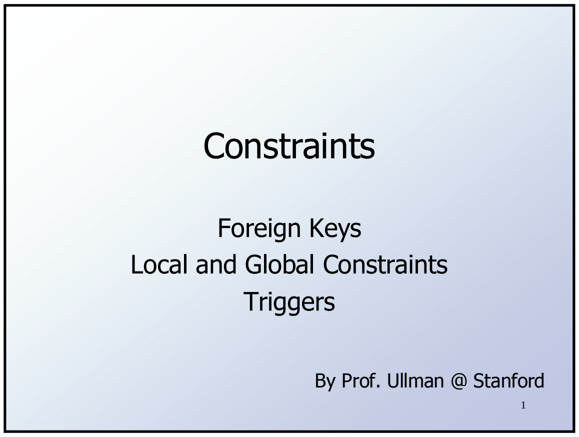
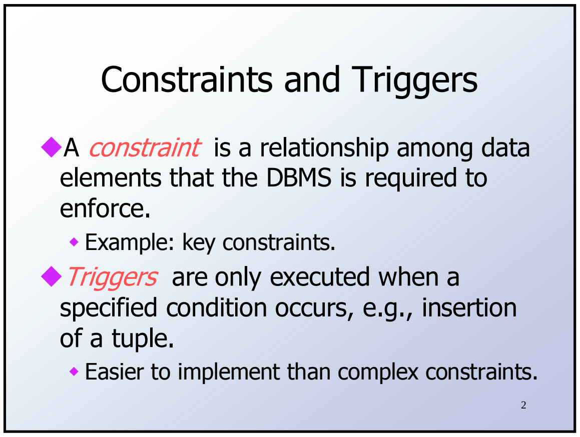


Constraints

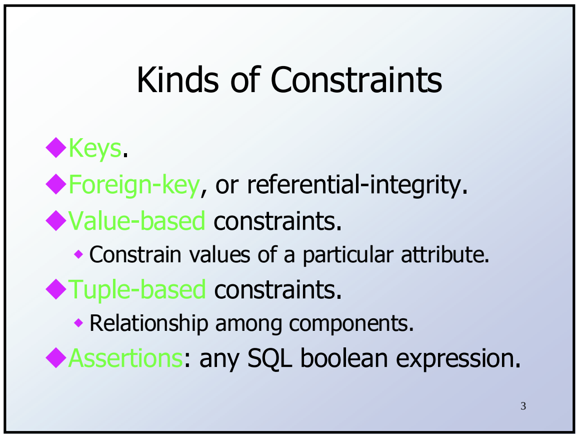
Foreign Keys
Local and Global Constraints
Triggers

By Prof. Ullman @ Stanford



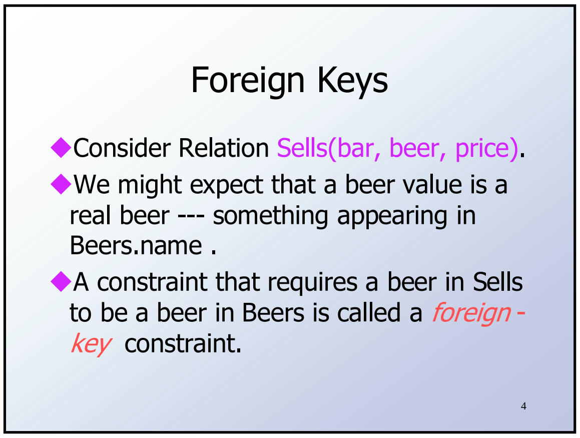
Constraints and Triggers

- 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.



