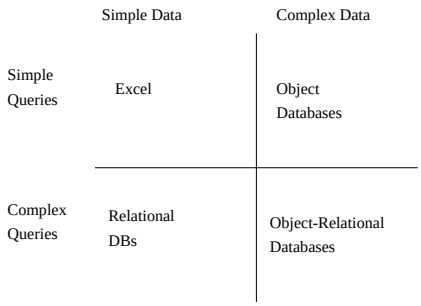
PostgreSQL for the SQL2 Experienced

April 9 2002

Michael Minock

What is Object-Relational?



Object-Relational is just relational technology with added support for table inheritance and more complex data types (BLOBs, spatial primitives, temporal primitives, arrays, lists, sets, references etc). Plus triggers and stored procedures. Plus ...

We will talk about the SQL1999 standard and some commercial offerings (and hype) later.

But let's put off this off and deal with a specific object-relational system: *PostgreSQL*

Object-Relational Features (from last term)

- Inheritance
- Row Types
- Collection Types
- (Encapsulated) User Defined Types
- Reference Types
- Explicit Identity

PostgreSQL Overview

PostgreSQL is the only open source Object-Relational Database available. (Although IBM has made DB2 available for academic use)

PostgreSQL is not entirely SQL1999 compatible (yet - or ever?).

PostgreSQL has good (better than Access, worse than Oracle) SQL2 support.

We are going to cover:

- PostgreSQL features
- The History of PostgreSQL
- The (On-line) Resources
- The Architecture and Components of PostgreSQL
- The Catalog Tables
- Administration and Operation (for this class)

PostgreSQL Features

MAJOR FEATURES BEYOND SQL2:

- Inheritance
- Extended Data Types (Built-in/User Defined)
- Functions - AKA stored procedures.
- Object Identity through OIDs.

OTHER (MAJOR) FEATURES:

- Constraints
- Rules
- Triggers
- Transactions

Inheritance

```
CREATE TABLE Student (
  PID CHAR(9) PRIMARY KEY,
  lname VARCHAR(30),
  fname VARCHAR(20),
  email VARCHAR(20)
);

CREATE TABLE MastersStudent (
  advisor VARCHAR(20),
  thesisTitle  VARCHAR(60)
) INHERITS (Student);

INSERT INTO MastersStudent(PID,lname,fname,email,
                           advisor,thesisTitle)
VALUES ('123456789','Jones','Sammy','sjones@loop.edu',
        'Mike Minock','Querying Query Queries');
```


Inheritance

The following yields no result:

```
SELECT * FROM ONLY Student;
```

Either of these return the tuple:

```
SELECT * FROM MastersStudent;
SELECT * FROM Student;
```

The first query reports more attribute values.

Note that PostgreSQL sidesteps *jagged* reports of more specific tuples.

Note that once a tuple is inserted, its 'class' membership is decided once and for all.

Multiple Inheritance

You may have multiple inheritance:

```
CREATE TABLE StudentWorker (
  taxCredit NUMERIC(5,2)
) INHERITS (Student,Worker);
```

Merges the same named attributes at different levels in the hierarchy.

Fails on type clash.

Inheritance

What does 'inheritance' do?

- Attributes inherited at table construction time.
- Existence constraints - you can not drop parent tables.
- Inclusion Dependencies
- NULL Value constraints

Inheritance

What does it not do (regrettably) ?

- Pass down key constraints
- Pass down rules
- Pass down triggers.

You must manually patch your hierarchy to observe the proper semantics - boo! (10,000 extra credit points to go in and hack PostgreSQL to observer the proper behavior - must also have code added to a future version of PostgreSQL).

Build in Types

Built in types:

```
Character String:  TEXT, VARCHAR(l), CHAR(l)
Number:            INTEGER, INT2, INT8, OID,
                    NUMBERIC(p,d), FLOAT, FLOAT4
Temporal:          DATE, TIME, TIMESTAMP, INTERVAL
Logical:           BOOLEAN
Geometric:         POINT, LSEG, PATH, BOX,
                    CIRCLE, POLYGON
Network:           INET, CIDR, MACADDR
```

Also PostrgreSQL has (multi-dimensional) Arrays and Lists, and BLOBS (Binary large objects). 7.1 and beyond support CLOBS (Character large objects).

