NSWI096 – Internet

12th December 2011

MySQL and PHP

Martin Svoboda

Department of Software Engineering
Faculty of Mathematics and Physics
Charles University in Prague

Outline

- MySQL
  - Relational model
  - Data definition
  - Data manipulation
- PHP and MySQL
  - Function reference

Introduction

- Relational databases
  - SQL = Structured Query Language
    - Data definition language
    - Data manipulation language
  - Implementations
    - Oracle, MS SQL Server, PostgreSQL, …
- MySQL
  - Widely used for web applications
  - http://dev.mysql.com/doc/

Introduction

- Table
  - Set of columns
    - Names and types
  - Set of integrity constraints
    - Column and table constraints
- Database
  - Set of tables
- Data × schema

column1	column2	…

Data Definition

- Data definition queries
  - **CREATE** …
    - DATABASE
    - TABLE
    - VIEW
    - INDEX
  - **ALTER** …
  - **RENAME** …
  - **DROP** …

Create Table

- Create table statement
  - **CREATE TABLE** [IF NOT EXISTS] table_name (

    create_definition, …

)
  - Allowed definitions
    - Columns with column integrity constraints
    - Table integrity constraints
    - Table indices

Create Table

- Create table statement
  - Column definition with constraints
    - column_name data_type
      [NOT NULL | NULL]
      [DEFAULT default_value]
      [AUTO_INCREMENT]
      [UNIQUE KEY | PRIMARY KEY]
      [REFERENCES referenced_table (column)]
  - Column integrity constraints are always defined over the particular column

Create Table

- Create table statement
  - Table constraints
    - [**CONSTRAINT** [constraint_name]] …

      PRIMARY KEY (col1, …)

      UNIQUE KEY (col1, …)

      FOREIGN KEY (col1, …) REFERENCES rtable (rcol1, …)
  - Table indices
    - **INDEX** [index_name] (col1, …)

Integrity Constraints

- Default value
- Not NULL
  - Empty values (NULL) are not allowed
- Primary key
  - Values cannot be empty and must be unique
- Unique key
  - All values other than NULL must be unique

Integrity Constraints

- Foreign key
  - Referencing table may only contain values that are present in the referenced table
  - Example
    - CREATE TABLE student (

      id INT PRIMARY KEY, …

)
    - CREATE TABLE email (

      student INT REFERENCES student (id), …

)

Data Types

- Numeric types
  - Exact numeric types
    - INT(m) or INTEGER(m)
      - *m* is the maximal display width in a number of digits
    - TINYINT, SMALLINT, MEDIUMINT, INT, BIGINT
      - 1, 2, 3, 4 and 8 bytes
    - BOOLEAN
      - true, false
    - DECIMAL(m, d)
      - *m* is the total number of digits
      - *d* is the number of digits after the decimal point

Data Types

- Numeric types
  - Approximate numeric types
    - FLOAT
    - DOUBLE
  - Modifiers
    - UNSIGNED – disallows negative values
    - ZEROFILL – left-padded with 0 up to the given length

Data Types

- Date and time types
  - Basic types
    - DATE
      - 'YYYY-MM-DD'
    - DATETIME
      - 'YYYY-MM-DD HH:MM:SS'
    - TIME
      - 'HH:MM:SS'
  - Note
    - Literals can be defined using ordinary strings

Data Types

- String types
  - Basic string types
    - VARCHAR(p) – variable-length string
      - *p* is the maximal allowed number of characters
    - CHAR(p) – fixed-length string
      - Strings are automatically right-padded with spaces
    - TEXT
  - String literals
    - 'value' or "value"
    - Escape sequences using backslash \

Data Types

- String types
  - Special string types
    - ENUM('value1', …) – one of the allowed string values
      - Internally implemented using integers
    - SET ('value1', …) – set of the allowed string values
      - Only 64 values are supported in the set definition

Create Table

- Sample statement
  - CREATE TABLE student (

```
id INT AUTO_INCREMENT PRIMARY KEY,
name VARCHAR(100) NOT NULL,
birthday DATE NOT NULL,
type ENUM('M.Sc.', 'Ph.D.') NOT NULL,
active BOOLEAN NOT NULL, -- studying = true
year INT(1) NOT NULL DEFAULT '1'
```

