Cool Things You Should and Shouldn't Do With PostgreSQL

PLUG Seminar

March 2004

Bernard Blackham

To be covered ...

- Overview of SQL
- Transactions
- Types
- Functions
- Triggers
- Privileges
- Modularity

Overview of SQL

- Database for storing data,
  - eg accounting system:

```
CREATE TABLE users (
    username varchar(8),
    password varchar(20),
    ip_address inet,
    location point
);
```

Overview of SQL

```
CREATE TABLE connections (
    username varchar(8) references users(username),
    start_time timestamp without timezone,
    end_time timestamp without timezone,
    .
    .
    .
    .
);
```

Inserting Data

```
INSERT INTO users
    (username, password, ip_address, location)
    VALUES ('bernard', 'password',
                    '10.11.0.235', '(5,3)');
```

Retrieving Data

- SELECT password, ip_address FROM users
  WHERE username = 'bernard';

- SELECT start_time, end_time
  FROM users u INNER JOIN connections c
  ON u.username = c.username
  WHERE username LIKE 'b%';

Transactions

- BEGIN WORK;
- DELETE FROM user_balances;
- INSERT INTO user_balances VALUES (....);
- ....
- something bad might happen here... ROLLBACK!
- ....
- COMMIT WORK;

Data Types ...

- String, Number, Boolean, etc...
- IP address, Network Addresses
- Geometric Types – Circle, Polygon, Lines
- Arrays

... and their operators

- SELECT username FROM users
  WHERE ip_address << '10.11.4.0/23';
- SELECT username FROM users
  WHERE circle '<(1,1),3)>' ~ location;

Functions in SQL

- SELECT username FROM users
  WHERE circle '<(1,1),3)>' ~ location OR
  box '((8,2),(10,5))' ~ location OR
  polygon '[(0,0),(-10,0),(-5,0))]' ~ location;

- SELECT username FROM users
  WHERE NOT
  (circle '<(1,1),3)>' ~ location OR
  box '((8,2),(10,5))' ~ location OR
  polygon '[(0,0),(-10,0),(-5,0))]' ~ location);

Functions in SQL

- ```
  CREATE FUNCTION in_hotspot(point)
      RETURNS boolean LANGUAGE SQL AS
   'SELECT (circle ''<(1,1),3)>'' ~ location OR
           box ''((8,2),(10,5))'' ~ location OR
           polygon ''[(0,0),(-10,0),(-5,0))]''
   ~ location);'
  ```

- ```
  SELECT username FROM users WHERE
           in_hotspot(location);
  ```
- ```
  SELECT username FROM users WHERE
           NOT in_hotspot(location);
  ```
  ```

Functions in plPgSQL

```
CREATE FUNCTION add_user(varchar, point)
    LANGUAGE plpgsql AS '
 DECLARE
  ip_address inet;
  password varchar;
 BEGIN
  password = random_password();
  IF $1 LIKE '%e%' THEN
    ip_address = get_ip(username);
  ELSE
    RAISE EXCEPTION ''Name has no e''''s'';
  END IF;
  INSERT INTO users VALUES ($1, password,
                        ip_address, $2);
 END;
```

Functions in plPgSQL

- SELECT add_user('bernard', '(1,3)');
- SELECT add_user('bob', '(6,5)');
  EXCEPTION: Username has no e's

Functions in Other Languages

- ```
  CREATE FUNCTION
    perl_max (integer, integer)
    RETURNS integer AS '
      if ($_[0] > $_[1]) { return $_[0]; }
      return $_[1];
    ' LANGUAGE plperl;
  ```
  ```

Functions in Other Languages

```
CREATE FUNCTION get_ip_address(character varying) RETURNS inet
    AS '
    import pyrad.client
    srv = pyrad.client.Client(server="localhost",
       secret="secret", dict=pyrad.dictionary.Dictionary
         ("/usr/lib/snap/radius/dictionary"))
    req = srv.CreateAuthPacket(code=pyrad.packet.AccessRequest,
  User_Name=args[0], NAS_Identifier="localhost")
    req["User-Password"] = req.PwCrypt("password")

    reply = srv.SendPacket(req)
    if reply.code == pyrad.packet.AccessAccept:
        if reply.has_key("Framed-IP-Address") and \
              len(reply["Framed-IP-Address"]):
            return reply["Framed-IP-Address"][0]
        else:
            return None
    else:
        return None
    '
    LANGUAGE plpythonu;
```

Why Functions?

- As with views, hide database complexities: allows for internal changes without client software needing rewriting.
- Restrict privileges:
  - SECURITY INVOKER
  - SECURITY DEFINER
- Real-world interfacing
- Extensibility

Triggers

- ON INSERT
- ON UPDATE
- ON DELETE

Triggers

```
CREATE TABLE users (
  username varchar,
  balance integer
);

CREATE TABLE requests (
  username varchar REFERENCES users(username),
  item_id integer REFERENCES items(id)
);

CREATE FUNCTION is_vip(username) AS ....;

INSERT INTO requests (username, item_id);
```

Triggers

```
CREATE FUNCTION check_request()
 RETURNS trigger LANGUAGE plpgsql AS '
DECLARE
  user_bal integer;
BEGIN
  SELECT INTO user_bal balance
       FROM users
       WHERE username = NEW.username;
  IF (user_bal - cost < 0) AND
       NOT is_vip(username) THEN
    RAISE EXCEPTION ''Balance too low''
  END IF;
  RETURN NEW;
END;
```

Triggers

```
CREATE TRIGGER requests_check
       BEFORE INSERT ON requests
       FOR EACH ROW
       EXECUTE PROCEDURE check_request();
```

- Similarly to update balances...

Writing Modules

- Shared library (normally written in C)
- Similar to writing procedural functions but trickier...

Conclusion

- This is just the surface...
- PostgreSQL has some infinitely useful features.
- Some really should not be abused.
- Though some should :)
- More info:
  `http://www.postgresql.com/`