SQL - Access control

- Why an access control ?
- Views
- Authorization

Claudine Tacquard, Ecole Polytechnique de l'Université de Tours - France

Presentation based on the slides created by Prof. Jeffrey D. Ullman

POLYTECH

Access control : Why ?

- Secrecy: Users should not be able to see things they are not supposed to.
- Integrity: Users should not be able to modify things they are not supposed to.
- Availability: Users should be able to see and modify things they are allowed to.

Access control : Views

- Views can be used to present necessary information (or a summary), while **hiding details** in underlying relation(s).
- A view is a "**virtual table**" : a relation that is defined in terms of the contents of other tables and views.
- In contrast, a relation whose value is **really stored** in the database is called a ***base table***.

Declare by:

CREATE VIEW <name> **AS** <query>;

Access control : Views
View Definition

- Example:

  Person (Id_Pers, first_name, last_name, #id_City)

  City (id_City, CityName, Area)

Persons from Paris ?

```
CREATE VIEW PersonFromParis AS
    SELECT first_name, last_name
    FROM Person, City
    WHERE Person.id_City = City.id_City
          and CityName = 'Paris';
```

Access control : Views
Accessing a View

- You may query a view as if it were a base table.
  - There is a limited ability to modify views if the modification makes sense as a modification of the underlying base table.
- Example:

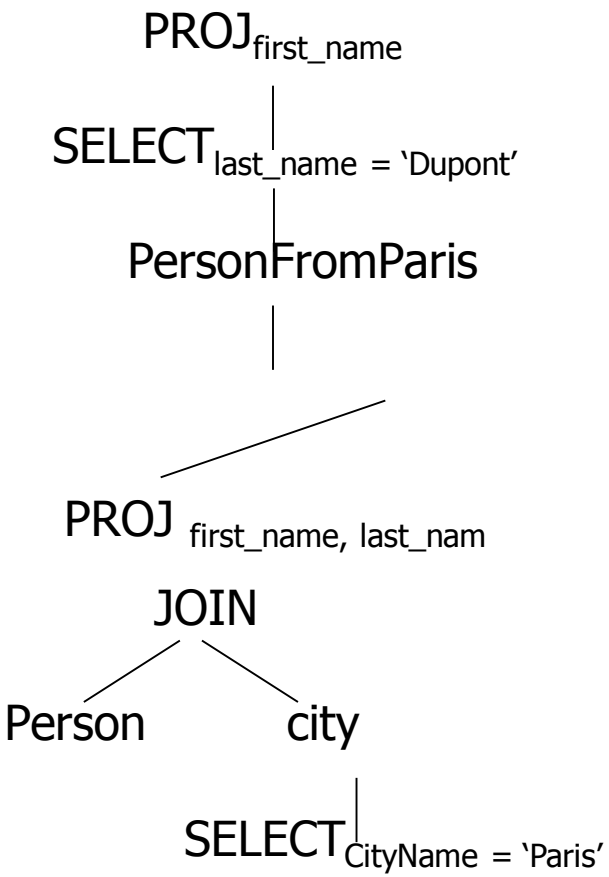
```
SELECT first_name FROM PersonFromParis

WHERE = last_name = 'Dupont';
```

Access control : Views
What Happens When a View Is Used?

1. The DBMS starts by interpreting the query as if the view were a base table.
   - Typical DBMS turns the query into something like relational algebra.
2. The queries defining any views used by the query are also replaced by their algebraic equivalents, and “spliced into” the expression tree for the query.