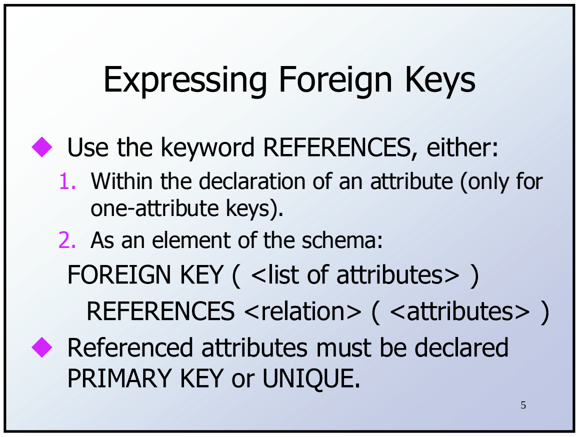
Kinds of Constraints

- 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.



Foreign Keys

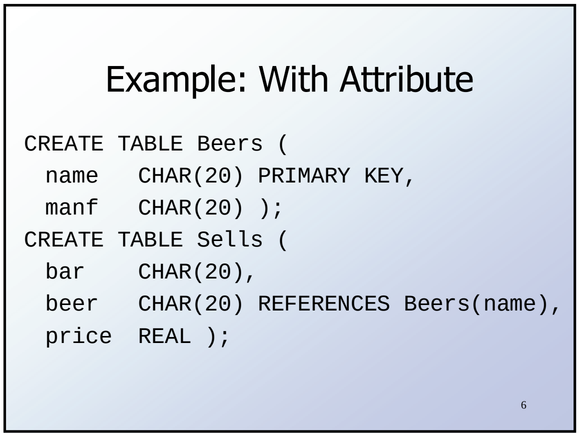
- Consider Relation Sells(bar, beer, price).
- We might expect that a beer value is a real beer --- something appearing in Beers.name .
- A constraint that requires a beer in Sells to be a beer in Beers is called a *foreign-key* constraint.



Expressing Foreign Keys

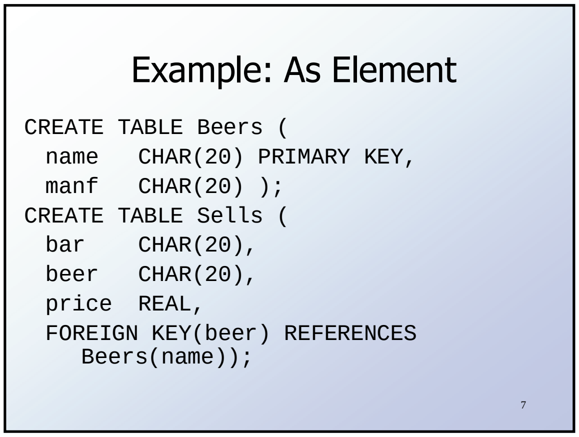
- Use the keyword REFERENCES, either:
  1. Within the declaration of an attribute (only for one-attribute keys).
  2. As an element of the schema:

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



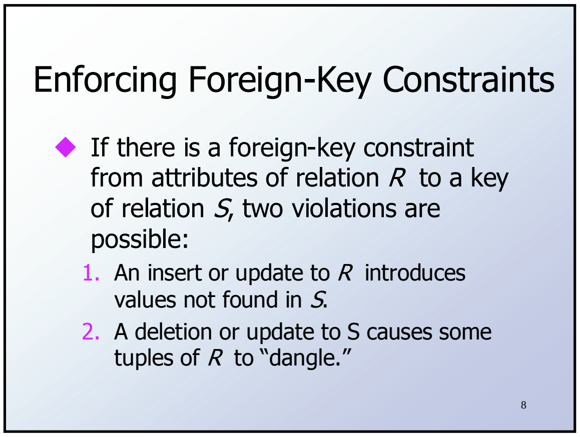
Example: With Attribute

```
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 );
```



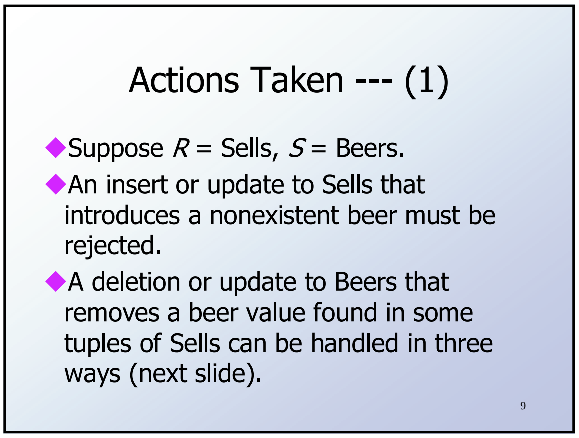
Example: As Element

```
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));
```



