"Building Portable Database Applications"

php|works 2006 in Toronto

Lukas Kahwe Smith
smith@pooteeweet.org

Agenda: Overview

- Introduction
- ext/PDO
- PEAR::MDB2
- ORM and ActiveRecord
- SQL Syntax
- Result Sets
- High Level API
- Error Handling
- Schema Management

Introduction: Motivation

- Support for multiple RDBMS
  - Increase market opportunities
- Forward compatibility
  - new RDBMS versions
  - new PHP (extension) like MySQLi and PDO
- Safe Money
  - Prevent vendor Lock in
  - Reduce train costs
- Pushing your own preference

Introduction: Levels of Abstraction

- Client API
  - Portable class/method/function names
  - Ex.: sqlite_logon vs. mysql_connect
- Server API
  - Portable SQL
  - Ex.: CONCAT(a, b) vs. a || b
- Higher Level API
  - Do common tasks automatically
  - Ex.: $mdb2->queryAll($query);

Introduction: Code Structuring

- One applications for each RDBMS
- Separate business logic from
  - Database interaction
  - SQL queries
- Write portable SQL with a database abstraction layer (DBAL)
  - Generic abstraction layer
  - Domain specific abstraction layer
- Mix and Match!

ext/PDO: Overview

- C Extension
- Thin-Layer
- Bundled since PHP 5.1 but also available through PECL
- OO interface
- Client API unification
- Little Server API unification
- Little High Level API
- Marked with !

PEAR::MDB2: Overview

- PHP userland code
- Fairly thin, with many optional features
- Available through PEAR
- Compatible with PHP4 and PHP5
- OO Interface
- Client API unification
- Server API unification
- High Level API
- Marked with **!**

ORM and ActiveRecord: Object Relational Mapper

- Provide a language native API
- No need to write SQL
  - Usually allow writing (parts) of the SQL
  - Can do a lot more for portability than a DBAL
- Most run on top of an DBAL
- I have not really been drawn in by anyone that is around
  - DB_DataObject, DB_Table, [M]DB_Querytool, Propel, ezPersistance ...

SQL Syntax: Data Types

- Data types (!)
  - Date/Time/Timestamp: ISO Date (!)
  - Boolean: True/False (!)
  - Decimal: Fixed Precision (!)
  - Float: Approximate Precision (!)
  - LOB: Large Objects (!!)

```
$mdb2->quote($var, 'text');
$mdb2->quote($date, 'timestamp');
$mdb2->quote($bool, 'boolean');
$mdb2->quote($fp, 'blob');
```

SQL Syntax: Prepared Statements

- Positional ? placeholders (!!)

```
// create sql string
$sql = 'INSERT INTO foo (?, ?, ?)';

// send off query
$stmt = $pdo->prepare($sql);

// bind values
$stmt->bindValue(1, $text);
$stmt->bindValue(2, $decimal);
$stmt->bindValue(3, $fp);
```

SQL Syntax: Prepared Statements

- Named placeholders (!!)

```
$sql = 'INSERT INTO foo (:url, :dm, :file)';

// prepare query
$types = array('text', 'decimal', 'blob')
$stmt = $mdb2->prepare($sql, $types,
  MDB2_PREPARE_MANIP);

// bind variables
$stmt->bindParam('url', $text);
$stmt->bindParam('dm', $decimal);
$stmt->bindParam('file', $fp);
```

SQL Syntax: Prepared Statements

- Execute statement (!!)

```
// send off query
$result = $stmt->execute();

// modify bound variables
$text = 'foo';
$decimal = 23453/8;
$fp = 'bar.pdf';

// send off query
$result = $stmt->execute();
```

SQL Syntax: Unique Key Generation

- Sequences vs. auto increment
  - Sequences can emulate auto increment with a trigger (!!)

```
$id = $db->lastInsertID($sequence);
```

  - Auto increment can emulate sequences with a table (!)

```
$id = $mdb2->getBeforeID($table);
$sql = "INSERT INTO $table (".$mdb2->quote($id,
  'integer').', 'foo')';
$affectedRows = $mdb2->exec($sql);
$id = $mdb2->getAfterID($id, $table);
```

SQL Syntax: LIMIT

- SELECT ... LIMIT (!)
  - Non Standard SQL extension

```
// set limit
$mdb2->setLimit(10, 20);
$mdb2->query('SELECT * FROM foo');

// MySQL : SELECT * FROM foo LIMIT 10 OFFSET 20
// MS SQL: SELECT * FROM foo TOP 30
// Oracle: SELECT * FROM (
      SELECT a.*, ROWNUM mdb2rn FROM (
        SELECT * FROM foo
      a WHERE ROWNUM <=30)
    WHERE mdb2rn >=11
```

SQL Syntax: JOINs and Subqueries

- JOINs
  - OUTER JOINs: (+) vs. * vs. ANSI
  - JOINs in UPDATE/DELETE

```
UPDATE t1, t2 SET t1.p=t2.p WHERE t1.id=t2.id
```

- Subqueries
  - Less optimized
  - Row comparisons limitations:

```
WHERE (v1, v2) < ALL (SELECT ..)
```

  - UPDATE/DELETE on the same table:

```
DELETE FROM t1 WHERE .. (SELECT .. FROM t1 ..)
```

SQL Syntax: Functions and Operators

- Functions (!)
  - SUBSTRING() vs. SUBSTR()
  - NOW() vs. CURRENT_TIMESTAMP
  - RAND() vs. RANDOM() vs. ((RANDOM()+2147483648/4294967296)
- Operators
  - Pattern Matching
    - LIKE vs. ILIKE vs. .. (!)
    - REGEXP vs. SIMILAR
  - AS: required vs. optional vs. forbidden (!)
  - CONCAT(a, b) vs. a || b (!)

