Functions, Triggers, Rules

Procedures and Functions

- A procedure (or function) is a module performing one or more actions
- https://www.postgresql.org/docs/10/plpgsql-structure.html

- Pros:
  - Server performs heavy-lifting
  - Minimize traffic between client-server (no need to send data to the client)
  - Extend the functionality of the database
- Cons:
  - Software Architecture/vendor dependency
  - Code is in the database

Functions

- The syntax for creating a function:

```
CREATE [OR REPLACE] FUNCTION function_name
   (parameter list)
   RETURN datatype
AS
BEGIN
   <body>
   RETURN (return_value);
END;
```

- To call it:

```
SELECT function_name(parameters,…)
```

Example

```
CREATE OR REPLACE FUNCTION add_event(
  title text, starts timestamp, ends timestamp,
  venue text, postal varchar(9), country char(2))
  RETURNS boolean AS $$
DECLARE
  did_insert boolean := false;
  found_count integer;
  the_venue_id integer;
BEGIN
  SELECT venue_id INTO the_venue_id
  FROM venues v
  WHERE v.postal_code=postal AND
  v.country_code=country AND v.name ILIKE venue
  LIMIT 1;
  IF the_venue_id IS NULL THEN
       INSERT INTO venues (name, postal_code,
       country_code)
          VALUES (venue, postal, country)  RETURNING
       venue_id INTO the_venue_id;
       did_insert := true;
  END IF;
  -- Note: this is a notice, not an error as in
  some programming languages
  RAISE NOTICE 'Venue found %', the_venue_id;
  INSERT INTO events (title, starts, ends,
  venue_id)
      VALUES (title, starts, ends, the_venue_id);
  RETURN did_insert;
END;
$$ LANGUAGE plpgsql;
```

- To execute it:

```
SELECT add_event('Modern Marvels', '2020-12-10 14:00',
'2020-12-10 17:00', 'Seamans Center', '52242', 'us');
```

Triggers

- Programs executed (fired) automatically when a given SQL operation (like INSERT, UPDATE or DELETE) affects the table associated with the trigger.
- Unlike a procedure, or a function, which must be invoked explicitly, database triggers are invoked implicitly.
- Database triggers can be used to perform any of the following:
    - Audit data modification
    - Log events transparently
    - Enforce complex business rules
    - Derive column values automatically
    - Implement complex security authorizations
    - Maintain replicate tables

Triggers

- To use a trigger, we need to first define a trigger procedure, then create the trigger which will execute the trigger procedure
- A trigger procedure is created with the CREATE FUNCTION command, declaring it as a function with no arguments and a return type of trigger.
- The function must be declared with no arguments even if it expects to receive arguments specified in CREATE TRIGGER — trigger, arguments are passed via TG_ARGV

Triggers

- Let’s create a logs event table (to make sure no one changes an event and tries to deny it later)

```
CREATE TABLE logs (
    event_id integer,
    old_title varchar(255),
    old_starts timestamp,
    old_ends timestamp,
    logged_at timestamp DEFAULT current_timestamp
);
```

Trigger example

```
CREATE OR REPLACE FUNCTION log_event() RETURNS trigger AS $$
DECLARE
BEGIN
    INSERT INTO logs (event_id, old_title, old_starts, old_ends)
        VALUES (OLD.event_id, OLD.title, OLD.starts, OLD.ends);
    RAISE NOTICE 'Someone just changed event #%', OLD.event_id;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

Trigger example (cont)

```
CREATE TRIGGER log_events
    AFTER UPDATE ON events
    FOR EACH ROW EXECUTE PROCEDURE log_event();
```

- Now let’s update the Modern Marvels event

```
    UPDATE events
    SET ends=‘2020-12-10 18:00:00'
    WHERE title=‘Modern Marvels’;
```

- And make sure the old event is logged:

```
SELECT event_id, old_title, old_ends, logged_at
FROM logs;
```

Views/Materialized views

```
CREATE VIEW holidays AS
    SELECT event_id AS holiday_id, title AS name, starts AS date
    FROM events
    WHERE title LIKE '%Day%' AND venue_id IS NULL;
```

Can be queried as any other table:

```
SELECT name, to_char(date, 'Month DD, YYYY') AS date
    FROM holidays
    WHERE date <= '2018-04-01';
```

- Materialized view
  - `CREATE MATERIALIZED VIEW mymatview;`
- To refresh it
  - `REFRESH MATERIALIZED VIEW mymatview;`
- It is a table that you can indexed

Updatable views

- Simple views are automatically updatable: the system will allow INSERT, UPDATE and DELETE statements to be used on the view in the same way as on a regular table.
- A view is automatically updatable if it satisfies all of the following conditions:
  - The view must have exactly one entry in its FROM list, which must be a table or another updatable view.
  - The view definition must not contain WITH, DISTINCT, GROUP BY, HAVING, LIMIT, or OFFSET clauses at the top level.
  - The view definition must not contain set operations (UNION, INTERSECT or EXCEPT) at the top level.
  - The view's select list must not contain any aggregates, window functions or set-returning functions.

Update events through the holidays view?

Alter the events table to have an array of associated colors

```
ALTER TABLE events
ADD colors text ARRAY;
```

Update the VIEW query to contain the colors array.

```
CREATE OR REPLACE VIEW holidays AS
SELECT event_id AS holiday_id, title AS name,
starts AS date, colors
FROM events
WHERE title LIKE '%Day%' AND venue_id IS NULL;
```

Now let’s try to update the colors for Christmas

```
UPDATE holidays SET colors = '{"red","green"}'
where name = 'Christmas Day';
```

Update Rules

Rules defined on `INSERT`, `UPDATE`, and `DELETE`

```
CREATE [ OR REPLACE ] RULE name AS ON event TO table
[ WHERE condition ] DO [ ALSO | INSTEAD ] { NOTHING |
command | ( command ; command ... ) }
```

`CREATE RULE` command allows:

- To have no action.
- Can have multiple actions.
- Can be `INSTEAD` or `ALSO` (the default).
- The pseudorelations `NEW` and `OLD` become useful.
  - NEW contains the values we're setting
  - OLD contains the values we query by

Rule Example (cont)

```
CREATE RULE update_holidays AS ON UPDATE TO holidays DO INSTEAD
UPDATE events
SET title = NEW.name,
starts = NEW.date,
colors = NEW.colors
WHERE title = OLD.name;
```

Now try inserting 'New Years Day' on *2020-12-31*

```
CREATE RULE insert_holidays AS ON INSERT TO holidays DO INSTEAD
INSERT INTO …
```

For today

- Create a rule that captures DELETEs on venues and instead sets the active flag (you added in a previous class assignment) to FALSE.

- Try:
- Delete from venues
  where name='University of South Carolina';