CSI 2132 Lab #6

- Assertions and Triggers

24 FEB 2019

Integrity Constraints - Recap

Types of Integrity Constraints

- Primary key constraints
- Foreign key constraints
- Referential integrity constraints
- Domain constraints
- General constraints

Note - Operations that violate any integrity constraint at the tuple level are disallowed.

Integrity Constraints - Recap

Primary key constraints: By default, DBMS checks that the combination of values for those attributes declared as primary key remains unique in the relation and that none of them are null

```
CREATE TABLE artist
(
  aname character varying(20) NOT NULL,
  birthplace character varying(20),
  style character varying(20),
  dateofbirth date,
  country character varying(20),
  CONSTRAINT "pk of artist" PRIMARY KEY (aname)
)
```

Integrity Constraints - Recap

Foreign key constraints: Control what attribute values can be stored in the relation holding the foreign key field. It can point to a primary key attribute in another relation or to a non-PK attribute having the UNIQUE constraint.

```
CREATE TABLE artwork
(
  title character varying(20) NOT NULL,
  "year" integer,
  "type" character varying(20),
  price numeric(8,2),
  aname character varying(20),
  CONSTRAINT artwork_pkey PRIMARY KEY (title),
  CONSTRAINT artwork_aname_fkey FOREIGN KEY (aname)
      REFERENCES artist (aname) MATCH SIMPLE
      ON UPDATE CASCADE ON DELETE CASCADE
)
```

Integrity Constraints - Recap

Referential integrity constraints: Specifies what happens to the tuples in the foreign relation when a deletion or update of a primary key attribute value is about to occur in the main table.

```
CREATE TABLE artwork
(
  title character varying(20) NOT NULL,
  "year" integer,
  "type" character varying(20),
  price numeric(8,2),
  aname character varying(20),
  CONSTRAINT artwork_pkey PRIMARY KEY (title),
  CONSTRAINT artwork_aname_fkey FOREIGN KEY (aname)
      REFERENCES artist (aname) MATCH SIMPLE
      ON UPDATE CASCADE ON DELETE CASCADE
)
```

Integrity Constraints - Recap

Domain constraints: Restricts the set of values an attribute can take to lie within a particular domain

- A CHECK clause is added to the attribute definition.

```
CREATE TABLE customer
(
  custid integer NOT NULL,
  "name" character varying(20),
  address character varying(20),
  amount numeric(8,2),
  rating integer,
  CONSTRAINT customer_pkey PRIMARY KEY (custid),
  CONSTRAINT customer_rating_check CHECK (rating >= 1 AND rating <= 10)
)
```

Integrity Constraints - Recap

General constraints:

Additional constraints applicable to the environment being modeled.

- They are highly model-specific and cannot be captured by any of the previous types of constraints.
- The way to do this in SQL is through declarative assertions.

Assertions

- special type of integrity constraint and shares the same namespace as other constraints
- assertion is not necessarily dependent on one particular table

```
CREATE ASSERTION <name>
CHECK (<condition>)
```

<name> is a mandatory identifier for the constraint.

It can be used later on to modify or drop the constraint.

<condition> can be written as in the WHERE clause

If it holds true, the assertion is not violated and the integrity of the data is guaranteed.

Assertions

Example:

- Limit the number of sailors and boats to 100 in total

```
create assertion smallClub
check (
          (select count(*) from sailors s) +
          (select count(*) from boats b) < 100
        )
```

Note - PostgreSQL does not implement assertions at present

Triggers

- database callback functions, which are automatically performed/ invoked when a specified database event occurs

Components of a Trigger

- Event(s): An INSERT, UPDATE or DELETE operation on a particular tuple.
- Condition: Determines whether the action should be executed. If no condition is specified, the trigger is executed once the event takes place.
- Action: Usually a sequence of SQL statements, but could be also a database transaction or running an external program.