(Extended) Types

```
CREATE TABLE House(
 id SERIAL,
 floor_plan POLYGON,
 description TEXT,
 available BOOLEAN,
 picture OID
);

INSERT INTO House
  VALUES('((0,0),(3,0),(3,2),(1,2),(1,1),(0,1))',
         'A nice house in country...',
          true,
          lo_import('/tmp/house1.jpg'));

INSERT INTO House
  VALUES('((0,0),(1,0),(1,1),(0,1))',
         'A shack in the city...',
          true,
          lo_import('/tmp/house2.jpg'));
```

BLOB Exports

```
SELECT description,
  lo_export(house.picture,'/tmp/outresult.jpg')
FROM House
WHERE area(box(floor_plan)) > 4 AND
  available;

       description          | lo_export
----------------------------+-----------
 A nice house in country... |         1
```

100 extra credit points if you write a new version of area that operates over POLYGONS. Must have code accepted into future version of postgreSQL.

Collection Types

```
CREATE TABLE transactions(
  tnum int PRIMARY KEY,
  items int[]);

INSERT INTO transactions VALUES
(55,'{2,3,6,8}');

INSERT INTO transactions VALUES
(56,'{6,7,22,88}');

INSERT INTO transactions VALUES
(57,'{2}');
```


Collection Types

```
SELECT * FROM transactions;

SELECT items FROM transactions
WHERE items[2] = 3;
```

But how about getting those transactions that have item number 6. Does not seem achievable in 7.2. Boo!

Not easy to write a PL/PGSQL function either. Boo!

Would also like the basic union, difference, intersection operations between arrays (list) types.

(5,000 extra credit points to go in and add C functions to PostgreSQL to observer the proper behavior - must also have code added to a future version of PostgreSQL).

Type Extensibility

`CREATE TYPE` provides TYPE extensibility, but you have to then write the constructors, destructors, etc. for the new type.

You may also wish to `CREATE OPERATOR` to allow for simple syntactic operations over your new type (for example overloading of +,-,=, etc.)

But this is hard. No one achieved this last year under 7.02. Want to try this year?

But note that all new tables also induce new types. This gives hope for creating row types (why). You may actually define tables of the row type. But it seems impossible to even do inserts. Boo!

(10,000 extra credit points to go in and hack the PostgreSQL to observer the proper behavior - must also have code added to a future version of PostgreSQL).

Functions

Server side functions improve performance and promote uniformity.

There are already a large number of functions built in (issue `\df`) to list them all. I wish there were more array functions though.

```
CREATE FUNCTION distance(numeric, numeric, numeric, numeric)
RETURNS float8
AS 'Select sqrt(($1 - $3)^2  + ($2 - $4)^2);'
LANGUAGE 'sql';
```

Note that the languages 'C', 'JAVA', 'PLPGSQL' must be added with `createlang` command line operation.

Sequences

We may wish to have more control over numbering. Sequences are the answer.

```
CREATE SEQUENCE my_numbers;
SELECT nextval('my_numbers');
 nextval
---------
       1
```

The functions `currval` and `setval` also exist.

```
CREATE TABLE product (
  product_id INTEGER DEFAULT nextval('my_numbers')
  weight NUMERIC(10,5));
INSERT INTO product VALUES (100);
```


OIDs - object identity

Every tuple receives an OID. This number in unique and immutable and thus **identifies** the tuple.

```
select oid,* from transactions;
  oid   | tnum |    items
-------+------+-------------
 19166 |   55 | {2,3,6,8}
 19173 |   56 | {6,7,22,88}
 19191 |   57 | {2}
```

OIDs are generated across a database installation and thus are not sequential.

Not backed up by default - this could potentially lead to problems.

Serial

Even easier - SERIAL type.

```
CREATE TABLE Customer(
  customer_id SERIAL,
  name CHAR(30)
);
```

Note you must use parameterized inserts over sequences. Boo!

Object versus Value Oriented

Do such constructs adequately address the object oriented versus value-oriented controversy?

Constraints

- NOT NULL
- UNIQUE
- PRIMARY KEY
- FOREIGN KEY/ REFERENCES
- CHECK

CHECKS are similar to ASSERTIONS and DOMAINS in SQL2.

Example

```
CREATE TABLE Owns (
  owner CHAR(9) REFERENCES PERSON(PNUM),
  asset CHAR(4) NOT NULL,
  shares NUMERIC(10,4),
  purchase DATE DEFAULT CURRENT_DATE,
  type CHAR(1) CHECK (type IN ('R','B'))
);
```


Cascade

When you delete a person, shall you delete their participation in relationships?

