Constraints and Triggers in SQL

Chapter 7/Chapter 6

November 20, 2002

1 Keys and Foreign Keys

Defining Primary Keys.

1. Declared an attribute using the keyword PRIMARY KEY after the definition of the attribute, for example,

   create table test (maker char(1) default not null primary key,
   model char(10), type char(10))

2. Declared a list of attributes using the keyword PRIMARY KEY then the list of attributes in parentheses, for example,

   create table test (maker char(1) default not null,
   model char(10) default not null,
   type char(10),
   primary key (maker, model))

Note: The first method is only used when primary key contains ONLY one attribute. DB2 requires that attributes belonging to the key cannot have NULL values.

Key Constrains. One relation/table only has one primary key. It can have more than one keys. In that case, we can enforce these constraints by creating an unique index:

   create unique index model on product(model)

The second way to define the constraint is to have it in the CREATE TABLE command by adding the keyword UNIQUE after the type declaration of the attribute, for example

   create table test (maker char(1) default not null,
   model char(10) default not null unique,
   type char(10),
   primary key (maker, model))

In this example, the primary key is maker,model but we also want the model to be unique.
2 Referential Integrity and Foreign Keys

Declaring Foreign-Key Constraints. Conditions:

1. The referenced attributes of the second relation must be declared the primary key or unique for their relation;

2. There is a referential integrity constraint connecting the two tables.

Two ways:

1. Use `REFERENCES table(<attribute>)` after the definition of the attribute, for example,

   ```sql
   create table pc (model int references product(model), hd int, ...)
   ```

   this works if `model` is the primary key of the table `product`.

2. Use `FOREIGN KEY <attributes> REFERENCES table(<attribute>)` if the foreign key contains more than one attribute, for example,

   ```sql
   create table newmanufacturer (maker char(1),
   model char(10),
   type char(10),
   foreign key (maker, model)
   references product (maker, model))
   ```

   In both cases, we can define the referential constraint to be `DEFERRABLE` or `NOT DEFERRABLE` followed by ‘INITIALLY DEFERRED’ or ‘INITIALLY IMMEDIATE’ that makes the system to check for the constraint after or before a transaction is made, respectively. For example,

   ```sql
   create table newmanufacturer (maker char(1),
   model char(10),
   type char(10),
   foreign key (maker, model)
   references product (maker, model))
   deferrable initially deferred
   ```

   We can give `name` to constraints and then specify them to be deferred or not deferred by the command

   ```sql
   set constraint Name_Of_Constraint deferred
   ```

   or

   ```sql
   set constraint Name_Of_Constraint immediate
   ```
For example, in the following command, we define a constraint with the name 'PCLaptop-Printer' that refers to the constraint that the attribute 'type' can only take three values 'pc', 'printer', or 'laptop':

```sql
create table newmanufacturer (maker char(1),
   model char(10),
   type char(10)
   constraint PCLaptopPrinter
   check (type in ('pc','laptop','printer'))
```

Maintaining Referential Integrity.

1. **Rejection:**
   (a) Insert into test2 a tuple whose values in (maker,model) cannot be found in test will be rejected.
   (b) Update in test2 will be rejected if the key is not in test.
   (c) Delete/Update in test will be rejected if there is some tuple in test2 referring to the tuple.

2. **Cascade:** Delete/Update of the third type will be carried out and tuples referring to the one in test2 will be deleted/updated correspondingly.

3. **Set-null:** Instead of deleting/updating, the tuples will be updated with NULL values.

