

Database Access with PHP

Internet Applications, ID1354

Contents

- Relational Databases and SQL
- Database Access With PHP

Section

Relational
Databases and SQL

Database Access
With PHP

- Relational Databases and SQL
- Database Access With PHP

Database

- ▶ A **database** is a collection of data, organized in tables.

Relational
Databases and SQL

Database Access
With PHP

Database

- ▶ A **database** is a collection of data, organized in tables.
- ▶ A **table** is a named collection of rows.
 - ▶ One table represents one abstraction, corresponds to **class** in object oriented programming.

Database

- ▶ A **database** is a collection of data, organized in tables.
- ▶ A **table** is a named collection of rows.
 - ▶ One table represents one abstraction, corresponds to **class** in object oriented programming.
- ▶ A **row** in a table has a number of columns.
 - ▶ Each row represents an instance of the abstraction represented by the table. Row corresponds to **object** in object oriented programming.

Database

- ▶ A **database** is a collection of data, organized in tables.
- ▶ A **table** is a named collection of rows.
 - ▶ One table represents one abstraction, corresponds to **class** in object oriented programming.
- ▶ A **row** in a table has a number of columns.
 - ▶ Each row represents an instance of the abstraction represented by the table. Row corresponds to **object** in object oriented programming.
- ▶ A **column** is a single data item having name, type, and value.
 - ▶ A column corresponds to a **field** in a class in object oriented programming. All rows in the same table has the same columns.

Structured Query Language, SQL

- ▶ SQL is an industry-standard **language for creating, updating and querying** relational databases.

Structured Query Language, SQL

- ▶ SQL is an industry-standard language for creating, updating and querying relational databases.
- ▶ Developed by IBM in the 1970s

Structured Query Language, SQL

Relational
Databases and SQL

Database Access
With PHP

- ▶ SQL is an industry-standard **language** for **creating, updating and querying** relational databases.
- ▶ Developed by IBM in the 1970s
- ▶ A single SQL statement can be very expressive and can initiate **high-level actions**, such as sorting and merging.

SQL Primer

- ▶ Create a table:

```
create table <table name> (<column name>, <data type>  
                           [,<column name>, <data type>]...)
```

SQL Primer

Relational
Databases and SQL

Database Access
With PHP

- ▶ Create a table:

```
create table <table name> (<column name>, <data type>  
    [,<column name>, <data type>]...)
```

- ▶ Example:

```
create table person (name varchar(100), age int(3),  
    phone varchar(12))
```

SQL Primer

- ▶ Create a table:

```
create table <table name> (<column name>, <data type>  
    [, <column name>, <data type>]...)
```

- ▶ Example:

```
create table person (name varchar(100), age int(3),  
    phone varchar(12))
```

- ▶ **varchar(100)** means a **string** of length 100.

SQL Primer

- ▶ Create a table:

```
create table <table name> (<column name>, <data type>  
    [, <column name>, <data type>]...)
```

- ▶ Example:

```
create table person (name varchar(100), age int(3),  
    phone varchar(12))
```

- ▶ `varchar(100)` means a **string** of length 100.
- ▶ `int(3)` means an **integer** with three digits.

SQL Primer

- ▶ Create a table:

```
create table <table name> (<column name>, <data type>  
    [, <column name>, <data type>]...)
```

- ▶ Example:

```
create table person (name varchar(100), age int(3),  
    phone varchar(12))
```

- ▶ `varchar(100)` means a **string** of length 100.
 - ▶ `int(3)` means an **integer** with three digits.
- ▶ Delete a table:

```
drop table <table name>
```

SQL Primer, Cont'd

Relational
Databases and SQL

Database Access
With PHP

- ▶ Retrieve a set of rows and columns:

```
select <column names> from <table name>  
  where <search condition>  
  [order by <column name> [asc | desc]]
```


SQL Primer, Cont'd

Relational
Databases and SQL

Database Access
With PHP

- ▶ Retrieve a set of rows and columns:

```
select <column names> from <table name>  
    where <search condition>  
    [order by <column name> [asc | desc]]
```