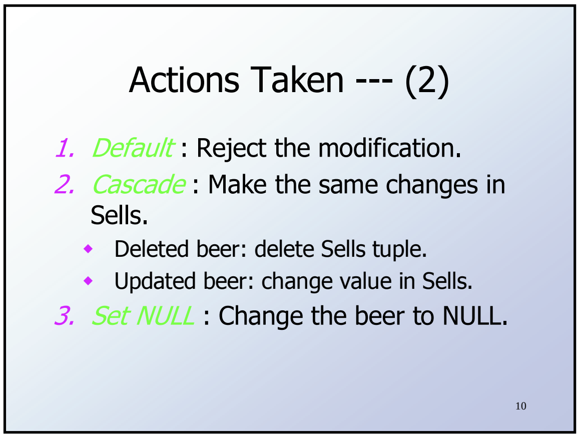
Enforcing Foreign-Key Constraints

- If there is a foreign-key constraint from attributes of relation *R* to a key of 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."



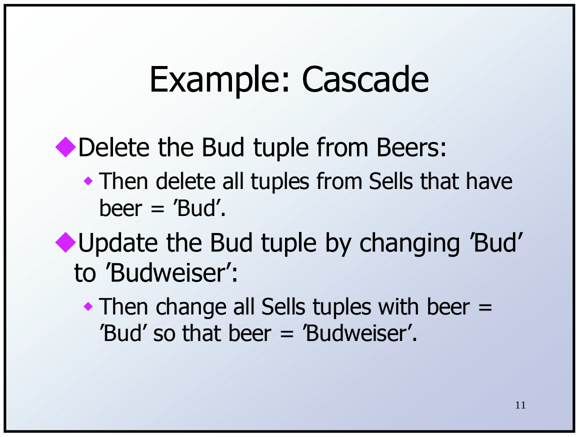
Actions Taken --- (1)

- 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).



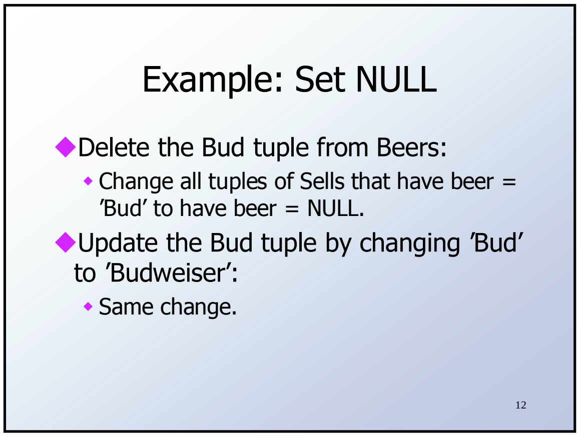
Actions Taken --- (2)

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' so that 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.

