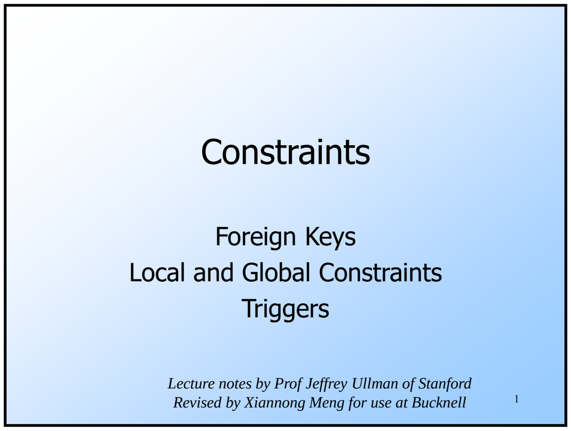
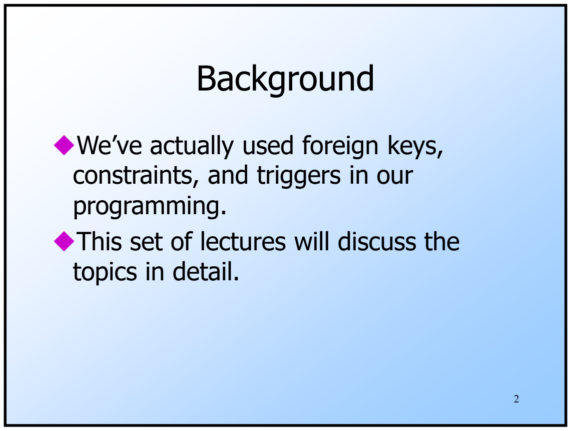


Constraints

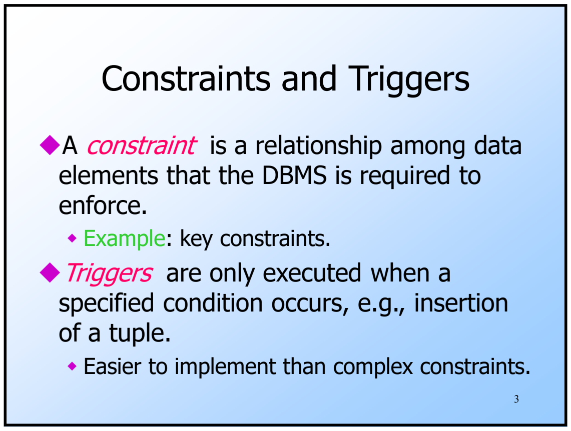
Foreign Keys
Local and Global Constraints
Triggers

Lecture notes by Prof Jeffrey Ullman of Stanford
Revised by Xiannong Meng for use at Bucknell



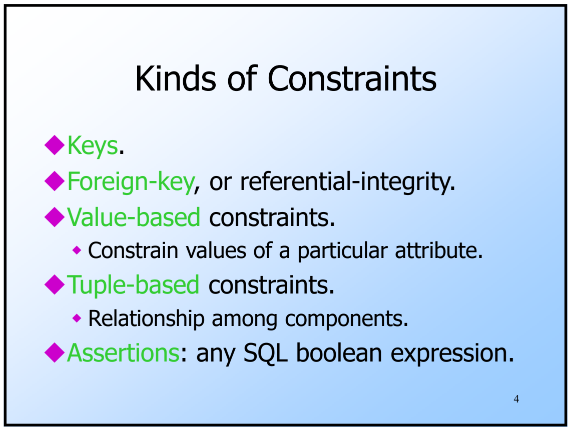
Background

- We've actually used foreign keys, constraints, and triggers in our programming.
- This set of lectures will discuss the topics in detail.



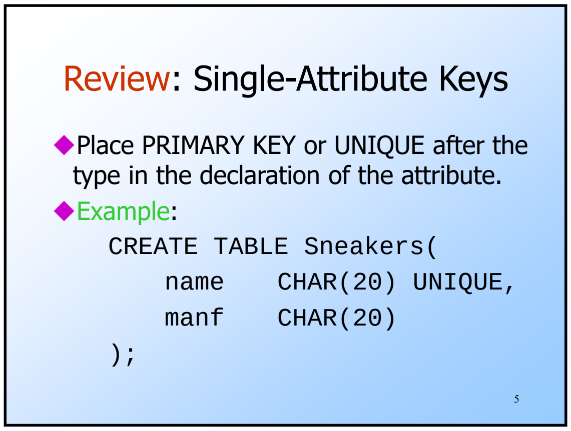
Constraints and Triggers

- 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

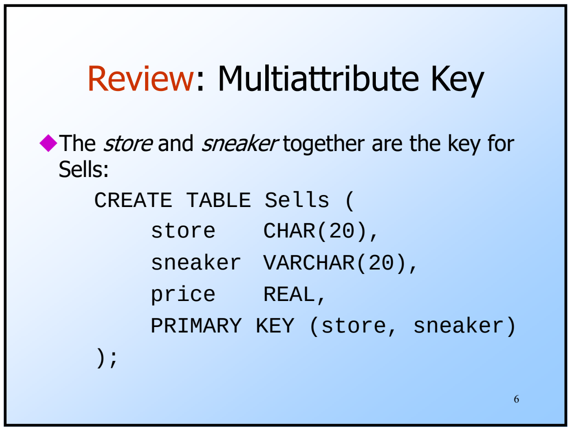
- Keys.
- Foreign-key, or referential-integrity.
- Value-based constraints.
  - Constrain values of a particular attribute.
- Tuple-based constraints.
  - Relationship among components.
- Assertions: any SQL boolean expression.