Specifying how referential constraints are enforced: using **ON DELETE** and **ON UPDATE**, for example,

```sql
create table newmanufacturer (maker char(1),
   model char(10),
   type char(10),
   foreign key (maker, model)
   references product (maker, model)
   on delete set null
   on update cascade)
```

### 3 Constraints on Attributes and Tuples

**Constraints on Values of attributes:** specified when defining the attribute, using

1. **NOT NULL,** e.g.,
   ```sql
   create table test (maker char(1) not null, ...)
   ```

2. **CHECK-based,** e.g.,
   ```sql
   create table test (maker char(1) CHECK (maker IN ('A','B','C')), ...)
   ```
Domain Constraints. Create a new type such as

```sql
create domain makerPd char(1) check (value in ('A','B','C'))
```

**NOTE:** This is available in SQL2 – our current DB2 does not have these features. However, it has a command for defining a new type from built-in types, for example,

```sql
create distinct type New_Type as Built_In_Type with comparisons
```

Global Constraints. Defined in create table command using the CHECK clause:

```sql
create table pctest (model int, hd int, price int,
  CHECK (hd >= 10 or price <= 1500))
```

One can create an assertion in SQL2 that can specify a global constraint. An assertion can be created with the command `CREATE ASSERTION <name> CHECK (<condition>)` where

1. **name** is the name of the assertion and
2. **condition** specifies the condition which must be satisfied by the tables.

Example (Will not work in DB2):

```sql
create assertion NoPCLapTop
  check (not exists
    ((select distinct maker from product, pc where
      product.model = pc.model)
    intersect
    (select distinct maker from product, laptop where
      product.model = laptop.model)))
```

4 Modification of Constraints

We can give name to constraint, e.g.,

```sql
create table newmanufacturer (maker char(1),
  model char(10),
  type char(10)
  constraint PCLaptopPrinter
    check (type in ('pc','laptop','printer'))
```

and then we can modify the constraint by `ALTER TABLE ...`, for example,

```sql
alter table newmanufacturer drop constraint PCLaptopPrinter
```

or

```sql
alter table newmanufacturer add constraint ModelIsKey
  primary key (model)
```
5 Triggers

Also called Event-Condition-Action Rules (or ECA rules) – this is a method for maintaining constraints automatically. A trigger is awakened whenever its condition becomes true (some event happens that makes the condition true). In this case, the actions associated to the rule will be executed. SQL allows

1. actions can be executed either before or after the triggering event.
2. the actions can refer to both old tuples or new tuples that were inserted, deleted, or updated in the event that triggered the action.
3. Update event might be limited to a particular attribute or a set of attributes.
4. A condition may be specified by WHEN; the action is executed only if the rule is triggered and the condition holds.
5. The programmer has an option of specifying that the action is performed either (a) once for each modified tuple; or once for all the tuples that changed in one database operation.

The syntax:

```
create trigger Trigger_Name
  before | after Event_Condition
  referencing Reference_To
  [for each row]
  [when Additional_Conditions]
  [SQL_Command | begin SQL_List_Of_Commands end]
```

where

1. Event_Condition: Condition that wakes up the trigger, for example, ‘DELETE ON pc’ or ‘UPDATE OF price ON pc’;
2. Reference_To: how to refer to the old and new tuples, for example, ‘OLD TABLE AS oldStuff’, ‘NEW TABLE AS newStuff’, ‘OLD ROW AS oldRow’, etc.
3. if for each row is omitted then the trigger is executed once;
4. Additional_Conditions: the condition specifying when the actions are executed
5. SQL_Command: (or SQL_List_Of_Commands) an SQL command (or a list of SQL commands)

Example: Trigger for price of pc cannot become higher:
create trigger NoHigherPrice
  after update of price on pc
  referencing
    old row as oldRow
    new row as newRow
  for each row
  when (oldRow.price < newRow.price)
    update pc
      set price = oldRow.price
      where model = oldRow.model

Instead-Of Trigger:  Allow for before/after of the trigger definition to be ‘instead-of’; This will instruct the system to execute the actions of the trigger instead of the current event. This can be used for modifying view, for example, if allMaker is a view of the product relation that does not have the 'type' attribute, we can have

create trigger insertPr
  instead of insert on allMaker
  referencing
    new row as newRow
  for each row
  insert into product(maker,model,type)
    values(newRow.maker, newRow.model,'pc')