CS252:HACD Fundamentals of Relational Databases
Notes for Section 9: Database Design Issues, Part II

1. Cover slide

Now at last we can look at *normal forms*. In so doing we will discover some issues that have still not been satisfactorily resolved.

2. First Normal Form (1NF)

For example, in the 5th edition (1990) of C.J. Date's *Introduction to Database Systems* we find this:

> A [relvar][1] is in *first normal form* (1NF) if and only if all underlying simple domains contain atomic values only.

The "underlying simple domains" here are just what we now call the declared types of the attributes. The trouble is, Codd did not originally give a definition of atomic; and when pressed to do so he came up with something that was not acceptable. He defined a value to be atomic if and only if it cannot be "decomposed by the DBMS", without defining "decomposed". Can a character string be decomposed into its constituent characters? people asked, and can an integer be decomposed into its prime factors?

In later editions of Date's book this definition does not appear and Date effectively defines a relvar to be in 1NF if and only if at all times its value is a relation.

3. 2NF and 3NF

The notes for this slide are for historical interest only. We do not use 2NF and 3NF on CS252 and you are not required to understand these terms.

Codd's definition of 2NF, as given by Date in 1990, is

> A [relvar] is in *second normal form* (2NF) if and only if it is in 1NF and every nonkey attribute is [irreducibly][2] dependent on the primary key.

The assumption that the primary key is the only key is clearly unsafe in general. In *The Relational Database Dictionary* (2006) Date provides the following revision:

> Relvar *R* is in second normal form (2NF) if and only if every attribute *A* that is not contained in any key of *R* is such that the set {*A*} is irreducibly dependent on every key of *R*.

Under both definitions ENROLMENT is clearly not in 2NF, because the FD {StudentId, CourseId} → {Name} is left-reducible to just {StudentId} → {Name}.

Codd's definition of 3NF, as given by Date in 1990, is

> A [relvar] is in *third normal form* (3NF) if and only if it is in 2NF and every nonkey attribute is nontransitively dependent on the primary key.

The term transitively dependent refers to situations like that shown in Slide 7, where we have {StudentId, CourseId} → {TutorId} and {TutorId} → {TutorName}, so the nonkey attribute TutorName is dependent on the primary key under the theorm of transitivity. Again we have an assumption that the primary key is the only key, and later definitions addressed that problem. The fact remains that 3NF does not fully address the problem it was intended to address, and the

[1] The word "relation" actually appears here, because Codd did not make the careful distinction between relations and relation variables that we now make. In later editions Date is careful to use the word he coined himself, relvar.
[2] The word "fully" appears here but "irreducibly" is now preferred.

formerly recommended design procedure of starting with relvars in 1NF, then decomposing as necessary to achieve 2NF, then decomposing as necessary again to achieve 3NF, is now discredited.

4. Boyce/Codd Normal Form (BCNF)

An FD is *trivial* if and only if its determinant is a superset of its dependant set. (Everything determines itself.)

ENROLMENT is not in BCNF because {StudentId} is the determinant of a nontrivial FD, {StudentId} → {Name}, and yet is not a superkey.

5. Splitting ENROLMENT (bis)

Note very carefully how the split is done. We make IS_CALLED by projecting ENROLMENT over the attributes appearing on both sides of the "rogue" FD, { StudentId } → { Name }. And we make IS_ENROLLED_ON by projecting it over all its attributes except for those appearing on the right-hand side of the rogue FD. The theorem giving rise to this general rule for decomposing around a rogue FD was proved by Ian Heath in 1971 and is therefore known as Heath's Theorem:

> Let *A, B,* and *C* be subsets of the heading of relvar *R* such that the union of *A, B,* and *C* is equal to that heading. Let *AB* denote the union of *A* and *B,* and similarly for *AC.* If *R* satisfies the FD *A* → *B*, then *R* is equal to the join of its projections over *AB* and *AC.*

Now we can see how every FD actually represents a join dependency. In our example the FD { StudentId } → { Name }, holding in ENROLMENT, represents the JD *{ {StudentId, Name}, {StudentId, CourseId} } holding in that same relvar. In general, the FD *A* → *B* holding in *R* represents the JD *{ *AB, AC* } holding in *R*. The JD immediately gives us the projections for the desired decomposition.