- NO ACTION
- CASCADE
- SET NULL
- DEFAULT

These are all arguments to the `REFERENCES` constraint.

Rules

Rules execute SQL on INSERT, UPDATE, or DELETE.

```
CREATE RULE integcon1 AS
ON INSERT TO Owns
WHERE NOT EXISTS
 (SELECT *
  FROM Asset
  WHERE Asset.symbol = new.asset)
DO INSTEAD NOTHING;
```


Triggers

Procedure executes BEFORE or AFTER each INSERT, UPDATE, or DELETE per ROW or STATEMENT.

```
CREATE TRIGGER check_critical
AFTER INSERT OR UPDATE ON reactor_status
FOR EACH ROW
EXECUTE PROCEDURE check_meltdown(new.temp, old.tem
```

Does STATEMENT work in 7.2?

Transactions

```
BEGIN WORK;
 UPDATE Owns
 SET shares = shares - 6.0::numeric(8,4)
 WHERE owner = '999999999' AND
   asset = 'MJMC';

 UPDATE Owns
 SET shares = shares + 6.0::numeric(8,4)
 WHERE owner = '999999999' AND
   asset = 'PGE';
COMMIT;

BEGIN WORK;
 UPDATE Owns
 SET shares = shares + 6.0::numeric(8,4)
 WHERE owner = '999999999' AND
   asset = 'MJMC';

 UPDATE Owns
 SET shares = shares - 6.0::numeric(8,4)
 WHERE owner = '999999999' AND
   asset = 'PGE';
COMMIT;
```


Transactions

PostgreSQL will occasionally suspend or event abort and roll-back your transactions.

Transactions have an isolation level: READ COMMITTED and SERIALIZABLE

Miscellaneous

- Regular Expressions with LIKE operator
- Outer Joins
- ...

History of PostgreSQL

The University of California, Berkeley Database Research Group, led by Michael Stonebraker, developed two key, pioneering systems:

- Ingres (1977-1985)
  One of the first relational systems with QUEL query language.
  Spawned Ingres Corporation, which was then acquired by Computer Associates.
- Postgres (1986-1994)
  One of the first object-relational systems.
  Illustra commercialized Postgres and then was acquired by Informix in 1996 - the price tag: 400,000,000$!
  Informix Integrated Illustra into their flagship Universal Server product.

History of PostgreSQL

Thankfully the Postgresql code went another direction as well.

Berkeley graduate students Joly Chen and Andrew Yu added SQL capabilities culminating in Postgres95.

Both students left Berkeley, but Chen still maintained Postgre95 and kept a mailing list

In the Summer of '96 four principle people (in three countries) started seriously (in their spare time) patching up the system (250,000 lines of C code) (Fournier, Lockhart, Mikheev, Momjian)

History of PostgreSQL

Late '96 the name of the system was changed to PostgreSQL.

Remote CVS used by a growing number of global developers.

Releases cycles every every three to five months:

- 1-3 months development
- 1 month beta test
- MAJOR RELEASE
- few weeks of sub-releases to fix bugs

History of PostgreSQL

Code got cleaner, better SQL support, more reliable, etc. PostgreSQL started getting a name for itself.

PostgreSQL added to Red Hat's Linux distribution

PostgreSQL is fully an Open Source system and inherits a BSD (Berkeley Software Distribution) license. Which is very flexible and allows for proprietary specialization of the system.

Great Bridge software has now hired some of the PostgreSQL principles (Momjian, etc.) but it is touting itself as a "Support" company.

The Last Year...

Principles seem to be trying to cash in on book deals.

Never ending threat posed by mySQL.

Improvements (from 7.02 to 7.2):

- 4K limit on TEXT fields lifted
- OUTER JOINS
- Performance
- Small Syntax Changes
- ...

Our PostgreSQL

The latest release of PostgreSQL is 7.2 - and we have it!

How do I know?

```
SELECT version();
                                  version
-------------------------------------------------------------------
 PostgreSQL 7.2 on sparc-sun-solaris2.8, compiled by GCC 2.95.2
```

This system is around 400,000 lines of code.

