# Chapter 7 Constraints and Triggers

- Keys and foreign keys
- Constraints on attributes and tuples
- Modification of constraints
- Assertions
- triggers

# Why use integrity constraints?

- **To catch data-entry errors.**
- **As correctness criteria when writing database updates.**
- **To enforce consistency across data in the database.**
- **To tell the system about the data - it may choose to store the data or process a queries accordingly.**

# 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 many constraints.**

# Types of Constraints

**(1) Non-null, unique**

**(2) Key**

**(3) Referential integrity (Foreign-keys)**

**(4) Attribute-based Check**

**(5) Tuple-based Check**

**(6) General assertions = global constraints**

# Constraints with key, not null and unique

- **key constraints: not null, unique.**
- **Not null constraints: not null.**
- **Unique constraints : can be null, but unique.**
- **Many unique constraints in a table, but only one key constraints.**
- **Key constraint forbids null's in the attributes of the key, but unique permits them.**

# Foreign Keys

**In relation R a clause that "attribute A references S(B)" says that whatever non-null values appear in the A column of R must also appear in the B column of relation S. B must be declared the primary key for S.**

Example:

CREATE TABLE Beers ( name CHAR(20) PRIMARY KEY, manf CHAR(20) );

CREATE TABLE Sells ( bar CHAR (20), beer CHAR(20) REFERENCES Beers(name), price REAL) ;

We expect a beer value is a real beer --- something appearing in Beers.name

# Expressing Foreign Keys

- **Use the keyword REFERENCES, either:**
  1. **Within the declaration of an attribute, when only one attribute is involved.**
  2. **As an element of the schema, as:**

**FOREIGN KEY ( <list of attributes> )**

**REFERENCES <relation>**
**( <attributes> )**

- **Referenced attributes must be declared PRIMARY KEY or UNIQUE.**

# Example: With Attribute

```
CREATE TABLE Beers (
  name    CHAR(20) PRIMARY KEY,
  manf    CHAR(20) );
CREATE TABLE Sells (
  bar     CHAR(20),
  beer    CHAR(20) REFERENCES
  Beers(name),
  price   REAL );
```

# Example: As Element

```
CREATE TABLE Beers (
  name    CHAR(20) PRIMARY KEY,
  manf    CHAR(20) );
CREATE TABLE Sells (
  bar     CHAR(20),
  beer    CHAR(20),
  price   REAL,
  FOREIGN KEY(beer) REFERENCES
     Beers(name));
```

# What happens when a foreign key Constraint is violated ?

**Two ways:**

1. **Insert or update a Sells tuple so it refers to a nonexistent beer → always rejected.**
2. **Delete or update a Beers tuple that has a beer value some Sells tuples refer to**

a) **Default: reject**

b) **Cascade: Ripple changes to referring Sells tuple**

c) **Set null**

What happens when a foreign key Constraint is violated ? (Cascade )

**Example:**

- **Delete “Bud” Cascade deletes all Sells tuples that mention Bud.**
- **Update “Bud” → “Budweiser” change all Sells tuples with “Bud”in beer column to be “Budweiser.”**

# What happens when a foreign key Constraint is violated ? (cont.)

**Set null: Change referring tuples to have null in referring components.**

**Example:**

- **Delete “Bud.” Set-null makes all Sells tuples with “Bud” in the beer component have Null there.**
- **Update “Bud”→ “ Budweiser” same change**

# Selecting a Policy: "Correct" policy is a design decision

**Add ON [DELETE, UPDATE] [CASCADE, SET NULL] to declaration of foreign key.**

**Example**

```
CREATE TABLE Sells (
   bar CHAR (20),  beer CHAR (20),
   price REAL,
  FOREIGN KEY beer REFERENCES
    Beers (name) ON DELETE SET NULLL
    ON UPDATE CASCADE);
```

**Otherwise, the default (reject) is used**

# Attribute-based Checks

**Follow an attribute by a condition that must hold for that attribute in each tuple of its relation.**

- **Form: CHECK (condition)**

1) **Condition may involve *the checked attribute.***
2) ***Other attributes and relations* may be involved, but only in subqueries.**

- **Condition is checked only when the associated attribute changes (i.e., an insert or update occurs)**

