Databases

Chapter 8

Why Databases?

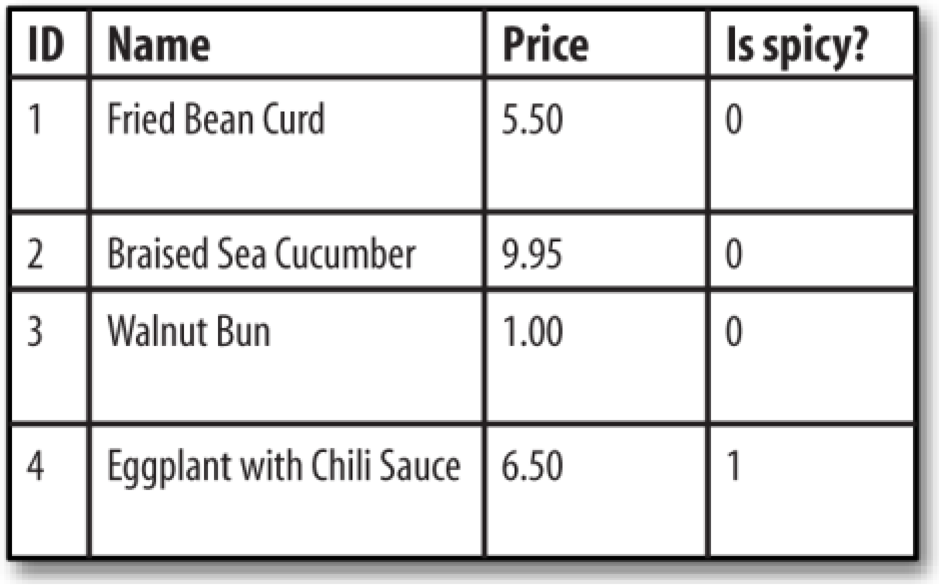
- Why not storing data in simple files? Why databases?
- There are three big reasons why using databases:
  - **Convenience**: if data is in a file, you must read, search to find data, change it and write back data!
  - **Simultaneous access:** if 2 users change same data at the same time, which change get the final effect?
  - **Security:** we cannot control access to part of a data file, but in the database we can

Organizing Data in a Database

- Data in your database is organized in tables
- Each table has rows and columns (fields)

- Structured Query Language (SQL) is a language used to ask questions of and give instructions to the database program.

- SQL is case-insensitive for its keywords
- But sensitive (by default) for the string values



ID	Name	Price	Is spicy?
1	Fried Bean Curd	5.50	0
2	Braised Sea Cucumber	9.95	0
3	Walnut Bun	1.00	0
4	Eggplant with Chili Sauce	6.50	1

Connecting to a Database Program

- To establish a connection to a database program, create a new PDO object.
- You pass the PDO constructor a string that describes the database you are connecting to
- It returns an object that you use in the rest of your program to exchange data with the database program.

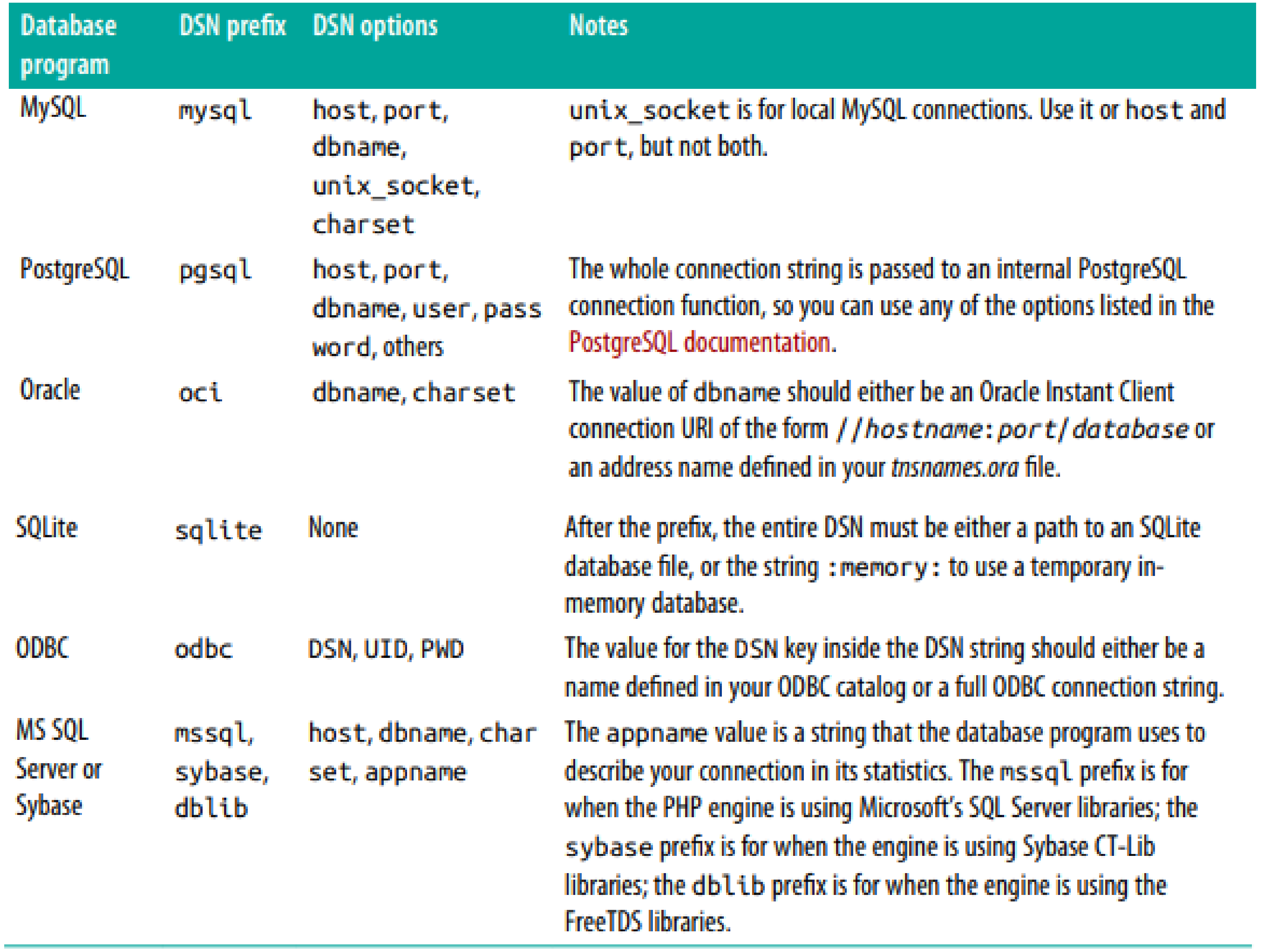
```
$db = new PDO('mysql:host=db.example.com;dbname=restaurant','penguin','top^hat');
```

