

# CIS4301 Notes:

Ryan Roden-Corrent

February 19, 2014

## 1 Database Modifications

### 1.1 Insert

Listing 1: multiple value insertion

---

```
INSERT INTO Likes
VALUES ('Sally', 'Bud'), ('Jim', 'Miller'); --comma separated tuples to enter
```

---

#### 1.1.1 Default Values

Listing 2: price defaults to 5 if not specified

---

```
...,
price Real DEFAULT 5, --make sure price is a reasonable, non-NULL value
...,
```

---

Listing 3: another default example

---

```
CREATE TABLE Drinkers (
  name CHAR(30) PRIMARY KEY,
  addr CHAR(50)
    DEFAULT '123 Sesame St.',
  phone CHAR(16)
);
```

---

#### 1.1.2 Subqueries in insertion

Listing 4: insertion via subquery

---

```
INSERT INTO PotBuddies
(
```

```

SELECT d2.drinker
FROM Frequents d1, Frequents d2
WHERE d1.drinker = 'Sally' AND
d2.drinker <> 'Sally' AND
d1.bar = d2.bar
);

```

---

Find all the drinkers at the bars Sally frequents and insert them into PotBuddies (Potential Buddies, what did you think it stands for?)

d1.bar	d2.bar
'Sally'	NOT 'Sally'
'Sally'	NOT 'Sally'

## 1.2 Deletion

Listing 5: Sally no longer likes Bud

```

DELETE FROM Likes
WHERE drinker = 'Sally' AND
beer = 'Bud';

```

---

Delete all rows where the drinker is 'Sally' and the beer is 'Bud'

Listing 6: clear out entire table

```

DELETE FROM Likes; -- no WHERE clause needed

```

---

Listing 7: delete with subquery

```

DELETE FROM Beers b
WHERE EXISTS ( --check if another beer is made by the same manufacturer
SELECT name FROM Beers --implicit join of Beers with itself
WHERE manf = b.manf AND
name <> b.name
);

```

---

Delete all beers where there is another beer by the same manufacturer.

name	manf	
Bud	Budweiser	mark as dirty
BudLite	Budweiser	mark as dirty

Delete is a **mark-and-sweep** process: first mark items for deletion, then delete all marked items. (If items were deleted immediately, it could disrupt the condition for deleting other items during the same deletion process).

## 1.3 Updates

---

Listing 8: UPDATE template

---

```
UPDATE <relation>
SET <list of attribute assignments>
WHERE <condition on tuples>;
```

---

---

Listing 9: Change Fred's Phone number

---

```
UPDATE Drinkers
SET phone = '555-1212'
WHERE name = 'Fred';
```

---

---

Listing 10: set maximum price on beers

---

```
UPDATE Sells
SET price = 4.00
WHERE price > 4.00;
```

---

---

Listing 11: add tax to price

---

```
UPDATE Sells
SET price = 1.05 * price --value can be result of a computation on attributes
WHERE price > 4.00;
```

---

## 2 Constraints

**constraint** relations enforced by DBMS

**trigger** only executed when a condition occurs

**Keys** must be unique and non-null

**Foreign-keys** referential integrity

**value-based** constrain value of attribute

**tuple-based** relationships between components

**assertions** boolean expression

### 2.1 Keys

#### 2.1.1 Single Attribute Keys

Listing 12: ensure names are unique

---

```
CREATE TABLE Beers (  
  name CHAR(20) UNIQUE, --note: name can still be NULL!  
  manf CHAR(20)  
);
```

---

### 2.1.2 Multi Attribute Keys

Listing 13: tuple as a primary key

---

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

---

### 2.1.3 Foreign Keys

Indicate that a key REFERENCES another relation and is used as a key. Referenced attributes must be declared PRIMARY KEY or UNIQUE.

Listing 14: Foreign key

---

```
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);
```

---

### 2.1.4 Enforcing Foreign Key Constraints

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

- An insert/update to R introduces values not found in S
- A delete/update to S causes some tuples of R to "dangle"

Actions Taken:

**Default** Reject the modification

**Cascade** Make the same changes in Sells

- Deleted beer: delete Sells tuple
- Updated beer: change value in Sells

**Set NULL** Change the beer to NULL

Example Cascade:

Delete Bud tuple from Beers → delete all tuples from sells that have beer = 'bud' Example Cascade:

Delete Bud tuple from Beers → change tuples from sells that have beer = 'bud' to have beer = NULL

Listing 15: setting policy

---

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

---

### 2.1.5 Attribute level checks

Add CHECK(*condition<sub>i</sub>*) 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.**

Listing 16: attribute check

---

```
CREATE TABLE Sells (
  bar CHAR(20),
  beer CHAR(20) CHECK (beer IN --make sure beer exists in Beers relation
    (SELECT name FROM Beers)), --implementation of FOREIGN KEY with a CHECK
  price REAL CHECK ( price <= 5.00) --make sure price is >= 5.00
);
```

---

Select subqueries inside check (like above) are **not** supported in Postgres

Listing 17: tuple-based check

---

```
CREATE TABLE Sells (
  bar CHAR(20),
  beer CHAR(20),
  price REAL,
  CHECK (bar = 'Joe's' OR --allow Joe to sell beer at > 5.00
    price <= 5.00));
```

---

Listing 18: operation in check

---

```
CREATE TABLE Sells (  
  bar CHAR(20),  
  beer CHAR(20),  
  price REAL,  
  tax REAL DEFAULT .05,  
  CHECK (price * tax <= 5.00));
```

---

### 2.1.6 Assertions

Can be messy, try to avoid. Just know that they are schema-level constraints

Listing 19: assertion

---

```
CREATE ASSERTION NoRipoffBars CHECK (  
  NOT EXISTS ( --no bar may charge an average of more than 5.00  
    SELECT bar FROM Sells  
    GROUP BY bar  
    HAVING 5.00 < AVG(price)  
  ));
```

---

## 2.2 Triggers

Motivation:

- Assertions are powerful, but cannot turn them off.
- attribute-level checks are controllable but not powerful
- Triggers let the user decide when they should be checked.

Also known as **ECA (event-condition-action)** rule.

**Event** type of db modification (e.g. insert on sells)

**Condition** SQL boolean expression

**Action** SQL statement

Example: After every insertion on sells, check the Beers table to see if the beer you are trying to insert exists. If it doesn't, insert that name into the Beers table.

Listing 20: Trigger definition

---

```
CREATE TRIGGER BeerTrig  
AFTER INSERT ON Sells      --the event  
REFERENCING NEW ROW AS NewTuple
```

```
FOR EACH ROW
WHEN (NewTuple.beer NOT IN --the condition
      (SELECT name FROM Beers))
INSERT INTO Beers(name) --the action
VALUES(NewTuple.beer);
```

---

### 2.2.1 Options

- CREATE TRIGGER < *name* >
- CREATE OR REPLACE TRIGGER < *name* >

Triggers:

- BEFORE
- AFTER
- INSTEAD OF - used for Views