Databasesystemer, forår 2005
IT Universitetet i København

Forelæsning 3: Business rules, constraints & triggers.

3. marts 2005

Forelæser: Rasmus Pagh

Today's lecture

Constraints and triggers

- Uniqueness constraints (identifiers/candidate keys, identifier/primary key)
- Assertion-based constraints
- Foreign keys
- Triggers

What you should remember from previously

In this lecture I will assume that you remember:

- Identifiers in E-R diagrams.
- Cardinality constraints in E-R diagrams.
- How to convert an E-R diagram into relations.
- The SQL used when modifying or adding tuples in a relation.

Next: Uniqueness constraints.

Identifiers in E-R diagrams

When translated to relations, identifiers in E-R diagrams serve to:

- Uniquely identify tuples in relations corresponding to entities.
- Together, identifiers of participating entities uniquely identify tuples in relations corresponding to relationships.

We would like the DBMS to make sure that data conforms to these uniqueness constraints. This is done by declaring the unique identifiers as "primary keys".

Declaring a primary key

Add to the relation schema a line of the form:

```
PRIMARY KEY (<list of attributes>)
```

If the primary key has just one attribute, we may instead write `PRIMARY KEY` immediately after the definition of the data type of the attribute, e.g.:

```
id INT PRIMARY KEY,
```

`NULL` values are not allowed in attributes of a primary key.

Declaring other candidate keys

If we want the DBMS to check other uniqueness constraints, we may add to the SQL relation schema any number of lines of the form:

```
UNIQUE (<list of attributes in key>)
```

Uniqueness is *not* guaranteed for tuples having `NULL` values in the key attributes. However, `NULL` values can be prevented by adding a `NOT NULL` constraint after the declaration of each key attribute.

When a key constraint is violated

When a key constraint is violated, an error message is produced.

The **state** of the database (i.e., the data it contains) is restored to what it was *before* the action that caused the violation.

Updates in SQL are grouped in units called **transactions** (more about transactions later in the course).

Constraint-violating transactions are undone (or **rolled back**).

Should every relation have a primary key?

Short answer: Not necessarily.

Consider for example the relation corresponding to a simple multivalued attribute in an E-R diagram:

- Typically, the only candidate key would consist of both attributes.
- Thus, a primary key constraint would only serve the purpose of eliminating duplicate tuples.

Next: Assertion-based constraints.

NOT NULL constraints

The constraint NOT NULL may be specified for any attribute in a relation schema, indicating that NULL is not a legal value.

In general, any attribute that does not correspond to an optional attribute in the E-R diagram should be declared NOT NULL.

CHECK constraints

Many business rules can be expressed as so-called CHECK constraints, which are **assertions** (i.e., conditions that must be true) about attributes or tuples of a relation.

- A CHECK constraint on an attribute is checked every time
  - a value of this attribute is modified.
  - a new tuple is inserted.
- A CHECK constraint on tuples is checked every time
  - an attribute value changes.
  - a new tuple is inserted.
- If a constraint is violated, the current transaction is rolled back, and an error message is produced.

Writing attribute-based CHECK constraints

A constraint C on an attribute is declared by writing

CHECK C

immediately after the datatype definition.

The condition C may refer to other attributes of the relation, and even to other relations, using a subquery.

(However, Oracle does not allow SQL queries in C.)

Examples:

- `percentage INT CHECK (percentage>=0 AND percentage<=100)`
- `cpr CHAR(10) CHECK (cpr IN (SELECT cpr FROM students))`

Writing tuple-based CHECK constraints

A constraint C on tuples is declared by adding the line

CHECK C

to the relation schema definition.

The only difference to attribute-based CHECK constraints is *when* the constraint is checked.

Examples:

- `CHECK (upper-bound => lower-bound)`
- `CHECK (cpr IN (SELECT cpr FROM students))`

Next: Foreign keys.

Foreign key constraints

A **foreign key constraint** on an attribute is a constraint saying that its attribute values can *always be found in exactly one place in another relation*.