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

```
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

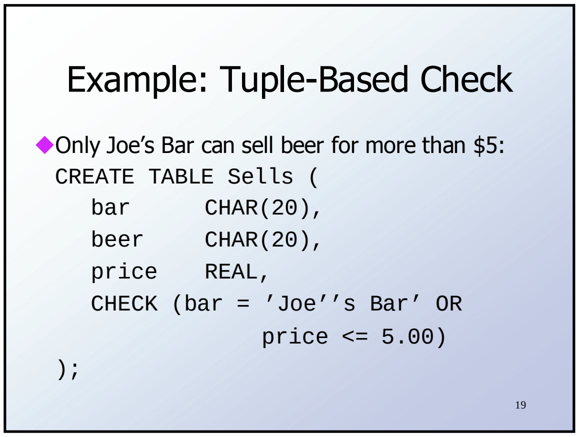
```
CREATE TABLE Sells (
  bar    CHAR(20),
  beer   CHAR(20)   CHECK ( beer IN
         (SELECT name FROM Beers)),
  price REAL CHECK ( price <= 5.00 )
);
```

Timing of Checks

- Attribute-based checks 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

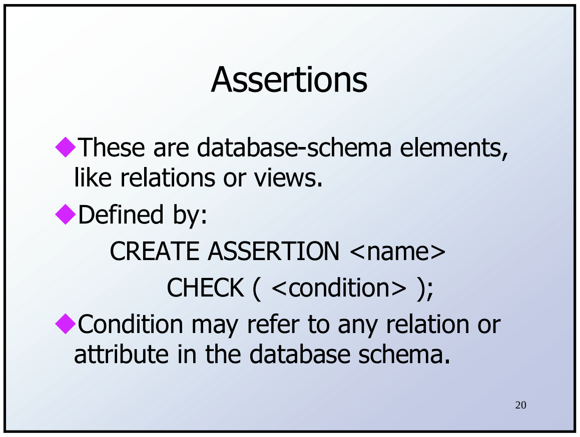
- CHECK (<condition>) may be added as a relation-schema element.
- The condition may refer to any attribute of the relation.
  - But any other attributes or relations require a subquery.
- Checked on insert or update only.



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)
);
```



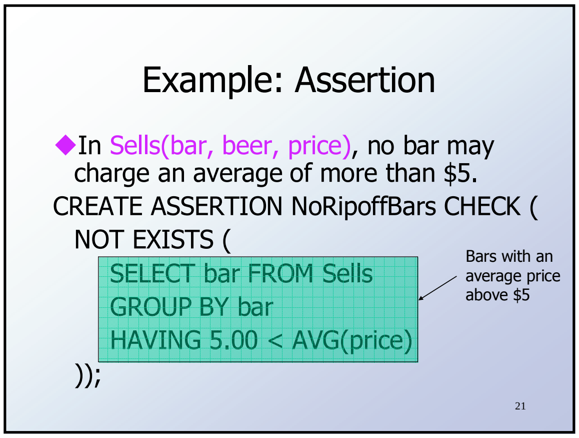
Assertions

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

  CREATE ASSERTION <name>

      CHECK (<condition>);

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

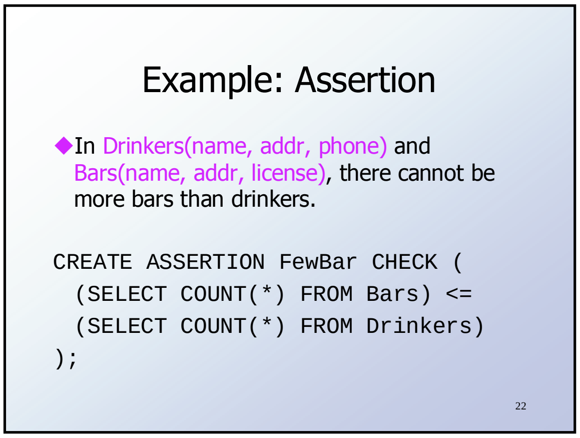


Example: Assertion

- 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)
  ));

