Constraints

FOREIGN KEYS

LOCAL AND GLOBAL CONSTRAINTS

TRIGGERS

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.

Foreign-key, or referential-integrity.

Value-based constraints.

- Constrain values of a particular attribute.

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 Beers (
      name   CHAR(20) UNIQUE,
      manf   CHAR(20)
);
```

Review: Multiattribute Key

The bar and beer together are the key for Sells:

```
CREATE TABLE Sells (
    bar       CHAR(20),
    beer      VARCHAR(20),
    price     REAL,
    PRIMARY KEY (bar, beer)
);
```

Foreign Keys

Values appearing in attributes of one relation must appear together in certain attributes of another relation.

Example: in Sells(bar, beer, price), we might expect that a beer value also appears in Beers.name .

Expressing Foreign Keys

Use keyword REFERENCES, either:

1. After an attribute (for one-attribute keys).
2. 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 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 Schema 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));
```

Enforcing Foreign-Key Constraints

If there is a foreign-key constraint from relation *R* to relation *S*, two violations are possible:

1. An insert or update to *R* introduces values not found in *S*.
2. A deletion or update to S causes some tuples of *R* to "dangle."

Actions Taken --- (1)

Example: suppose *R* = Sells, *S* = Beers.

An insert or update to Sells that introduces a nonexistent beer must be rejected.

A deletion or update to Beers that removes a beer value found in some tuples of Sells can be handled in three ways (next slide).

Actions Taken --- (2)

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

Example: Cascade

Delete the Bud tuple from Beers:

- Then delete all tuples from Sells that have beer = ’Bud’.

Update the Bud tuple by changing ’Bud’ to ’Budweiser’:

- Then change all Sells tuples with beer = ’Bud’ to beer = ’Budweiser’.

Example: Set NULL

Delete the Bud tuple from Beers:

- Change all tuples of Sells that have beer = ’Bud’ to have beer = NULL.

Update the Bud tuple by changing ’Bud’ to ’Budweiser’:

- Same change as for deletion.

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

Attribute-Based Checks

Constraints on the value of a particular attribute.

Add CHECK(<condition>) to the declaration for the attribute.

The condition may use the name of the attribute, but any other relation or attribute name must be in a subquery.

Example: Attribute-Based Check

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

Timing of Checks

Attribute-based checks are performed only when a value for that attribute is inserted or updated.

- Example: `CHECK (price <= 5.00)` checks every new price and rejects the modification (for that tuple) if the price 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

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.

Checked on insert or update only.

Example: Tuple-Based Check

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)
);
```

Assertions

These are database-schema elements, like relations.

Defined by:

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

Condition may refer to any relation or attribute in the database schema.

Example: Assertion

In Sells(bar, beer, price), no bar may charge an average of more than $5.

```
CREATE ASSERTION NoRipoffBars CHECK (
 NOT EXISTS (
        SELECT bar FROM Sells
        GROUP BY bar
        HAVING 5.00 < AVG(price)
 ));
```

Bars with an average price above $5

Example: Assertion

In Drinkers(name, addr, phone) and Bars(name, addr, license), there cannot be more bars than drinkers.

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

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.

- Example: No change to Beers can affect FewBar. Neither can an insertion to Drinkers.

Triggers: Motivation

Assertions are powerful, but the DBMS often cannot tell when they need to be checked.

Attribute- and tuple-based checks are checked at known times, but are not powerful.

Triggers let the user decide when to check for any condition.

Event-Condition-Action Rules

Another name for “trigger” is *ECA rule*, or *event-condition-action* rule.

Event : typically a type of database modification, e.g., “insert on Sells.”

Condition : Any SQL boolean-valued expression.

Action : Any SQL statements.

Preliminary Example: A Trigger

Instead of using a foreign-key constraint and rejecting insertions into Sells(bar, beer, price) with unknown beers, a trigger can add that beer to Beers, with a NULL manufacturer.

Example: Trigger Definition

```
CREATE TRIGGER BeerTrig
AFTER INSERT ON Sells
 REFERENCING NEW ROW AS NewTuple
 FOR EACH ROW
 WHEN (NewTuple.beer NOT IN
        (SELECT name FROM Beers))
 INSERT INTO Beers(name)
        VALUES(NewTuple.beer);
```

The event

The condition

The action

Options: CREATE TRIGGER

CREATE TRIGGER <name>

Or:

REPLACE TRIGGER <name>

- Useful if there is a trigger with that name and you want to modify the trigger.

Options: The Event

AFTER can be BEFORE.

INSERT can be DELETE or UPDATE.

- And UPDATE can be UPDATE . . . ON a particular attribute.

Options: FOR EACH ROW

Triggers are either “row-level” or “statement-level.”

FOR EACH ROW indicates row-level; its absence indicates statement-level.

Row level triggers : execute once for each modified tuple.

Statement-level triggers : execute once for a SQL statement, regardless of how many tuples are modified.

Options: REFERENCING

INSERT statements imply a new tuple (for row-level) or table (for statement-level).

- The “table” is the set of inserted tuples.

DELETE implies an old tuple or table.

UPDATE implies both.

Refer to these by

[NEW OLD][ROW TABLE] AS <name>

Options: The Condition

Any boolean-valued condition.

Evaluated on the database as it would exist before or after the triggering event, depending on whether BEFORE or AFTER is used.

- But always before the changes take effect.

Access the new/old tuple/table through the names in the REFERENCING clause.

Options: The Action

There can be more than one SQL statement in the action.

- Surround by BEGIN . . . END if there is more than one.

But queries make no sense in an action, so we are really limited to modifications.

Another Example

Using Sells(bar, beer, price) and a unary relation RipoffBars(bar), maintain a list of bars that raise the price of any beer by more than $1.

The Trigger

```
CREATE TRIGGER PriceTrig
AFTER UPDATE OF price ON Sells
REFERENCING
        OLD ROW AS ooo
        NEW ROW AS nnn
FOR EACH ROW
WHEN(nnn.price > ooo.price + 1.00)
INSERT INTO RipoffBars
        VALUES(nnn.bar);
```

- AFTER UPDATE OF price ON Sells: The event – only changes to prices
- REFERENCING OLD ROW AS ooo / NEW ROW AS nnn: Updates let us talk about old and new tuples
- FOR EACH ROW: We need to consider each price change
- WHEN(nnn.price > ooo.price + 1.00): Condition: a raise in price > $1
- INSERT INTO RipoffBars VALUES(nnn.bar);: When the price change is great enough, add the bar to RipoffBars