)

Data Manipulation

- Data manipulation queries
  - **INSERT** …
  - **UPDATE** …
  - **DELETE** …
  - **SELECT** …

Insert Queries

- Data insertion
  - **INSERT** INTO table_name [(col1, …)]
    VALUES (val1, …)
  - INSERT INTO … ON DUPLICATE KEY UPDATE …
    - When an insertion would violate primary or unique key constraints, an update of the original row is performed
  - Sample query
    - INSERT INTO student VALUES
      (DEFAULT, 'Jan Novak', '2001-01-01', 'Ph.D.', true, 2)

Update Queries

- Data updates
  - **UPDATE** table_name SET col1 = val1, …

    [WHERE *search_condition*]

    [ORDER BY *order_definition*] [LIMIT row_count]
  - New values
    - *expression* | DEFAULT
    - Assignments are evaluated from the left to the right
      - UPDATE table SET col1 = col1 + 1, col2 = col1;
  - Sample query
    - UPDATE student SET year = year + 1 WHERE active

Delete Queries

- Data deletion
  - **DELETE** FROM table_name

    [WHERE *search_condition*]

    [ORDER BY *order_definition*] [LIMIT row_count]
  - Sample query
    - DELETE FROM student WHERE id = 123 LIMIT 1

Select Queries

- Query components
  - **SELECT** – columns to be selected
  - **FROM** – tables with queried data
  - **WHERE** – filtering condition on rows
  - **GROUP BY** – columns used for aggregation
  - **HAVING** – filtering condition on aggregated rows
  - **ORDER BY** – columns used for ordering
  - **LIMIT** – limiting number of results

Select Queries

- Data querying
  - **SELECT** [ALL | DISTINCT] *select_expressions*
    [FROM *table_references*]
    [WHERE *search_condition*]
    [GROUP BY {col1 | expr1}, …]
    [HAVING *search_condition*]
    [ORDER BY {col1 | expr1} [ASC | DESC], …]
    [LIMIT row_count [OFFSET row_offset]]

Select Queries

- Sample query
  - Selects all active doctoral students

```
SELECT id, name
FROM student
WHERE (active) AND (type = 'Ph.D.')
ORDER BY name ASC
```

Search Conditions

- Simple conditions
  - expression [NOT] **BETWEEN** lower AND upper
    - Tests whether a given value is within allowed bounds
  - expr1 ◯ expr2
    - Expression comparison using operators:
      - < <= = <> >= >
  - expr1 ◯ [ALL | ANY] (*nested_query*)
    - All comparisons / at least one comparison must be satisfied in order the entire condition to be satisfied

Search Conditions

- Simple conditions
  - expression **IS** [NOT] **NULL**
    - Tests whether an expression is equal to NULL
      - expression = NULL is always evaluated to false (unknown)!
  - expression [NOT] **LIKE** pattern
    - Tests whether a given string corresponds to a pattern
      - % matches arbitrary number of characters
      - _ matches exactly one character
  - **EXISTS** (*nested_query*)
    - Tests whether the nested query has at least one row

Search Conditions

- Complex conditions
  - Derived using logical operators
    - expr1 **AND** expr2 – conjunction
    - expr1 **OR** expr2 – disjunction
    - expr1 **XOR** expr2 – exclusive or
    - **NOT** expression – negation
  - Logic with three values
    - true, false and unknown

Aggregated Queries

- Sample query
  - Counts all active students in each year

```
SELECT year, COUNT(*) AS count
FROM student
WHERE (active)
GROUP BY year
ORDER BY year DESC
```

Select Queries

- Sample query
  - Student counts…

id	name	*	year
1	Peter	…	1
2	Alice	…	4
3	John	…	3
4	Andrew	…	1
5	Jane	…	4
6	Anna	…	1

id	name	*	year
1	Peter	…	1
4	Andrew	…	1
6	Anna	…	1
2	Alice	…	4
5	Jane	…	4
3	John	…	3

year	count
4	2
3	1
1	3

Aggregated Queries

- Basic functions
  - **COUNT** – number of values
  - **SUM** – sum of values
  - **AVG** – average value
  - **MIN** – minimal value
  - **MAX** – maximal value
- DISTINCT alternatives
  - Duplicated values are ignored

