CS252:HACD Fundamentals of Relational Databases
Notes for Section 7: Constraints and Updating

1. Cover slide

Now that we have learned relational algebra, we are in a position to tackle the two topics of this section.

2. CONSTRAINTS

"… the database is at all times in a *consistent* state"—we need to clarify "at all times". In **Tutorial D** it means that consistency is enforced at the end of every innermost statement—in other words, from a syntactic point of view, every time the system encounters a semicolon in the commands that are given to it. There is no requirement for consistency to be enforced at a lower level of granularity than that, because intermediate states arising during the execution of a command are not visible to any users.

Some authorities require constraints to be satisfied only at the ends of transactions—in other words, immediately following execution of COMMIT commands. This weaker approach allows an inconsistent state to arise during a transaction and to be visible to the user running that transaction, but still protects other users. (This approach was once thought necessary to overcome the problem described on Slide 20, but see the solution described on Slides 21 and 22.)

The SQL counterparts of KEY are PRIMARY KEY and UNIQUE. That of IS_EMPTY is NOT EXISTS.

3. KEY Constraints

The constraint, if declared, has to be satisfied for every relation ever assigned to EXAM_MARK.

Note that the current value shows that { Mark } is not a superkey, because two distinct tuples have the same value for Mark. We might wonder if { CourseId, Mark } or { StudentId, Mark } is a superkey, as both of those subsets do satisfy the constraint in the relation shown. However, our knowledge of the enterprise tells us that in general a student might get the same mark on two different courses and that different students might get the same mark on the same course.

The longhand shown on the slide groups the marks for each combination of StudentId and CourseId, then picks out just those tuples containing more than one mark in the Marks group, then requires that result to be empty by projecting it over no attributes and comparing the result of that with the empty relation of degree zero (a.k.a. TABLE_DUM).

In general, the longhand involves grouping the nonkey attributes, then picking out the tuples having groups of cardinality greater then one, then projecting over no attributes, then comparing that with TABLE_DUM. If the comparison yields FALSE, then the KEY constraint is violated.

Here's a shorter longhand for the same KEY constraint:

COUNT (EXAM_MARK) = COUNT (EXAM_MARK { StudentId, CourseId })

If the number of tuples in the base relvar is equal to the number of tuples in its projection over the key attributes, then the KEY constraint is satisfied; otherwise it is not.

4. When a Superkey Is a Key

No notes.

5. The KEY Shorthand

Note that **Tutorial D** supports keys but does not support primary keys. Primary keys are not essential. Multiple KEY clauses are permitted in the same relvar definition. No declared key can be a superset of another one.

6. Multiple Keys

No notes.

7. Degenerate Cases of Keys

Recall the definition of key. First we defined superkey as meaning a certain kind of uniqueness constraint, and then we defined a key as being a superkey of which no proper subset is a superkey.

As for the special property implied by the empty key, recall that if *k* is a superkey of *rv,* then COUNT(*rv*) = COUNT(*rv* { *k* }). What is the maximum value of COUNT(*rv* {})?

8. "Foreign Key" Constraints

The term "foreign key" was introduced in the 1970s and taken up by SQL. **Tutorial D** has no direct counterpart, for reasons we shall shortly see. Not everybody finds the term very intuitive. The special kind of constraint it refers to is certainly a very common one indeed, and comparatively easy to implement with good performance in a DBMS.

The constraint involves matching certain attributes (just one in the example) with those of a key of "another" relvar (called the *referenced relvar*). I wrote the word "another" in quotes, because in fact the referenced relvar is permitted to be the same as the referencing relvar; but the fact that the two relvars are usually different ones perhaps explains the use of the word "foreign". The referenced attributes are a key in a foreign place, so to speak, but unfortunately the term "foreign key" refers to the referenc*ing* attributes!

9. Inclusion Dependency

By using a relation comparison such as the one shown on this slide, we lose those restrictions that, for reasons that are not entirely clear, are imposed on foreign keys:

- The referencing relation does not have to be specifically a base relvar reference (here it is a projection of a base relvar).
- Nor does the referenced relation (here again it is a projection of a base relvar).
- And the matching attributes no longer have to constitute a declared key of the referenced relation (though here they are).

Because the reasons are unclear, the shorthand is not supported in **Tutorial D**, so you will have to write such constraints using as relation comparisons or the IS_EMPTY shorthand shown on the next slide. In any case, even if **Tutorial D** did support FOREIGN KEY, it would have to be done differently from SQL, because in SQL the specification depends on column order when more than one column is involved.

Rel alert: Don't forget that *Rel* uses <= and >= for $\subseteq$ and $\supseteq$, respectively.

10. A Special Case of Inclusion Dependency

Recall that TABLE_DUM is the relation with no attributes and no tuples. If TABLE_DUM is a superset of *r,* then *r* too must be empty (and in fact must be TABLE_DUM).

