1 The Next Generation: MySQL 5 + PHP 5

-
- ApacheCon Europe 2005
  -
  - July 19, 2005 :: Stuttgart
  -
  - Georg Richter & Zak Greant

2 About Georg Richter

- Author/maintainer of PHP's MySQL and ncurses extensions
- Author of MySQL Connector/OO.org
- ASF Member
- MySQL AB Senior Developer

3 About Zak Greant

- Co-maintainer of PHP’s MySQL extensions
- Works with eZ systems as their Director, Free Software and Open Source
- Author, PHP Functions Essential Reference

4 Questions ?

- If something isn’t clear, just ask
- ... or wait for a break
- ... or wait for the end of the tutorial
- ... or send mail to apachecon@greant.com

5 How many of you use: (in production)

- PHP 4.x ?
- PHP 5.0.x ?
- PHP 5.1 ?
-
- MySQL 3.23 ?
- MySQL 4.0 ?

- MySQL 4.1 ?
- MySQL 5.0 ?
- MaxDB ?

6 An Overview of ext/mysqli

- ... or, why make another MySQL API for PHP?

7 The PHP 5 MySQL API

- Called ext/mysqli, with the 'i' standing for any one of: improved, interface, ingenious, incompatible or incomplete (and hopefully not for: idiotic, impaired, etc.)
- Supports all modern MySQL versions. (Older versions (< 4.1.x) do not support all features)
- Needs version 4.1.3+ of the MySQL client library.
- Written by Georg Richter.

8 Why was ext/mysqli created?

- ext/mysql was difficult to extend (due to design flaws like: optional connections and arguments, many deprecated functions, lots of nasty code to support all this)
- New features in MySQL 4.1.+ could not be easily supported in ext/mysql
- Better mapping between the ext/mysqli and the MySQL C API will make it easier to maintain this extension in the future

9 Why use ext/mysqli: Safer

- Safer connections with SSL and strong password hashing
- Safer queries with prepared statements
- No default connections or links make it harder to accidentally compromise or damage databases or the server.

10 Why use ext/mysqli: Faster

- New MySQL binary protocol is more efficient
- Prepared statements can give massive performance enhancements (1+ orders of magnitude) over large data sets
- Faster overall code

11 Why use ext/mysqli: Simpler

- OO interface is simple, concise and extensible
- Prepared statements make certain operations simpler
- No persistent connections
- Less to go wrong

12 Comparing new and old

- The procedural interfaces are very similar, with the exception of some additional functions and the lack of default links and connections.
- For the most part, we will focus on the object-oriented interface. If you don't like OO, don't worry - you can easily mix the OO interface into procedural code.
- Note that code based on the OO interface is easier to extend

13 No Default Data Sources

- Unlike the old extension, a default connection is never created or set. This prevents queries accidentally getting sent to the wrong place if the php.ini file is modified.
- Calling mysqli_query() without a valid connection to MySQL always fails, unlike mysql_query()
- Calling mysqli_query() without specifying a link also fails, unlike mysql_query()

14 Procedural vs. OO

- Connecting to a MySQL server
  - $link = mysqli_connect($h, $u, $p, $db);
  - $link = new mysqli($h, $u, $p, $db);
- Sending a query
  - $result = mysqli_query($link,'SELECT 1');
  - $result = $link->query('SELECT 1');
- Getting results
  - $row = mysqli_fetch_row($result);
  - $row = $result->fetch_row();

15 Using ext/mysqli

- More Fun.

16 Connecting to the server

- Each parameter is optional.
  - $link = new mysqli($host, $user, $password, $db, $port, $socket);

17 Don't Use Defaults!

- file::/etc/php.ini
  - mysqli.default_host = "staging"
- mysqli.default_host = "live"

-
- file::/../test.php
- $link->query("DROP DATABASE foo");
    - code to recreate db for testing suite
-
    - great way to accidentally trash the production database
    - Hopefully, we can remove this "feature" in future versions of ext/mysqli

18 Making Queries

- Just as you would expect
    - $result = $link->query('SELECT 1');
- Optional last parameter allows use of buffered or unbuffered queries
- Unbuffered queries provide more rapid access to the first elements of large data sets, but tie up the
- Buffered queries require more storage on the client side, and require all of a result to be transferred before it can be used.