Trigger Syntax

CREATE TRIGGER name {BEFORE | AFTER}

{ event [OR…] } [OF attribute] ON table

[FOR [EACH] {ROW | STATEMENT}]

[WHEN (condition)]

EXECUTE PROCEDURE funcname(arguments)

Trigger Syntax

- BEFORE = constraints are checked before the operation is attempted
- AFTER = constraints are checked after the operation has been carried out
- OF = Column associated with the UPDATE operation
- ROW = The trigger is invoked once per row affected by the underlying operation. Individual attribute values per row are available.
- STATEMENT = The trigger is invoked only once for the entire operation no matter how many rows are affected. No attribute values are available.

Example

Execute the check_sailor_rating_age( ) function whenever a row of the SAILORS table is about to be updated

```
CREATE TRIGGER check_sailor
        BEFORE UPDATE ON sailors
        FOR EACH ROW
        EXECUTE PROCEDURE check_sailor_rating_age();
```

Example

- Same as before, but only if the sailor's rating is to be updated

```
CREATE TRIGGER check_sailor
        BEFORE UPDATE OF rating ON sailors
        FOR EACH ROW
        EXECUTE PROCEDURE check_sailor_rating_age();
```

Example

- Same as before, but only if the sailor's age will in fact change its value. Notice the WHEN clause

```
CREATE TRIGGER check_sailor
        BEFORE UPDATE ON sailors
        FOR EACH ROW
        WHEN (OLD.age IS DISTINCT FROM NEW.age)
        EXECUTE PROCEDURE check_sailor_rating_age();
```

Example

- Call a function to log any sailors' updates, but only if something changed

```
CREATE TRIGGER log_sailor_update
        AFTER UPDATE ON sailors
        FOR EACH ROW
        WHEN (OLD.* IS DISTINCT FROM NEW.*)
        EXECUTE PROCEDURE log_sailor_update();
```

Trigger Procedures(functions)

- Must be defined before the CREATE TRIGGER statement can execute.
- Must be declared as a function taking no arguments and returning type trigger.
- Can be written in C (low-level) or PL-PGSQL (high-level)
- We will use PL-PGSQL (Procedural Language for PostgresSQL) from pgAdmin

Writing Trigger Procedures with PL-PGSQL

- Open the Query Editor and type the following

```
CREATE FUNCTION check_sailor_name_age()
  RETURNS trigger AS
$BODY$
BEGIN

-- Check sailor's name
IF NEW.sname IS NULL THEN
        RAISE EXCEPTION 'The sailor must have a name';
END IF;

-- Check sailor's age
IF NEW.age > 50 THEN
        RAISE EXCEPTION 'The sailor must be 50 or below';
END IF;

RETURN NEW;

END

$BODY$ LANGUAGE plpgsql;
```

Writing Trigger Procedures with PL-PGSQL

- Open the Query Editor and type the following

```
CREATE TRIGGER check_sailor
BEFORE UPDATE ON sailors
FOR EACH ROW
EXECUTE PROCEDURE check_sailor_name_age()
```

Writing Trigger Procedures with PL-PGSQL

Some variables that are often needed:

- **NEW** = Holds the contents of the row to insert or update.
- **OLD** = Holds the contents of the original row.
- **TG_NARGS** = Number of input arguments passed on to the trigger procedure.
- **TG_ARGV[ ]** = Text array containing the arguments, accessed as $1, $2, etc.

Writing Trigger Procedures with PL-PGSQL

Testing the Trigger Procedure

- Go to the Edit Data window of the Sailors table.
- Try to update Peter's age to 51. What do you get?
- Now update it to 23. Was the operation successful?

References

- **Triggers in PostgreSQL**

  http://www.postgresql.org/docs/8.1/static/triggers.html

- **Quick Intro to PL-PGSQL**

  http://www.codeproject.com/KB/database/howto_write_pl_pgsql_func.aspx

- **Basic PL-PGSQL statements**

  http://developer.postgresql.org/pgdocs/postgres/plpgsql-statements.html

- **Creating trigger procedures with PL-PGSQL**

  http://www.postgresql.org/docs/8.1/static/plpgsql-trigger.html