Constraints and Updating

Hugh Darwen

hugh@dcs.warwick.ac.uk
www.dcs.warwick.ac.uk/~hugh

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

1

Constraints

Constraints express the integrity rules for a database.

Enforcement of constraints by the DBMS ensures that the database is at all times in a *consistent* state.

A constraint is a *truth-valued* expression, such as a *comparison*, declared as part of the *logical schema* of the database.

The comparands of a constraint are typically relation expressions or invocations of aggregate operators.

But the commonest kinds of constraint are expressed using special shorthands, like KEY, FOREIGN KEY, IS_EMPTY.

2

KEY Constraints

The constraint shown below is a "uniqueness" constraint, meaning that no two distinct tuples can match on both StudentId and CourseId.

{ StudentId, CourseId } is a *superkey* of EXAM_MARK

EXAM_MARK

StudentId	CourseId	Mark
S1	C1	85
S1	C2	49
S2	C1	49
S3	C3	66
S4	C1	93

```
( ( EXAM_MARK GROUP { Mark } AS Marks
WHERE COUNT ( Marks ) > 1 ) { } ) = RELATION { } { }
```

3

When a Superkey Is a Key

If no proper subset of superkey *K* is a superkey, then *K* is a *key*.

So { StudentId, CourseId } is in fact a key of EXAM_MARK, and is in fact the only key of EXAM_MARK.

In general a relvar can have several keys, in which case it is sometimes useful to nominate one of them as being the *primary key*. For that reason, keys are sometimes referred to as *candidate keys*. When a primary key is nominated, any other keys are called *alternate keys*.

4

The KEY Shorthand

Traditionally, a KEY constraint is declared as part of the definition of the relvar to which it pertains, thus:

```
VAR EXAM_MARK BASE RELATION {
        StudentId SID,
        CourseId CID,
        Mark INTEGER }
 KEY { StudentId, CourseId } ;
```

5

Multiple Keys

Recall PLUS (*a* + *b* = *c*):

a	b	c
1	2	3
2	3	5
2	1	3

Not a variable, of course, but we can still observe that {a, b}, {a, c} and {b, c} are all keys. We might even nominate {a, b} to be the primary key (for psychological reasons only).

6

Degenerate Cases of Keys

The entire heading can be a key. In that case it is the only key (why?).

The empty set can be a key. In that case it is the only key (why?). What special property is implied by such a key?

7

"Foreign Key" Constraints

IS_CALLED

StudentId	Name
S1	Anne
S2	Boris
S3	Cindy
S4	Devinder
S5	Boris

IS_ENROLLED_ON

StudentId	CourseId
S1	C1
S1	C2
S2	C1
S3	C3
S4	C1

KEY { StudentId }

Every StudentId value here must also appear in IS_CALLED { StudentId }

8

Inclusion Dependency

FOREIGN KEY { StudentId } REFERENCING IS_CALLED included in declaration of IS_ENROLLED_ON is shorthand for:

IS_CALLED { StudentId } ⊇ IS_ENROLLED_ON { StudentId }

Such constraints in general are sometimes called *inclusion dependencies.* An inclusion dependency is a foreign key if the heading common to the two comparands is a key of the referenced relvar.

9

A Special Case of Inclusion Dependency

Consider:
TABLE_DUM ⊇ *r* { }
≡ RELATION { } { } ⊇ *r* { }

In **Tutorial D** we can write this as **IS_EMPTY** (*r*).

Also:
r1 ⊇ *r2* ≡ IS_EMPTY (*r2* MINUS *r1*)

≡ *r2* NOT MATCHING *r1*

10

IS_EMPTY Example

EXAM_MARK

StudentId	CourseId	Mark
S1	C1	85
S1	C2	49
S2	C1	49
S3	C3	66
S4	C1	93

This might be subject to the constraint:
0 ≤ Mark ≤ 100

IS_EMPTY (EXAM_MARK WHERE Mark < 0 OR Mark > 100)

11

Generalisation of Inclusion Dependency

IS_EMPTY (*r1* **NOT MATCHING** *r2*)

E.g., to express that foreign key in IS_ENROLLED_ON:

IS_EMPTY (IS_ENROLLED_ON **NOT MATCHING** IS_CALLED)

But now the operands can be arbitrary relation expressions, without the restrictions of FOREIGN KEY.