11. IS_EMPTY Example

Because the operand of IS_EMPTY is a relation expression of arbitrary complexity, and because of the completeness of the relational algebra, any constraint can be expressed using IS_EMPTY, in theory. But sometimes the more general notation for inclusion dependencies is more convenient. The KEY and FOREIGN KEY shorthands, when applicable, are always more convenient.

Note that expressions using IS_EMPTY often seem like double negatives. We want to ensure that every mark is between 0 and 100 but we have to declare instead that no mark shall not be between 0 and 100.

12. Generalisation of Inclusion Dependency

Note that the operands now do not have to be of the same type (have the same heading). So we don't need those projections that we had to use in the inclusion dependency. As a consequence, the expression now includes no attribute names. This makes it immune to certain changes in the database definition, but vulnerable to others. It is unaffected when the name StudentId is changed to the same new name in both relvars, but it is vulnerable to the case where it is changed in just one of them. To be 100% safe, but vulnerable to both kinds of change, the constraint could be written like this:

```
IS_EMPTY ( IS_ENROLLED_ON { StudentId } NOT MATCHING
IS_CALLED { StudentId } )
```

But use of IS_EMPTY (… NOT MATCHING …) expressions poses big performance challenges for the DBMS. Shorthands are often good for the system as well as the user. The FOREIGN KEY shorthand used in SQL does have the advantage of being easy to implement with great efficiency. The question is, do we really need *all* of the observed restrictions before we can achieve the same efficiency?

13. "Exclusion Dependency"?

"Exclusion Dependency" is in quotes because I made it up for this lecture. You won't find it in the literature.

Unlike the FOREIGN KEY shorthand, inclusion dependency has an obvious and occasionally useful inverse, neatly captured in **Tutorial D** by omission of the word NOT.

Note that the operands of MATCHING can now be placed in either order, which is not the case with the inclusion dependency.

14. Constraint Declaration

As usual, a **Tutorial D** declaration consists of a key word indicating the kind of thing being declared, followed by a name by which it can be subsequently referenced, followed by the rest of the declaration.

15. Relational Update Operators

No notes.

16. INSERT, UPDATE, DELETE

Loosely speaking especially in the case of UPDATE. To speak of updating tuples is to speak of the tuples in a relvar as if they too were variables (and thus vairables within variables).

Each of these operators is actually shorthand for some form of assignment. For example, the assignment shown for enrolling student S5 on course C1 is the longhand for a certain invocation of the INSERT operator …

17. INSERT

This example records two new enrolments. The SQL counterpart of **Tutorial D**'s `RELATION{…}` expressions is the `VALUES` expression. Many SQL implementations restrict `VALUES` to denoting just a single row, even though the international standard allows any number of rows to be included, as in **Tutorial D**'s `RELATION{…}`.

18. UPDATE

If the WHERE clause is omitted, the specified attribute updates apply to *every* tuple in the current value of the target relvar. In other words, the WHERE clause defaults to WHERE TRUE.

19. DELETE

No notes.

20. An Occasional Problem with Updating

The example shown on this slide cannot happen with Oracle SQL because the second constraint cannot be expressed in Oracle. This is because Oracle, like nearly all industrial SQL implementations, does not allow subqueries to appear in constraint declarations.[1] However, the impasse can still arise if, for example, base table T1 is defined with a foreign key referencing base table T2 and T2 is defined with a foreign key referencing T1.

21. Proposed Solution to The Impasse

Multiple assignment has been proposed as a solution to the updating impasse that occasionally arises, but is not available in the technology of 2005. SQL systems have rather unsatisfactory ad hoc workarounds.

Notice the comma after the first INSERT. It indicates that constraints are not to be checked yet. In **Tutorial D**, constraints are checked at every semicolon (so to speak) only.

The detailed semantics of multiple assignment are rather complicated and beyond the scope of this course. Perhaps its most important aim is to ensure that no inconsistent database state is ever "visible", even if can notionally arise in the middle of a operation. For that reason, there is a rule to the effect that all expressions involved in the multiple assignment are evaluated before any updates are done (see the next slide).

22. A Note on Multiple Assignment

Rel's support for multiple assignment is inconsistent with the aspect described on this slide. In other words, if one of the individual assignments references a variable updated by an early assignment in the same statement, the value of that reference is the result of that earlier update instead of being its value at the very beginning of the statement. If that earlier update resulted in a temporary inconsistent state of the database, then the later assignment is "seeing" that inconsistent state—the phenomenon that **Tutorial D**'s multiple assignment is intended to avoid is being allowed to arise. The deficiency in Rel's support for multiple assignment need not bother CS252 students. The important thing for CS252 is that the "impasse" described on Slide 20 can be easily addressed.

End of Notes

[1] The SQL international standard does allow them and also includes a CREATE ASSERTION statement that supports everything that can be expressed using **Tutorial D**'s CONSTRAINT statement. We are not aware of any SQL implementations that support CREATE ASSERTION.