SQL Syntax: Other ..

- GROUP BY

```
SELECT A, B, MAX(C) FROM T1 GROUP BY A
```

- Transactions
  - START TRANSACTION vs. BEGIN (!!)
  - Different ISOLATION LEVEL (!)
- Multi value INSERT

```
INSERT INTO t1 VALUES (i) (1),(2),(3)
```

- CSV import/export
  - LOAD DATA vs. COPY vs. ..
- Identifier quoting (!)
  - `foo` vs. “foo” vs. [foo]

Result Sets: Data Fetching

- Buffered results (!)
- NULL vs. empty strings (!!)
- Associative fetches (!!)
  - Removing qualifiers from key (!)
- Removing string padding (!)
- Introspection (!!)
- Data types (!)

```
// create sql string
$sql = 'SELECT dateBirth, decMoney FROM foo';
// send off query
$types = array('date', 'float')
$res = $mdb2->query($sql, $types);
```

High Level API: Result "Buffering" in PHP

- fetchAll() (!!)
  - Read result into a multi dimensional array
  - Effective result buffering
  - Rekey
    - Use the first column as the key for the first array dimension instead of the row number
  - Group
    - All rows with the same value in the first column are grouped inside another array
  - Various other optional features

High Level API: “Nested” Transactions

```
$mdb2->beginNestedTransaction();
$query = "INSERT INTO autoinc (id) VALUES (?)";
$stmt = $mdb2->prepare($query);
$stmt->execute(array(1));

function foo() {
  $savepoint = $mdb2->beginNestedTransaction();
  ..
  $mdb2->rollback($savepoint);
}

foo();
$stmt->execute(array(2));
$mdb2->completeNestedTransaction();
```

High Level API: The Sky is the Limit

- Abstract data types (!)

```
$mdb2->quote(array(1, 2, 3), 'serialize');
```

- Iterators (!)
  - Via custom result wrapper (!)
- REPLACE (!)
- DML SQL builder (!)
- Schema management (!)
- Schema reverse engineering (!)
- Single column non-correlated subquery emulation (!)

Error Handling: Error Codes

- Different error codes
- No support for error codes
- Portable error codes (!!)
  - Connection errors
  - Syntactical errors
  - Semantic errors
  - Access violations
  - etc.

Error Handling: Exceptions, PEAR_Error

```
// PDO has optional exceptions ..
$pdo->setAttribute(PDO::ATTR_ERRMODE,
  PDO::ERRMODE_EXCEPTION);

// MDB2 allows setting a global error handler
PEAR::setErrorHandling(PEAR_ERROR_DIE);
// expect no such table error
$mdb2->expectError(MDB2_ERROR_NOSUCHTABLE);
$res = $mdb2->queryAll($sql);
$mdb2->popExpect();
// table didnt exist?
If (MDB2::isError($res,MDB2_ERROR_NOSUCHTABLE))
```

Schema Management: Main Considerations

- Optimize schema
  - Portability
    - Triggers for auto increment Emulation
  - Performance
    - Normalization
    - Partitioning
    - Buffer sizes
- Cross joining databases/schema

```
SELECT * FROM db1.t1, db2.t2
```

  - Stick everything in one database/schema or under one user for Oracle
- GRANT handling is a nightmare

Schema Management: Schema Evolution

- Getting the DDL and DML
  - SQL diff tools can generate DDL
    - Usually RDBMS specific
  - Data synchronisation very tricky
    - Especially if DDL and DML needs to be mixed
  - Deal with object dependencies
    - Foreign Keys, Views, Stored Routines
  - Column order matters with sloppy code
- No additional steps during development
- Less steps during packaging
- Allow releases to be skipped

Schema Management: Example

- Version 1.0.0
  - User table with a single phone number
- Version 1.1.0
  - Allow infinite phone numbers per user
    - Add new table phone numbers

```
CREATE TABLE phone_nums (user_id INT REFERENCES
  user, phone_num CHAR(20))
```

    - Move all data into the new table

```
INSERT INTO phone_nums (user_id, phone_num)
  SELECT user_id, phone_num FROM users
```

    - Drop the old phone numbers column

```
ALTER TABLE users DROP COLUMN phone_num
```

Schema Management: Tools

- Diff Tools
  - Generate DDL by comparing
  - Does not handle DML
- ER Modeling tools
  - Visually design schema
  - Synchronize model
  - Reverse engineer model
- Replication and Logging
  - Very RDBMS specific
- Generic Schema Formats
- SCM Tools

Thank you for listening ..
Comments? Questions?

smith@pooteeweet.org

References:

- These slides
  - http://pooteeweet.org/files/phptek06/building_portable_db_apps.pdf
- „SQL Performance Tuning“ by Peter Gulutzan and Trudy Pelzer
- SQL Syntax Tips
  - http://www.analysisandsolutions.com/presentations/portability/slides/index.htm
  - http://troels.arvin.dk/db/DBMS/
- WebBuilder2 Schema Manager
  - http://svn.oss.backendmedia.com/modules/schema_manager/schema_manager.phps