18. Normal Forms

We will study only those normal forms shown in bold on the slide (BCNF, 5NF, and 6NF). The others are really just early "mistakes" in the development of the theory, mostly attributable to E.F. Codd himself. Many textbooks still give definitions of 1NF, 2NF, 3NF, and 4NF, but they are all problematical and in one case (1NF) not even precise! Suffice it to say that BCNF covers all the cases that 2NF and 3NF were intended to cover. 4NF covers some additional cases that are rather rare but was discovered to be still incomplete and was superseded by a correction, 5NF. 5NF is sufficient to cover all cases of the kind of redundancy that normalisation addresses. 6NF is just an even stricter form that is occasionally useful.

19. Normalisation

Notice exactly how we derive a JD from a given FD. First of all, note that a JD determined by an FD is always of degree 2 (always consists of exactly two projections). One of these projections is over the attributes involved in both sides of the FD; the other is over the determinant of that FD plus any remaining attributes in *RV*, the relvar we are normalising.

20. Purposes of Normalisation

No notes.

End of Notes