Review: Single-Attribute Keys

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

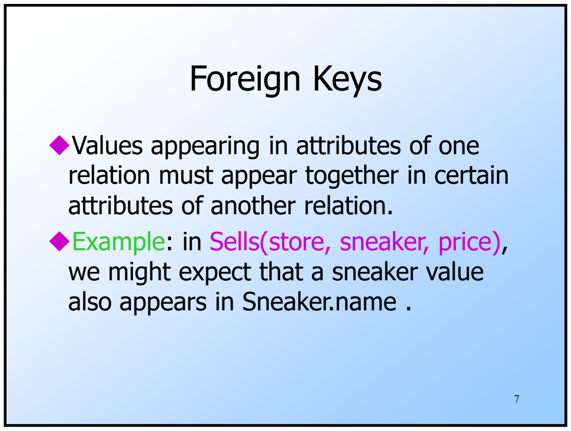
```
CREATE TABLE Sneakers(
    name    CHAR(20) UNIQUE,
    manf    CHAR(20)
);
```



Review: Multiattribute Key

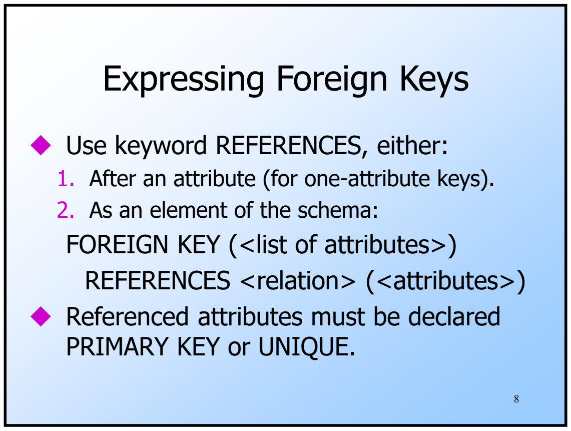
- The *store* and *sneaker* together are the key for Sells:

```
CREATE TABLE Sells (
    store    CHAR(20),
    sneaker  VARCHAR(20),
    price    REAL,
    PRIMARY KEY (store, sneaker)
);
```



Foreign Keys

- Values appearing in attributes of one relation must appear together in certain attributes of another relation.
- Example: in Sells(store, sneaker, price), we might expect that a sneaker value also appears in Sneaker.name .

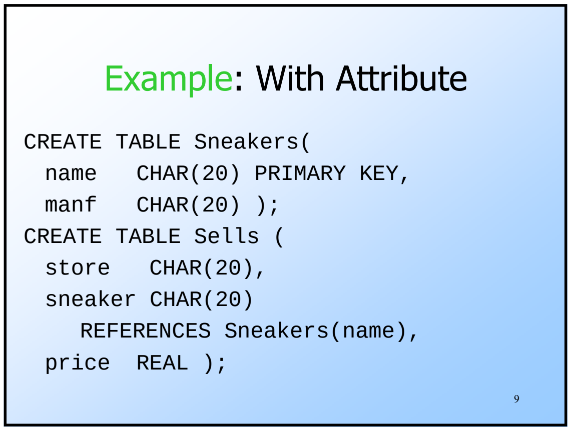


Expressing Foreign Keys

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

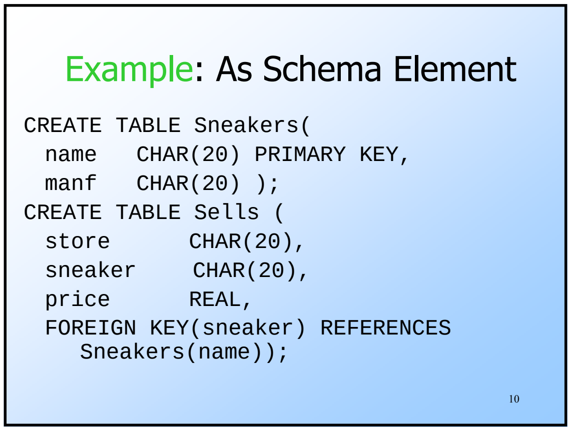
  FOREIGN KEY (<list of attributes>)

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



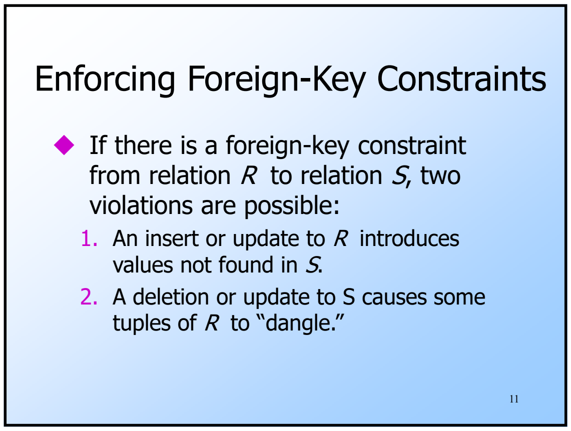
Example: With Attribute

```
CREATE TABLE Sneakers(
  name   CHAR(20) PRIMARY KEY,
  manf   CHAR(20) );
CREATE TABLE Sells (
  store   CHAR(20),
  sneaker CHAR(20)
     REFERENCES Sneakers(name),
  price  REAL );
```