# Example

```
CREATE TABLE Sells
 ( bar CHAR (20),
   beer CHAR(20) CHECK ( beer IN
   SELECT name FROM Beers)),
   price REAL CHECK ( price <= 5.00));
```

# Attribute-based Check

- **Effect  when a value for that attribute is inserted or updated.**
  - **Example: CHECK (price <= 5.00) checks every new price and rejects it if it is more than $5.**
  - **Example: CHECK (beer IN (SELECT name FROM Beers)) not checked if a beer is deleted from Beers (unlike foreign-keys).**

# Tuple- Based Checks

**Separate element of table declaration.**

- **Form: like attribute-based check.**
- **Condition can refer to any attribute of the relation. Other relations/attributes require a subquery.**

- **Checked whenever a tuple is inserted or updated.**

# Example

**Only Joe's Bar can sell beer for more than $5.**

```
CREATE TABLE Sells (
  bar CHAR (20),
  beer CHAR (20),
  price REAL,
  CHECK (bar= 'Joe''s Bar' OR price
   <=5.00 ) );
```

# CHECK with and without subquery

```
CREATE TABLE Sells (
    bar CHAR (20),
    beer CHAR(20) CHECK ( beer IN
  SELECT name FROM Beers)),
   price REAL CHECK ( price <= 5.00))
```

Insert or update on Sells **invoke** checks.

Changes on Beers: **nothing happen.**

# SQL Assertions

- **Database-schema constraint.**
- **Condition may refer to any relation or attribute in the database schema. (Not present in Oracle).**
- **Checked whenever a mentioned relation changes.**
- **Syntax:**

**CREATE ASSERTION <name>**

**CHECK (< condition>);**

# Example: No bar may charge an average of more than $5 for beer

**Sells (bar, beer, price)**

```
CREATE ASSERTION NoRipoffbars
CHECK (NOT EXISTS (
        SELECT bar
        FROM   Sells
        GROUP BY bar
      HAVING 5.0 < AVG(price));
```

Bars with an average price above $5

- **Checked whenever Sells changes**

# Example

**There cannot be more bars than drinkers.**

**Bar (name, addr, license)**

**Drinkers (name, addr, phone)**

```
CREATE ASSERTION FewBar
CHECK( (SELECT COUNT(*) FROM Bars)
   <=(SELECT COUNT(*) FROM Drinkers));
```

- **Checked whenever Bars or Drinkers changes.**

# Timing of Assertion Checks

- **In principle, assertion checked every time after every modification.**
- **A clever system can observe that only certain changes could cause a given assertion to be violated.**
  - **Example: No change to Beers can affect FewBar. Neither can an insertion to Drinkers.**

# Comparison of Constraints

| Type of constraints | Where Declared | When Activated | Guarranteed 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 change to any mentioned relation | Yes |

# Modification of Constraints

- **Name your Constraints**

**Example,**

1) **Gender Char(1) CONSTRAINT NoAndro CHECK (gender in ('F','M')),**
2) **Name Char(30) CONSTRAINT NameIsKey PRIMARY KEY,**

# Modification of Constraints(cont.)

- **Altering Constraints on Tables**

**Examples,**

1) **ALTER TABLE Student DROP CONSTRAINT NoAndro;**
2) **ALTER TABLE Student ADD CONSTRAINT NameIskey PRIMARY KEY(name);**

# DEMO
# ABOUT KINDS OF CONSTRAINTS

create table students(sid int primary key, name char[10] not null, dept char[2],age int default 20);

create table courses(cid int primary key, cname char[40], spring boolean, teacher char[10]);

create table sc (**sid** int references students(sid),ON DELETE CASCADE ON UPDATE CASCADE, **cid** int check (cid in(1,2,3,4,5,6,7,8,9)), semester int, cname char[40], grade int);

- insert into students(sid) values (11);

/* rejected: name NOT NULL */

- insert into students(sid,name) values(11,'Dan');

/* default value */

- select cid, cname from courses;
- insert into sc(sid,cid) values (11,11);

/* rejected: check clause */

- Delete from courses where cid=1; /* problems */

- select sid, cid from sc where sid=1;
- delete from students where sid=1;

/* to see all the courses sid=1 chosen has been deleted 级联删除 */

- select sid,cid from sc;