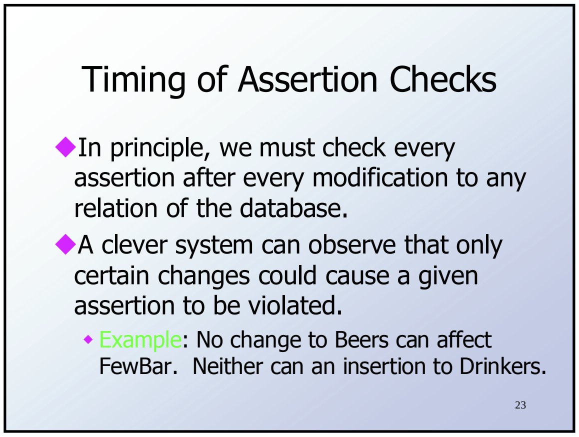
Bars with an average price above $5



Example: Assertion

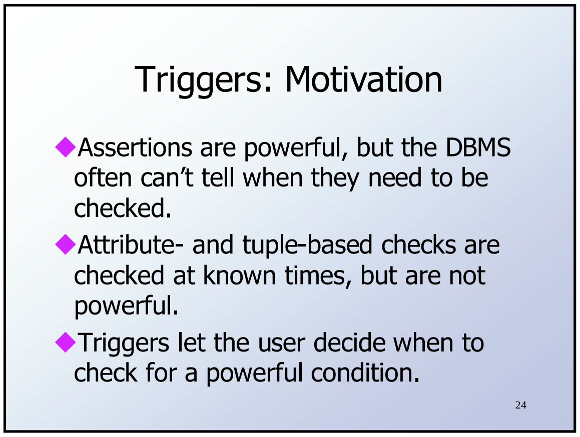
- In Drinkers(name, addr, phone) and Bars(name, addr, license), there cannot be more bars than drinkers.

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



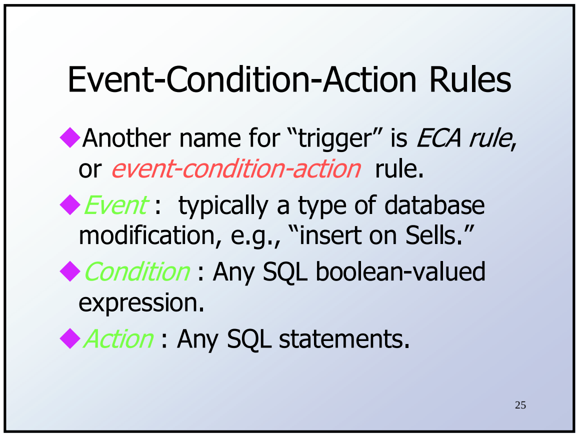
Timing of Assertion Checks

- 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.



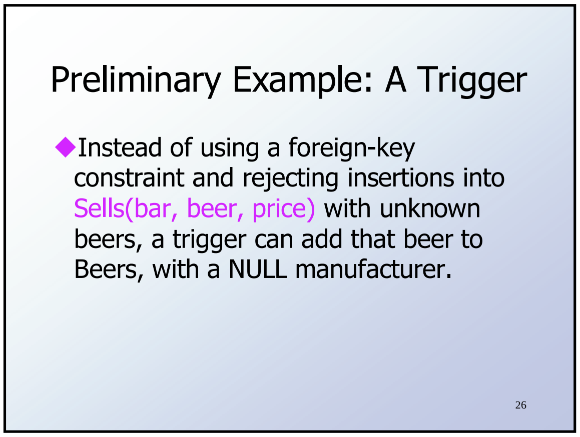
Triggers: Motivation

- Assertions are powerful, but the DBMS often can’t 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 a powerful condition.