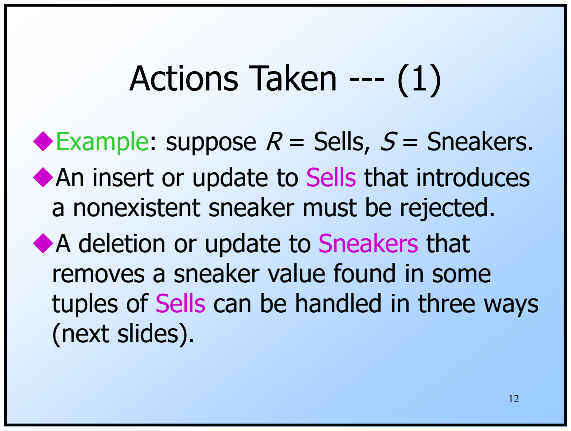
Example: As Schema Element

```
CREATE TABLE Sneakers(
  name   CHAR(20) PRIMARY KEY,
  manf   CHAR(20) );
CREATE TABLE Sells (
  store      CHAR(20),
  sneaker    CHAR(20),
  price      REAL,
  FOREIGN KEY(sneaker) REFERENCES
     Sneakers(name));
```



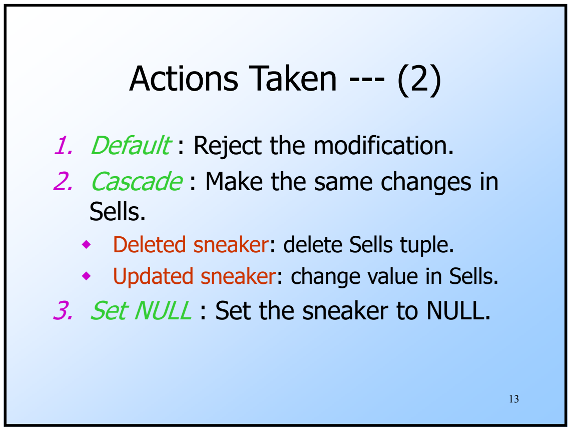
Enforcing Foreign-Key Constraints

- 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."



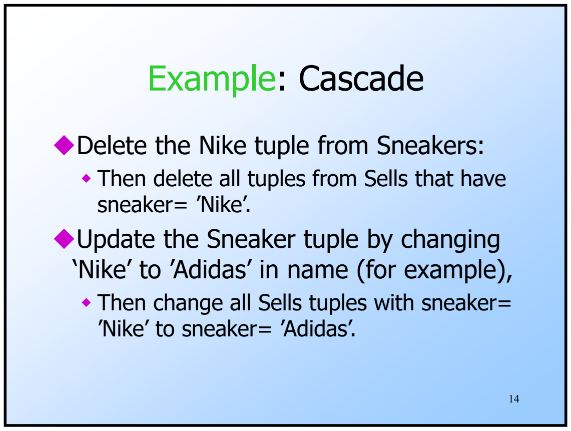
Actions Taken --- (1)

- Example: suppose *R* = Sells, *S* = Sneakers.
- An insert or update to Sells that introduces a nonexistent sneaker must be rejected.
- A deletion or update to Sneakers that removes a sneaker value found in some tuples of Sells can be handled in three ways (next slides).



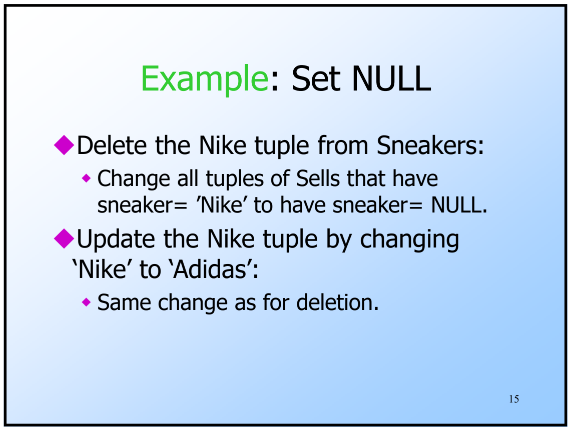
Actions Taken --- (2)

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



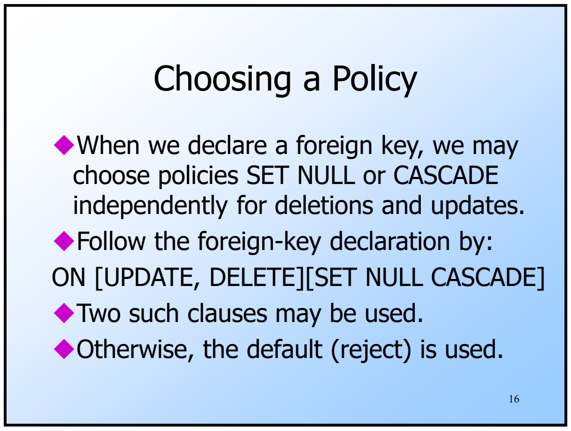
Example: Cascade

- Delete the Nike tuple from Sneakers:
  - Then delete all tuples from Sells that have sneaker= ’Nike’.
- Update the Sneaker tuple by changing ‘Nike’ to ’Adidas’ in name (for example),
  - Then change all Sells tuples with sneaker= ’Nike’ to sneaker= ’Adidas’.



