Lecture Notes for Lecture 7 of CS 5200 (Database Management System) for the Summer 1, 2019 session at the Northeastern University Silicon Valley Campus.

Foreign Key Constraints and Triggers

Philip Gust,
Clinical Instructor
Department of Computer Science

Lecture 6 Review

- Once tables are created, they should be tested using representative samples of actual data sets
- Input data is often “dirty”, and must be ”scrubbed” first to avoid adding incorrect or inconsistent data in the database.
- SQL supports integrity constraints including key, domain, and referential integrity constraints.
- Six types of integrity constraints were presented, with examples of how to specify each in the schema, and to add, alter, or drop them from existing tables.
- In many cases, constraints can be named to make it easier to alter or drop them afterwards.
- Auto-increment fields were also presented as a way to create a unique key for a table row.

Today’s Topics

- In this lecture we will discuss two conditional mechanisms for automating operations in a database:
- Foreign key constraints enable the database to perform simple actions as table rows are added, altered and removed.
- Triggers enable the database to perform more general actions to ensure that constraints are maintained as table rows are added, altered and removed.

Foreign Key Constraints and Triggers

- A *foreign key constraint* is a mechanism that can be specified as part of a foreign key. It specifies an action for the database system to perform when a row is updated, or dropped.
- Foreign key constraints enable the database to perform simple actions to ensure that constraints are maintained.
- Foreign key constraints are useful for ensuring referential integrity of 1:1 a 1:N relations.

Foreign Key Constraints and Triggers

- Consider again the structure of the following two tables.

```
CREATE TABLE Customer (
        id INT NOT NULL,
        name VARCHAR (20) NOT NULL,
        age INT NOT NULL CHECK (age >= 18),
        address CHAR (25) ,
        PRIMARY KEY (id)
)

CREATE TABLE Orders (
        id INT NOT NULL,
        date DATETIME,
        amount double NOT NULL (amount >= 0.00),
        customer_id INT;
        PRIMARY KEY (id)
        FOREIGN KEY(customer_id) references Customers(id),
)
```

Foreign Key Constraints and Triggers

- Here is a version of Orders with a foreign key constraint that specifies what happens to orders associated with a customer if the customer entry is deleted:

```
CREATE TABLE Orders (
        id INT NOT NULL,
        amount double NOT NULL (amount >= 0.00),
        customer_id INT;
        PRIMARY KEY (id)
        FOREIGN KEY(customer_id) references Customers(id) ON DELETE CASCADE,
)
INSERT INTO Customers VALUES (1, ‘Joe’, 42, ‘123 Main St.’),
INSERT INTO Customers VALUES (2, ’Mary’, 19, ’12 First St.’);
INSERT INTO Orders(ref) VALUES (100,  25.10, 1)
INSERT INTO Orders(ref) VALUES (101, 102.57, 2);

DELETE FROM Customers WHERE id = 1;
SELECT * FROM Orders;
--> will return only the row (id: 101, amount: 102.57, customer_id: 2)
```

Foreign Key Constraints and Triggers

Referential actions

- You can specify an ON DELETE clause and/or an ON UPDATE clause, followed by the appropriate action (CASCADE, RESTRICT, SET NULL, or NO ACTION) when defining foreign keys.
- These clauses specify whether Derby should modify corresponding foreign key values or disallow the operation, to keep foreign key relationships intact when a primary key value is updated or deleted from a table
- You specify the update and delete rule of a referential constraint when you define the referential constraint.

Foreign Key Constraints and Triggers

On UPDATE { NO ACTION | RESTRICT }

- The update rule applies when a row of either the parent or dependent table is updated. The choices are NO ACTION and RESTRICT.
- If the update rule is NO ACTION, checks the dependent tables for foreign key constraints *after* all deletes have been executed but *before* triggers have been executed.
- When a value in a column of the parent table's primary key is updated and the update rule has been specified as RESTRICT, checks dependent tables for foreign key constraints.