Aggregate Functions

- Notes
  - NULL values
    - Empty values are usually ignored
    - However, COUNT(*) ignores them not
  - Empty arguments
    - SUM, AVG, MIN and MAX functions return NULL
    - COUNT function returns 0

Table Joining

- Join types
  - table1 **JOIN** table2
    - Cross join corresponds to the Cartesian product of all rows from the left table and all rows from the right one

T1.A	T1.*
1	…
2	…
3	…

T2.A	T2.*
1	…
4	…

T1.A	T1.*	T2.A	T2.*
1	…	1	…
1	…	4	…
2	…	1	…
2	…	4	…
3	…	1	…
3	…	4	…

Table Joining

- Join types
  - table1 NATURAL **JOIN** table2
    - Natural join combines only those pairs of rows from the left table and the right table that have equal values in all shared columns (columns with the same names)

T1.A	T1.*
1	…
2	…
3	…

T2.A	T2.*
1	…
4	…

T1.A = T2.A	T1.*	T2.*
1	…	…

Table Joining

- Join types
  - table1 **JOIN** table2 ON *join_condition*
    - Theta join combines only those pairs of rows that satisfy the specified joining condition

T1.A	T1.*
1	…
2	…
3	…

T2.A	T2.*
1	…
4	…

T1.A	T1.*	T2.A	T2.*
1	…	1	…
1	…	4	…
2	…	4	…
3	…	4	…

    - Sample join condition
      - T1.A <= T2.A

Table Joining

- Join types
  - table1 [LEFT | RIGHT] OUTER **JOIN** table2 ON …
    - Left/right outer join attempts to combine pairs of rows with respect to the given condition, but includes even rows from the left/right table that cannot be joined

T1.A	T1.*
1	…
2	…
3	…

T2.A	T2.*
1	…
4	…

T1.A	T1.*	T2.A	T2.*
1	…	1	…
2	…	NULL	NULL
3	…	NULL	NULL

    - Sample left outer join
      - T1.A = T2.A

Functions and operators

- Arithmetic operators
  - + - * / DIV MOD or %
- Comparison operators
  - < <= = <> >= >
- Logical operators
  - and or && or or || xor not or !

Functions and operators

- String functions
  - **CONCAT**(string1, …)
    - Returns the concatenated string
  - **LENGTH**(string)
    - Returns the length of a string in bytes
  - **SUBSTRING**(string, position, length)
    - Returns the substring of a given string

Functions and operators

- Date and time functions
  - **NOW**()
    - Returns the current date and time
  - **DATE**(datetime) and **TIME**(datetime)
    - Extracts the date/time part
- Miscellaneous functions
  - **MD5**(string) and **SHA1**(string)
    - Calculates MD5/SHA1 hash codes

PHP and MySQL

- **MySQL** extension
  - Older PHP and MySQL versions
  - Procedural style
- **MySQLi** extension
  - Works with MySQL server 4.1 and newer
  - Requires PHP 5
  - Procedural and object oriented styles

MySQLi Extension

- Procedural interface
  - Functions
- Object oriented interface
  - Class **mysqli**
    - Database connection
  - Class **mysqli_result**
    - Query results
  - Class **mysqli_stmt**
    - Prepared statements

Connection Functions

- **mysqli::__construct**(string $host, string $username, string $password, string $dbname, int $port)
  - mysqli mysqli_connect(…)
  - Opens a new connection to the MySQL server
  - Example
    - $mysqli = new mysqli('localhost', 'root', '…', 'sis');
    - $mysqli = mysqli_connect('localhost', 'root', '…', 'sis');

Connection Functions

- int **mysqli::connect_errno**
  - int mysqli_connect_errno()
  - Returns an error code from the last connect call
- string **mysqli::connect_error**
  - string mysqli_connect_error()
  - Returns a description of the last connect error
- bool **mysqli::close**()
  - bool mysqli_close(mysqli $link)
  - Closes a previously opened database connection

Connection Functions

- int **mysqli::errno**
  - int mysqli_errno(mysqli $link)
  - Returns a code of the most recent error
- string **mysqli::error**
  - string mysqli_error(mysqli $link)
  - Returns a string description of the last error