- The first argument is called a data source name (DSN):
  - It begins with what kind of database program to connect to (e.g. mysql)
  - then has a colon :
  - then some key=value pairs separated by semicolon providing information about how to connect.
- If the database connection needs a username and password, these are passed as the second and third arguments

- PDO support many DBMSs but if it not, you will get could not find driver message when creating a PDO object

The charset option, available with some database programs, specifies how the database program should handle non-English characters.

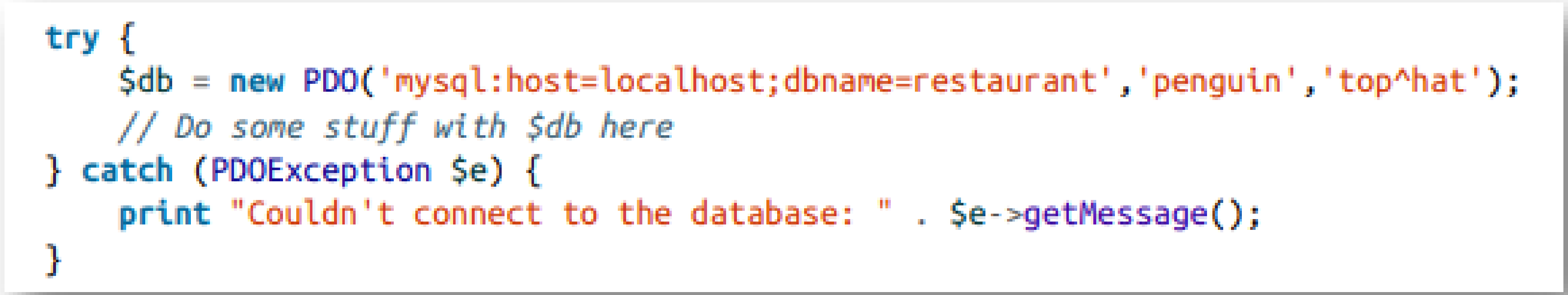
Table 8-1. PDO DSN prefixes and options



Database program	DSN prefix	DSN options	Notes
MySQL	`mysql`	`host`, `port`, `dbname`, `unix_socket`, `charset`	`unix_socket` is for local MySQL connections. Use it or `host` and `port`, but not both.
PostgreSQL	`pgsql`	`host`, `port`, `dbname`, `user`, `password`, others	The whole connection string is passed to an internal PostgreSQL connection function, so you can use any of the options listed in the PostgreSQL documentation.
Oracle	`oci`	`dbname`, `charset`	The value of `dbname` should either be an Oracle Instant Client connection URI of the form `//`*`hostname`*`:`*`port`*`/`*`database`* or an address name defined in your *tnsnames.ora* file.
SQLite	`sqlite`	None	After the prefix, the entire DSN must be either a path to an SQLite database file, or the string `:memory:` to use a temporary in-memory database.
ODBC	`odbc`	`DSN`, `UID`, `PWD`	The value for the `DSN` key inside the DSN string should either be a name defined in your ODBC catalog or a full ODBC connection string.
MS SQL Server or Sybase	`mssql`, `sybase`, `dblib`	`host`, `dbname`, `charset`, `appname`	The `appname` value is a string that the database program uses to describe your connection in its statistics. The `mssql` prefix is for when the PHP engine is using Microsoft's SQL Server libraries; the `sybase` prefix is for when the engine is using Sybase CT-Lib libraries; the `dblib` prefix is for when the engine is using the FreeTDS libraries.