19 Fetching meta-data

- Via functions, as in ext/mysql
- By accessing a property of an object (faster)
- Properties are fetched as required. Using var_dump() won't reveal them.
    - # dump all connection properties
    - foreach(array('affected_rows', 'client_info', 'client_version', 'errno', 'error', 'field_count', 'host_info', 'info', 'insert_id', 'protocol_version', 'sqlstate', 'thread_id', 'warning_count') as $p){ echo $p,': ', $link->$p, "\n"; }

20 Fetching the insert id

- $link->query('CREATE TEMPORARY TABLE foo (id int(11) NOT NULL auto_increment, bar text, PRIMARY KEY
-
- $link->query('INSERT foo (bar) VALUES (NOW());
-
- echo "Insert ID: ", $link->insert_id,
-     "\n";
-
    - Insert ID: 1

21 Prepared Statements I

- A method of running queries that provides performance and security benefits.

- Allows separation of query preparation (syntactic validation, parsing, query execution plan, ...) from query execution (modifying a table or fetching a result set)
- Works with CREATE TABLE, DELETE, DO, INSERT, REPLACE, SELECT, SET, UPDATE, and many SHOW statements

22 Prepared Statements II

- Queries are split into two parts
- ... statements with optional placeholders
  - SELECT name, count FROM birds
  - SELECT name, count FROM birds WHERE station = ?
- ... and data corresponding to the placeholders
  - 'ENSN' # Skien, Norway weather station

23 Prepare

- The statement is sent to the server
  - $query = 'SELECT title, review, year FROM movie WHERE actor LIKE ?';
  - $stmt = $link->prepare($query);
- The server syntactically validates, parses and (possibly) plans the query.
- If the query is successfully prepared, the prepared statement is saved and a statement handle is returned.

24 Bind Parameters

- Bind local variables to any placeholders
  - # bind variable to prepared statement
  - $stmt->bind_param('s', $actor);
- Parameters can be of the following types:
  - b: blob (send max_allowed_package chunks)
  - d: double/float
  - i: integer
  - s: string (includes enum, set and string representations of numbers, such as decimal)

25 Execute

- Request that the server execute the query referenced by the link, passing any bound parameters with the request.
  - $stmt->execute();

26 Bind Results

- If the query returned rows of data, bind fields in the query to local variables.

- $stmt->
  bind_result(
    $title,
    $review,
    $year
  );

27 Fetch Data

- Then fetch a row from the result set. Each field is bound into the corresponding variable from the bind_result call.
  - while($stmt->fetch()){
  -     printf("Actor: %s, Title: %s (%s)
  - Review: %0.1d/5\n",
  -         $actor, $title, $year, $review);
  - }

28 Simple Prepared SELECT

- $link = new mysqli($h, $u, $p, 'information_schema');\
-
- $query = 'SELECT TABLE_NAME FROM VIEWS';
-
- $stmt = $link->prepare($query);
  $stmt->execute();
  $stmt->bind_result($name);
-
- while($stmt->fetch()){
     echo $name, "\n";
  }

29 Simple Prepared INSERT

- $link = new mysqli($h, $u, $p, 'test');
-
  $stmt = $link->prepare('INSERT movie (actor, review, title, year) VALUES (?, ?, ?, ?)');
-

- $stmt->bind_param('sdsi', $actor, $review, $title, $year);
  $actor = 'Audrey Tautou';
  $review = 5;
  $title = 'Amelie';
  $year = 2001;
  $stmt->execute();

30 Error Handling

- Most functions return false on failure
- For more info, use properties from mysqli or mysqli_stmt objects
  - $link->error()
  - $stmt->error()
- ... or a function-based idiom, like ext/mysqli
  - mysql_connect_error()
  - mysql_error()

31 Report Functions

- Provides information to help debugging and development
- Report instances where indexes are not used
- Report errors in function calls (which usually need to be explicitly requested)

32 Basic Reporting Example

- mysqli_report(MYSQLI_REPORT_ALL);
- $link = new mysqli($h, $u, $p, 'world');
- $result = $link->query('SELECT * FROM city WHERE name LIKE "%k%" LIMIT 10');
-
```
while($row = $result->fetch_row()){
  echo join(" ", $row), "\n";
}
```
  - PHP Warning: mysqli::query(): No index used in query/prepared statement SELECT * FROM city WHERE name LIKE "%k%" LIMIT 10 in /Users/zag/Projects/Sessions/mysqluc05/prepared_2.php on line 4

33 Exceptions I

- ext/mysqli has been recently extended to throw exceptions
- This helps prevent standard ugly procedural error handling code:

- $link = new mysqli(...);
- if(FALSE === $link){ ... }
-
- $result->query(...);
- if(FALSE === $result){ ... }
-
- # etc.