Foreign Key Constraints and Triggers

On UPDATE { NO ACTION | RESTRICT }

- If any row in a dependent table violates a foreign key constraint, the statement is rejected.
- When a value in a column of the dependent table is updated, and that value is part of a foreign key, NO ACTION is the implicit update rule.
- NO ACTION means that if a foreign key is updated with a non-null value, the update value must match a value in the parent table's primary key when the update statement is completed.
- If the update does not match a value in the parent table's primary key, the statement is rejected.

Foreign Key Constraints and Triggers

On DELETE { NO ACTION | RESTRICT | CASCADE | SET NULL }

- The delete rule applies when a row of the parent table is deleted and that row has dependents in the dependent table of the referential constraint.
- If rows of the dependent table are deleted, the delete operation on the parent table is said to be *propagated* to the dependent table.
- If the dependent table is also a parent table, the action specified applies, in turn, to its dependents.
- SET NULL can be specified only if some column of the foreign key allows null values.

Foreign Key Constraints and Triggers

On DELETE { NO ACTION | RESTRICT | CASCADE | SET NULL }

- If the delete rule is:
  - NO ACTION: Derby checks the dependent tables for foreign key constraints *after* all deletes have been executed but *before* triggers have been executed. If any row in a dependent table violates a foreign key constraint, the statement is rejected.
  - RESTRICT: Derby checks dependent tables for foreign key constraints. If any row in a dependent table violates a foreign key constraint, the statement is rejected.
  - CASCADE: The delete operation is propagated to the dependent table (and that table's dependents, if applicable).
  - SET NULL: Each nullable column of the dependent table's foreign key is set to null. (Again, if the dependent table also has dependent tables, nullable columns in those tables' foreign keys are also set to null.)

Foreign Key Constraints and Triggers

On DELETE { NO ACTION | RESTRICT | CASCADE | SET NULL }

- Each referential constraint in which a table is a parent has its own delete rule; all applicable delete rules are used to determine the result of a delete operation.
- Thus, a row cannot be deleted if it has dependents in a referential constraint with a delete rule of RESTRICT or NO ACTION.
- Similarly, a row cannot be deleted if the deletion cascades to any of its descendants that are dependents in a referential constraint with the delete rule of RESTRICT or NO ACTION.

Foreign Key Constraints and Triggers

On DELETE { NO ACTION | RESTRICT | CASCADE | SET NULL }

- Deleting a row from the parent table involves other tables. Any table involved in a delete operation on the parent table is said to be delete-connected to the parent table.
- The delete can affect rows of these tables in the following ways:
    - If the delete rule is RESTRICT or NO ACTION, a dependent table is involved in the operation but is not affected by the operation. (That is, Derby checks the values within the table, but does not delete any values.)
    - If the delete rule is SET NULL, a dependent table's rows can be updated when a row of the parent table is the object of a delete or propagated delete operation.
    - If the delete rule is CASCADE, a dependent table's rows can be deleted when a parent table is the object of a delete.
    - If the dependent table is also a parent table, the actions described in this list apply, in turn, to its dependents.

Foreign Key Constraints and Triggers

Foreign key constraints can work for simple 1:N relations. A more general mechanism is needed for more general constraint management.

- A ***trigger*** defines a set of actions that are executed when a database event occurs on a specified table. A *database event* is a delete, insert, or update operation. For example, if you define a trigger for a delete on a particular table, the trigger's action occurs whenever someone deletes a row or rows from the table.
- Along with constraints, triggers can help enforce data integrity rules with actions such as cascading deletes or updates.
- Triggers can also perform a variety of functions such as issuing alerts, updating other tables, sending e-mail, and other useful actions.
- You can define any number of triggers for a single table, including multiple triggers on the same table for the same event.
- You can create a trigger in any schema except one that starts with *SYS*. The trigger need not reside in the same schema as the table on which it is defined.