PDO

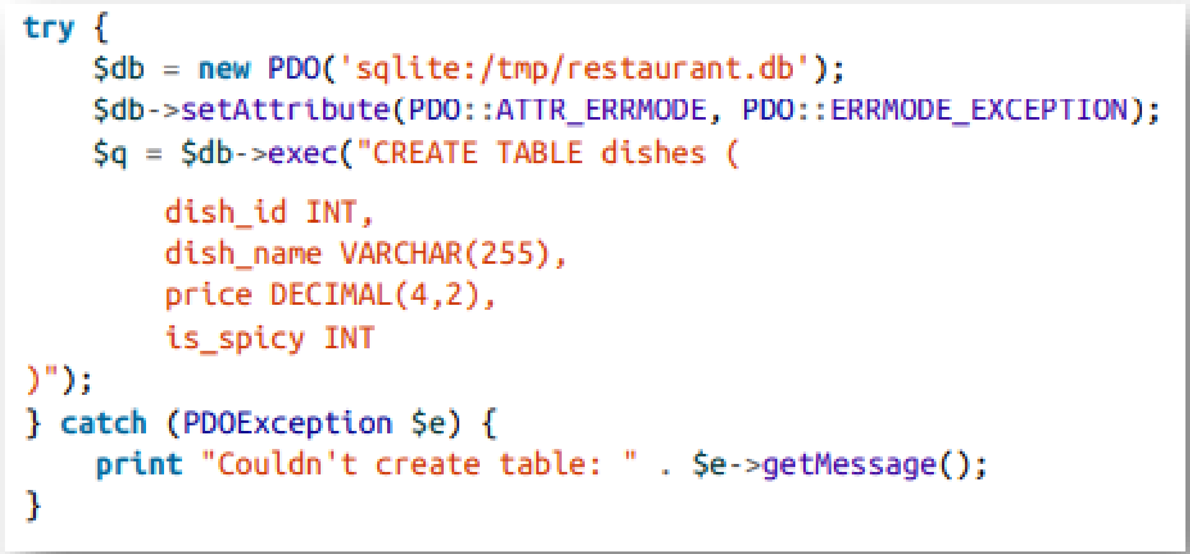
- If all goes well with new PDO(), it returns an object that you use to interact with the database.
- If there is a problem connecting, it throws a PDOException exception. Make sure to catch exceptions so you can verify that the connection succeeded before going forward in your program.



```
try {
    $db = new PDO('mysql:host=localhost;dbname=restaurant','penguin','top^hat');
    // Do some stuff with $db here
} catch (PDOException $e) {
    print "Couldn't connect to the database: " . $e->getMessage();
}
```

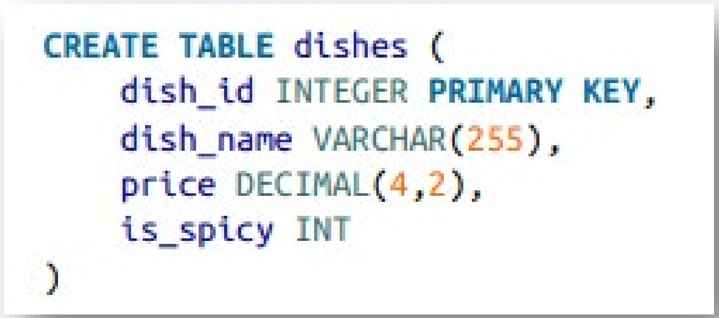
Creating a Table

- First you must create a table. This is usually a one-time operation.

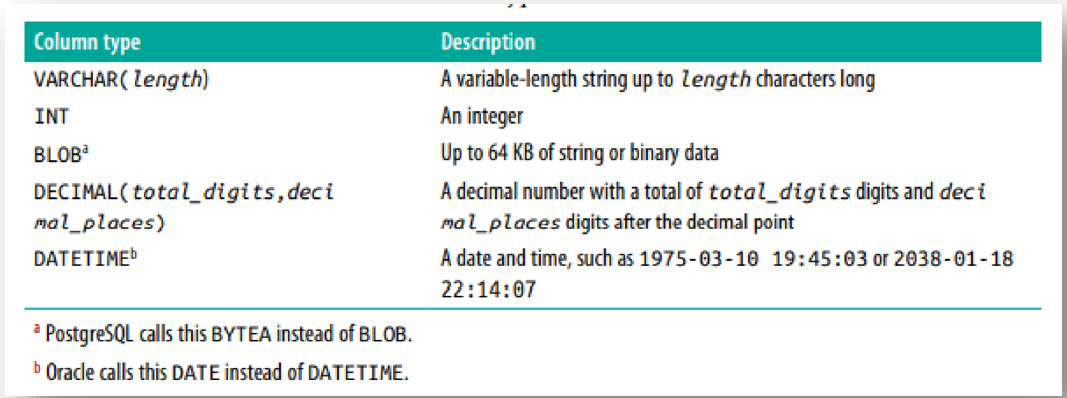


```
try {
    $db = new PDO('sqlite:/tmp/restaurant.db');
    $db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    $q = $db->exec("CREATE TABLE dishes (

        dish_id INT,
        dish_name VARCHAR(255),
        price DECIMAL(4,2),
        is_spicy INT
)");
} catch (PDOException $e) {
    print "Couldn't create table: " . $e->getMessage();
}
```



```
CREATE TABLE dishes (
    dish_id INTEGER PRIMARY KEY,
    dish_name VARCHAR(255),
    price DECIMAL(4,2),
    is_spicy INT
)
```



Column type	Description
VARCHAR(*length*)	A variable-length string up to *length* characters long
INT	An integer
BLOB[a]	Up to 64 KB of string or binary data
DECIMAL(*total_digits*,*decimal_places*)	A decimal number with a total of *total_digits* digits and *decimal_places* digits after the decimal point
DATETIME[b]	A date and time, such as 1975-03-10 19:45:03 or 2038-01-18 22:14:07

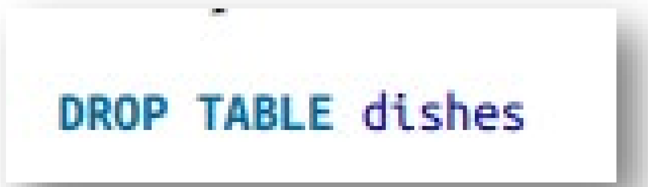
[a] PostgreSQL calls this BYTEA instead of BLOB.

[b] Oracle calls this DATE instead of DATETIME.