34 Exceptions II

- With exceptions, you get nice clean code like:
  ```
  try {
    $my = new my_mysqli($h, $u, $p);
    $result = $my->query("SELECT NOW()");
    var_dump($result->fetch_row());
    $result->free();
    $my->close();
  } catch ( Exception $e ){
    # error handling here
  }
  ```

35 Exceptions III

- Use specific catch blocks for specific errors. A generic catch block could also be used.
  ```
  try {
    $my = new my_mysqli($h, $u, $p);
    $result = $my->query("SELECT NOW()");
    var_dump($result->fetch_row());
  } catch(ConnectException $exception) {
    echo "Connection Error\n";
    var_dump($exception->getMessage());
  } catch(QueryException $exception) {
    echo "Query Error\n";
    var_dump($exception->getMessage());
  }
  ```

36 Extending ext/mysqli

- Adding a new method.

  -
    ```
    class my_mysqli extends mysqli {
      function quick_fetch($query) {
        if(!$result = $this->query($query)){
          return FALSE;
        }
        return array_pop($this->query($query)->fetch_row());
      }
    }
    $my = new my_mysqli($, $u, $p);
    echo $link->quick_fetch('SELECT NOW()');
    ```

37 Migrating is a Piece of Cake

- The similarities of ext/mysql and ext/mysqli make migration simple
- The major choices are choosing whether or not to use OO and prepared statements

38 Migrating is a Tough Cookie

- Don’t trust new code for a production setting
- The old MySQL extension has been in production use for years.
- ext/mysqli hasn’t. There may be bugs or subtle change in behavior

39 Migration: Duplicate Environment

- Duplicate all or part of your application environment (or create your desired app. environment)
- Replicate data from your current MySQL install to a newer version of MySQL
- Use rsync to sync file data
- Write simple scripts to automate all the process - you will likely need several tries to get it right and doing it all by hand gets boring

40 Migration: Live Data

- Ensure that your duplicate environment can’t trash data on shared servers
- Crank up the error reporting, logs, etc
- Use socat or ipfilters to split traffic between your real environment and your test environment
- Fix what you forgot to do
- Try again

41 Migration: Followup

- Compare the state of the MySQL databases at the end of a test run

- Use mysqldump to dump data in a format that can easily be diffed
- Comparing log files
- Run test suites
- etc.

42 Coffee Break?

43 A Quick Trip Through MySQL Feature Land

44 UNIREG

- Ancient History

45 MySQL 3.x

- Rest In Peace.

46 MySQL 4.0.x

- Very Stable.
- Mostly Harmless.
- General Availability.

47 MySQL 4.1.x

- General Availability.

48 MySQL 4.1 Major Features

- Error and Warnings Reporting System, Improved Client/Server Protocol, Improved I18L, Integrated Help, Stored Procedures, Subqueries

49 Errors and Warnings

- Better reporting for warnings and errors
- Use SHOW WARNINGS/ERRORS to view warning and error messages
- Each query resets the warning/error message cache

50 Showing warnings and errors

- # display last 10 errors from prior query
- SHOW ERRORS LIMIT 10;
-
- # display the total number of errors
- SHOW COUNT(*) ERRORS;
-
- # fetch the total number of warnings

- SELECT @@warning_count;
-
- # fetch max. # of error messages that will be stored for a single query
- SELECT @@max_error_count;

51 Sample warning display

- DROP TABLE IF EXISTS no_such_table;
- SHOW WARNINGS\G
-

