Extensions are good for business logic

Beamer Theme: Amsterdam

Dimitri Fontaine

Oct 20, 2011

Extensions? Logic? Fuzzy Business? Say what?

1 Extensions and Business Logic
What's an Extension?
MVC: Where's the Model
Packaging your in database Model

2 Managing upgrades
Extension upgrades
From development to production
Managing Rollouts

3 Conclusion
Any question?

Extensions? Logic? Fuzzy Business? Say what?

1 Extensions and Business Logic
   What's an Extension?
   MVC: Where's the Model
   Packaging your in database Model

2 Managing upgrades
   Extension upgrades
   From development to production
   Managing Rollouts

3 Conclusion
   Any question?

Extensions? Logic? Fuzzy Business? Say what?

1 Extensions and Business Logic
- What's an Extension?
- MVC: Where's the Model
- Packaging your in database Model

2 Managing upgrades
- Extension upgrades
- From development to production
- Managing Rollouts

3 Conclusion
- Any question?

Extensions

Extensions	PostgreSQL is very extensible, and with full support now. Almost all about SQL solving is possible to implement as an extension.
Full Support	Wait, I wish you were here • Stuttgart, December 2010, PgDay • Brussels, February 2011, FOSDEM • Ottawa, May 2011, PgCon
Featuring	dump & restore, versioning, upgrades, dependencies

Extensions

Extensions
: PostgreSQL is very extensible, and with full support now. Almost all about SQL solving is possible to implement as an extension.

Full Support
: Wait, I wish you were here

  - Stuttgart, December 2010, PgDay
  - Brussels, February 2011, FOSDEM
  - Ottawa, May 2011, PgCon

Featuring
: dump & restore, versioning, upgrades, dependencies

Extensions

Extensions	PostgreSQL is very extensible, and with full support now. Almost all about SQL solving is possible to implement as an extension.
Full Support	Wait, I wish you were here • Stuttgart, December 2010, PgDay • Brussels, February 2011, FOSDEM • Ottawa, May 2011, PgCon
Featuring	dump & restore, versioning, upgrades, dependencies

Some extensions example

46 Contribs, Community extensions, Private ones...

- cube
- ltree
- citext
- hstore
- intagg

- adminpack
- pgq
- pg_trgm
- wildspeed
- dblink

- PostGIS
- ip4r
- temporal
- prefix
- pgfincore

- pgcrypto
- pg_stattuple
- pg_freespacemap
- pg_stat_statements
- pg_standby

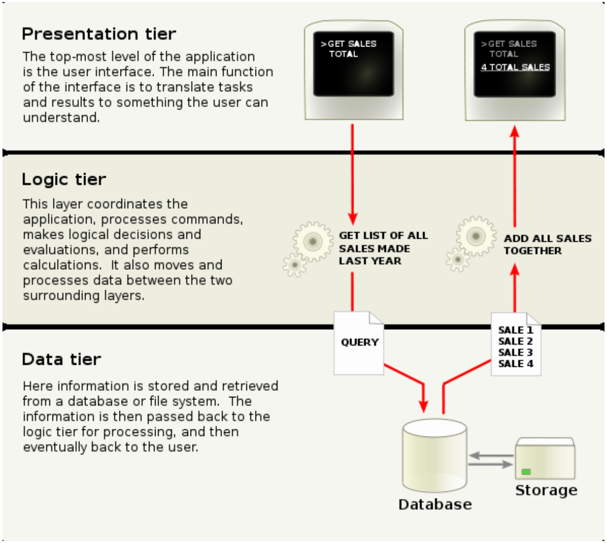
Some extensions are simpler than that

For the sake of this talk, if you have some *business logic functions* in your database, you have an extension. Even `VIEW` qualifies.

Example (Very simple extension)

```
CREATE OR REPLACE FUNCTION accounting.vat(numeric)
 RETURNS numeric
 LANGUAGE SQL
AS $$
  RETURN $1 * 0.196;
$$;
```

MVC: Where's the Model



Presentation tier

The top-most level of the application is the user interface. The main function of the interface is to translate tasks and results to something the user can understand.

Logic tier

This layer coordinates the application, processes commands, makes logical decisions and evaluations, and performs calculations. It also moves and processes data between the two surrounding layers.

Data tier

Here information is stored and retrieved from a database or file system. The information is then passed back to the logic tier for processing, and then eventually back to the user.

Put the logic into the database layer

35.15. Packaging Related Objects into an Extension 1/6

Example (pair–1.0.sql)

```
CREATE TYPE pair AS ( k text, v text );

CREATE OR REPLACE FUNCTION pair(anyelement, text)
RETURNS pair LANGUAGE SQL AS 'SELECT ROW($1, $2)::pair';

CREATE OR REPLACE FUNCTION pair(text, anyelement)
RETURNS pair LANGUAGE SQL AS 'SELECT ROW($1, $2)::pair';

CREATE OR REPLACE FUNCTION pair(anyelement, anyelement)
RETURNS pair LANGUAGE SQL AS 'SELECT ROW($1, $2)::pair';

CREATE OR REPLACE FUNCTION pair(text, text)
```

35.15. Packaging Related Objects into an Extension 2/6

Example (pair–1.0.sql)

```
CREATE OPERATOR ~> (LEFTARG = text, RIGHTARG = anyelement,
                    PROCEDURE = pair);
CREATE OPERATOR ~> (LEFTARG = anyelement, RIGHTARG = text,
                    PROCEDURE = pair);
CREATE OPERATOR ~> (LEFTARG = anyelement, RIGHTARG = anyeler
                    PROCEDURE = pair);
CREATE OPERATOR ~> (LEFTARG = text, RIGHTARG = text,
                    PROCEDURE = pair);
```

35.15. Packaging Related Objects into an Extension 3/6

PostgreSQL needs some *metadata* about your extension, fill in the `control` file.