- setAttribute() ensures that PDO throws exceptions if there are problems with queries, not just a problem when connecting.

Dropping a table

- The opposite of CREATE TABLE is DROP TABLE.

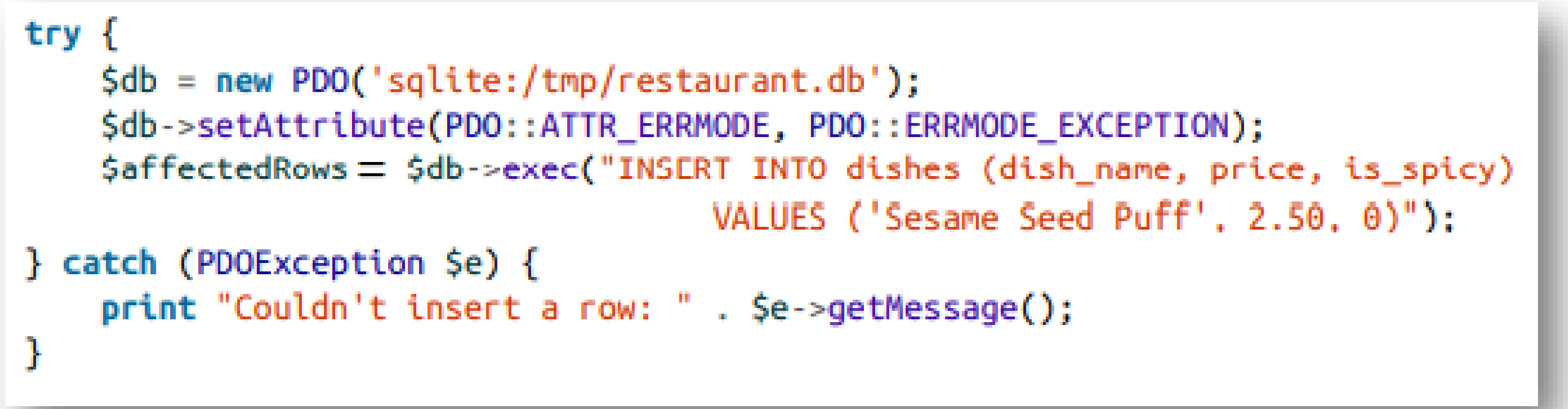


```
DROP TABLE dishes
```

- NOTE: It removes a table and the data in it from a database

Putting Data into the Database

- To put some data into the database, pass an INSERT statement to the object's exec() method



```
try {
    $db = new PDO('sqlite:/tmp/restaurant.db');
    $db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    $affectedRows = $db->exec("INSERT INTO dishes (dish_name, price, is_spicy)
                                 VALUES ('Sesame Seed Puff', 2.50, 0)");
} catch (PDOException $e) {
    print "Couldn't insert a row: " . $e->getMessage();
}
```

- The exec() method returns the number of rows affected by the SQL statement. In this case, it returns 1

- If something goes wrong with INSERT, an exception is thrown.

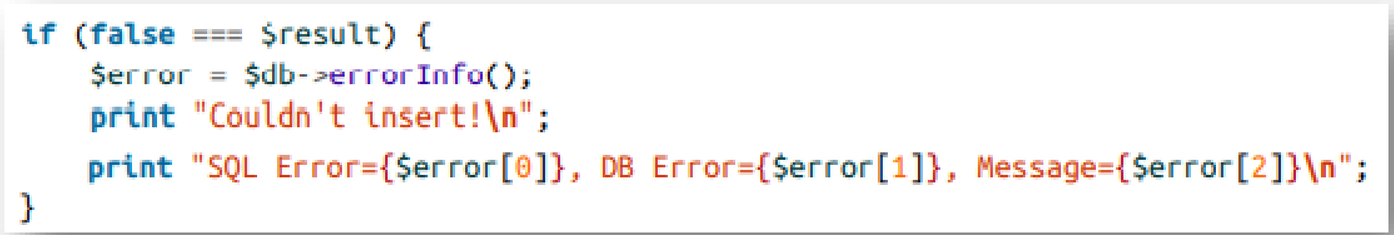
PDO error modes

- PDO has 3 error modes:
  - Silent
  - Warning
  - Exception
- The silent mode is the default.
- The exception error mode is activated by $db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION)
- The other two error modes require you to check the return values from your PDO function calls
  - If there is an error, use additional PDO methods to find information about the error.

- The warning mode is activated by setting the PDO::ATTR_ERRMODE attribute to PDO::ERRMODE_WARNING

errorinfo()

- Like other PDO methods, if exec() fails at its task, it returns false. (Question: why === instead of ==?)

