Constraints: Attribute-based checks

- We may want to ensure attributes are constrained to a specific range of values
  - That is, constrained above-and-beyond the rules for the attribute's type
- To accomplish this, use **check (<condition>)** to the attribute declaration
- The condition may use the name of the attribute...
- ... but any other relation or attribute name in the condition must be in a subquery.
- Condition must evaluate to **true** or **false**.

Example: attribute-based check

```
create table Sells (
    pub    char(20),
    beer   char(20)
        check (beer in (select name from beers)),
    price  real
        check (price <= 6.00)
);
```

Timing of checks

- Attribute-based checks are performed:
  - when a value for the attribute is inserted
  - when a value for the attribute is updated
- Example: **check (price <= 5.00)**
  - Examines every new price and rejects a modification (for the corresponding tuple) if the price is more then 5.00.
- Example: **check (beer in (select name from beers))**
  - Not checked if a beer is deleted from the Beers relation...!

Constraints: Tuple-based checks

- A **check condition** can be associated with the schema
- In this case, the condition may refer to any of the relation's attributes
  - Note that other attributes from other relations can only be accessed via a subquery.
- Conditions are only evaluated on insert or update operations for the corresponding tuple.
- Example: Only Pat's pub can sell beer for more than $8.00.

Example: tuple-based check

```
create table Sells (
    pub    char(20),
    beer   char(20),
    price  real,
    check (bar = 'Pat''s Pub' or price <= 8.00)
);
```

Constraints: Database-level checks

- Some conditions of interest to us might not fall within one relation or another
- Rather, they might span more than one relation
- For these we could use **assertions**
  - These are database-schema elements
  - They are at the same level as relations and views.
- Defined by: **create assertion <name> check (<condition>);**
- Condition (must be boolean!) can refer to any relation or attribute in the database schema

Example: assertion

- In Sells(pub, beer, price):
  - No pub may charge an average of more than $5 for a beer.
  - Note the condition will probably involve an aggregation operator

```
create assertion NoPriceyPubs check (
    not exists (
        select pub from Sells
        group by pub
        having 5.00 < avg(price)
    )
);
```

Example: assertion

- In Patrons(name, addr, phone) and Pubs(name, addr, license):
  - There cannot be more pubs than there are patrons.

```
create assertion HealthyPubEconomy check (
    (select count(*) from pubs) <=
    (select count(*) from patrons)
);
```

Timing of assertion checks

- In principle:
  - DBMS must check every assertion after every modification to any relation in the database
- However, this can be expensive
- Some cleverness can observe that only certain kinds of changes could cause a particular assertion to be violated
- Example:
  - No possible change to Beers can affect the assertion HealthyPubEconomy
  - No insertion into Patrons can cause a violation (although a deletion could cause a violation)

Triggers

- **Assertions** are powerful...
- ... but the DBMS often cannot determine the most efficient way to check them.
- **Postgres does not yet have assertions.**
- **Attribute- and tuple-based checks** are evaluated at known times...
- ... but they are not as powerful as assertions
- **Triggers** let the database programmer decide when to check for any condition.

Event-Condition-Action Rules

- Another name for a trigger (ECA rule)
- **Event**
  - Typically a type of database modification
  - Example: insert into Sells
- **Condition**
  - Can be any SQL boolean-valued statement
  - Can also be quite complex as it is expressed in **PL/pgSQL**
- **Action**
  - Can be SQL statements
  - Can also be an arbitrary mixing of SQL statements and PL/pgSQL code
- Specifying the trigger means:
  - Constructing a PL/pgSQL function describing the **condition** + **action**
  - Using **create trigger** to link the effect to the condition/action function

Trigger: Introductory example

- Instead of using a foreign-key constraint between Sells and Beers...
- ... which rejects insertions into Sells given an unknown beer...
- ... we can add a trigger that adds the beer to Beer (but with a null manufacturer)
- (Our syntax at this point will depart significantly from that in the textbook.)

Trigger: defining condition + action

```
create function beer_trig_func() returns trigger as $beer_trig$
    begin
        if NEW.beer not in (select name from beers) then
            insert into Beers(name) values (NEW.beer);
        end if;
        return new;
    end;
$beer_trig$ language plpgsql;
```

function prelude and postlude

condition

action

Trigger: linking event with condition+action

```
create trigger BeerTrig after insert on Sells
    for each row
    execute procedure beer_trig_func();
```

Trigger: creation options

- Syntax:
  - **create trigger <name>**
  - **create or replace trigger <name>**
  - Latter form is useful if there already exists a trigger with that name which needs to be replaced
- **after** can be **before**
  - There is also an **instead of** when we want the trigger applied to a view instead of a relation
  - Further to views: Can have triggers modify views by modifying the appropriate base tables
- **insert** can be **delete** or **update**
  - **update** can be further restricted to a specific attribute as **update ... on**

Trigger: row- vs. statement-level

- The number of times the action is triggered can be somewhat controlled
- row-level: execute once for each modified tuple
  - **for each row**
- statement-level: execute once for an SQL statement regardless of how many tuples are modified
  - **for each statement**

Trigger: referencing

- **insert** statements:
  - These imply either a new tuple (for row-level)...
  - ... or a new table (for statement-level) where the table is the set of inserted tuples
- **delete**:
  - implies an old tuple or table
- **update**:
  - implies both insert and delete
- For postgres: only tuples