- ▶ Example:

```
select name, age from person  
    where name = 'nisse'
```

SQL Primer, Cont'd

Relational
Databases and SQL

Database Access
With PHP

- ▶ Insert rows:

```
insert into <table name> [( <column names> )]  
values ( <expressions> )
```

SQL Primer, Cont'd

Relational
Databases and SQL

Database Access
With PHP

- ▶ Insert rows:

```
insert into <table name> [( <column names> )]  
values (<expressions>)
```

- ▶ Example:

```
insert into person  
values ('frida', 76, '878345745')
```

SQL Primer, Cont'd

Relational
Databases and SQL

Database Access
With PHP

- ▶ Update rows:

```
update <table name>  
  set <column name = <expression>,  
      [, <column name> = <expression>] ...  
  where <search condition>
```

SQL Primer, Cont'd

Relational
Databases and SQL

Database Access
With PHP

- ▶ Update rows:

```
update <table name>  
  set <column name = <expression>,  
      [, <column name> = <expression>] ...  
  where <search condition>
```

- ▶ Example:

```
update person set age = 12  
             where name = 'nisse'
```

SQL Primer, Cont'd

Relational
Databases and SQL

Database Access
With PHP

- ▶ Delete rows:

```
delete from <table name>  
      where <search condition>
```

SQL Primer, Cont'd

Relational
Databases and SQL

Database Access
With PHP

- ▶ Delete rows:

```
delete from <table name>  
      where <search condition>
```

- ▶ Example:

```
delete from person where age = 52
```

MySQL

- ▶ A free, efficient, **widely used database system**.

MySQL

- ▶ A free, efficient, **widely used database system**.
- ▶ Available from **`http://www.mysql.org`** or as a part of a *XAMP* package.

Question 1

Section

- Relational Databases and SQL
- Database Access With PHP

MySQL APIs in PHP

- ▶ PHP offers three different APIs to connect to MySQL: `mysql`, `PDO_MySQL` and `mysqli`.

MySQL APIs in PHP

- ▶ PHP offers three different APIs to connect to MySQL: `mysql`, `PDO_MySQL` and `mysqli`.
- ▶ `mysql` should not be used, it has been deprecated as of PHP 5.5.0 and will eventually be removed.

MySQL APIs in PHP

- ▶ PHP offers three different APIs to connect to MySQL: `mysql`, `PDO_MySQL` and `mysqli`.
- ▶ `mysql` should not be used, it has been deprecated as of PHP 5.5.0 and will eventually be removed.
- ▶ The main difference between the other two is that `PDO_MySQL` is only object oriented, while `mysqli` has both an object oriented and a procedural API.

MySQL APIs in PHP

- ▶ PHP offers three different APIs to connect to MySQL: `mysql`, `PDO_MySQL` and `mysqli`.
- ▶ `mysql` should not be used, it has been deprecated as of PHP 5.5.0 and will eventually be removed.
- ▶ The main difference between the other two is that `PDO_MySQL` is only object oriented, while `mysqli` has both an object oriented and a procedural API.
- ▶ The examples on the following slides use the object oriented API of `mysqli`.

Configure Error Handling

- ▶ The following statement makes `mysqli` throw an exception of class `mysqli_sql_exception` when an error occurs.

```
mysqli_report(MYSQLI_REPORT_ERROR  
              MYSQLI_REPORT_STRICT);
```


Configure Error Handling

- ▶ The following statement makes `mysqli` throw an exception of class `mysqli_sql_exception` when an error occurs.

```
mysqli_report(MYSQLI_REPORT_ERROR  
             MYSQLI_REPORT_STRICT);
```

- ▶ Without this statement, it is necessary to check for error numbers to know if an operation succeeded.

Connect to a Database

- ▶ The following statement **connects to the database persons** on the MySQL server on **localhost**, using the username **user** and the password **pass**.

```
$personDb = new \mysqli('localhost', 'user',  
                        'pass', 'persons');
```

Connect to a Database

- ▶ The following statement **connects to the database persons** on the MySQL server on **localhost**, using the username **user** and the password **pass**.

```
$personDb = new \mysqli('localhost', 'user',  
                        'pass', 'persons');
```

- ▶ The created connection is represented by an instance of the class **mysqli**, which is stored in the variable **\$personDb**.