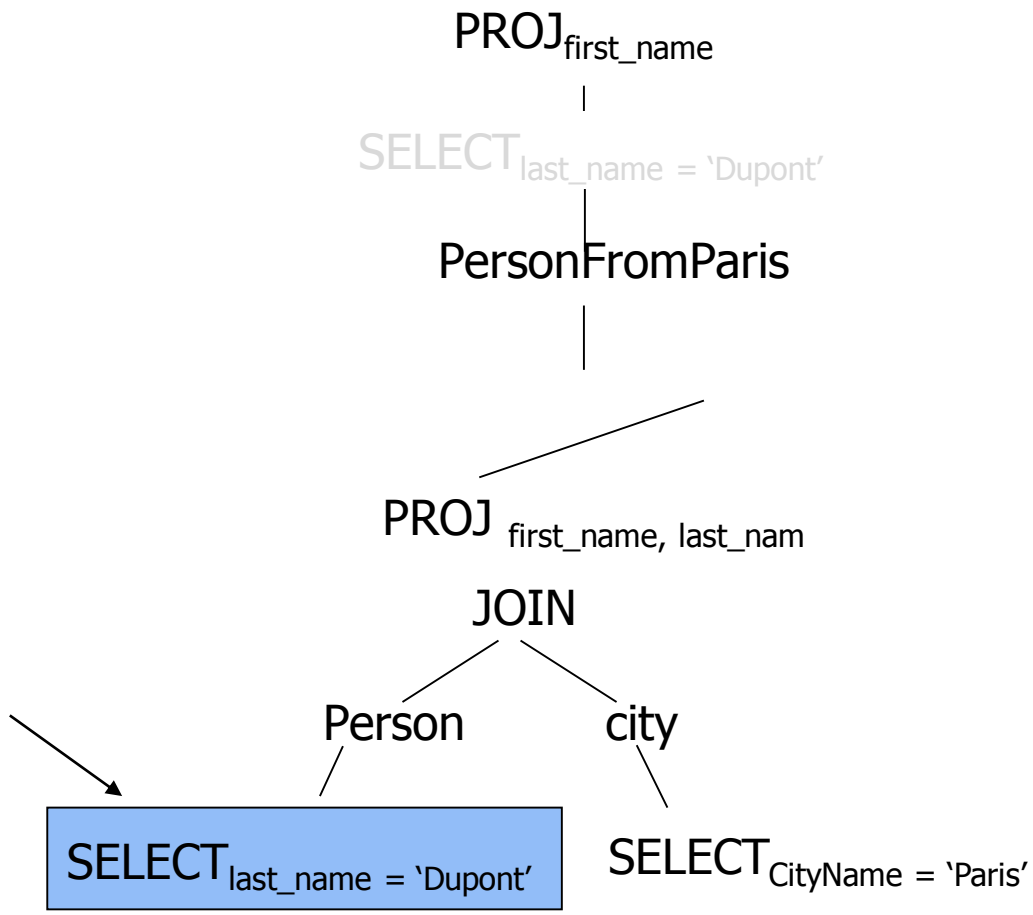
Access control : Views
Example: View Expansion



Access control : Views
DMBS Optimization

- It is interesting to observe that the typical DBMS will then “optimize” the query by transforming the algebraic expression to one that can be executed faster.
- Key optimizations:
  1. Push selections down the tree.
  2. Eliminate unnecessary projections.

Access control : Views
Example: Optimization



Most tuples are eliminated from Person before the expensive join.

Access control : Materialized View

- A **materialized view** (MV) is a database object that stores the **results of a query** at a single point in time. Unlike a view, materialized view is not virtual.
- MV may be stored locally or remotely in other site.
- MV are used by applications that do not require current data or that require data valid at a specific point in time.
- MV increases query performance since it contains results of a query

Access control : Materialized View

Refresh Methods :

1. **Complete Refresh**: essentially re-creates the materialized view

2. **Fast Refresh:**
   - First identifies the changes that occurred since the most recent refresh of the materialized view and then applies these changes to the materialized view.
   - Fast refreshes are **more efficient** than complete refreshes when there are few changes or the view is refreshed frequently.

CREATE MATERIALIZED VIEW <name> **AS** <query>; (not supported by SQLite)

Access control : Authorization

- A file system identifies certain privileges on the objects (files) it manages.
  - Typically read, write, execute.
- A file system identifies certain participants to whom privileges may be granted.
  - Typically the owner, a group, all users.

Access control: Authorization Privileges (1)

- SQL identifies a more detailed set of privileges on objects (relations, views,…) than the typical file system.
- Nine privileges in all, some of which can be restricted to one column of one relation.

Access control: Authorization Privileges (2)

Some important privileges on a relation:

- SELECT = right to query the relation.
- INSERT = right to insert tuples (May apply to only one attribute)
- DELETE = right to delete tuples.
- UPDATE = right to update tuples (May apply to only one attribute).