IS_CALLED and IS_ENROLLED_ON are in 2NF. As it happens, they are also in 3NF, BCNF, 4NF, 5NF and 6NF, so we need to look at some different examples to discover cases where a lower NF holds while a higher one is violated.

It seems as if the decomposition also allows us to add enrolments for students who have no name, and that is true unless a constraint is declared to the effect that every StudentId value appearing in IS_ENROLLED_ON also appears as a StudentId value in IS_CALLED.

Exercise: Write a **Tutorial D** declaration for this constraint.

6. Advantages of BCNF

The problems referred to on the slide give rise to what have been called “update anomalies”:

1. Inability to record some fact (in this case, somebody’s name) without at the same time recording some other fact (in this case, an enrolment).
2. Loss of a fact (in this case, somebody’s name) when some other recorded fact ceases to be true (in this case, the only enrolment for the student in question).
3. The same update (e.g., correction of somebody’s name) might need to be done in more than one place (so to speak).
4. When a fact is recorded (in this case, an enrolment), the accompanying fact, if already recorded (e.g. S1’s name), must be copied faithfully.

Exercise: Write a **Tutorial D** constraint declaration to express the FD mentioned in the slide.

7. Another Kind of Rogue FD

The “rogue” FD here is {TutorId} → {TutorName}. The difference in kind with respect to the first example was used by Codd in his definitions of 2NF and 3NF but is arguably insignificant: it’s just that in the first example the problematical determinant is a proper subset of a key, whereas here we have a determinant, {TutorId}, that isn’t a subset, proper or not, of any key. Therefore {TutorId is not a superkey; therefore TUTORS_ON is not in BCNF.

We have the same problems as with ENROLMENT: T1’s name is recorded twice (so we need to make sure it is always spelled the same way and not, for example, Hugh in one place and Huw in another), and we cannot record the name of a tutor who hasn’t been assigned for any enrolment yet.

8. Splitting TUTORS_ON

Note in passing that in TUTORS_ON_BCNF, { StudentId, CourseId } should be a foreign key referencing IS_ENROLLED_ON, to make sure that the enrolment to which a tutor is assigned does exist. As we have seen, that constraint can be declared in **Tutorial D** thus:

```
CONSTRAINT FK_FOR_TUTOR_ON
IS_EMPTY ( TUTOR_ON_BCNF NOT MATCHING IS_ENROLLED_ON ) ;
```

9. Dependency Preservation

If we use Heath’s theorem to decompose some relvar, it is highly desirable, where possible, to ensure that every FD that holds in that relvar still holds in one of the relvars resulting from the decomposition. The example on this slide shows that you sometimes need to take care, when there is more than one rogue FD, to address those rogues in an appropriate order. However, this aim cannot always be achieved, as we shall see later.

10. Try 1

11. Try 2

12. Try 3

13. An FD That Cannot Be Preserved

Under the assumed FD we have the possibly unrealistic situation, for the sake of an example, where no teacher teaches more than one course.

We have exactly the same problems of redundancy and lack of orthogonality, and yet TUTOR_FOR is in 3NF according to our definition, because TutorId is not a nonkey attribute.

Note that although each tutor teaches only one course, and exactly one tutor is assigned for a single enrolment, the same course can have several tutors (T1 and T3 both teach C1).

Note that the primary key, {StudentId, CourseId} is not the only key of TUTOR_FOR. Because {Tutor} → {CourseId} holds, {Tutor, StudentId} → {CourseId} also holds, under left-augmentation, therefore {Tutor, StudentId} is also a key.

But if {Tutor, StudentId} is chosen to be the primary key, then TUTOR_FOR is not in 2NF!!

So pursuit of 2NF and 3NF fails to achieve its objective in certain cases. Note that these “rogue” cases can only arise when there is more than one key and two of the keys “overlap” (have an attribute in common). However, having overlapping keys doesn’t *necessarily* lead to a violation of 3NF.