Execute a SQL Statement

- ▶ The **query** method in the **mysqli** instance is used to **execute a SQL statement**.

```
$personDb->query('drop table if exists person');
```

Prepared Statements

- ▶ A prepared statement execution consists of **two** stages: **prepare** and **execute**.

Prepared Statements

- ▶ A prepared statement execution consists of **two stages: prepare and execute**.
- ▶ At the **prepare stage**, a statement template is sent to the database server. The server performs a syntax check and initializes server resources for later use.

Prepared Statements

- ▶ A prepared statement execution consists of **two stages: prepare and execute**.
- ▶ At the **prepare stage**, a statement template is sent to the database server. The server performs a syntax check and initializes server resources for later use.
- ▶ During the **execute stage**, the client binds parameter values and sends them to the server. The server creates a statement from the statement template and the bound values and executes it.

Prepared Statements

- ▶ A prepared statement execution consists of **two stages: prepare and execute**.
- ▶ At the **prepare stage**, a statement template is sent to the database server. The server performs a syntax check and initializes server resources for later use.
- ▶ During the **execute stage**, the client binds parameter values and sends them to the server. The server creates a statement from the statement template and the bound values and executes it.
- ▶ Prepared statements are **more secure**, more about this on coming lectures.

Prepared Statements

- ▶ A prepared statement execution consists of **two stages: prepare and execute**.
- ▶ At the **prepare stage**, a statement template is sent to the database server. The server performs a syntax check and initializes server resources for later use.
- ▶ During the **execute stage**, the client binds parameter values and sends them to the server. The server creates a statement from the statement template and the bound values and executes it.
- ▶ Prepared statements are **more secure**, more about this on coming lectures.
- ▶ Prepared statements are **faster** than ordinary statements when executing the same statements multiple times, since they are **interpreted only once** by the database server.

Create and Execute a Prepared Statement

```
1 $updateStmt =
2     $personDb->prepare(
3         "update person set age = ?, phone = ? where name = ?"
4     );
5 $updateStmt->bind_param('iss', $age, $phone, $name);
6 $updateStmt->execute();
```

- ▶ The **prepare** method in the **mysqli** instance creates a prepared statement, lines one to four.

Create and Execute a Prepared Statement

```
1 $updateStmt =  
2     $personDb->prepare(  
3         "update person set age = ?, phone = ? where name = ?"  
4     );  
5 $updateStmt->bind_param('iss', $age, $phone, $name);  
6 $updateStmt->execute();
```

- ▶ The `prepare` method in the `mysqli` instance creates a prepared statement, lines one to four.
- ▶ The question marks in the SQL statement on line three are parameters that shall be bound to values before the statement is executed.

Create and Execute a Prepared Statement

```
1 $updateStmt =
2     $personDb->prepare(
3         "update person set age = ?, phone = ? where name = ?"
4     );
5 $updateStmt->bind_param('iss', $age, $phone, $name);
6 $updateStmt->execute();
```

- ▶ The `prepare` method in the `mysqli` instance creates a prepared statement, lines one to four.
- ▶ The question marks in the SQL statement on line three are parameters that shall be bound to values before the statement is executed.
- ▶ The `bind_param` method, line five, binds those parameters to the values of the php variables `$age`, `$phone` and `$name`, in that order.

Create and Execute a Prepared Statement

```
1 $updateStmt =
2     $personDb->prepare(
3         "update person set age = ?, phone = ? where name = ?"
4     );
5 $updateStmt->bind_param('iss', $age, $phone, $name);
6 $updateStmt->execute();
```

- ▶ The `prepare` method in the `mysqli` instance creates a prepared statement, lines one to four.
- ▶ The question marks in the SQL statement on line three are parameters that shall be bound to values before the statement is executed.
- ▶ The `bind_param` method, line five, binds those parameters to the values of the php variables `$age`, `$phone` and `$name`, in that order.
- ▶ The string `'iss'` on line five tells the types of the parameters: **integer**, **string**, **string**.

