Lecture 5

Database Objects

(part 1)

- Constraints in Relational Algebra
- Keys and Foreign Keys
- Constraints on Attributes and Tuples
- Modification of Constraints
- Assertions
- Triggers

Relational Algebra as a Constraint Language

- Two ways
  - R is an expression of relational algebra

    $R = \emptyset$

    - The value of R must be empty
    - There are no tuples in the result of R
  - R and S are expressions of relational algebra

    $R \subseteq S$

    - Every tuple in the result of R must also be in the result of S

Referential Integrity Constraints

- Referential integrity constraint 参照完整性

$$\pi_A(R) \subseteq \pi_B(S)$$

  - If we have any value *v* as the component in attribute *A* of some tuple in one relation *R*, then because of our design intentions we may expect that *v* will appear in a particular component, say for attribute *B*, of some tuple of another relation *S*

Referential Integrity Constraints (Cont'd)

- Example

  Movies (title, year, length, genre, studioName, producerC#)

  MovieExec (name, address, cert#, netWorth)

$$\pi_{producerC\#}(Movies) \subseteq \pi_{cert\#}(MovieExec)$$

  - The producerC# of each Movies tuple must also appear in the cert# of some MovieExec tuple

Referential Integrity Constraints (Cont'd)

- Example

  StarsIn (movieTitle, movieYear, starName)

  Movies (title, year, length, genre, studioName, producerC#)

$$\pi_{movieTitle,movieYear}(StarsIn) \subseteq \pi_{title,year}(Movies)$$

Key Constraints

- Example

  MovieStar (name, address, gender, birthdate)

$$\sigma_{MS1.name=MS2.name \text{ AND } MS1.address \neq MS2.address}(MS1 \times MS2) = \varnothing$$

  – No two tuples agree on the *name* component

    - If two tuples agree on *name*, then they must also agree on *address*

    - MS1 is shorthand for the renaming

$$\rho_{MS1(name,address,gender,birthdate)}(MovieStar)$$

Additional Constraint Examples

- Example

  MovieStar (name, address, gender, birthdate)

$$\sigma_{gender \neq 'F' \text{ AND } gender \neq 'M'}(MovieStar) = \varnothing$$

  - The set of tuples in *MovieStar* whose *gender* is equal to neither ‘F’ nor ‘M’ is empty

Additional Constraint Examples (Cont'd)

- Example

  MovieExec (name, address, cert#, netWorth)

  Studio (name, address, presC#)

$$\sigma_{netWorth<10000000}(Studio \bowtie_{presC\#=cert\#} MovieExec) = \varnothing$$

  - One must have a net worth of at least $10,000,000 to be the president of a movie studio

$$\pi_{presC\#}(Studio) \subseteq \pi_{cert\#}(\sigma_{netWorth\geq 10000000}(MovieExec))$$

Constraints and Triggers

- A *constraint* is a relationship among data elements that the DBMS is required to enforce.
  - Example: key constraints.
- *Triggers* are only executed when a specified condition occurs, e.g., insertion of a tuple.
  - Easier to implement than complex constraints.

Kinds of Constraints

- Keys, (Primary key, Unique Key).
  - Uniqueness of value for a particular attribute or attributes
- Foreign-key, or referential-integrity.
- Attribute-based constraints.
  - Constrain values of a particular attribute.
  - Not-Null Constraints
- Tuple-based constraints.
  - Relationship among components.
- Assertions: any SQL boolean expression.

Review: Single-Attribute Keys

- Place PRIMARY KEY or UNIQUE after the type in the declaration of the attribute.
- Example:

```
CREATE TABLE movieexec (
    name      CHAR(30),
    address   VARCHAR(255),
    cert      INT       PRIMARY KEY,
    netWorth INT
);
```

Review: Multiattribute Key

- The title and year together are the key for movies:

```
CREATE TABLE movies (
    title       CHAR(100),
    year        INT,
    length      INT,
    genre       CHAR(10),
    studioName  CHAR(30),
    producerC   INT,
    PRIMARY KEY (title, year)
);
```

Foreign Keys

- Values appearing in attributes of one relation must appear in certain attributes of another relation.
- Example: in studio(name, address, presC), we might expect that a presC value also appears in movieexec.cert

Expressing Foreign Keys

- To declare a Foreign Key, Use keyword REFERENCES, either:
  - With Attribute : After an attribute (for one-attribute keys).