Foreign Key Constraints and Triggers

Syntax

CREATE TRIGGER TriggerName

{ AFTER | NO CASCADE BEFORE }

{ INSERT | DELETE | UPDATE } [OF *column-Name* [, column-Name]*]

ON *table-Name*

[*ReferencingClause*]

FOR EACH { ROW | STATEMENT } MODE DB2SQL

Triggered-SQL-statement

- Triggers are defined as either *Before* or *After* triggers.
  - Before triggers fire before the statement's changes are applied and before any constraints have been applied. Before triggers can be either *row* or *statement triggers*.
  - After triggers fire after all constraints have been satisfied and after the changes have been applied to the target table. After triggers can be either *row* or *statement triggers*.

Foreign Key Constraints and Triggers

Statement vs. row triggers

- You have the option to specify whether a trigger is a *statement trigger* or a *row trigger.*
- If it is not specified in the CREATE TRIGGER statement via FOR EACH clause, then the trigger is a *statement trigger* by default
  - *statement triggers.* A statement trigger fires once per triggering event and regardless of whether any rows are modified by the insert, update, or delete event.
  - *row triggers.* A row trigger fires once for each row affected by the triggering event. If no rows are affected, the trigger does not fire.

  Note: An update that sets a column value to the value that it originally contained (for example, UPDATE T SET C = C) causes a row trigger to fire, even though the value of the column is the same as it was prior to the triggering event.

Foreign Key Constraints and Triggers

Triggered SQL statements

- The action defined by the trigger is called the triggered-SQL-statement. It has the following limitations:
    - It must not contain any dynamic parameters (?).
    - It must not create, alter, or drop the table upon which the trigger is defined.
    - It must not add an index to or remove an index from the table on which the trigger is defined.
    - It must not add a trigger to or drop a trigger from the table upon which the trigger is defined.
    - It must not commit or roll back the current transaction or change the isolation level.
    - Before triggers cannot have INSERT, UPDATE or DELETE statements as their action

Foreign Key Constraints and Triggers

Triggered SQL statements

- The triggered-SQL-statement can reference database objects other than the table upon which the trigger is declared. If any of these database objects is dropped, the trigger is invalidated.
- If the trigger cannot be successfully recompiled upon the next execution, the invocation throws an exception and the statement that caused it to fire will be rolled back.
- When a database event occurs that fires a trigger, actions are performed in this order:
  - It fires *No Cascade Before* triggers.
  - It performs constraint checking (primary key, unique key, foreign key, check).
  - It performs the insert, update, or delete.
  - It fires *After* triggers.

Foreign Key Constraints and Triggers

Example: Airline Flights

```
-- use a check constraint to allow only appropriate
-- abbreviations for the meals
CREATE TABLE FLIGHTS (
        FLIGHT_ID CHAR(6) NOT NULL ,
        SEGMENT_NUMBER INTEGER NOT NULL ,
        ORIG_AIRPORT CHAR(3),
        DEPART_TIME TIME,
        DEST_AIRPORT CHAR(3),
        ARRIVE_TIME TIME,
        MEAL CHAR(1) CONSTRAINT MEAL_CONSTRAINT
                CHECK (MEAL IN ('B', 'L', 'D', 'S')),
        PRIMARY KEY (FLIGHT_ID, SEGMENT_NUMBER)
);
```

Foreign Key Constraints and Triggers

Example: Airline Flights

```
-- create a table with a table-level primary key constraint
-- and a table-level foreign key constraint
CREATE TABLE FLTAVAIL (
        FLIGHT_ID CHAR(6) NOT NULL,
        SEGMENT_NUMBER INT NOT NULL,
        FLIGHT_DATE DATE NOT NULL,
        ECONOMY_SEATS_TAKEN INT,
        BUSINESS_SEATS_TAKEN INT,
        FIRSTCLASS_SEATS_TAKEN INT,
        CONSTRAINT FLTAVAIL_PK PRIMARY KEY (FLIGHT_ID, SEGMENT_NUMBER),
        CONSTRAINT FLTS_FK FOREIGN KEY (FLIGHT_ID, SEGMENT_NUMBER)
        REFERENCES Flights (FLIGHT_ID, SEGMENT_NUMBER)
);
```