12

"Exclusion Dependency"?

IS_EMPTY (*r1* **MATCHING** *r2*)

E.g., to enforce disjointness of part-time and full-time employees:

IS_EMPTY (PART_TIMER **MATCHING** FULL_TIMER)

Equivalently:

IS_EMPTY (FULL_TIMER **MATCHING** PART_TIMER)

13

Constraint Declaration

In **Tutorial D** (in addition to KEY specifications written inside relvar declarations):

CONSTRAINT *name expression* ;

E.g.: CONSTRAINT Marks_out_of_100 IS_EMPTY
(EXAM_MARK WHERE Mark < 0 OR Mark > 100) ;

And to cancel this constraint:
DROP CONSTRAINT Marks_out_of_100 ;

14

Relational Update Operators

In theory, only assignment is needed. For example, to enrol student S5 on course C1:

```
IS_ENROLLED_ON :=
   IS_ENROLLED_ON
   UNION
   RELATION { TUPLE { StudentId SID ( 'S5' ),
                      CourseId CID ( 'C1' ) } } ;
```

But that's not always convenient, and not easy for the system to do the update quickly, either.

15

INSERT, UPDATE, DELETE

The following shorthands are universally agreed on:

- INSERT, for adding tuples to a relvar
- UPDATE, for updating existing tuples in a relvar
- DELETE, for removing tuples from a relvar

loosely speaking!

16

INSERT

In **Tutorial D**:

INSERT *relvar-name relation-expression* ;

E.g.

```
INSERT IS_ENROLLED_ON
        RELATION { TUPLE { StudentId SID ( 'S5' ),
                           CourseId CID ( 'C1' ) },
                   TUPLE { StudentId SID ( 'S4' ),
                           CourseId CID ( 'C4' ) } } ;
```

17

UPDATE

In **Tutorial D**:

UPDATE *relvar-name* [**WHERE** …] (*attribute-updates*) ;

E.g.

```
UPDATE EXAM_MARK WHERE CourseId = CID ( 'C1' )
  ( Mark := Mark + 5 ) ;
```

When it was decided that the exam for C1 had been a little too difficult, perhaps. Everybody who sat the exam gets 5 more marks.

18

DELETE

In **Tutorial D**:

DELETE *relvar-name* [**WHERE** *condition*] ;

E.g.

DELETE IS_CALLED WHERE Name = NAME ('Boris') ;

(Did we mean to do that? — there's more than one Boris!)

19

An Occasional Problem with Updating

Suppose the following constraints are in effect:

CONSTRAINT EnrolRecognisedStudentsOnly
IS_EMPTY (IS_ENROLLED_ON NOT MATCHING IS_CALLED);

CONSTRAINT RegisterEnrolledStudentsOnly
IS_EMPTY (IS_CALLED NOT MATCHING IS_ENROLLED_ON);

We can't enrol a student before we have named her and we can't name her before we have enrolled her on some course. *Impasse?*

20

Proposed Solution to The Impasse

"Multiple assignment": updating several variables simultaneously.

In **Tutorial D**:

```
INSERT IS_CALLED
        RELATION { TUPLE { StudentId SID ( 'S6' ),
                           Name NAME ( 'Zoë' ) } } ,
INSERT IS_ENROLLED_ON
        RELATION { TUPLE { StudentId SID ( 'S6' ),
                           CourseId CID ( 'C1' ) } } ;
```

21

A Note on Multiple Assignment

Would the following have the same effect?

```
INSERT IS_CALLED
        RELATION { TUPLE { StudentId SID ( 'S6' ),
                           Name NAME ( 'Zoë' ) } } ,
INSERT IS_ENROLLED_ON
    EXTEND IS_CALLED WHERE Name = NAME ('Zoë' )
      ADD  (CID('C1') AS CourseId) {StudentId, CourseId } ;
```

No! The second INSERT cannot see Zoë.

22