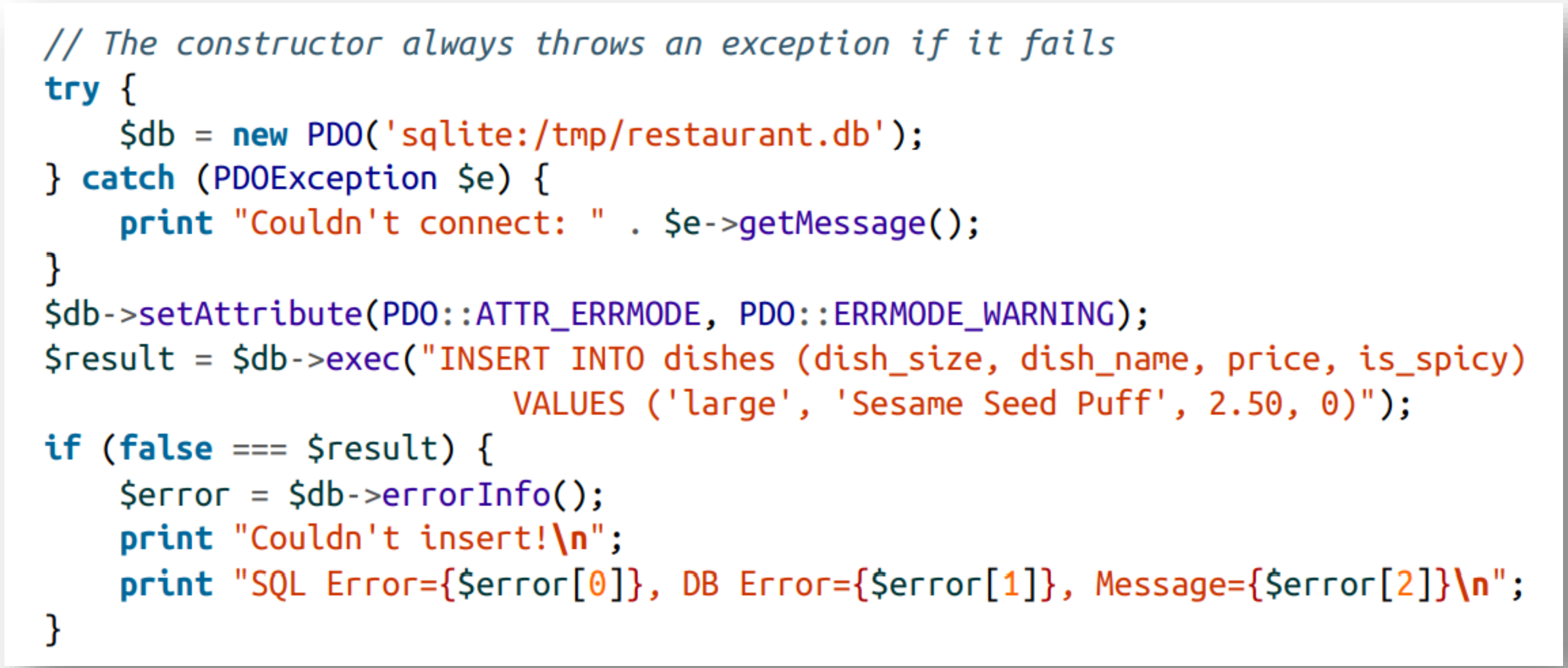


```
if (false === $result) {
    $error = $db->errorInfo();
    print "Couldn't insert!\n";
    print "SQL Error={$error[0]}, DB Error={$error[1]}, Message={$error[2]}\n";
}
```

- errorInfo() returns a three-element array with error information.
  - The **first** element is an SQLSTATE error code. These are error codes that are mostly standardized across different database programs. In this case, HY000 is a catch-all for general errors.
  - The **second** element is an error code specific to the particular database program in use.
  - The **third** element is a textual message describing the error.

Warnign mode

- In this mode, functions behave as they do in silent mode—no exceptions, returning false on error—but the PHP engine also generates a warning-level error message.
- Depending on how you’ve configured error handling, this message may get displayed on screen or in a log file.



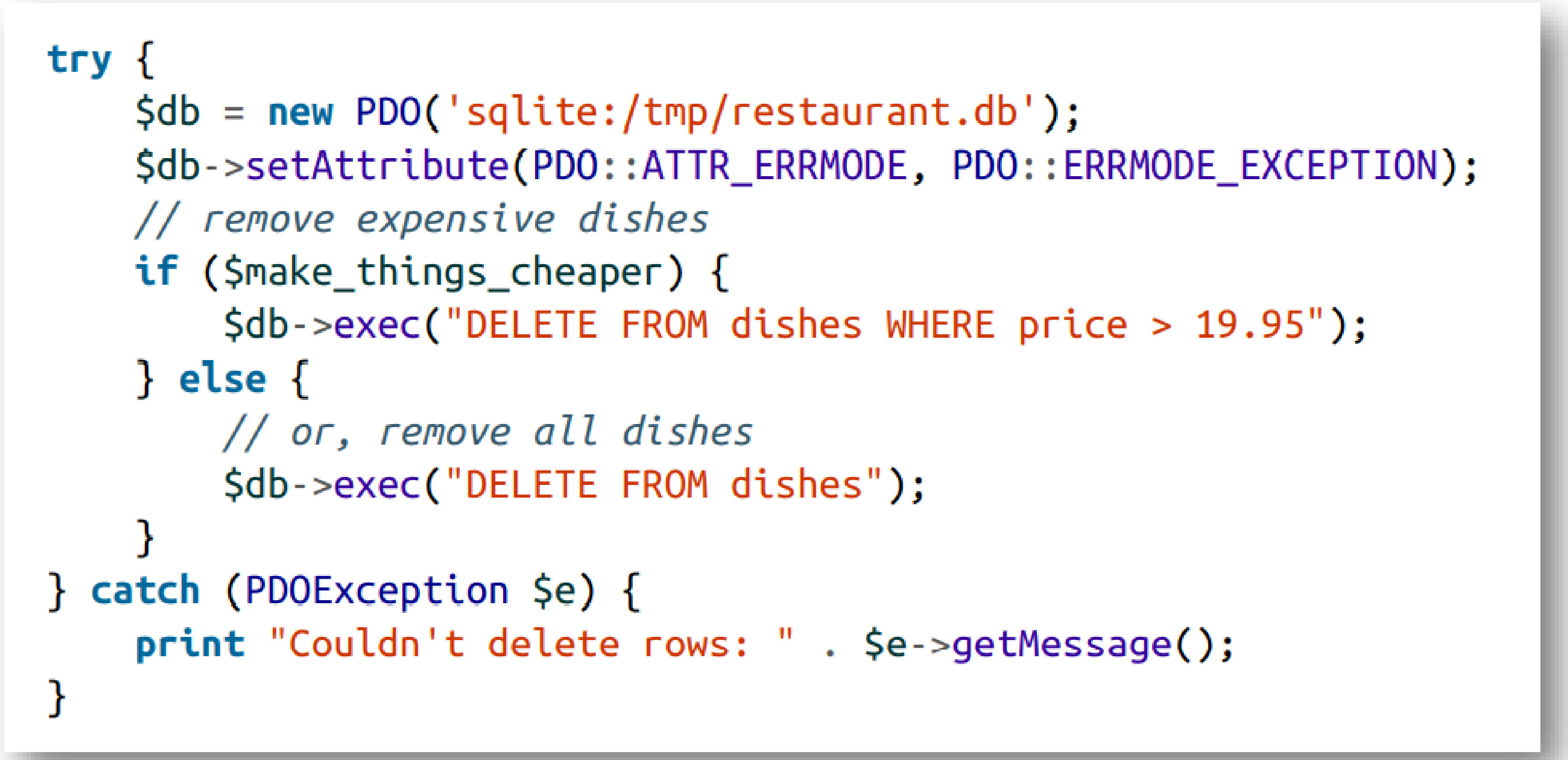
```php
// The constructor always throws an exception if it fails
try {
    $db = new PDO('sqlite:/tmp/restaurant.db');
} catch (PDOException $e) {
    print "Couldn't connect: " . $e->getMessage();
}
$db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_WARNING);
$result = $db->exec("INSERT INTO dishes (dish_size, dish_name, price, is_spicy)
                         VALUES ('large', 'Sesame Seed Puff', 2.50, 0)");
if (false === $result) {
    $error = $db->errorInfo();
    print "Couldn't insert!\n";
    print "SQL Error={$error[0]}, DB Error={$error[1]}, Message={$error[2]}\n";
}
```

