Constraints and Triggers
Constraints and Triggers

- **Constraint:**
  - relationship among data elements
  - DBMS should enforce the constraints
  - Example: key constraints

- **Triggers:**
  - Actions that are executed when a specified condition occurs
  - Easier to implement than many constraint
  - Example: insert a tuple
Constraints

- Keys
- Foreign-key (referential integrity)
- Value-based constraints
- Tuple-based constraints
- Assertions (SQL boolean expression)
Foreign Keys

- Requires that values for certain attributes must appear in other relations.

- Example: Dog(name, breed, Owner) the the value for the owner of the dog must appear in the Owner relation.
Foreign keys

- Keyword REFERENCES either
  - Within the declaration of an attribute when only one attribute involved OR
  - As an element of the schema

- Example:
  FOREIGN KEY (<attributes>) REFERENCES <relation> (<attributes>);

- Referenced attributes must be declared as PRIMARY KEY
Example

Dog(name,breed,owner)

CREATE TABLE Owner (  
   name CHAR (20) PRIMARY KEY,  
   SSN INT,  
   phone CHAR (20));

CREATE TABLE Dog (  
   name CHAR(20) PRIMARY KEY,  
   breed CHAR(10),  
   owner CHAR(20) REFERENCES Owner(name) );
Example

Dog(name, breed, owner)

CREATE TABLE Owner (  
    name CHAR (20) PRIMARY KEY,  
    SSN INT,  
    phone CHAR (20));

CREATE TABLE Dog (  
    name CHAR(20) PRIMARY KEY ,  
    breed CHAR(10),  
    owner CHAR(20),  
    FOREIGN KEY (owner) REFERENCES Owner(name) );
Enforcing Foreign-Key Constraints

- If there is a foreign-key constraints form attributes $A_1, \ldots, A_n$ of a relation $R$ to the primary-key $B_1, \ldots, B_m$ of an other relation $S$, then possible violations:
  - Insert or update on $R$ may introduce values not in $S$
  - Deletion or update on $S$ may remove values needed for tuples in $R$
- Tuples of $R$ without a matching primary-key in $S$ are called dangling tuples
Actions to prevent Foreign-Key Violations

- Insertions or updates that would create a dangling tuple must be REJECTED
- Example: If a new dog is inserted into the Dog relation before the owner’s data is inserted into the Owner relation.
Actions for Deleting/Modifying Tuples Needed for Foreign-Key

- Three possible ways to handle:
  1. Default: REJECT the modification
  2. Cascade: make the same changes in the referencing relation
  3. Set NULL: change the referencing attribute to NULL
Example: Cascade

- Suppose owner ‘Alexandra Smith’ is deleted from Owner relation
  - Delete all tuples from Dog where the owner attribute value is ‘Alexandra Smith’

- Suppose Alexandra Smith wants to update her name to ‘Alexandra Ray’
  - Change the owner attribute values of all tuples in Dog from ‘Alexandra Smith’ to ‘Alexandra Ray’
Example: Set NULL

- Suppose owner ‘Alexandra Smith’ is deleted from Owner relation
  - Change all tuples from Dog where the owner attribute value is ‘Alexandra Smith’ to owner=NULL

- Suppose Alexandra Smith wants to update her name to ‘Alexandra Ray’ (same as before)
  - Change the owner attribute values of all tuples in Dog from ‘Alexandra Smith’ to ‘Alexandra Ray’
Choosing a Policy

- When declaring a foreign key, policy can be set independently for deletions and updates
- If not declared then default is used
- Example:

```sql
CREATE TABLE Dog (  
  name CHAR(20) PRIMARY KEY ,
  breed CHAR(10),
  owner CHAR(20),
  FOREIGN KEY (owner) REFERENCES Owner(name)
  ON DELETE SET NULL);
```
Attribute-Based Constraints

- Constraint the value of a particular attribute
- CHECK(<condition>) is added to the declaration of the attribute
- Condition may use the name of the attribute or any other relation or attribute name may be in a sub-query
CREATE TABLE Dog ( 
  name CHAR(20) PRIMARY KEY , 
  breed CHAR(10), 
  owner CHAR(20) CHECK (owner IN 
    (SELECT name FROM Owner)), 
  weight REAL CHECK (0 <weight AND 
    weight < 120) );
Timing of Checks

- Attribute value check is checked only when the value of the attribute is inserted or updated.

- Example:
  - CHECK (0 < weight AND weight < 120) is verified every time a new weight value is inserted/updated the Dog database.
  - CHECK (owner IN (SELECT name FROM Owner)) is not checked when an owner is deleted from Owner – NOT LIKE FOREIGN KEY.
Tuple-Based Checks

- Check (<condition>) may be added during schema definition.
- Condition may refer to any attribute of the relation but other relations and their attributes require sub-queries.
- Checked during insert or update.
CREATE TABLE Dog (  
    name CHAR(20) PRIMARY KEY ,  
    breed CHAR(10),  
    owner CHAR(20),  
    weight REAL,  
    CHECK (owner = 'Alexandra Smith' OR  
        breed = 'G.S.'));
Assertions

- Holds on database-schema elements like relations and views
- Must always be true
- Condition may refer to any relation or attribute in the database schema
- CREATE ASSERTION <name>
  CHECK (<condition>);
Example

- In Dog relation(name, breed, weight, owner) tiny dogs selected

CREATE ASSERTION TinyDogs
CHECK NOT EXISTS (SELECT name
FROM Dog
WHERE weight < 12 );
Example

In Owners and Dogs cannot be more owners than dogs.