Access control: Authorization
Example: Privileges

- For the statement below:

```
INSERT INTO R1 (A)
    SELECT A FROM R2
    WHERE NOT EXISTS
        (SELECT * FROM R1
         WHERE R2.A = R1.A);
```

- We require privileges SELECT on R1 and R2, and INSERT on R1 or R1.A

R1	A	B
	a1	b1
	a2	b1
	a3	b1
	a4	b1
	a5	b3
	a6	**@**

R2	A	C
	a6	c5
	a3	c1

Access control: Authorization Authorization ID's

- Privileges are actually assigned to authorization ids, which can denote a **single user** or a **group of users.**
- For a single user the *authorization ID*, is typically his name.
- There is an authorization ID PUBLIC.

  Granting a privilege to PUBLIC makes it available to any authorization ID.

CREATE USER <user> [IDENTIFIED BY <password>]

```
ex : CREATE USER peter IDENTIFIED BY 'mypass'
```

Access control: Authorization
Granting Privileges

- You have all possible privileges on the objects, such as relations, that you create.
- You may grant privileges to other users (authorization ID's), including PUBLIC.
- You may also grant privileges WITH GRANT OPTION, which lets the grantee also grant this privilege.

Access control: Authorization
The GRANT Statement

- To grant privileges, say:

  GRANT <list of privileges>

  ON <relation or other object>

  TO <list of authorization ID's>;

- If you want the recipient(s) to be able to pass the privilege(s) to others add:

  WITH GRANT OPTION

Access control: Authorization
Example: GRANT

- Suppose you are the owner of Person. You may say:

```
GRANT SELECT, UPDATE(id_City)

ON Person

TO peter;
```

- Now Peter has the right to issue any query on Person and can update the id_City component only.

Access control: Authorization
Example: Grant Option

- Suppose we also grant:

```
GRANT UPDATE ON Person TO peter
WITH GRANT OPTION;
```

- Now, Peter can not only update any attribute of Person, but can grant to others the privilege UPDATE ON Person.
  - Also, he can grant more specific privileges like UPDATE(id_City) ON Person.

Access control: Authorization
Revoking Privileges

REVOKE <list of privileges>

ON <relation or other object>

FROM <list of authorization ID's>;

- Your grant of these privileges can no longer be used by these users to justify their use of the privilege.
  - But they may still have the privilege because they obtained it independently from elsewhere.

Access control: Authorization
Set of Privileges

- A **role** is a set of privileges that can be granted to users or to other role.

CREATE ROLE role_name [AUTHORIZATION owner_name]

Ex :

```
CREATE ROLE resource_management;
GRANT SELECT, INSERT, UPDATE ON Person
TO resource_management;
CREATE USER jean;
GRANT resource_management TO jean;
```

SQL - Trigger

- Motivation
- Définition
- Exemple

Presentation based on slides created by Prof. Jeffrey D. Ullman

POLYTECH

Trigger Motivation

- Attribute and tuple-based checks have limited capabilities.
  - CHECK(<condition>) →Attribute or tuple
  - They are checked at known times : only on insert or update.

- Assertions are sufficiently general for most constraint applications, but they are hard to implement efficiently.
  - In principle, they should be checked after every modification to any relation of the database
  - The DBMS must have real intelligence to avoid checking assertions that couldn't possibly have been violated.

Trigger Solution

- A trigger allows the user to specify **when the check occurs**.
- Like an assertion, a trigger has a general-purpose condition and also can perform any sequence of SQL database modifications.