Example: Set NULL

- Delete the Nike tuple from Sneakers:
  - Change all tuples of Sells that have sneaker= ’Nike’ to have sneaker= NULL.
- Update the Nike tuple by changing ‘Nike’ to ‘Adidas’:
  - Same change as for deletion.

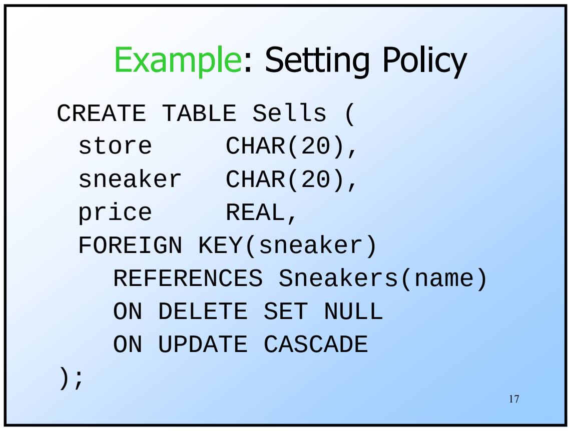


Choosing a Policy

- 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:

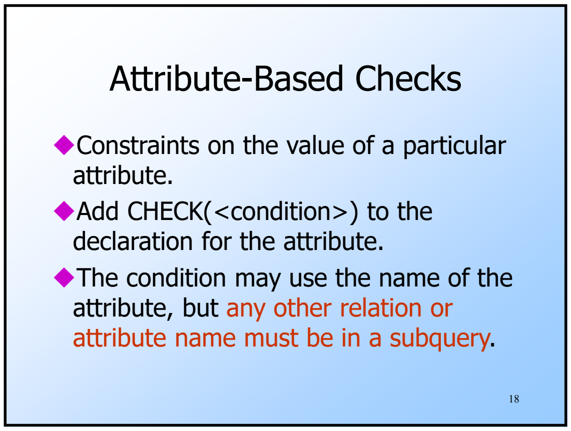
ON [UPDATE, DELETE][SET NULL CASCADE]

- Two such clauses may be used.
- Otherwise, the default (reject) is used.



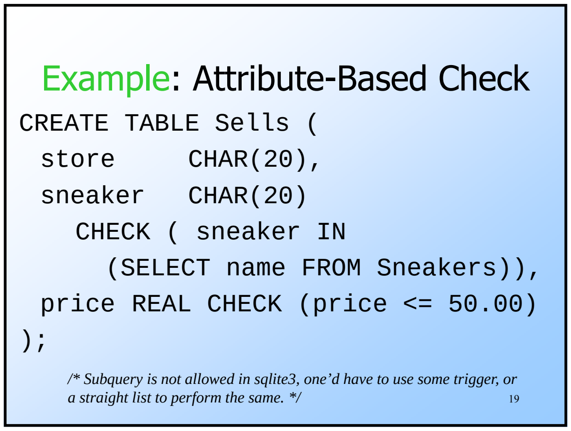
Example: Setting Policy

```
CREATE TABLE Sells (
  store     CHAR(20),
  sneaker   CHAR(20),
  price     REAL,
  FOREIGN KEY(sneaker)
     REFERENCES Sneakers(name)
     ON DELETE SET NULL
     ON UPDATE CASCADE
);
```



Attribute-Based Checks

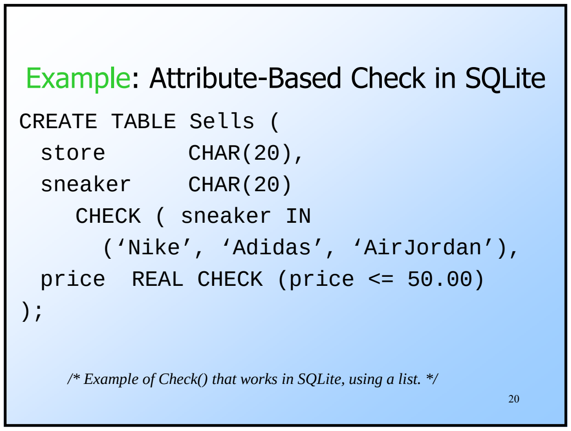
- Constraints on the value of a particular attribute.
- Add CHECK(<condition>) to the declaration for the attribute.
- The condition may use the name of the attribute, but any other relation or attribute name must be in a subquery.