    REFERENCES <relation> (<attributes>)
  - As an element of the schema:

    FOREIGN KEY (<list of attributes>)

    REFERENCES <relation> (<attributes>)
- Referenced attributes must be declared PRIMARY KEY or UNIQUE.

Example: With Attribute

```
CREATE TABLE movieexec (
    name        CHAR(30),
    address     VARCHAR(255),
    cert        INT         PRIMARY KEY,
    netWorth    INT
);
CREATE TABLE studio (
    name        CHAR(50)   PRIMARY KEY,
    address     VARCHAR(255),
    presC       INT         REFERENCES movieexec(cert)
);
```

Example: As Schema Element

```
CREATE TABLE movieexec (
    name          CHAR(30),
    address       VARCHAR(255),
    cert          INT           PRIMARY KEY,
    netWorth      INT
);
CREATE TABLE studio (
    name          CHAR(50)   PRIMARY KEY,
    address       VARCHAR(255),
    presC         INT,
    FOREIGN KEY(presC) REFERENCES movieexec(cert)
);
```

Essence behind of Foreign Keys

studio → movieexec

- Values appearing in attributes of `studio.presC` must appear in `movieexec.cert`.

[studio.presC]⊆[movieexec.cert]+NULL

- Modification on **studio** and **movieexec**

	studio	**movieexec**
Delete	OK!	?
Insert	?	OK!
Update	?	?

Enforcing Foreign-Key Constraints

- If there is a foreign-key constraint from relation **`studio` (Referencing)** to relation **`movieexec` (Referenced)**, two violations are possible:
  1. An insert or update to **`studio`** introduces values not found in **`movieexec.`**
  2. A deletion or update to **`movieexec`** causes some tuples of **`studio`** to “dangle.”

Actions Taken --- (1)

- An insert or update to studio that introduces a nonexistent executive in movieexec must be

Rejected!

Actions Taken --- (2)

A deletion or update to movieexec that removes a presC value found in some tuples of studio can be handled in three ways.

1. *Default* : Reject the modification.
2. *Cascade* : Make the same changes in studio.
   - Deleted executive: delete studio tuple.
   - Updated executive: change value in studio.
3. *Set NULL* : Change the studio.`presC` to NULL.

Example: Cascade

- Delete the cert=123 tuple from movieexec:
  - Then delete all tuples from studio that have presC=123.
- Update the cert=345 tuple by changing 345 to 346:
  - Then change all studio tuples with presC=345 to presC=346.

Example: Set NULL

- Delete the cert=123 tuple from movieexec:
  - Change all tuples of studio that have presC=123 to have presC=NULL.
- Update the cert=345 tuple by changing 345 to 34:
  - Change all tuples of studio that have presC=345 to have presC=NULL.

Choosing a Policy

- When we declare a foreign key, we may choose policies SET NULL or CASCADE independently for deletions and updates.
- Follow the foreign-key declaration by:

  ON [UPDATE, DELETE]

  [SET NULL, CASCADE]
- Two such clauses may be used.
- Otherwise, the default (reject) is used.

Example: Setting Policy

```
CREATE TABLE studio (
        name CHAR(50)  PRIMARY KEY,
        address      VARCHAR(255),
        presC        INT
        REFERENCES movieexec(cert)
        ON DELETE SET NULL
        ON UPDATE CASCADE
);
```

Deferred Checking of Constraints

- The situation:
  - Arnold Schwarzenegger decides to found a movie studio, called La Vista, of which he is the president
  - violate the foreign-key constraint

```
INSERT INTO studio
 VALUES ('La Vista', 'New York' , 23456)
```

We are in trouble because there is no tuple of movieexec with certificate number 23456

Solution

1) First to insert the tuple for `'La Vista'` without a president's certificate

```
INSERT INTO studio(name, address)
    VALUES ('La Vista','New York') ;
```

Set **presC as** Null

2) Then insert a tuple for Arnold Schwarzenegger into movieexec

3) Update **Studio**

```
UPDATE Studio
   SET presC# = 23456
   WHERE name = 'La Vista';
```

Any problem for this solution?

