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 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 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

\[
\{ \text{EXAM_MARK GROUP \{Mark\} AS Marks WHERE COUNT (Marks) > 1 \} \} = \text{RELATION \{\} \{\}}
\]

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

\[
\{ \text{EXAM_MARK GROUP \{Mark\} AS Marks WHERE COUNT (Marks) > 1 \} \} = \text{RELATION \{\} \{\}}
\]

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.

3 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

\[
\{ \text{EXAM_MARK GROUP \{Mark\} AS Marks WHERE COUNT (Marks) > 1 \} \} = \text{RELATION \{\} \{\}}
\]

4 The KEY Shorthand

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

\[
\text{VAR EXAM\_MARK BASE RELATION \{} \text{StudentId SID, CourseId CID, Mark INTEGER } \} \text{ KEY \{ StudentId, CourseId \};}
\]

5 Multiple Keys

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

\[
\begin{array}{ccc}
\text{a} & \text{b} & \text{c} \\
1 & 2 & 3 \\
2 & 3 & 5 \\
\end{array}
\]

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).
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?

“Foreign Key” Constraints

<table>
<thead>
<tr>
<th>StudentId</th>
<th>Name</th>
<th>IS_CALLED</th>
<th>StudentId</th>
<th>CourseId</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>Anne</td>
<td></td>
<td>S1</td>
<td>C1</td>
</tr>
<tr>
<td>S2</td>
<td>Boris</td>
<td></td>
<td>S1</td>
<td>C2</td>
</tr>
<tr>
<td>S3</td>
<td>Cindy</td>
<td></td>
<td>S2</td>
<td>C1</td>
</tr>
<tr>
<td>S4</td>
<td>Devinder</td>
<td></td>
<td>S3</td>
<td>C3</td>
</tr>
<tr>
<td>S5</td>
<td>Boris</td>
<td></td>
<td>S4</td>
<td>C1</td>
</tr>
</tbody>
</table>

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

Inclusion Dependency

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

IS_CALLED { StudentId } \supseteq 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.

A Special Case of Inclusion Dependency

Consider:

\[
\text{TABLE_DUM} \supseteq r \{ \}
\]

\[
\text{RELATION} \{ \{ \} \supseteq r \{ \}
\]

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

Also:

\[
rl \supseteq r2 \equiv IS\_EMPTY ( r2 \text{ MINUS } rl ) = r2 \text{ NOT MATCHING } rl
\]

IS_EMPTY Example

This might be subject to the constraint:

\[ 0 \leq \text{Mark} \leq 100 \]

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

<table>
<thead>
<tr>
<th>StudentId</th>
<th>CourseId</th>
<th>Mark</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>C1</td>
<td>85</td>
</tr>
<tr>
<td>S1</td>
<td>C2</td>
<td>49</td>
</tr>
<tr>
<td>S2</td>
<td>C1</td>
<td>49</td>
</tr>
<tr>
<td>S3</td>
<td>C3</td>
<td>66</td>
</tr>
<tr>
<td>S4</td>
<td>C1</td>
<td>93</td>
</tr>
</tbody>
</table>

Generalisation of Inclusion Dependency

IS_EMPTY ( rl 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.
“Exclusion Dependency”?

\[
\text{IS\_EMPTY} ( r1 \text{ MATCHING } r2 )
\]

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

\[
\text{IS\_EMPTY} ( \text{PART\_TIMER MATCHING} \text{ FULL\_TIMER})
\]

Equivalently:

\[
\text{IS\_EMPTY} ( \text{FULL\_TIMER MATCHING} \text{ PART\_TIMER})
\]

Constraint Declaration

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

\[
\text{CONSTRAINT name expression ;}
\]

E.g.: \text{CONSTRAINT Marks\_out\_of\_100 IS\_EMPTY ( EXAM\_MARK WHERE Mark < 0 OR Mark > 100 ) ;}

And to cancel this constraint:

\[
\text{DROP CONSTRAINT Marks\_out\_of\_100 ;}
\]

Relational Update Operators

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

\[
\text{IS\_ENROLLED\_ON :=}
\]

\[
\text{IS\_ENROLLED\_ON UNION}
\]

\[
\text{RELATION \{ TUPLE \{ StudentId SID ( 'S5' ), CourseId CID ( 'C1' ) \} \} ;}
\]

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

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!

INSERT

In Tutorial D:

\[
\text{INSERT relvar-name relation-expression ;}
\]

E.g.

\[
\text{INSERT IS\_ENROLLED\_ON}
\]

\[
\text{RELATION \{ TUPLE \{ StudentId SID ( 'S5' ), CourseId CID ( 'C1' ) \}, TUPLE \{ StudentId SID ( 'S4' ), CourseId CID ( 'C4' ) \} \} ;}
\]

UPDATE

In Tutorial D:

\[
\text{UPDATE relvar-name [ WHERE … ] (attribute-updates) ;}
\]

E.g.

\[
\text{UPDATE EXAM\_MARK WHERE CourseId = CID ( 'C1' ) ( Mark := Mark + 5 ) ;}
\]

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.
DELETE

In Tutorial D:

DELETE relvar-name [ WHERE condition ] ;

E.g.

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

(Do we mean to do that? — there’s more than one Boris?)

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?

Proposed Solution to The Impasse

“Multiple assignment”: updating several variables simultaneously.

In Tutorial D:

INSERT IS_CALLED
RELATION { TUPLE { StudentId SID (’S6’), 
Name NAME (’Zoe’) } } ,

INSERT IS_ENROLLED_ON
RELATION { TUPLE { StudentId SID (’S6’), 
CourseId CID (’C1’) } } ;

A Note on Multiple Assignment

Would the following have the same effect?

INSERT IS_CALLED
RELATION { TUPLE { StudentId SID (’S6’), 
Name NAME (’Zoe’) } } ,

INSERT IS_ENROLLED_ON
EXTEND IS_CALLED WHERE Name = NAME (’Zoe’) 
ADD (CID(’C1’) AS CourseId) {StudentId, CourseId} ;

No! The second INSERT cannot see Zoe.