Update Data in a Table

```
try {
    $db = new PDO('sqlite:/tmp/restaurant.db');
    $db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    // Eggplant with Chili Sauce is spicy
    // If we don't care how many rows are affected,
    // there's no need to keep the return value from exec()
    $db->exec("UPDATE dishes SET is_spicy = 1
                WHERE dish_name = 'Eggplant with Chili Sauce'");
    // Lobster with Chili Sauce is spicy and pricy
    $db->exec("UPDATE dishes SET is_spicy = 1, price=price * 2
               WHERE dish_name = 'Lobster with Chili Sauce'");
} catch (PDOException $e) {
    print "Couldn't insert a row: " . $e->getMessage();
}
```

Delete Data From a Table

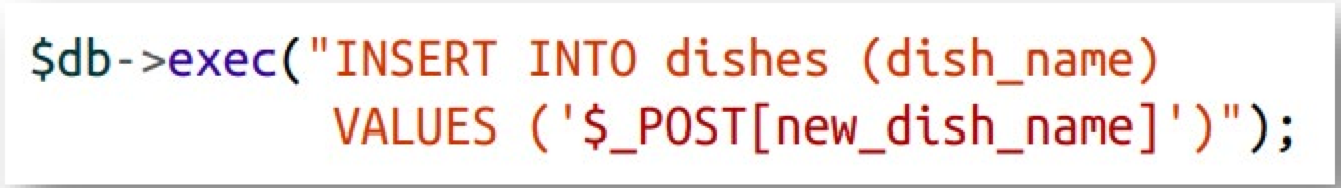
- Remember that exec() returns the number of rows changed or removed by an UPDATE or DELETE statement.
- Use the return value to find out how many rows that query affected.



```php
try {
    $db = new PDO('sqlite:/tmp/restaurant.db');
    $db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    // remove expensive dishes
    if ($make_things_cheaper) {
        $db->exec("DELETE FROM dishes WHERE price > 19.95");
    } else {
        // or, remove all dishes
        $db->exec("DELETE FROM dishes");
    }
} catch (PDOException $e) {
    print "Couldn't delete rows: " . $e->getMessage();
}
```

Inserting Form Data Safely

- Using unsanitized form data in SQL queries can cause a problem, called an “SQL injection attack.”



```
$db->exec("INSERT INTO dishes (dish_name)
           VALUES ('$_POST[new_dish_name]')");
```

- If the submitted value for new_dish_name is reasonable, such as Fried Bean Curd, then the query succeeds.
- A query with an apostrophe in it causes a problem.
  - If the submitted value for new_dish_name is *General Tso's Chicken*, DBMS thinks that the apostrophe between Tso and s ends the string, so the s Chicken' after the second single quote is an unwanted syntax error.

SQL Injection Attack

- If the PHP code is as below, then the malicious user can inject their own malicious SQL code

```
$db->exec("INSERT INTO dishes (dish_name)
           VALUES ('$_POST[new_dish_name]')");
```

- If the user enters the following input in the new_disj_name field of the from

```
x'); DELETE FROM dishes; INSERT INTO dishes (dish_name) VALUES ('y.
```

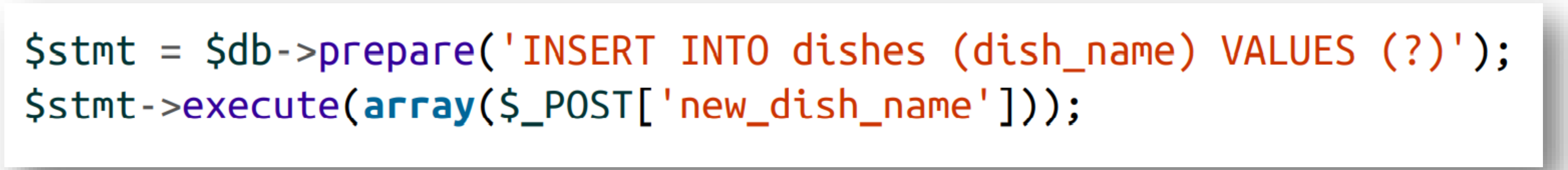
- The SQL code becomes as below, which user can delete ALL dishes and insert another value:

```
INSERT INTO DISHES (dish_name) VALUES ('x');
DELETE FROM dishes; INSERT INTO dishes (dish_name) VALUES ('y')
```

Prepared Statements

- With Prepared Statements, you separate your query execution into two steps.