Trigger
Event-Condition-Action Rules

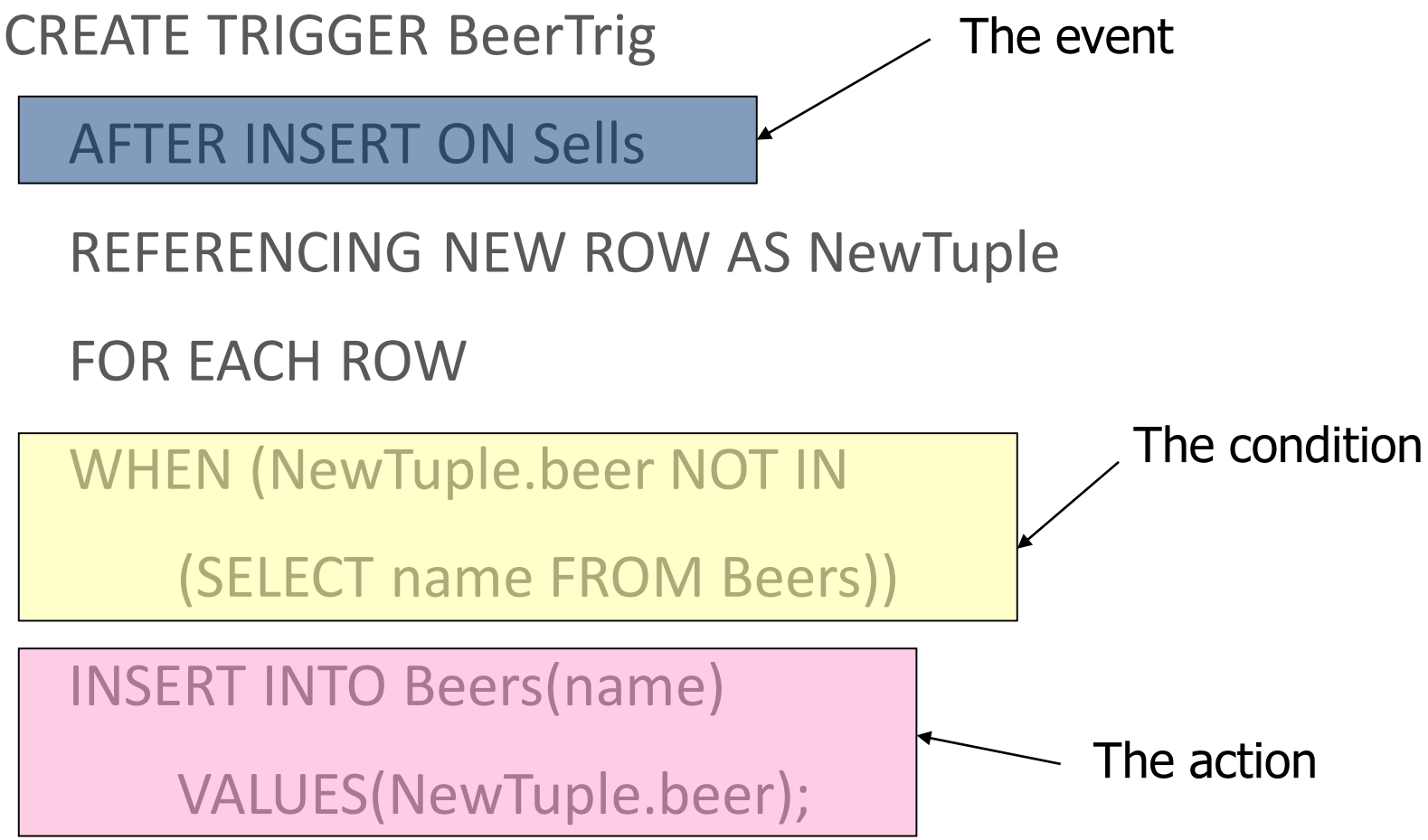
- Another name for “trigger” is *ECA rule*, or Event-Condition-Action rule.
- *Event* : typically a type of database modification, e.g., “insert on Table R1”
- *Condition* : Any SQL Boolean-valued expression.
- *Action* : Any SQL statements.

Trigger Example

Beers(name, manf)
Bars(name, addr, license)
Drinkers(name, addr, phone)
Likes(drinker, beer)
Sells(bar, beer, price)
Frequents(drinker, bar)

- 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.

Trigger
Example of Trigger Definition



Trigger
CREATE TRIGGER

CREATE TRIGGER <name>

Option:

CREATE OR REPLACE TRIGGER <name>

if you want to modify an existing trigger.

Trigger
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.

Trigger
Event 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 are executed once for each modified tuple.

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

Trigger
Event Options: REFERENCING

- INSERT statements imply a new tuple (for row-level) or new set of tuples (for statement-level).
- DELETE implies an old tuple or table.
- UPDATE implies both.
- Refer to these by

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

Trigger
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.