Example: Attribute-Based Check

```
CREATE TABLE Sells (
  store     CHAR(20),
  sneaker   CHAR(20)
     CHECK ( sneaker IN
        (SELECT name FROM Sneakers)),
  price REAL CHECK (price <= 50.00)
);
```

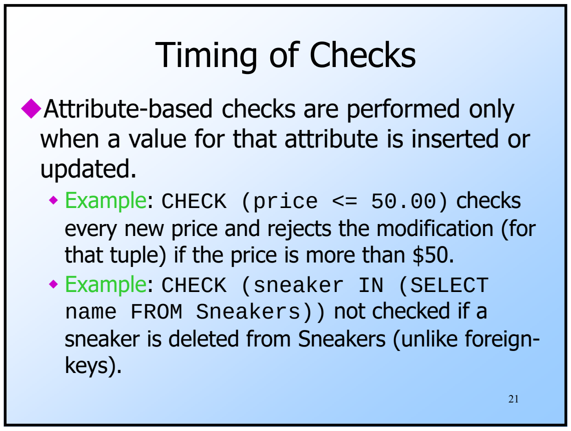
/ Subquery is not allowed in sqlite3, one’d have to use some trigger, or a straight list to perform the same. */*



Example: Attribute-Based Check in SQLite

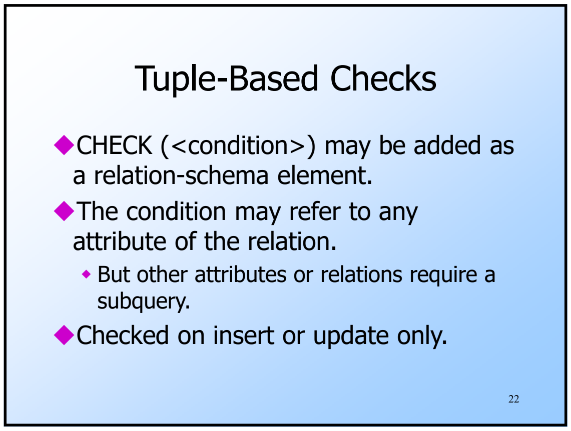
```
CREATE TABLE Sells (
  store      CHAR(20),
  sneaker    CHAR(20)
     CHECK ( sneaker IN
       (‘Nike’, ‘Adidas’, ‘AirJordan’),
  price  REAL CHECK (price <= 50.00)
);
```

/ Example of Check() that works in SQLite, using a list. */*



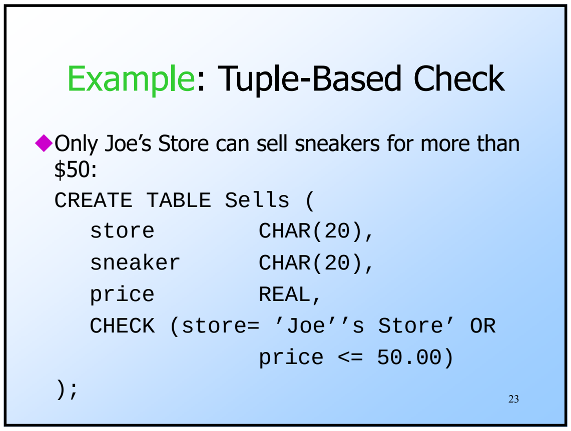
Timing of Checks

- Attribute-based checks are performed only when a value for that attribute is inserted or updated.
  - Example: `CHECK (price <= 50.00)` checks every new price and rejects the modification (for that tuple) if the price is more than $50.
  - Example: `CHECK (sneaker IN (SELECT name FROM Sneakers))` not checked if a sneaker is deleted from Sneakers (unlike foreign-keys).



Tuple-Based Checks

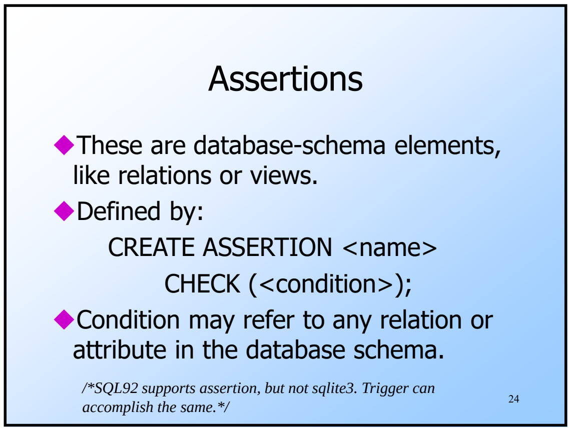
- CHECK (<condition>) may be added as a relation-schema element.
- The condition may refer to any attribute of the relation.
  - But other attributes or relations require a subquery.
- Checked on insert or update only.



Example: Tuple-Based Check

- Only Joe’s Store can sell sneakers for more than $50:

```
CREATE TABLE Sells (
   store       CHAR(20),
   sneaker     CHAR(20),
   price       REAL,
   CHECK (store= ’Joe’’s Store’ OR
               price <= 50.00)
);
```



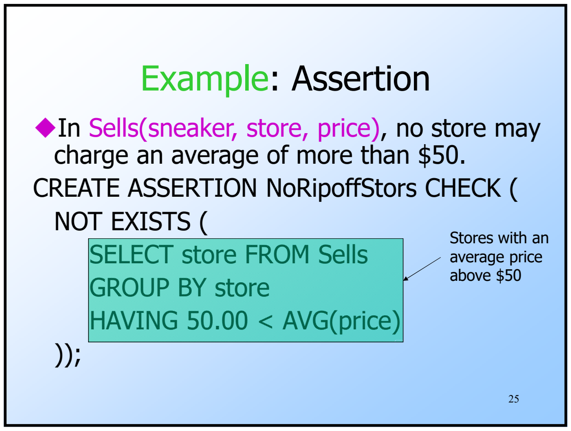
Assertions

- 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.