14. Splitting TUTOR_FOR

Yes, we have lost the constraint that was expressed by the key, {StudentId, CourseId} of TUTOR_FOR. If we do not somehow reinstate that constraint, it will be possible to enrol student S1 of course C1 for a second time, with tutor T3.

15. Reinstating The Lost FD

Any functional dependency, $A \rightarrow B$, in relvar R can be expressed as a constraint of the form:

COUNT (*R* { *A* , *B* }) = COUNT (*R* { *A* })

16. And The Lost Foreign Key

Perhaps it is just as well that **Tutorial D** does not have the FOREIGN KEY shorthand. It is easy to use the same longhand here as we did for the original TUTOR_FOR relvar, just replacing the name TUTOR_FOR by (TUTORS JOIN TEACHES).

17. In BCNF But Still Problematical

Perhaps the predicate for relvar TBC1 is "*Teacher* uses *Book* on course *CourseId*". In that case, the given JD tells us that if teacher *t* uses book *b* at all and *b* is used by anybody on course *c*, then *t* uses *b* on *c*.

The two books in the example are both used on C1 and C2 and teacher T1 uses both books. The appearance of the fourth tuple shown in the diagram is implied by the fifth (showing that somebody uses "Database in Depth" on C2) and the second (showing that T1 uses "Database in Depth" on some course). The JD also tells us that T2 does not use "Database Systems" at all, for otherwise that teacher would be compelled to use it on course C2.

Now, suppose we learn that T2 is to teach course C1 and has decided to use "Database Systems" on that course. Then under the JD (assuming it is declared as a constraint) the DBMS must reject any attempt to insert the tuple <T2, Database Systems, C1> unless the tuple <T2, Database Systems, C2> is inserted at the same time.

Once again we have redundancy: that T1 uses Database Systems is recorded twice, and that Database in Depth is used on C2 is also recorded twice. And once again we have lack of orthogonality. The simple predicates "*Teacher* uses *Book*" and "*Book* is used on course *CourseId*" are not treated independently.

18. Fifth Normal Form (5NF)

TBC1 is not in 5NF because the nontrivial JD *{ { Teacher, CourseId }, { Book, CourseId } } is not implied by the only key of TBC1, which is { Teacher, Book, CourseId }, the entire heading.

What Ronald Fagin really meant by a JD being "implied by the keys" is that the set of relations that satisfy all the key constraints for *R* is exactly the same as the set of relations that satisfy the JD.

The following notes are included for your possible interest. Because of the confusion they reveal as having pervaded the issue of 5NF, **you are not required to understand all this for CS252 assessment purposes.**

Until 2009 it was thought that a JD is "implied by the keys of *R*" if and only if each of its projections includes some key of *R* (as claimed by Date in his *Introduction to Database Systems*). However, consider relvar X{a,b,c} with keys {a,b}, {a,c}, and {b,c} and let the JD *{{a,b},{a,c},{b,c}} hold in X. It is fairly easy to find a relation that satisfies the key constraints but not the JD! However, X is not subject to any redundancy of the kind we have been looking at.

It seems, then, that 30 years after Fagin offered us 5NF, we found that his definition was actually a trifle too strong. Moreover, shortly after that discovery we found that Date's definition is a bit too strong too! If relvar *R* is in BCNF, then it is sufficient for each nontrivial JD to contain just one projection that is a superkey for *R*. With that small modification, Date's definition, arguably easier to understand, should be preferred—but in that case we can no longer use the term 5NF (we cannot change history). The term redundancy free normal form (RFNF) is under consideration for Date's definition (appropriately revised).

Note carefully, though, that the concepts of (a) holding (whether or not a given JD holds), and (b) being implied by keys, are orthogonal—a JD can hold and be implied by keys, can be implied by keys but not hold, can hold but not be implied by keys, or can neither hold nor be implied by keys. That said, we are normally interested to see if a JD is implied by keys only if that JD actually holds.