- But always before the changes take effect

Access the new/old tuple or set of tuples through the names declared in the REFERENCING clause.

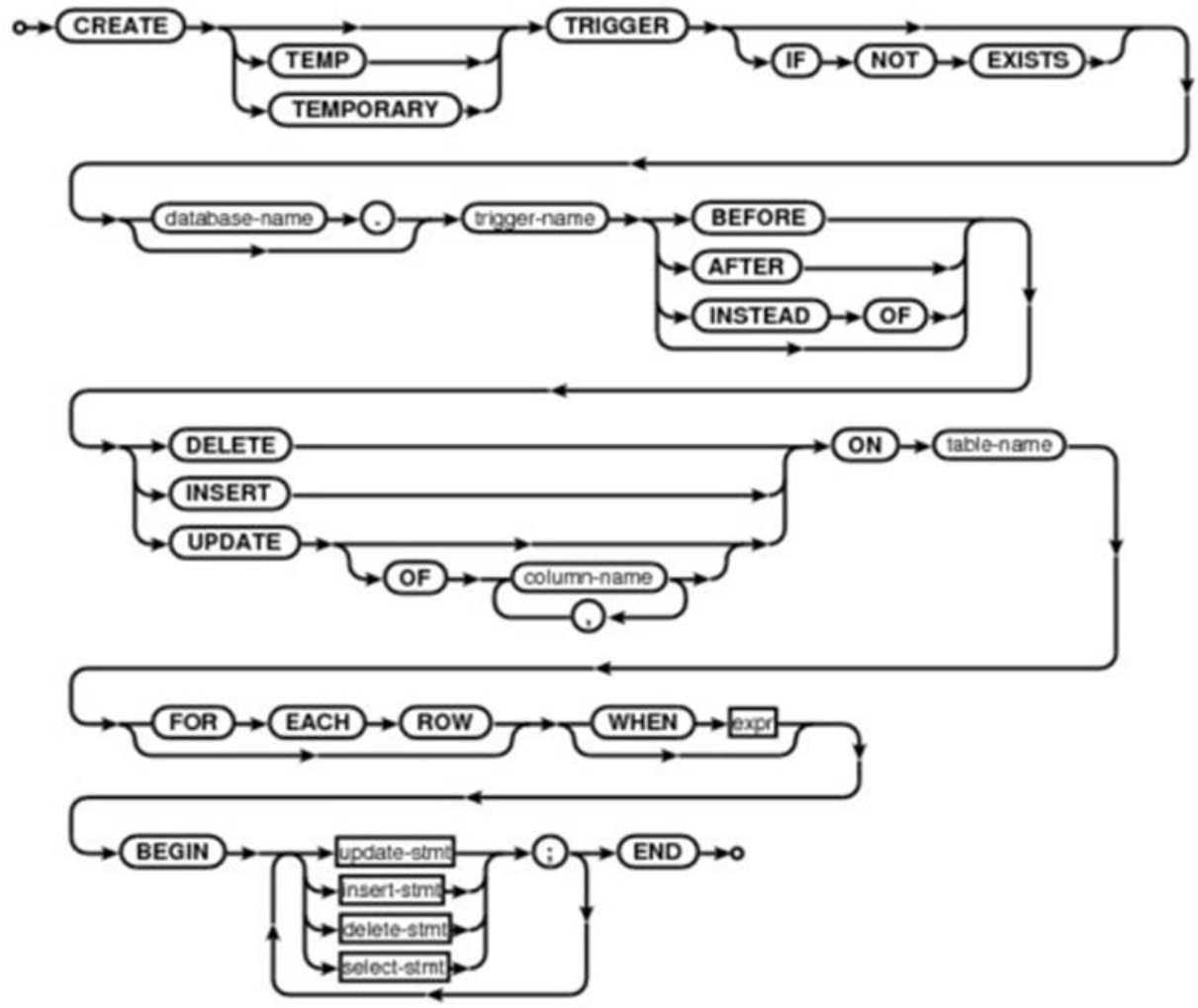
Trigger
The Action

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.