(On-line) Resources

ALL of the resources are reachable from:

`http://www.postgresql.org/`

Momjian's *PostgreSQL Introduction and Concepts* book The release documents are the basic materials:

- PostgreSQL Tutorial
- PostgreSQL Administrators Guide
- PostgreSQL User's Guide
- PostgreSQL Programmer's Guide
- PostgreSQL Developer's Guide

The man Pages

Architecture

Postgres uses a simple "process per-user" client server architecture

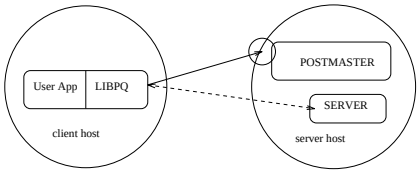
A session consists of the following cooperating UNIX processes:

- A supervisory daemon process (postmaster)
- The user's front end application (e.g. psql)
- One (or more) back-end database servers (the postgres process itself)

A single postmaster manages a given collection of databases on a single host. This collection is called an *installation*.

Architecture

To connect to a database, front end applications (using the library `libpg`) call the postmaster, which in turn spawns a back-end server process to handle the session.



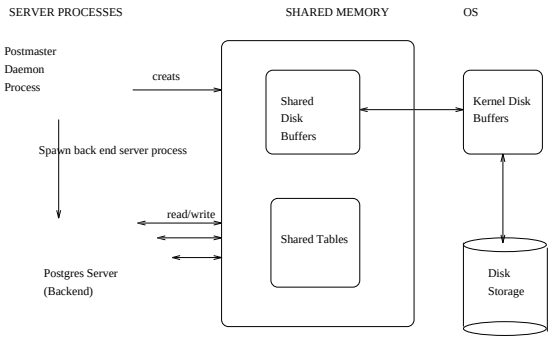
After spawning the back-end server process, the postmaster is out of the loop. Termination of the front end process, terminates the back-end process.

Note that multiple client libraries offer different APIs: libpg, ODBC, JDBC, Perl DBD

Architecture

The postgres back-end server must access a database (or set of databases) stored on the file system.

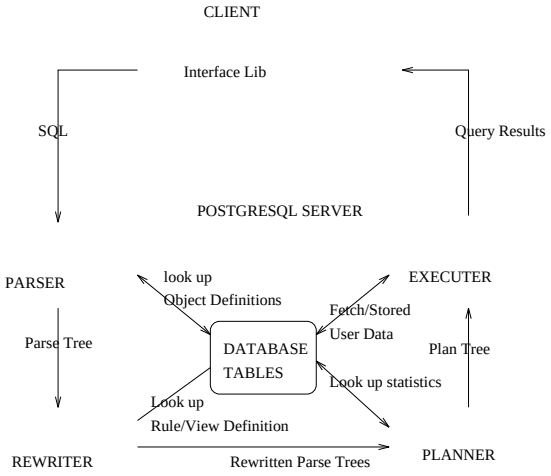
It must also access shared memory.



Architecture

- Hard separation of client and server good for security and reliability
- System works well in a networked environment
- Architecture promotes Portability.
- Shared memory for inter-server communication limits scalability (single server site can not be spread across multiple machines).
- Connection start up overhead is bad for short duration client tasks (use connection side connection pooling).

Components



System Catalogs

`pg_database`
`pg_class`
`pg_attribute`
`pg_index`
`pg_relcheck`
`pg_proc`
`pg_laguage`
`pg_aggregate`
`pg_operator`
`pg_type`
`pg_description`
`pg_group`
`pg_log`
`pg_rewrite`
`pg_shadow`
`pg_trigger`

Administration

The postgres system binaries are installed at: `/opt/postgres/`

Preparing a database installation: `initdb` (creates the shared catalog tables and creates the initial db *template1*) .

I have run `initdb -D DB` from `~mjm`. So everything will land in `/Home/staff/mjm/DB`.

Administration

Running the Server:

```
postmaster -D /Home/staff/mjm/DB -p 5006 -i
```

Create DBs through `createdb`.

```
createdb -h kant.cs.umu.se -p 5006 db01
```

I have created 5 databases for this class. `db01`,`db02`, ..., `db05`.

I shall create more as they are needed.

Client Connection

Let us suppose that you (and your) partners have been assigned to the database `db01`.

To connect (from unix) type:

```
/opt/postgres/bin/psql
   -h kant.cs.umu.se -p 5006
   -d db01 -U db01
```

You should then see:

```
Welcome to psql, the PostgreSQL interactive terminal.

Type:  \copyright for distribution terms
       \h for help with SQL commands
       \? for help on internal slash commands
       \g or terminate with semicolon to execute query
       \q to quit

db11=>
```


Conclusions

We covered the features, history, documentation, architecture, internals, and finally a rough set of instructions to get you up an running.

You should now be ready to take on an assignment.