-
  ```
   Level: Note
    Code: 1051
  Message: Unknown table 'no_such_table'
  ```

52 Improved Client/Server Protocol

- Supports prepared statements
- Allows blob/clob data to be sent in chunks to server without storing requiring client-side storage
- Lower overhead - transmits data in its natural representation
- Optional inline zlib compression
- Optional SSL connections

53 Improved I18L

- Much better support for character sets and collations
- Can mix character sets, etc. inside of any data context in the server, from databases to tables to queries.
- Supported in InnoDB, MEMORY and MyISAM storage engines
- Includes UNICODE support

54 Collations

- Rules for sorting character sets
- One character set can have many collations. e.g. latin1 has latin1_bin, latin1_german1_ci, latin1_german2_ci, etc.
- A string has zero or one default collations.
- Collations can only be used for the corresponding character set
  - # using a collation with ORDER BY
  - SELECT * FROM names ORDER BY name COLLATE latin1_bin;

55 A Binary Collation (ASCII)

- ... WHERE 'A' < 'B'

    - Comparison returns true, as the encoding of ‘A’ (65) is less than the encoding of ‘B' (66)
  - ... WHERE 'A' = 'a'
    - Comparison returns false, as the encoding of ‘A’ (65) is different than the encoding of ‘a' (97)

56 A Non-Binary Collation

- Non-binary collations use transformative rules to alter the comparison
  - “ü” == “ue”
  - “A” == “a”
  - “A” == “eh” // latin1_canadian ;)

57 Examining a String

-
  ```
  SET @str =
    CONVERT(_latin1'Foo!' USING utf8);
  ```
-

-
  ```
  SELECT CHARSET(@str),
    CHAR_LENGTH(@chr_str),
    BIT_LENGTH(@chr_str),
    COLLATION(@chr_str)\G
  ```
  -
  - Results

-
  ```
     CHARSET(@str): utf8
  CHAR_LENGTH(@str): 3
   BIT_LENGTH(@str): 24
    COLLATION(@str): utf8_general_ci
  ```

58 Examining a Table

- `SHOW CREATE TABLE mysql.user\G`
-

-
  ```
  CREATE TABLE user (
    Host char(60) collate utf8_bin NOT NULL default '',
    User char(16) collate utf8_bin NOT NULL default '',
    Password char(41) collate utf8_bin NOT NULL default '',
    Select_priv enum('N','Y') character set utf8 NOT NULL default 'N',
    ...
  ) ... DEFAULT CHARSET=utf8 COLLATE=utf8_bin ...
  ```

59 Charset/Collation Info

- Use SHOW CHARACTER SET to show the available character sets on a MySQL server
- Use SHOW COLLATION to show the available collations on a MySQL server
- Note that the collation names generally end in suffixes that indidicate if they are case-sensitive (_cs), case-insensitive (_ci) or binary (_bin) collations

60 Integrated Help

- Provides simple help on MySQL features and functions via queries.
- Help data is stored in the mysql.help_% tables on the MySQL server.
- Generated from the included manual using the fill_help_tables script
- Very handy if dealing with an unfamiliar feature or version of MySQL

61 Using Integrated Help

- HELP CONTENTS
- HELP SELECT
-
    - Use SQL wildcards
- HELP EL_
- HELP DATA MAN%

62 Subqueries

- Allow a query within another query to be treated as a table, list or scalar value
- More powerful and easier to use than joins
- Can be of correlated (where a table referenced in a subquery also appears in the outer query) or uncorrelated forms (where this is not the case or is forbidden (as in derived tables))

63 Simple Subquery

- # MEMORY tables/total # of tables
-
- SELECT (COUNT(*) FROM TABLES WHERE ENGINE = 'MEMORY'), (SELECT COUNT(*) FROM TABLES);

64 Subquery as Scalar

- Subqueries can go most places that a scalar value can be used
- Determine how many cities, from all of the cities listed in the world database are larger than the largest city in Norway.

        - SELECT COUNT(*), (SELECT COUNT(*) FROM city) FROM city WHERE city.population >
          (SELECT MAX(population) FROM city
          WHERE countrycode = 'NOR');

65 Subqueries and Exists

- Correlated subquery with exists
    - SELECT name, code FROM country
      WHERE NOT EXISTS
      (SELECT * FROM city
       WHERE countrycode = country.code);

66 MySQL 5.x

- Still a beta release.
-
- Don’t use it in production without a lot of testing.

67 MySQL 5.0 Major Features

- Information Schema
- Stored Procedures
- Triggers
- Views

68 Information Schema

- A consistent, query-based method for retrieving meta-data about the server
- Accessing meta-data becomes just another query, allowing much easier programmatic access of the meta-data.
- Provides access to meta-data on tables, columns, stored procedures, views, etc.

69 Stored Procedures

- A collection of SQL statements stored on the server and callable by name
- Greater independence from the client application
- Better network performance vs. more server load
- More secure – keeps operations on data on the server
- Not yet stable – still limited

70 Stored Procedure Example