1、studio.`presC` cannot be PK

2、Studio.`presC` cannot have a Not-Null Constraint

Circular Constraints

- There are cases of circular constraints
  - That cannot be fixed by judiciously ordering the database modification steps
  - Impossible to insert new studios with new presidents
    - We cannot insert a tuple with a new value of presC into studio. Violate the foreign-key presC→movieexec(cert)
    - We cannot insert a tuple with a new value of cert into movieexec. Violate the foreign-key cert→studio(presC)

Deferred Checking

- First，group several SQL statements (the two insertions - one into studio and the other into movieexec) into a single transaction
- Then tell the DBMS not to check the constraints until after the whole transaction has finished its actions and is about to commit.

```
CREATE TABLE Studio (
  name CHAR(30) PRIMARY KEY,
  address VARCHAR(255),
  presC# INT
      REFERENCES MovieExec (cert#)
      DEFERRABLE INITIALLY DEFERRED    );
```

Deferred Checking

- To inform the DBMS about point (2) in the previous slide
  - The declaration of any constraint may be followed by one of DEFERRABLE or NOT DEFERRABLE (default)
  - NOT DEFERRABLE (default)
    - Every time a database modification statement is executed, the constraint is checked immediately afterwards
  - DEFERRABLE
    - We have the option of having it wait until a transaction is complete before checking the constraint
    - DEFERRABLE INITIALLY DEFERRED
      - Checking will be deferred to just before each transaction commits
    - DEFERRABLE INITIALLY IMMEDIATE
      - The check will be made immediately after each statement

Deferred Checking

- Two additional points about deferring constraints
  - Constraints of any type can be given names
  - If a constraint has a name, say MyConstraint, then we can change a deferrable constraint from immediate to deferred by the SQL statement

```
SET CONSTRAINTS MyConstraint DEFERRED;
```

- And we can reverse the process by replacing DEFERRED in the above to IMMEDIATE

Deferred Checking

- NOT DEFERRABLE (default)
  - 不可延迟，并且约束也无法更改为可延迟状态。约束会在每一句sql statement 之后都进行 check，不符合则 roll back
- DEFERRABLE

  可延迟状态，在 deferrable 状态时又有2个选项
  - DEFERRABLE INITIALLY DEFERRED
    - 约束会在整个事务进行commit 时check，如果不符合则roll back
  - DEFERRABLE INITIALLY IMMEDIATE
    - 即约束会在每一句sql statement 之后都进行 check，效果等同于not deferrable，但是可以修改延迟状态

  在deferrable状态下，可以通过set constraints修改 immediate 或者deferred

OUTLINES

Not-Null Constraints

- Constraints on the value of a particular attribute: NOT NULL .
  - Disallow tuples in which this attribute is NULL

```
CREATE TABLE studio (
    name      CHAR(50) PRIMARY KEY,
    address   VARCHAR(255),
    presC     INT      REFERENCES
              movieexec(cert) NOT NULL
);
```

Not-Null Constraints

- NOT NULL has two consequences:
  - We could not insert a tuple into Studio by specifying only the name and address

```
INSERT INTO studio(name, address) VALUES ('La
Vista','New York');
```

  - We could not use the set-null policy which tells the system to fix foreign-key violations by making presc be NULL

Attribute-Based Checks

- Constraints on the value of a particular attribute.
- Add CHECK(<condition>) to the declaration for the attribute.
  - The condition can be anything that could follow WHERE in a SQL query.
- The condition may use the name of the attribute, but any other relation or attribute name must be in a subquery (not supported in pg).

Example: Attribute-Based Check

```
CREATE TABLE studio (
    name            CHAR(50)     PRIMARY KEY,
    address         VARCHAR(255),
    presc           INT          REFERENCES movieexec(cert)
      CHECK (presc >= 100000)
);
CREATE TABLE moviestar (
    name            CHAR(30) PRIMARY KEY,
    address         VARCHAR(255),
    gender          CHAR(1) CHECK (gender IN ('F', 'M')),
    birthdate       CHAR(10)
);
```

Timing of Attribute-Based Check

- Attribute-based checks are performed **only when a value for that attribute is inserted or updated**.
  - Example: `CHECK (presc >= 100000)` checks every new presc and rejects the modification (for that tuple) if the price is < 1000000.

Attribute-Based Check VS. FK

```
CREATE TABLE studio (
    name            CHAR(50)     PRIMARY KEY,
    address         VARCHAR(255),
    presc           INT,
  CHECK (presc IN (SELECT cert FROM movieexec))
);
```

1、Modification studio.presc NOT IN movieexec.cert Reject! As the effect of FK

2、Modify studio.presc to NULL (if NO NULL exist in movieexec.cert. Impossible! PK) Reject! Not as the effect of FK

3、Modification movieexec（Delete or Update）is **invisible** to the CHECK in studio. Not as the effect of FK. This will result in the CHECK contraint becoming violated.