Historical note: Actually, TBC1 is not in 4NF either. The original definition of 4NF was rather elaborate and involved a difficult concept referred to a *multivalued dependency.* But it turned out that 4NF was equivalent to 5NF but restricted to binary JDs only (i.e., JDs with just two projections, like the one in the example at hand. So we never needed to grapple with that difficult concept and 4NF can be cast to the dustbin of history. 5NF was defined by the original author of 4NF, Ronald Fagin again, when he discovered 4NF to be inadequate.

19. Normalising TBC1

No notes.

20. A JD of Degree >2

Still taking the predicate to be "*Teacher* uses *Book* on course *CourseId"*, the revised JD tells us that if teacher *t* uses book *b* at all and *b* is used by anybody on course *c*, and *t* teaches course *c*, then *t* uses *b* on *c*.

We are heading for a 3-way decomposition, as we were with the decomposition on WIFE_OF_HENRY_VIII, but this time there can be no intermediate step—we cannot do it by two 2-way decompositions. Our ternary JD here is what might called *essentially* ternary.

21. Normalising TBC2

Verify that TB JOIN BC JOIN TC = TBC.

22. Sixth Normal Form (6NF)

No notes.

23. Wives of Henry VIII in 6NF

Use 6NF only for nonkey attributes that are "optional" and independent from all other nonkey attributes. If every wife has a first name but not every wife has a last name and not every wife has a fate, then the 6NF design shown on this slide is a good one.

Decomposing a 5NF relvar introduces orthogonality but does not eliminate any redundancy because a 5NF relvar is free of redundancy.

Note that an optimal database design very often lies somewhere between 5NF and 6NF (in a manner of speaking). In other words, some of its relvars, if joined together, might still be in 5NF.

24. EXERCISE

The Warwick Wargs football club decided to record their 2015 fixtures and results in a relational database, based on the new international standard for **Tutorial D** that had been published in 2014 following the widespread uptake of that language in the industry, replacing the old-fashioned SQL systems that had caused so much frustration.

Anne and Boris, who had both attended CS252 as students, got together to design the database.

Boris proposed the following relvar (leaving aside the issue of attribute types):

```
fixture { Against, Venue, Date, GoalsFor, GoalsAgainst, Result }
```

Boris explained as follows:

- `Venue` is either "home" or "away" and is always agreed upon with the opposing team when the date is fixed.
- `Result` is "win" when `GoalsFor` is greater than `GoalsAgainst`, "lose" when `GoalsFor` is less than `GoalsAgainst`, and otherwise "draw".
- The predicate given for `fixture` is as follows:

  > "When Wargs played a(n) `Venue` game against `Against` on `Date`, Wargs scored `GoalsFor` goals and `Against` scored `GoalsAgainst` goals, so the result was `Result`."

- Wargs never play more than one match on the same day.
- In the league Wargs play in, each team plays each other team twice, once at home and once away.
- As the fixture list is available at the start of the season, Boris plans for it to be possible to record the date and venue of a fixture before the match in question has been played. Anne agrees with that idea.

Anne found some problems with Boris's proposal and suggested a decomposition of `fixture` into several 6NF relvars to record the same information.

Boris then found some problems with Anne's suggestion and they eventually agreed on a compromise that addressed all the problems they had both observed.

(a) Write down the nontrivial functional dependencies (FDs) that hold in `fixture`. For each one, state whether its determinant is a key of `fixture`.

(b) Decompose `fixture` into as few relvars as possible such that each one is in BCNF. Use Boris's notation for the relvars and state the key(s) of each one. For each BCNF relvar, state whether it is in 6NF.

(c) What problem, originally observed by Anne, has been solved in the BCNF design?

(d) For any BCNF relvar that is not in 6NF, decompose it into an equivalent set of 6NF relvars having the same key.

(e) What problem, originally observed by Anne and remaining in the BCNF design, has been solved in the 6NF design?

(f) What problem, originally observed by Anne, remains in the complete 6NF design?

(g) What new problems did Boris observe with the 6NF design?

(h) What compromise do you think Anne and Boris settled on, to solve all the problems they had both observed? Give the relvars and their keys, and express in **Tutorial D** any other constraints that you think need to be declared for this design.

End of Notes