*/*SQL92 supports assertion, but not sqlite3. Trigger can accomplish the same.*/*

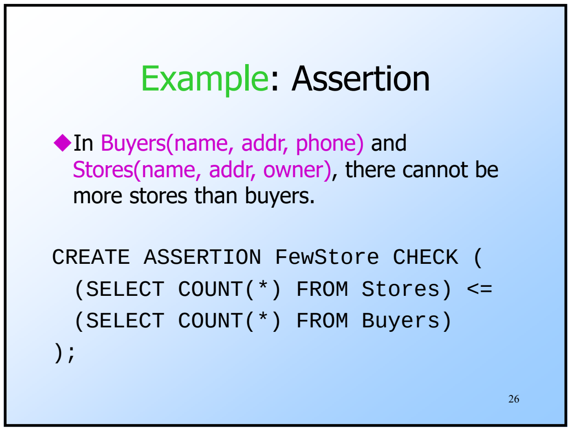


Example: Assertion

- In Sells(sneaker, store, price), no store may charge an average of more than $50.

```
CREATE ASSERTION NoRipoffStors CHECK (
  NOT EXISTS (
     SELECT store FROM Sells
     GROUP BY store
     HAVING 50.00 < AVG(price)
  ));
```

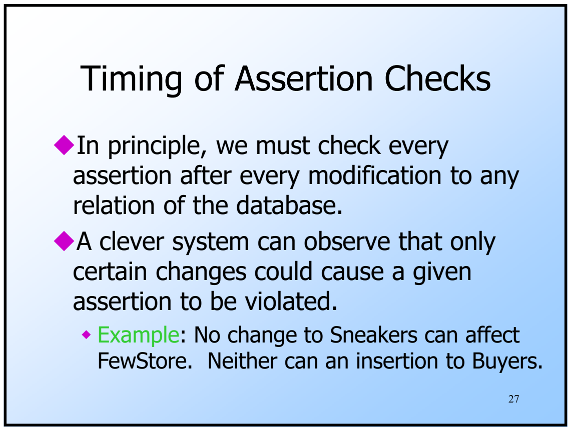
Stores with an average price above $50



Example: Assertion

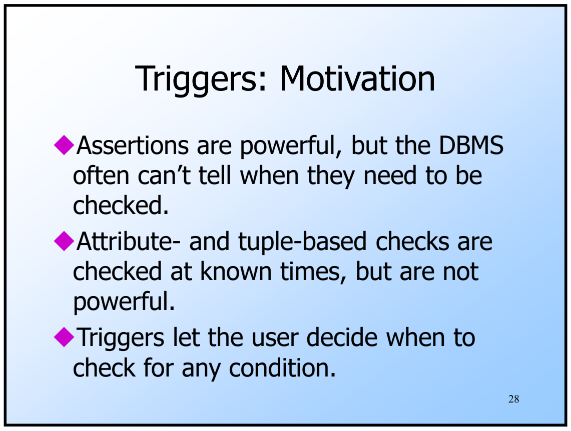
- In Buyers(name, addr, phone) and Stores(name, addr, owner), there cannot be more stores than buyers.

```
CREATE ASSERTION FewStore CHECK (
  (SELECT COUNT(*) FROM Stores) <=
  (SELECT COUNT(*) FROM Buyers)
);
```



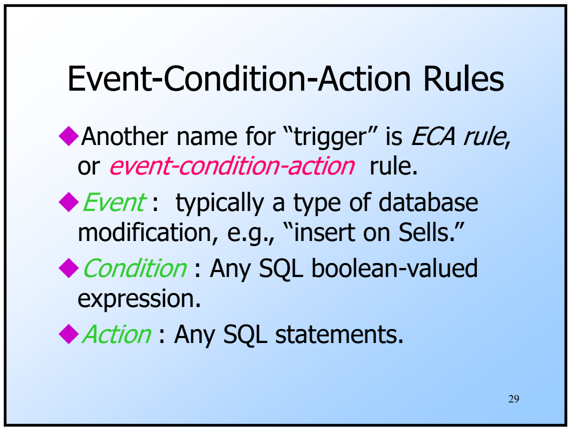
Timing of Assertion Checks

- 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 Sneakers can affect FewStore. Neither can an insertion to Buyers.



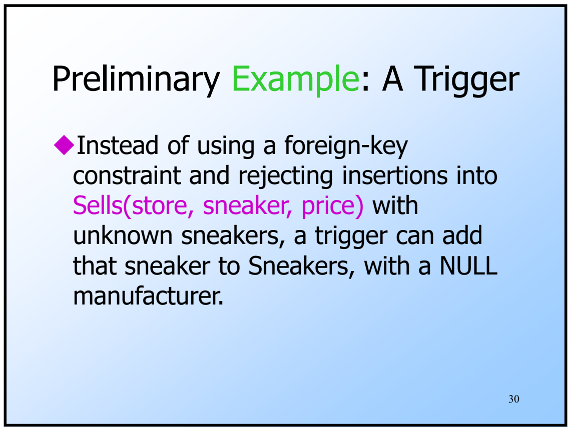
Triggers: Motivation

- 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 any condition.