-
  ```
  CREATE PROCEDURE withdraw(p_amt DECIMAL(6,2), p_tellerid INT, p_custid INT)
  MODIFIES SQL DATA
  BEGIN ATOMIC
    UPDATE customers
      SET balance=balance - p_amt;
    UPDATE tellers
          SET cashonhand=cashonhand - p_amt
          WHERE tellerid = p_tellerid;
    INSERT INTO transactions
      VALUES (p_custid, p_tellerid, p_amt);
  END
  ```

71 Triggers

- A chunk of SQL run when a data modification query is executed on a given table.
- Can be set to run before or after DELETE, INSERT and UPDATE queries.
- Created with syntax:
  - CREATE TRIGGER name BEFORE QUERY_TYPE ON table FOR EACH ROW statement(s);
- Trigger support is still rudimentary.

72 Simple Sample Triggers

- These just echo out a snippet of text on DELETE or INSERT.
  - CREATE TABLE test (i int NOT NULL, PRIMARY KEY (i));
  - CREATE TRIGGER show_insert BEFORE INSERT ON test FOR EACH ROW SELECT CONCAT('inserted ', NEW.i);
  - CREATE TRIGGER show_delete BEFORE DELETE ON test FOR EACH ROW SELECT CONCAT('deleted ', NEW.i);

73 Sample Trigger

- Keep track of the number of updates to a column
  - CREATE TRIGGER count_changes BEFORE UPDATE ON address FOR EACH ROW SET NEW.count = IFNULL (OLD.count, 1) + 1;

74 Views

- A logical table (rather than physical) created from a query
- Can be updated (but be careful)
- Has its own permissions
- Relies on the underlying table indexes for efficiency
- Managed much like a normal table: CREATE VIEW, SHOW VIEW, ALTER VIEW, DROP VIEW

75 Creating and Using a View

- CREATE VIEW scandinavia AS SELECT id, name, population, district, countrycode FROM city WHERE countrycode in ('DNK', 'NOR', 'SWE');
-
- SELECT name FROM scandinavia ORDER BY population DESC LIMIT 4;

```
+-----------------------+
| name                  |
+-----------------------+
| Stockholm             |
| Oslo                  |
| København             |
| Gothenburg [Göteborg] |
+-----------------------+
```

76 Creating a View of a View

- CREATE VIEW norway AS SELECT id, name, population, district FROM scandinavia WHERE countrycode = 'NOR';
-
- SELECT name, district FROM norway;

```
+-----------+--------------+
| name      | district     |
+-----------+--------------+
| Oslo      | Oslo         |
| Bergen    | Hordaland    |
| Trondheim | Sør-Trøndelag |
| Stavanger | Rogaland     |
| Bærum     | Akershus     |
+-----------+--------------+
```

77 Inserting Into a View

- Works much like expected
  - INSERT norway (name, population, district) VALUES ('Skien', 50507, 'Telemark');
- Watch our for missing defaults!

- SELECT Name, CountryCode as Country, Population as ‘Pop.’, District FROM city WHERE Name = 'Skien';

```
+--------+---------+----------+-------+
| Name  | Country | District | Pop.  |
+--------+---------+----------+-------+
| Skien |         | Telemark | 50507 |
+--------+---------+----------+-------+
```

78 Creating Alternate Views of Data

- CREATE VIEW privs AS SELECT host, user,

```
 (if(Select_priv = 'Y', 1 << 0, 0) |
  if(Insert_priv = 'Y', 1 << 1, 0) |
  if(Update_priv = 'Y', 1 << 2, 0) |
  if(Delete_priv = 'Y', 1 << 3, 0) |
  if(Create_priv = 'Y', 1 << 4, 0) |
  if(Drop_priv = 'Y', 1 << 5, 0) |
  if(Reload_priv = 'Y', 1 << 6, 0) |
  if(Shutdown_priv = 'Y', 1 << 7, 0) |
  ...
  if(Show_view_priv = 'Y', 1 << 22, 0))
```

- AS privmap FROM mysql.user;

79 Using the Alternate View

- mysql> SELECT * FROM privs;

```
+-------------+------+---------+
| host        | user | privmap |
+-------------+------+---------+
| localhost   | root | 8388607 |
| towel.local | root | 8388607 |
| towel.local |      | 0       |
| localhost   |      | 0       |
+-------------+------+---------+
4 rows in set (0.00 sec)
```

80 Questions?