



Relational Databases and Plone

Joel Burton

joel@joelburton.com

Plone Conference I

Why Relational Databases?

- Nervous manager / company policy
- Simultaneous access with other applications
- Query flexibility/speed
 - Rel DBs have index v. non-index
 - Query/reporting tools for rel Dbs

PostgreSQL

- High-performance, featureful, BSD-licensed
 - Full SQL: UNION, subselects, etc.
 - Transactions
 - User functions (Python, Perl, Ruby, Tcl, PLSQL)
 - Arrays, points, & many other data types
 - Views & rules
- ZpyscopgDA is excellent DA for PG

ZODB in DB

- DCOracle, DB in Sleepycat
- Entire ZODB tree kept in DB
- Entries are pickles—not useful
- Mostly beneficial to fit company policies, replication, scalability concerns

ZSQL Methods

- Wide variety of database adaptors
- Database transactions = Zope transaction
 - MySQL: use “-” at start

Caching Database Results

- Cache for s seconds
- Cache maximum n result sets
- Can't easily clear cache w/o restarting Zope
- Caches not shared across Zope threads
- Fix w/ DM's *CacheControlledZSQLMethods*
- Or, cache results of skins that uses obj.

Pluggable Brains

- Allow binding of on-disk Python class to database results
- Preferable to embedding logic in DB or keeping in dozens of skins.

```
SELECT first, last  
FROM Clients;
```

```
def full_name(self):  
    "Returns last name"  
    return self.first,  
           self.last
```

URL Traversal

- Can pass parameters in URL:
<http://z.com/zsql/first/joel/last/burton/skin>
- If only one parameter, can opt for simple:
<http://z.com/zsql/123/skin>
- Allows for de-coupling of ZSQL method & skin, rather than having to have $n \times m$ PythonScripts to combine or having to pass/process skin name

Traversal (cont'd)

- Can combine w/pluggable brain to make:
<http://z.com/zsql/id/123>
- Have brain have an `index_html` method which returns `self.SkinName()`

Editing ZSQL Methods

- TTW
- ExternalEditor
 - Don't forget to investigate SQL editing modes
- FTP
- WebDAV
 - Requires minor fix to ZSQL product (in Zope.org tips; link at *joelburton.com*)

Editing FSDV ZSQL Method

⊙ `<dtml-comment>`

`title: An on-disk ZSQL method`

`connection id: my_db_connection`

`arguments: id name:string='bob'`

`max_rows: 1000`

`max_cache: 500`

`cache_time: 500`

`</dtml-comment>`

Performance Tips

● Loop at lowest level

- in ZSQL statement: `<dtml-in>`,
`<dtml-var sql_delimiter>`
- in DB

● Prepare queries

- Using PREPARE/EXECUTE with loops
- or procedural languages (easier to cross over sessions, etc.)

Results By Agg. Functions

- Allow faster results for list of lists
- Good for comma-separated, html lists, etc.
- Moves some logic into DB
- Saves time, wire

```
CREATE FUNCTION
  add_comma ...

CREATE AGGREGATE
  commaify ...

SELECT name, commaify(tasks)

FROM Tasks
GROUP BY name;
```

Inline ZSQL

- Sometimes easier to follow SQL logic in skin or script than by decoding name
- Replace many disorganized ZSQL methods
- One **generic_sql** method where entire statement is dtml-var

User Authentication

- All users share one Database Adaptor
 - Unless your ZODB logic forces otherwise
- (PG-specific) Can “su” to DB user:
SET SESSION AUTHORIZATION joel;
- Database security is easier, cleaner, better-audited than Zope security.

Schemas

- PostgreSQL-specific
- Multiple namespaces in a single DB
- **CREATE TABLE joel.tasks (...)**
SELECT * FROM joel.tasks;
- All users have a “search-order” of namespaces
- Different Zope users could have different path, making DB change “miraculously”

APE

- ⊙ Part of ZODB stored entirely in DB (or fs...)
- ⊙ Different storages for different sections
- ⊙ Not final release, but used by many
- ⊙ Subobjects must be stored by APE
- ⊙ Objects must be in ZODB together to benefit

APE, cont'd

- Can write own mapping to map object to own table
 - “experimental”
- Excellent performance
- Archetypes may be moving to APE-level storage

SQL Storage

- Backend storage possibility for Archetypes
- Store some/all attributes of obj in DB
- Set on an attribute basis; can mix-and-match
- Tables are by class (very intuitive, useful)
- Subobjects can be in ZODB
- Objects can be all over ZODB

SQLStorage Table Creation

- Table is named for object type name
- SQLStorage automatically creates on use
 - You can modify afterwards
 - Or, better, you can create instead

SQLStorage Parent/Child

- All Archetypes objects get UID
- Allows moving of objects around ZODB & still finding
- Subobjects of Archetype objects get a *parentuid* field

SQLStorage Marshalling

- Can custom marshall Zope type to SQLStorage type
- Python list could be stored as:
 - newline-joined string
 - pickle/bytea data
 - classic parent/child table
 - array

Modifying Mapping

```
def map_lines(self, fld, val):  
    return '\n'.join(val)  
  
def unmap_lines(self, fld, val):  
    return val.split('\n')  
  
db_type_map =  
    { ..., 'lines': 'text', ... }
```

Domains

- Name for a database field type + restrictions
 - `home_phone varchar(15)`
`not null check (...)`
 - `home_phone phone`
- Same input/output but self-documenting and easy to keep consistent

Notifying Zope of Changes

- DB can notify Zope of changes
- Allows for caching of ZODB objects while still reflecting DB changes
- Allows for catalog reindexing on DB changes
- Allows for deletion/insertion to DB that delete/create ZODB object

Notifying Zope of Changes

- DB trigger calls *wget*
- *wget* calls PythonScript that recatalogs or clears cache
- Syntax in Archetypes' SQLStorage-HOWTO
- In future PostgreSQL, should be possible to have PG directly alter ZODB through PL/Python

SQLStorage and Views

- ⊙ Instead of using table, SQLStorage can query/write to a view
- ⊙ Allows us to marshall/modify/hide in DB
- ⊙ Interesting security checks in DB possible
- ⊙ Create VIEW first so SQLStorage will use it
- ⊙ Add rules to make view updatable