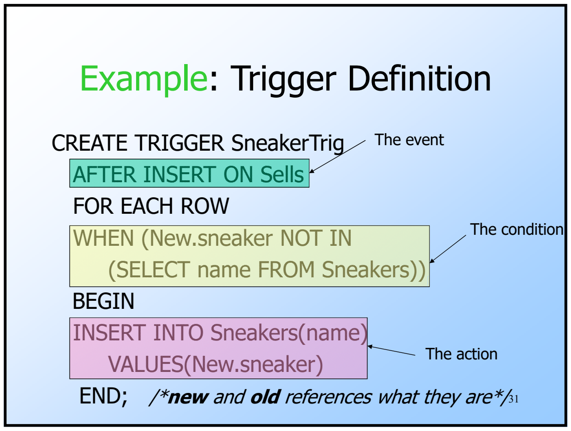
Event-Condition-Action Rules

- 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(store, sneaker, price) with unknown sneakers, a trigger can add that sneaker to Sneakers, with a NULL manufacturer.



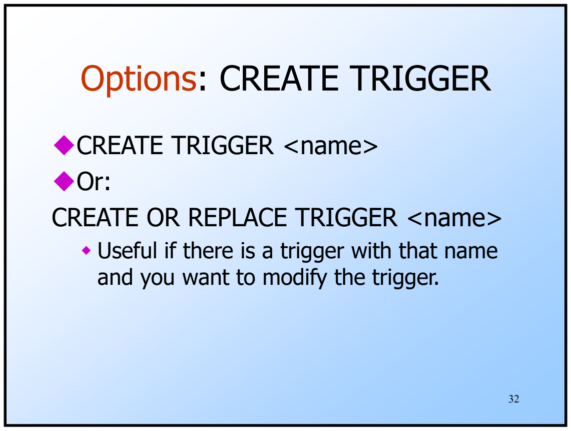
Example: Trigger Definition

```
CREATE TRIGGER SneakerTrig
  AFTER INSERT ON Sells
  FOR EACH ROW
  WHEN (New.sneaker NOT IN
     (SELECT name FROM Sneakers))
  BEGIN
  INSERT INTO Sneakers(name)
     VALUES(New.sneaker)
   END;   /*new and old references what they are*/
```

The event

The condition

The action

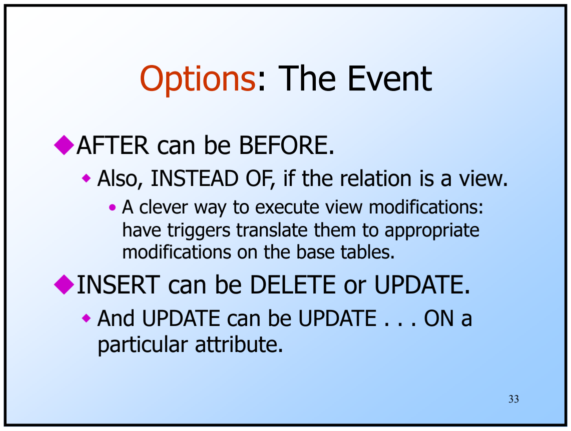


Options: CREATE TRIGGER

- CREATE TRIGGER <name>
- Or:

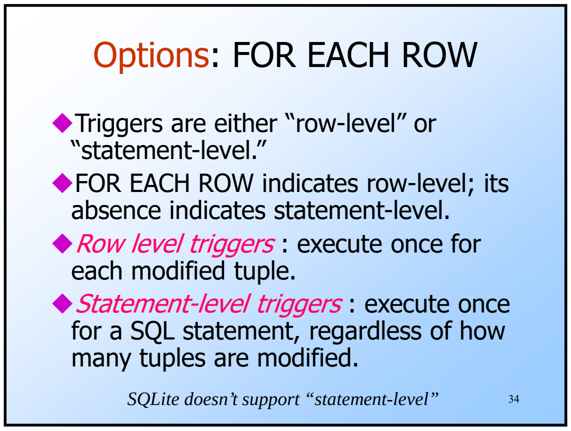
CREATE OR REPLACE TRIGGER <name>

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



Options: The Event

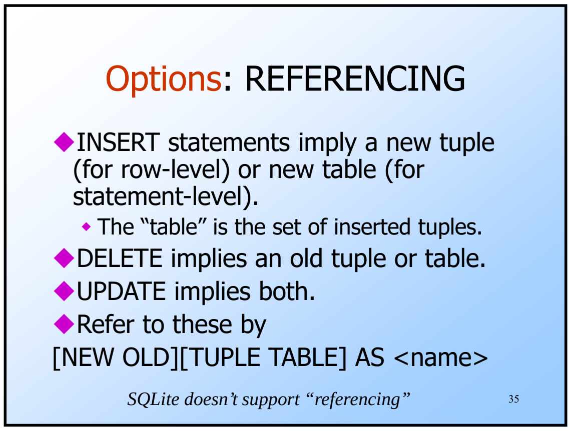
- AFTER can be BEFORE.
  - Also, INSTEAD OF, if the relation is a view.
    - A clever way to execute view modifications: have triggers translate them to appropriate modifications on the base tables.
- INSERT can be DELETE or UPDATE.
  - And UPDATE can be UPDATE . . . ON a particular attribute.



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 a SQL statement, regardless of how many tuples are modified.