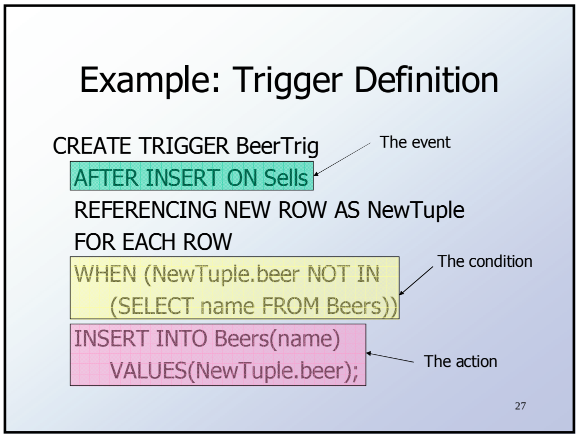
Event-Condition-Action Rules

- 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

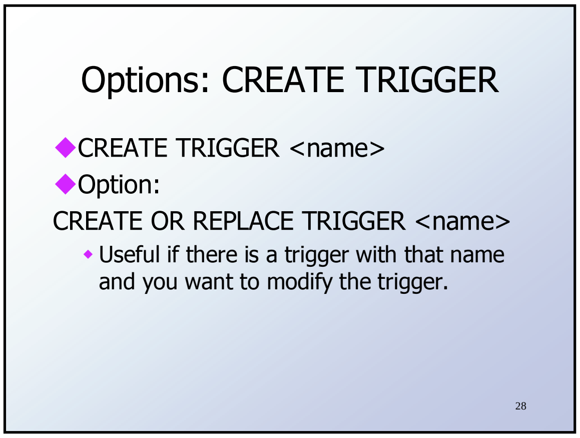
- Instead of using a foreign-key constraint and rejecting insertions into Sells(bar, beer, price) with unknown beers, a trigger can add that beer to Beers, with a NULL manufacturer.



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

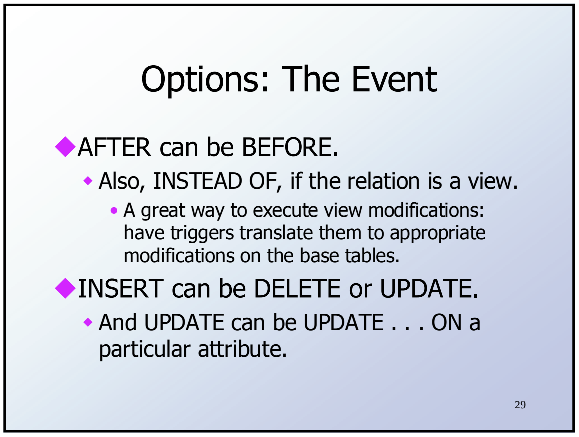


Options: CREATE TRIGGER

- CREATE TRIGGER <name>
- Option:

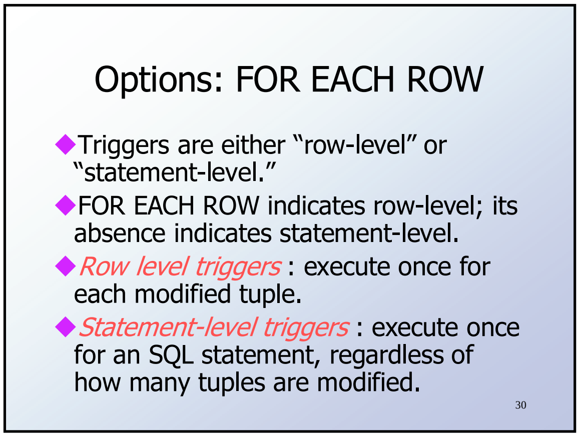
CREATE OR REPLACE TRIGGER <name>

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



Options: The Event

- AFTER can be BEFORE.
  - Also, INSTEAD OF, if the relation is a view.
    - A great way to execute view modifications: have triggers translate them to appropriate modifications on the base tables.
- 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 an SQL statement, regardless of how many tuples are modified.

Options: REFERENCING

- INSERT statements imply a new tuple (for row-level) or new 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][TUPLE TABLE] AS <name>