Foreign Key Constraints and Triggers

Example: Airline Flights

```
CREATE TRIGGER t1
    AFTER UPDATE ON x
    FOR EACH ROW MODE DB2SQL
            values app.notifyEmail('Jerry', 'Table x has been updated');

CREATE TRIGGER FLIGHTSDELETE
    AFTER DELETE ON FLIGHTS
    REFERENCING OLD_TABLE AS DELETEDFLIGHTS
    FOR EACH STATEMENT MODE DB2SQL
            DELETE FROM FLIGHTAVAILABILITY WHERE FLIGHT_ID IN
            (SELECT FLIGHT_ID FROM DELETEDFLIGHTS);

CREATE TRIGGER FLIGHTSDELETE3
    AFTER DELETE ON FLIGHTS
    REFERENCING OLD AS OLD
    FOR EACH ROW MODE DB2SQL
            DELETE FROM FLIGHTAVAILABILITY WHERE FLIGHT_ID = OLD.FLIGHT_ID;
```

Foreign Key Constraints and Triggers

Referencing transition variables and tables

- Triggered-SQL-statements often need to refer to data that is currently being changed by the database event that caused them to fire. For example, the triggered-SQL-statement might need to refer to the new (post-change or "after") values.
- There are a number of ways to refer to data that is currently being changed by the database event that caused the trigger to fire.
- Changed data can be referred to in the triggered-SQL-statement using *transition variables* or *transition tables*.
- The referencing clause allows you to provide a correlation name or alias for these transition variables by specifying OLD/NEW AS *correlation-Name* .

Foreign Key Constraints and Triggers

Referencing transition variables and tables

- For statement triggers, transition *tables* serve as a table identifier for the triggered-SQL-statement or the trigger qualification.
- The referencing clause allows you to provide a correlation name or alias for these transition tables by specifying OLD_TABLE/NEW_TABLE AS correlation-Name
- The referencing clause can designate only one new correlation or identifier and only one old correlation or identifier. Row triggers cannot designate an identifier for a transition table and statement triggers cannot designate a correlation for transition variables.

Foreign Key Constraints and Triggers

Using triggers for many-many relationships

- Triggers can be used to implement constraints between two tables that are in a many-to-many relationship through an intermediate relation table.
- For example, if the PublishedBy relationship between Publisher and a Journal is represented as a table, deleting a publisher from the Publisher table should cause all the Journals for the Publisher to be deleted, as well as the relationship entry.
- Deleting the PublishedBy entry for the deleted journal could be handled by a normal ON DELETE row constraint on the PublishedBy table.
- Similarly, deleting a journal from the Journal table should cause all articles in the Articles table to be deleted. as well the entries in PublishedBy and WrittenBy.
- Article authors in the Author table should not be deleted (unless perhaps there are no other articles by the author).

Foreign Key Constraints and Triggers

Finding Journals for a Publisher

- If the PublishedBy 1:n relationship is represented by an entry in the Journal table, deletion will occur by including an ON DELETE constraint on the Journal row.
- We could also find all Journals for the Publisher using a simple left inner join between the tables on the PublishedBy foreign key column.

```
select Publisher.Name as Name, Journal.ISSN as ISSN from Publisher
left join  Journal on Publisher.Name= Journal.PublishedBy
```

- If PublishedBy is represented as a relationship table, we need to use a query in the Trigger to select all the journals for the publisher being deleted.

```
select Publisher.Name as Name, Journal.ISSN as ISSN from Publisher
left join PublishedBy on Publisher.Name = PublishedBy.PublisherName
left join Journal on PublishedBy.ISSN = Journal.ISSN
```