Create and Execute a Prepared Statement

```
1 $updateStmt =
2     $personDb->prepare(
3         "update person set age = ?, phone = ? where name = ?"
4     );
5 $updateStmt->bind_param('iss', $age, $phone, $name);
6 $updateStmt->execute();
```

- ▶ The **prepare** method in the **mysqli** instance creates a prepared statement, lines one to four.
- ▶ The question marks in the SQL statement on line three are parameters that shall be bound to values before the statement is executed.
- ▶ The **bind_param** method, line five, binds those parameters to the values of the php variables **\$age**, **\$phone** and **\$name**, in that order.
- ▶ The string **'iss'** on line five tells the types of the parameters: **integer**, **string**, **string**.
- ▶ The **execute** method on line six executes the prepared statement.

Read the Search Result of a Select Statement

```
1 $persons = array();
2 $selectStmt = $personDb->prepare("select * from persons");
3 $selectStmt->execute();
4 $selectStmt->bind_result($name, $age, $phone);
5 while ($this->selectStmt->fetch()) {
6     $persons[] = new Person($name, $age, $phone);
7 }
```

- ▶ A **select** statement is **created** on line two.

Read the Search Result of a Select Statement

```
1 $persons = array();
2 $selectStmt = $personDb->prepare("select * from persons");
3 $selectStmt->execute();
4 $selectStmt->bind_result($name, $age, $phone);
5 while ($this->selectStmt->fetch()) {
6     $persons[] = new Person($name, $age, $phone);
7 }
```

- ▶ A **select** statement is **created** on line two.
- ▶ The prepared statement is **executed** on line three. This **returns a result set** with all rows and columns found by the **select**.

Read the Search Result of a Select Statement

```
1 $persons = array();
2 $selectStmt = $personDb->prepare("select * from persons");
3 $selectStmt->execute();
4 $selectStmt->bind_result($name, $age, $phone);
5 while ($this->selectStmt->fetch()) {
6     $persons[] = new Person($name, $age, $phone);
7 }
```

- ▶ A **select** statement is **created** on line two.
- ▶ The prepared statement is **executed** on line three. This **returns a result set** with all rows and columns found by the **select**.
- ▶ The result of the **select** is **bound to the php variables \$name, \$age and \$phone** on line four.

Read the Search Result of a Select Statement

```
1 $persons = array();
2 $selectStmt = $personDb->prepare("select * from persons");
3 $selectStmt->execute();
4 $selectStmt->bind_result($name, $age, $phone);
5 while ($this->selectStmt->fetch()) {
6     $persons[] = new Person($name, $age, $phone);
7 }
```

- ▶ A **select** statement is **created** on line two.
- ▶ The prepared statement is **executed** on line three. This **returns a result set** with all rows and columns found by the **select**.
- ▶ The result of the **select** is **bound to the php variables** `$name`, `$age` and `$phone` on line four.
- ▶ The values for all columns on the first row in the result set is **placed in the variables** on line five. Each following call to **fetch** will load a new row.

Read the Search Result of a Select Statement

```
1 $persons = array();
2 $selectStmt = $personDb->prepare("select * from persons");
3 $selectStmt->execute();
4 $selectStmt->bind_result($name, $age, $phone);
5 while ($this->selectStmt->fetch()) {
6     $persons[] = new Person($name, $age, $phone);
7 }
```

- ▶ A **select** statement is **created** on line two.
- ▶ The prepared statement is **executed** on line three. This **returns a result set** with all rows and columns found by the **select**.
- ▶ The result of the **select** is **bound to the php variables \$name, \$age and \$phone** on line four.
- ▶ The values for all columns on the first row in the result set is **placed in the variables** on line five. Each following call to **fetch** will load a new row.
- ▶ Each turn in the loop will create a new **Person** object and store that object in an array on line six.

Inserting HTTP Parameters in a Database

- ▶ When using HTTP parameters in database calls, the characters (' " \ and **NULL**) might cause problems.

Inserting HTTP Parameters in a Database

- ▶ When using HTTP parameters in database calls, the characters (' " \ and **NULL**) might cause problems.
- ▶ To escape these characters, use the function

real_escape_string(\$str)

```
$name = "O' Hara"
```

```
$name = $personDb->real_escape_string($name);
```

Question 2