Tuple-Based Checks

- CHECK (<condition>) may be added as a relation-schema element.
- The condition may refer to any attribute of the relation.
  - But other attributes or relations require a subquery (not supported in pg).

Example: Tuple-Based Check

- This constraint is true for every female movie star and for every star whose name does not begin with ‘Ms.’:

```
CREATE TABLE moviestar (
    name        CHAR(30) PRIMARY KEY,
    address          VARCHAR(255),
    gender      CHAR(1),
    birthdate   CHAR(10),
  CHECK (gender = 'F' OR name NOT LIKE 'Ms.%')
);
```

Timing of Tuple-Based Checks

- Checked every time a tuple is inserted into *R* and every time a tuple of *R* is updated
  - If false for the tuple (violated), and insertion or update is rejected!
  - Like an attribute-based CHECK, a tuple-based CHECK is invisible to other relations
    - Even a deletion from *R* can cause the condition to become false, if *R* is mentioned in a subquery
    - If a tuple-based check does not have subqueries, then we can rely on its always holding

Comparison of Attribute-,Tuple-Based Checks

Type of Constraint	Where Declared	When Activated	Guaranteed to Hold?
Attribute-based CHECK	With attribute	On insertion to relation or attribute update	Not if subqueries
Tuple-based CHECK	Element of relation schema	On insertion to relation or tuple update	Not if subqueries

- If a constraint on a tuple involves more than one attribute of that tuple, then it must be written as a tuple-based constraint
- If the constraint involves only one attribute of the tuple, then it can be written as either a tuple- or attribute-based constraint
  - But the tuple-based constraint will be checked more frequently than the attribute-based constraint. Probably less efficient

Modification of constraints

- Giving name

```
CREATE TABLE moviestar (
    name    CHAR(30) CONSTRAINT nameiskey PRIMARY KEY,
    address VARCHAR(255),
    gender  CHAR(1) CONSTRAINT noandro
            CHECK (gender IN ('F', 'M')),
    birthdate     CHAR(10),
      CONSTRAINT righttitle CHECK (gender = 'F' OR name NOT
LIKE 'Ms.%')
);
```

Modification of constraints

- Altering constraints on tables
  - `SET CONSTRAINT constraintName DEFERRED (or IMMEDIATE);`
  - `ALTER TABLE relationName DROP CONSTRAINT constraintName`
  - `ALTER TABLE relationName ADD CONSTRAINT constraintName CHECK(…)`
- Note
  - The added constraint must be of a kind that can be associated with tuples
    - Tuple-based constraints, key, or foreign-key
  - You cannot add a constraint to a table unless it holds at that time for every tuple in the table

Assertions

- These are database-schema elements, like relations or views.
- Defined by:

  CREATE ASSERTION <name>

  CHECK (<condition>);
- Condition may refer to any relation or attribute in the database schema.
- An assertion is a boolean-valued SQL expression that must be true at all times.
- DROP ASSERTION <name>

Example: Assertion

```
MovieExec (name, address, cert#, networth)
Studio(name, address, presC#)
```

No one can become the president of a studio unless their net worth is at least $10,000,000

```
CREATE ASSERTION RichPres CHECK
   (NOT EXISTS
     (SELECT * FROM studio, movieexec
      WHERE presc = cert AND networth < 10000000)
   );
```

Timing of Assertion Checks

- In principle, we must check every assertion after every modification to any relation of the database.
- A clever system can observe that only certain changes could cause a given assertion to be violated.

Comparison of constraints

Type of Constraint	Where Declared	When Activated	Guaranteed to Hold?
Attribute-based CHECK	With attribute	On insertion to relation or attribute update	Not if subqueries
Tuple-based CHECK	Element of relation schema	On insertion to relation or tuple update	Not if subqueries
Assertion	Element of database schema	On any change to any mentioned relation	Yes

Triggers

- *Triggers*
  - are only awakened when certain events, specified by the database programmer, occur
    - Insert, delete, or update to a particular relation
  - Once awakened by its triggering event, the trigger tests a condition
    - If the condition does not hold, then nothing else associated with the trigger happens
  - If the condition of the trigger is satisfied, the action associated with the trigger is performed by the DBMS

Event-Condition-Action Rules

- Another name for “trigger” is *ECA rule*, or *event-condition-action* rule.
- *Event* : typically a type of database modification
- *Condition* : Any SQL boolean-valued expression.
- *Action* : Any SQL statements.