1. First, you give PDO's prepare() a version of your query with a ? in the SQL in each place you want a value to go.
   - *This method returns a PDOStatement object.*
2. Then, you call execute() on your PDOStatement object, passing it an array of values to be substituted for the placeholding ? Characters



```
$stmt = $db->prepare('INSERT INTO dishes (dish_name) VALUES (?)');
$stmt->execute(array($_POST['new_dish_name']));
```

Prepared Statements

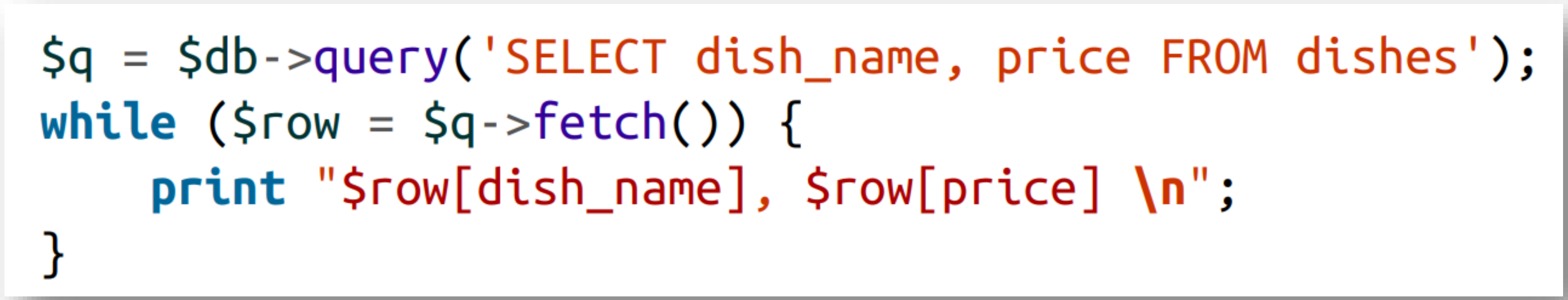
You don’t need to put quotes around the placeholder in the query. PDO takes care of it.

If you want to use multiple values in a query, put multiple placeholders (?) in the query and in the value array.

```
$stmt = $db->prepare('INSERT INTO dishes (dish_name,price,is_spicy) VALUES (?,?,?)');
$stmt->execute(array($_POST['new_dish_name'], $_POST['new_price'],
                     $_POST['is_spicy']));
```

Retrieving Data from the Database

- Use the query() method to retrieve information from the database
- Pass it an SQL query for the database.
- It returns a PDOStatement object that provides access to the retrieved rows.
- Each time you call the fetch() method of this object, you get the next row returned from the query.
- When there are no more rows left, fetch() returns a value that evaluates to false



```php
$q = $db->query('SELECT dish_name, price FROM dishes');
while ($row = $q->fetch()) {
    print "$row[dish_name], $row[price] \n";
}
```

Retrieving Data from the Database

```php
$q = $db->query('SELECT dish_name, price FROM dishes');
while ($row = $q->fetch()) {
    print "$row[dish_name], $row[price] \n";
}
```

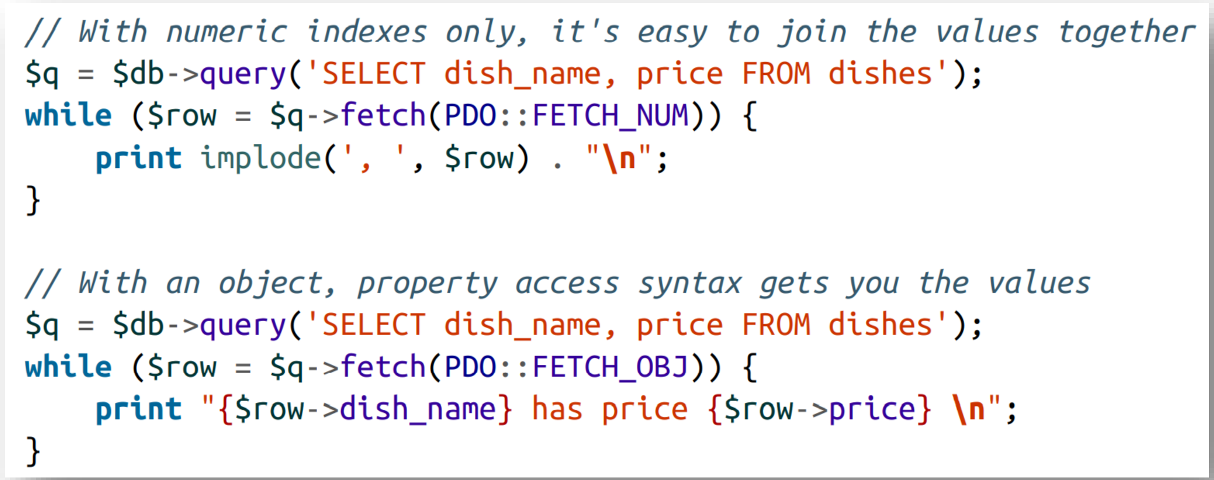
- fetch() returns an array with both numeric and string keys.
  - The numeric keys, starting at 0, contain each column's value for the row. ($row[0], $row[1]....)
  - The string keys do as well, with key names set to column names. ($row[dish_name], $row[price]....)
- If you have a small DB, use the fetchAll() method to put them into an array without looping

```php
$q = $db->query('SELECT dish_name, price FROM dishes');
// $rows will be a four-element array; each element is
// one row of data from the database
$rows = $q->fetchAll();
```