CREATE ASSERTION Few-owners CHECK
(
  (SELECT COUNT (*) FROM Owner) <=
  (SELECT COUNT (*) FROM Dog) );
Timing Assertion

- In general, check every assertion after every modification to any relation of the database
- Clever system: only certain changes can cause a given assertion to be violated → check only after these changes
Triggers

- Attribute and tuple-based checks → limited in capabilities
- Assertions: general and powerful but difficult to implement efficiently
- Triggers:
  - Allows the user to specify when the check occurs.
  - General purpose conditions and sequence of SQL database modifications
Triggers

- Also called event-condition-action (ECA) rules
  - Event: typically a type of database modification
  - Condition: and SQL boolean-valued expression
  - Action: any SQL statement
Example

- Instead of using foreign-key constraints to reject an insertion of a dog into Dog if the owner is not present in Owner, use trigger to insert the same owner into Owner with NULL for phone and SSN.

- CREATE TRIGGER OWNR
  AFTER INSERT ON Dog
  REFERENCING NEW ROW AS NewTuple
  FOR EACH ROW
  WHEN (NewTuple.owner NOT IN
       (SELECT name FROM Owner))
  INSERT INTO Owner(name)
       VALUES(NewTuple.owner);
Options: Create Trigger

CREATE TRIGGER <name>
Option:
CREATE OR REPLACE TRIGGER <name>
useful to modify existing trigger

CREATE TRIGGER OWNR
AFTER INSERT ON Dog
REFERENCING NEW ROW AS NewTuple
FOR EACH ROW
WHEN (NewTuple.owner NOT IN
(SELECT name FROM Owner))
INSERT INTO Owner(name)
VALUES(NewTuple.owner);
Options: Condition

- AFTER can be BEFORE or
  - INSTEAD OF for views (can be used to execute view modifications and translate them to modifications on the base relations)
- INSERT and be DELETE or UPDATE or UPDATE ... ON a particular attribute

CREATE TRIGGER OWNR
  AFTER INSERT ON Dog
  REFERENCING NEW ROW AS NewTuple
  FOR EACH ROW
  WHEN (NewTuple.owner NOT IN
        (SELECT name FROM Owner))
  INSERT INTO Owner(name)
  VALUES(NewTuple.owner);
Options: For Each Row

- **Triggers:**
  - Row-level
  - Statement-level

- **FOR EACH ROW** indicates row-level, its absence indicates statement-level

- Row-level triggers executed once for each modified tuple

- Statement-level triggers executed once for an SQL statement, regardless of the number of modified tuples

```sql
CREATE TRIGGER OWNR
  AFTER INSERT ON Dog
  REFERENCING NEW ROW AS NewTuple
  FOR EACH ROW
  WHEN (NewTuple.owner NOT IN (SELECT name FROM Owner))
  INSERT INTO Owner(name) VALUES(NewTuple.owner);
```
Options: Referencing

- INSERT statement implies a new tuple (row-level) or a new set of tuples (statement-level)
- DELETE implies and old tuple or table
- UPDATE implies both
- Format of reference:
  [NEW OLD] [TUPLE TABLE] AS <name>

CREATE TRIGGER OWNR
  AFTER INSERT ON Dog
  REFERENCING NEW ROW AS NewTuple
  FOR EACH ROW
  WHEN (NewTuple.owner NOT IN (SELECT name FROM Owner))
  INSERT INTO Owner(name)
  VALUES(NewTuple.owner);
Options: Condition

- Any boolean-valued condition is appropriate
- Evaluated before or after the triggering event, depending on whether BEFORE or AFTER was used
- Access the new/old tuples or set of tuples through names declared in the REFERENCING clause

```
CREATE TRIGGER OWNR
    AFTER INSERT ON Dog
    REFERENCING NEW ROW AS NewTuple
    FOR EACH ROW
    WHEN (NewTuple.owner NOT IN (SELECT name FROM Owner))
    INSERT INTO Owner(name)
        VALUES(NewTuple.owner);
```
Options: Action

- More than one SQL statements are allowed in action
- Surround be BEGIN ... END if there is more than one
- Action: modification

```sql
CREATE TRIGGER OWNR
    AFTER INSERT ON Dog
    REFERENCING NEW ROW AS NewTuple
    FOR EACH ROW
    WHEN (NewTuple.owner NOT IN
          (SELECT name FROM Owner))
    INSERT INTO Owner(name)
    VALUES(NewTuple.owner);
```
Triggers on Views

- Generally, it is impossible to modify a view because it does not exist.
- INSTEAD OF trigger lets us interpret view modifications.
Example

- Owner(name, phone, address)
- Owns(O.name, D.name, D.breed)
- Dog(name, age, weight, breed)
- Create a view with the owner’s name and phone, and the dog’s name and weight

CREATE VIEW dog-and-owner AS
(SELECT o.name, o.phone, d.name, weight
FROM Owner o, Owns, Dog d
WHERE o.name = O.name and d.name = D.name)
Example

CREATE TRIGGER View-Update
INSTEAD OF INSERT ON dog-and-owner
REFERENCING NEW ROW AS n
FOR EACH ROW
BEGIN
  INSERT INTO Dog(name,weight) VALUES(n.d.name, weight);
  INSERT INTO OWNER(name,phone) VALUES(n.o.name, n.o.phone);
  INSERT INTO OWNS(O.name,D.name) VALUES(n.o.name, n.o.phone);
END;