Example (pair.control)

```
# pair extension
comment = 'A key/value pair data type'
default_version = '1.0'
relocatable = true
```

35.15. Packaging Related Objects into an Extension 4/6

To easy the package installation process, you need a scary `Makefile`. Beware of `VPATH`, he's your friend, but he's very picky about it.

Example (Makefile)

```
EXTENSION = pair
DATA = pair--1.0.sql # avoid $(wildcard sql/*--*.sql)

PG_CONFIG = pg_config
PGXS := $(shell $(PG_CONFIG) --pgxs)
include $(PGXS)
```

35.15. Packaging Related Objects into an Extension 5/6

Now, relax and profit.

Example (psql)

```
CREATE EXTENSION pair SCHEMA utils;
```

35.15. Packaging Related Objects into an Extension 6/6

Oh, and maybe you wanted to use the extension, too.

Example (psql)

```
CREATE TABLE foo(kv pair);
INSERT INTO foo(kv)
     SELECT 'x' ~> 'y';
```

Upgrading an extension

That used to be a “guru” only operation...

Example (extension update)

```
ALTER EXTENSION pair UPDATE;
ALTER EXTENSION pair UPDATE TO '1.1';

SELECT * FROM pg_available_extensions();
SELECT * FROM pg_available_extension_versions();
```

Packaging upgrades in development

Example (update to 1.4)

```
ALTER EXTENSION pair UPDATE TO '1.1';
...
ALTER EXTENSION pair UPDATE TO '1.4';
```

- `pair--1.0.sql`
- `pair--1.0--1.1.sql`
- `pair--1.1--1.2.sql`
- `pair--1.2--1.3.sql`
- `pair--1.3--1.4.sql`

Packaging upgrades in development

Example (update to 1.4)

```
ALTER EXTENSION pair UPDATE TO '1.1';
...
ALTER EXTENSION pair UPDATE TO '1.4';
```

- `pair--1.0.sql`
- `pair--1.0--1.1.sql`
- `pair--1.1--1.2.sql`
- `pair--1.2--1.3.sql`
- `pair--1.3--1.4.sql`

Packaging upgrades in development

Example (update to 1.4)

```
ALTER EXTENSION pair UPDATE TO '1.1';
...
ALTER EXTENSION pair UPDATE TO '1.4';
```

- `pair--1.0.sql`
- `pair--1.0--1.1.sql`
- `pair--1.1--1.2.sql`
- `pair--1.2--1.3.sql`
- `pair--1.3--1.4.sql`

Packaging upgrades for production rollouts

Example (update to 1.4)

```
ALTER EXTENSION pair UPDATE TO '1.4';
```

- `\dx` shows we're at version `1.0`
- PostgreSQL will happily apply those files:
- `pair--1.0--1.1.sql`, `pair--1.1--1.2.sql`,
- `pair--1.2--1.3.sql`, `pair--1.3--1.4.sql`
- Check with `pg_available_extension_versions()`!

Packaging upgrades for production rollouts

Example (update to 1.4)

```
ALTER EXTENSION pair UPDATE TO '1.4';
```

- `\dx` shows we're at version `1.0`
- PostgreSQL will happily apply those files:
- `pair--1.0--1.1.sql`, `pair--1.1--1.2.sql`,
- `pair--1.2--1.3.sql`, `pair--1.3--1.4.sql`
- Check with `pg_available_extension_versions()`!

Packaging upgrades for production rollouts

Example (update to 1.4)

```
ALTER EXTENSION pair UPDATE TO '1.4';
```

- `\dx` shows we're at version `1.0`
- PostgreSQL will happily apply those files:
- `pair--1.0--1.1.sql`, `pair--1.1--1.2.sql`,
- `pair--1.2--1.3.sql`, `pair--1.3--1.4.sql`
- Check with `pg_available_extension_versions()`!

Packaging upgrades for production rollouts

Sometimes playing each step one after the other is not what you want.

- Prepare `pair--1.0--1.4.sql`
- PostgreSQL will happily prefer this file
- Check with `pg_available_extension_versions()`

Packaging upgrades for production rollouts

Sometimes playing each step one after the other is not what you want.

- Prepare `pair--1.0--1.4.sql`
- PostgreSQL will happily prefer this file
- Check with `pg_available_extension_versions()`

Packaging upgrades for production rollouts

Sometimes playing each step one after the other is not what you want.

- Prepare `pair--1.0--1.4.sql`
- PostgreSQL will happily prefer this file
- Check with `pg_available_extension_versions()`

Any question?

Now is a pretty good time to ask!

If you want to leave feedback, consider visiting
`http://2011.pgconf.eu/feedback`