Options: The Condition

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

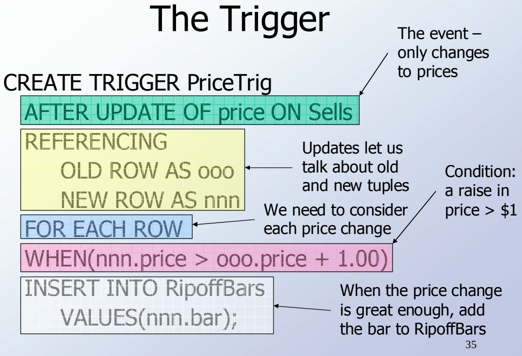
Options: The Action

- 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) created for the purpose, 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);
```

The event – only changes to prices

Updates let us talk about old and new tuples

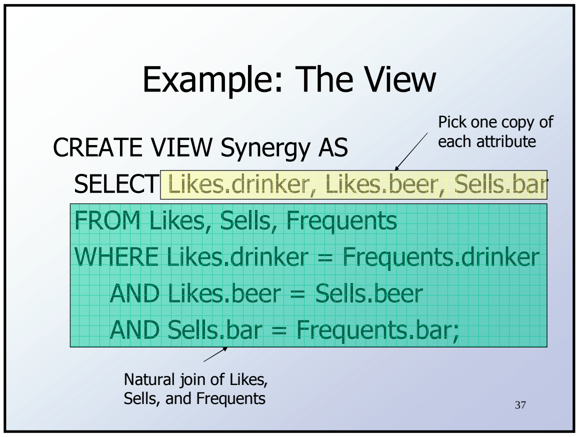
We need to consider each price change

Condition: a raise in price > $1

When the price change is great enough, add the bar to RipoffBars

Triggers on Views

- Generally, it is impossible to modify a view, because it doesn't exist.
- But an INSTEAD OF trigger lets us interpret view modifications in a way that makes sense.
- Example: We'll design a view Synergy that has (drinker, beer, bar) triples such that the bar serves the beer, the drinker frequents the bar and likes the beer.



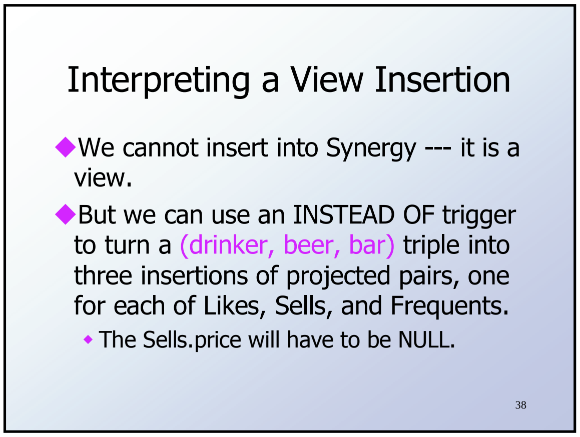
Example: The View

```
CREATE VIEW Synergy AS
  SELECT Likes.drinker, Likes.beer, Sells.bar
  FROM Likes, Sells, Frequents
  WHERE Likes.drinker = Frequents.drinker
     AND Likes.beer = Sells.beer
     AND Sells.bar = Frequents.bar;
```

Pick one copy of each attribute

Natural join of Likes, Sells, and Frequents

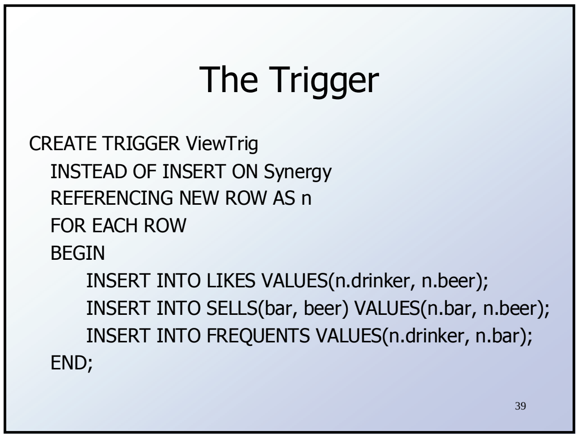
37



Interpreting a View Insertion

- We cannot insert into Synergy --- it is a view.
- But we can use an INSTEAD OF trigger to turn a (drinker, beer, bar) triple into three insertions of projected pairs, one for each of Likes, Sells, and Frequents.
  - The Sells.price will have to be NULL.

38



The Trigger

```
CREATE TRIGGER ViewTrig
  INSTEAD OF INSERT ON Synergy
  REFERENCING NEW ROW AS n
  FOR EACH ROW
  BEGIN
      INSERT INTO LIKES VALUES(n.drinker, n.beer);
      INSERT INTO SELLS(bar, beer) VALUES(n.bar, n.beer);
      INSERT INTO FREQUENTS VALUES(n.drinker, n.bar);
  END;
```

39