Foreign key constraints are typically used to express **referential integrity**, i.e., that values supposed to refer to tuples in other tables indeed do so.

If we want the DBMS to check foreign key constraints, we may add to the SQL relation schema any number of declarations of the form:

```
FOREIGN KEY (<attribute name>)
            REFERENCES <table name>(<attribute name>)
```

Composite foreign keys

Foreign keys may be **composite**, i.e., consist of several attributes.

The syntax for declaring composite foreign keys is the obvious extension of what we saw before:

```
FOREIGN KEY (<list of attribute names>)
        REFERENCES <table name>(<list of attribute names>)
```


Semantics of a foreign key constraint

Suppose the schema for relation R contains the declaration

FOREIGN KEY $(A_1, \ldots, A_n)$ REFERENCES S$(B_1, \ldots, B_n)$.

Then the relation S *must* have $B_1, \ldots, B_n$ as primary keys or contain a declaration like

UNIQUE $(B_1, \ldots, B_n)$.

This means that the DBMS checks that any values of $A_1, \ldots, A_n$ in a tuple of R can also be found as values of $B_1, \ldots, B_n$ in a tuple of S.

Problem session (5 minutes)

What is the difference (if any) between the CHECK constraint

```
cpr CHAR(10) CHECK (cpr IN (SELECT cpr FROM students))
```

and the referential integrity constraint

```
cpr CHAR(10) REFERENCES students(cpr)
```


Referential integrity from E-R diagrams

If a relationship in our E-R diagram has an "exactly one" cardinality constraint, it can be expressed as a foreign key constraint.

This means that the DBMS maintains the referential integrity of the relationship.

There seems to be no general way to express an "at least one" cardinality constraint.

Note that in supertype-subtype relationships there is an implicit "exactly one" cardinality constraint.

Maintaining referential integrity

The default (i.e., standard) policy when a transaction violates a foreign key constraint is to roll the transaction back.

However, for each referential constraint we may choose from two other policies for handling changes *to the referenced relation*:

- **The cascade policy**:
  - If the foreign key attribute values of a tuple were changed, change all references to this tuple to the new value.
  - If a tuple is deleted, delete all tuples referencing it.
- **The set-null policy**:
  - If some reference became invalid, set all its attribute values to `NULL`.

Next: Triggers.

Triggers

Triggers is a general mechanism for:

- Enforcing constraints/business rules, and more generally
- Making the DBMS perform actions on certain events.

The definition of a trigger consist of an event, a condition, and an action.

- Triggers are awakened (or triggered) when the **event**, a certain change to the database, occurs.
- If the **condition** associated with the trigger is true, then the **action** is performed.

Triggers in SQL

Key features of triggers in SQL:

- Triggering events are insertions, deletions, and updates of tuples.
- The action can be any SQL statement.
  (But most RDBMSs have restrictions on the SQL allowed in the action.)
- The action can refer to values from both before *and* after the event.
- The action can be performed either
  - After each event that activates the trigger, or
  - At the end of each transaction where one or more events activated the trigger.

Trigger definition syntax, simplified

Syntax in Oracle (differs slightly from SQL definition):

```
CREATE TRIGGER <name of trigger> AFTER
INSERT | DELETE | UPDATE
   [OF <attribute name>] ON <name of relation or view>
[REFERENCING OLD AS <name>, NEW AS <name>]
[FOR EACH ROW
 [WHEN <condition>]]
BEGIN
   <PL/SQL commands>
END;
```

Vertical lines | between alternatives. Brackets [] around optional parts.

Variables in `<PL/SQL commands>` must be prefixed by semicolon (`:old.a`).

Most important points in this lecture

As a minimum, you should after this week:

- Know how to declare key constraints and referential integrity (i.e., foreign key) constraints in SQL.
- Understand the basic mechanisms for maintaining referential integrity.
- Know how to declare tuple-based CHECK constraints, and know how these are checked.
- Understand how to define triggers, and the mechanism for executing triggers in SQL.