SQLite doesn't support "statement-level"

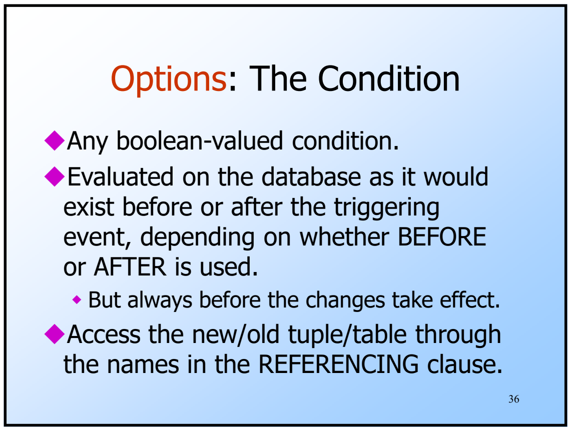


Options: REFERENCING

- 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

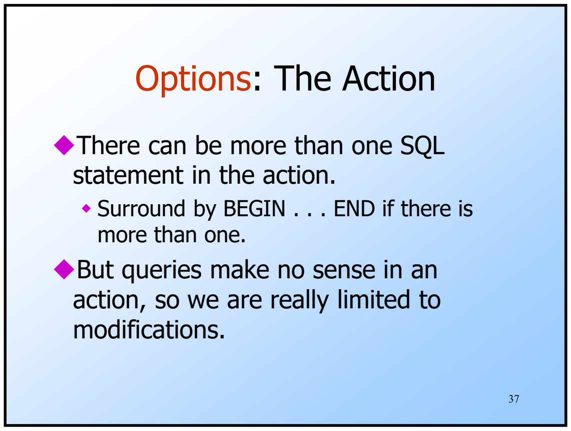
[NEW OLD][TUPLE TABLE] AS <name>

SQLite doesn't support "referencing"



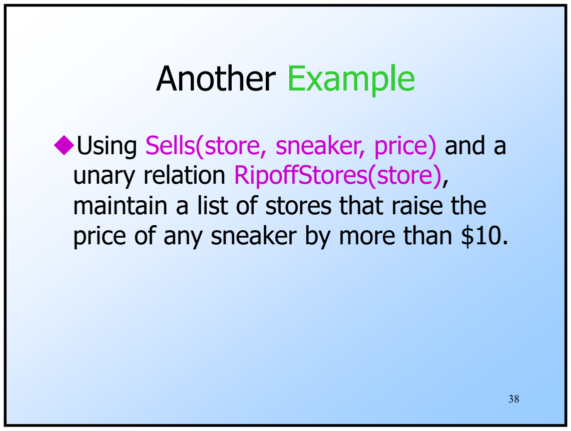
Options: The Condition

- 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.



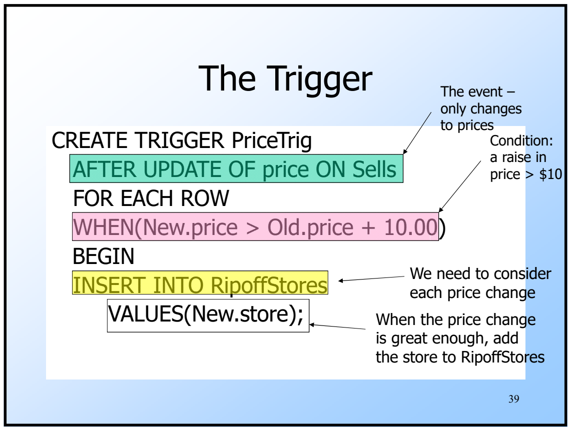
Options: The Action

- 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(store, sneaker, price) and a unary relation RipoffStores(store), maintain a list of stores that raise the price of any sneaker by more than $10.



The Trigger

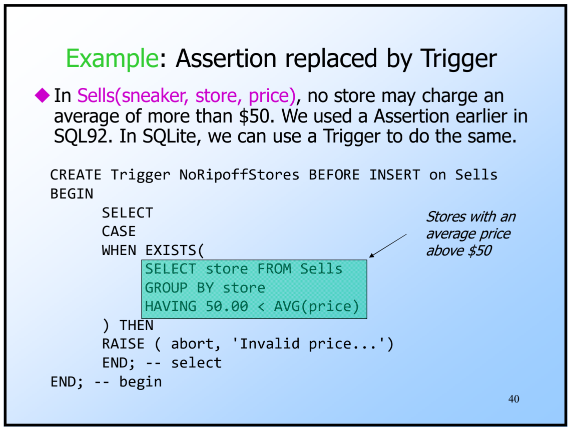
```
CREATE TRIGGER PriceTrig
  AFTER UPDATE OF price ON Sells
  FOR EACH ROW
  WHEN(New.price > Old.price + 10.00)
  BEGIN
  INSERT INTO RipoffStores
     VALUES(New.store);
```

The event – only changes to prices

Condition: a raise in price > $10

We need to consider each price change

When the price change is great enough, add the store to RipoffStores



Example: Assertion replaced by Trigger

- In Sells(sneaker, store, price), no store may charge an average of more than $50. We used a Assertion earlier in SQL92. In SQLite, we can use a Trigger to do the same.

```
CREATE Trigger NoRipoffStores BEFORE INSERT on Sells
BEGIN
      SELECT
      CASE
      WHEN EXISTS(
           SELECT store FROM Sells
           GROUP BY store
           HAVING 50.00 < AVG(price)
      ) THEN
      RAISE ( abort, 'Invalid price...')
      END; -- select
END; -- begin
```

Stores with an average price above $50