Trigger
SQLite syntax



Event

Condition

Action

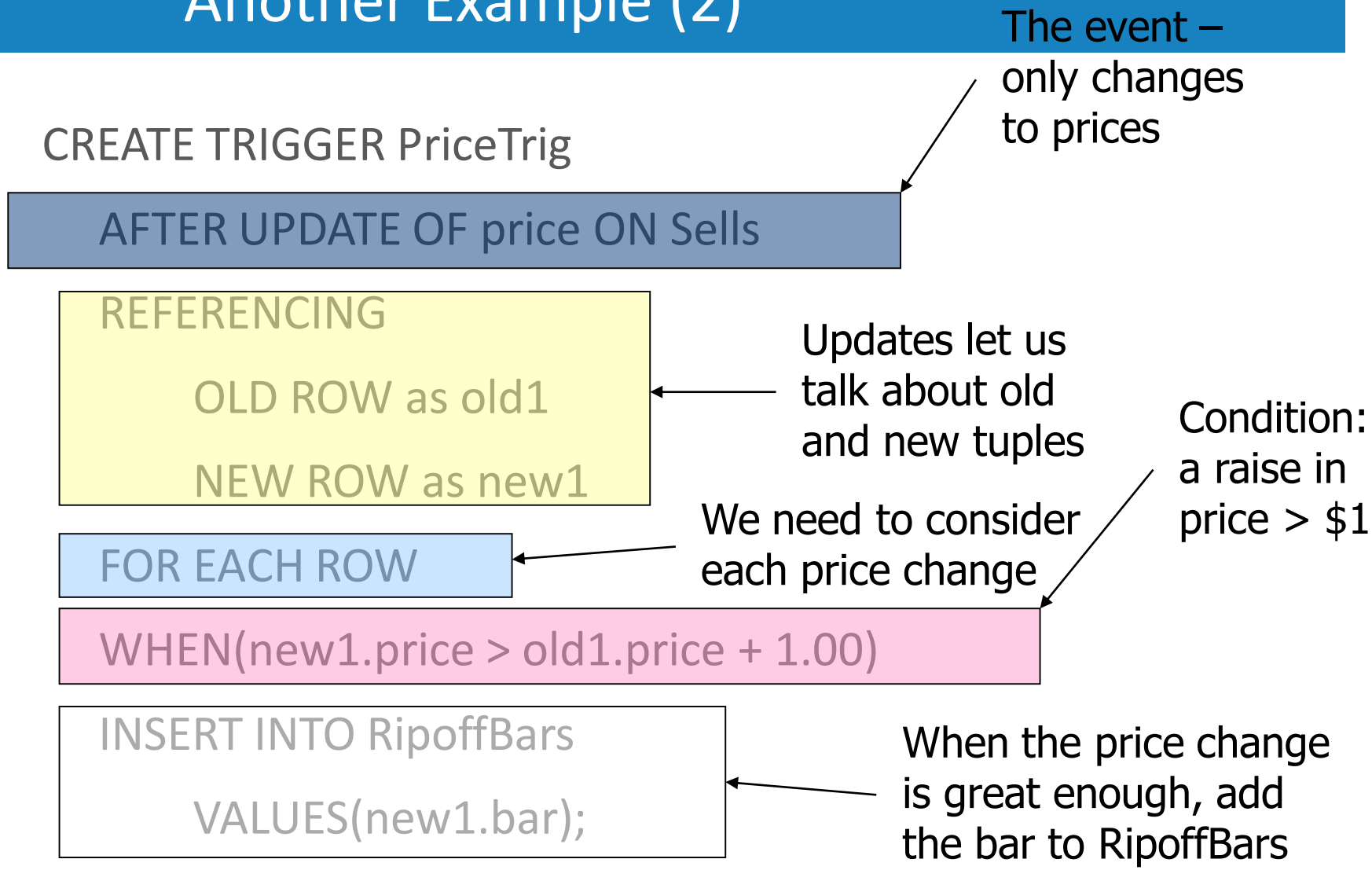
Trigger
Another Example (1)

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.

Trigger
Another Example (2)



Triggers and 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.

Trigger on View

Pick one copy of each attribute

```
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;
```

Natural join of Likes, Sells, and Frequents

Trigger on View

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.

Trigger on View

```
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;
```