Triggers in SQL

- The check of the trigger's condition and the action of the trigger may be executed
  - either on the state of the database that exists before the triggering event is itself executed
  - or on the state that exists after the triggering event is executed

Triggers in SQL

- The condition and action can refer to both old and/or new values of tuples that were updated in the triggering event
- It is possible to define update events that are limited to a particular attribute or set of attributes

Triggers in SQL

- The programmer has an option of specifying that the trigger executes either
  - Once for each modified tuple, or
    - Row-level trigger
  - Once for all the tuples that are changed in one SQL statement
    - Statement-level trigger

Example: Trigger Definition

```
CREATE TRIGGER NetWorthTrigger
  AFTER UPDATE OF netWorth ON MovieExec
  REFERENCING
      OLD ROW AS OldTuple,
     NEW ROW  AS  NewTuple
  FOR EACH ROW
  WHEN (OldTuple.netWorth >
         NewTuple.netWorth)
    UPDATE MovieExec
    SET netWorth = OldTuple.netWorth
    WHERE cert = NewTuple.cert;
```

Triggers in PostgreSQL

- The trigger can be specified to fire
  - before the operation is attempted on a row
    - before constraints are checked and
      the INSERT, UPDATE, or DELETE is attempted
  - or after the operation has completed
    - after constraints are checked and
      the INSERT, UPDATE, or DELETE has completed
  - instead of the operation
    - in the case of inserts, updates or deletes on a view

Triggers in PostgreSQL

- If the trigger fires before or instead of the event
  - the trigger can skip the operation for the current row
  - or change the row being inserted
- If the trigger fires after the event
  - all changes, including the effects of other triggers, are "visible" to the trigger.

Triggers in PostgreSQL

- A trigger that is marked
  - FOR EACH ROW is called once for every row that the operation modifies.
  - FOR EACH STATEMENT only executes once for any given operation, regardless of how many rows it modifies
    - an operation that modifies zero rows will still result in the execution of any applicable FOR EACH STATEMENT triggers

Triggers in PostgreSQL

- Triggers that are specified to fire INSTEAD OF the trigger event
  - must be marked FOR EACH ROW
  - and can only be defined on views
- BEFORE and AFTER triggers on a view
  - must be marked as FOR EACH STATEMENT

Triggers in PostgreSQL

- A trigger definition can specify a Boolean WHEN condition
  - which will be tested to see whether the trigger should be fired
  - In row-level triggers theWHEN condition can examine the old and/or new values of columns of the row
  - Statement-level triggers can also have WHEN conditions, although the feature is not so useful for them since the condition cannot refer to any values in the table

Triggers in PostgreSQL

- In FOR EACH ROW triggers
  - the WHEN condition can refer to columns of the old and/or new row values by writing OLD.column_name or NEW.column_name respectively
  - INSERT triggers cannot refer to OLD
  - DELETE triggers cannot refer to NEW
  - INSTEAD OF triggers do not support WHEN conditions

Example : Suppose we want to prevent the average net worth of movie executives from dropping below $500,000. This constraint could be violated by an insertion, a deletion, or an update to the networth column of

MovieExec(name, address, cert#, networth)

```
CREATE TRIGGER AvgNetWorthTrigger
AFTER UPDATE OF networth ON MovieExec
REFERENCING
    OLD TABLE AS OldStuff,
    NEW TABLE AS NewStuff
FOR EACH STATEMENT
WHEN (500000>(SELECT AVG (networth) FROM MovieExec))
BEGIN
    DELETE FROM MovieExec
      WHERE (name,address,cert#,networth) IN Newstuff;
    INSERT INTO MovieExec
         (SELECT * FROM Oldstuff);
END;
```

Example of BEFORE Trigger

Movies(title, year, length, genre, studioName, producerC#)

```
CREATE TRIGGER FixYearTrigger
BEFORE INSERT ON Movies
REFERENCING
    NEW ROW AS NewRow,
    NEW TABLE AS NewStuff
FOR EACH ROW
WHEN NewRow.year IS NULL
UPDATE NewStuff SET year=1915;
```

Summary

- Referential-integrity constraints
- Attribute-based check constraints
- Tuple-based check constraints
- Modifying constraints
- Assertions
- Invoking the checks
- Triggers