Querying Functions

- mixed **mysqli::query**(string $query)
  - mixed mysqli_query(mysqli $link, …)
  - Performs a query on the database
  - Return values
    - false on failure
    - mysqli_result for SELECT and some other queries
    - true for remaining types of queries
  - Example
    - $result = $mysqli->query("SELECT * FROM students");

Querying Functions

- int **mysqli::affected_rows**
  - int mysqli_affected_rows(mysqli $link)
  - Counts all rows affected by the last query
- int **mysqli_result::num_rows**
  - int mysqli_num_rows(mysqli_result $result)
  - Returns the number of rows in the result set

Querying Functions

- mixed **mysqli_result::fetch_row**()
  - mixed mysqli_fetch_row(mysqli_result $result)
  - Fetches one row of data from the query result
  - Return values
    - Array of values with integer keys
    - null when there are no more rows
  - Example
    - $result = $mysqli->query("SELECT * FROM students");

      $row = $result->fetch_row();

      // $row[0], $row[1], …

Querying Functions

- object **mysqli_result::fetch_object**()
  - object mysqli_fetch_object(mysqli_result $result)
  - Fetches one row of data from the query result
  - Return values
    - Object with properties named using column names
    - null when there are no more rows
  - Example
    - $result = $mysqli->query("SELECT * FROM students");

      $row = $result->fetch_object();

      // $row->id, $row->name, $row->city, …

Prepared Statements

- mysqli_stmt **mysqli::prepare**(string $query)
  - mysqli_stmt mysqli_prepare(mysqli $link, …)
  - Prepares a statement for execution
    - We can use ? as variable query parameters
  - Example
    - $query = "SELECT * FROM students WHERE id=?"

      $stmt = $mysqli->prepare($query);

Prepared Statements

- bool **mysqli_stmt::bind_param**(string $types, mixed &$var1, …)
  - bool mysqli_stmt_bind_param(mysqli_stmt $stmt, …)
  - Binds parameters to a prepared statement
  - Types
    - i integer d double s string b blob
  - Example
    - $id = 123456; $stmt->bind_param('i', $id);
    - … $other_stmt->bind_param('iss', $id, $name, $city);

Prepared Statements

- bool **mysqli_stmt::bind_result**(mixed &$var1, …)
  - bool mysqli_stmt_bind_result(mysqli_stmt $stmt, …)
  - Binds columns in the result set to variables
  - Example
    - $query = "SELECT name, city FROM students"

      …

      $stmt->bind_result($name, $city);

Prepared Statements

- bool **mysqli_stmt::execute**()
  - bool mysqli_stmt_execute(mysqli_stmt $stmt)
  - Executes a prepared query statement
- bool **mysqli_stmt::fetch**()
  - bool mysqli_stmt_fetch(mysqli_stmt $stmt)
  - Fetches results into the bound result variables
  - Return values
    - true on success
    - false on error
    - null when there are no more rows in the result

Prepared Statements

- Usage pattern
  - `$stmt = $mysqli->prepare($query);`
  - `// Definition of $param* variables`
  - `$stmt->bind_param($types, $param1, …);`
  - `$stmt->execute();`
  - `$stmt->bind_result($column1, …);`
  - `$stmt->fetch();`
  - `// Usage of $column* in a loop`

Prepared Statements

- int **mysqli_stmt::affected_rows**
  - int mysqli_stmt_affected_rows(mysqli_stmt $stmt)
- int **mysqli_stmt::num_rows**
  - int mysqli_stmt_num_rows(mysqli_stmt $stmt)
- int **mysqli_stmt::errno**
  - int mysqli_stmt_errno(mysqli_stmt $stmt)
- string **mysqli_stmt::error**
  - string mysqli_stmt_error(mysqli_stmt $stmt)

Conclusion

- MySQL statements
  - **CREATE** TABLE (…)
  - **INSERT** INTO … VALUES (…)
  - **UPDATE** … SET … WHERE …
  - **DELETE** FROM … WHERE …
  - **SELECT** … FROM … WHERE … GROUP BY … HAVING … ORDER BY … LIMIT …
- MySQLi extension
  - **mysqli**, **mysqli_result**, **mysqli_stmt**