Retrieving Data from the Database

If you want to count number of rows, use SELECT COUNT(*) SQL statement

- By default, PHP returns indexed and associative array. But it is not efficient usage of resources
- PDO::FETCH_NUM, as the first argument to fetch() or fetchAll() gets an indexed array
- PDO::FETCH_ASSOC gets associative arrays
- PDO::FETCH_OBJ To get a row back as an object instead of an array

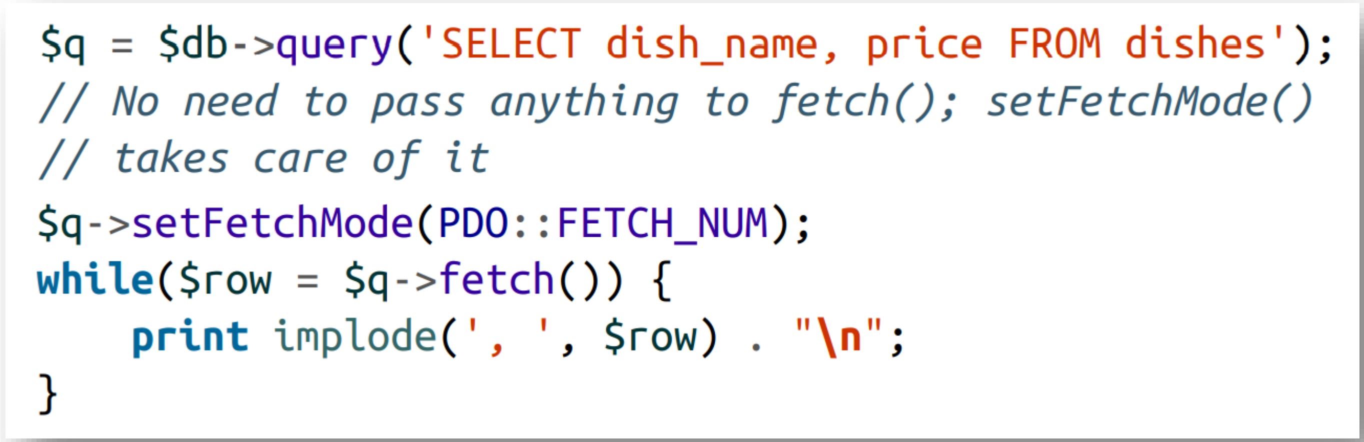


```php
// With numeric indexes only, it's easy to join the values together
$q = $db->query('SELECT dish_name, price FROM dishes');
while ($row = $q->fetch(PDO::FETCH_NUM)) {
    print implode(', ', $row) . "\n";
}

// With an object, property access syntax gets you the values
$q = $db->query('SELECT dish_name, price FROM dishes');
while ($row = $q->fetch(PDO::FETCH_OBJ)) {
    print "{$row->dish_name} has price {$row->price} \n";
}
```

Fetch Mode

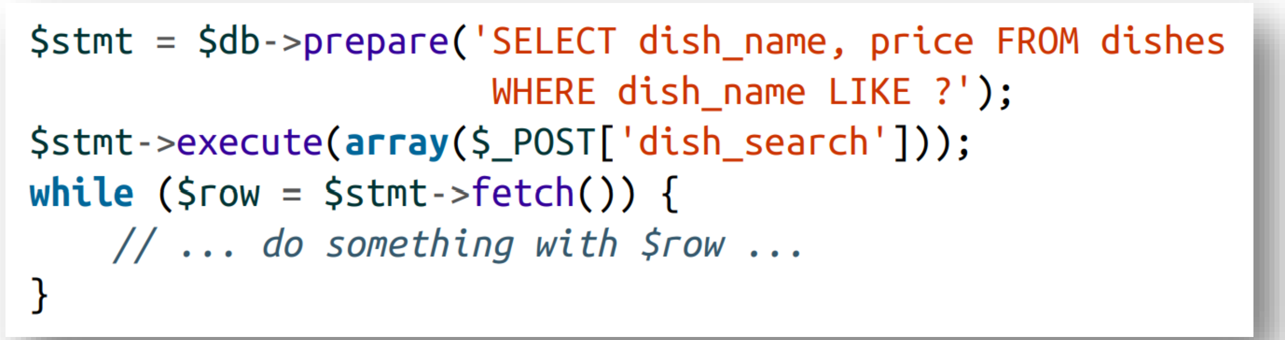
- You can set the mode for all queries you issue on a given connection by setFetchMode()



```php
$q = $db->query('SELECT dish_name, price FROM dishes');
// No need to pass anything to fetch(); setFetchMode()
// takes care of it
$q->setFetchMode(PDO::FETCH_NUM);
while($row = $q->fetch()) {
    print implode(', ', $row) . "\n";
}
```

Retrieving Form Data Safely

- It’s possible to use placeholders with SELECT statements
- NOTE: be careful of wildcards! (*, %, _, ? ...)

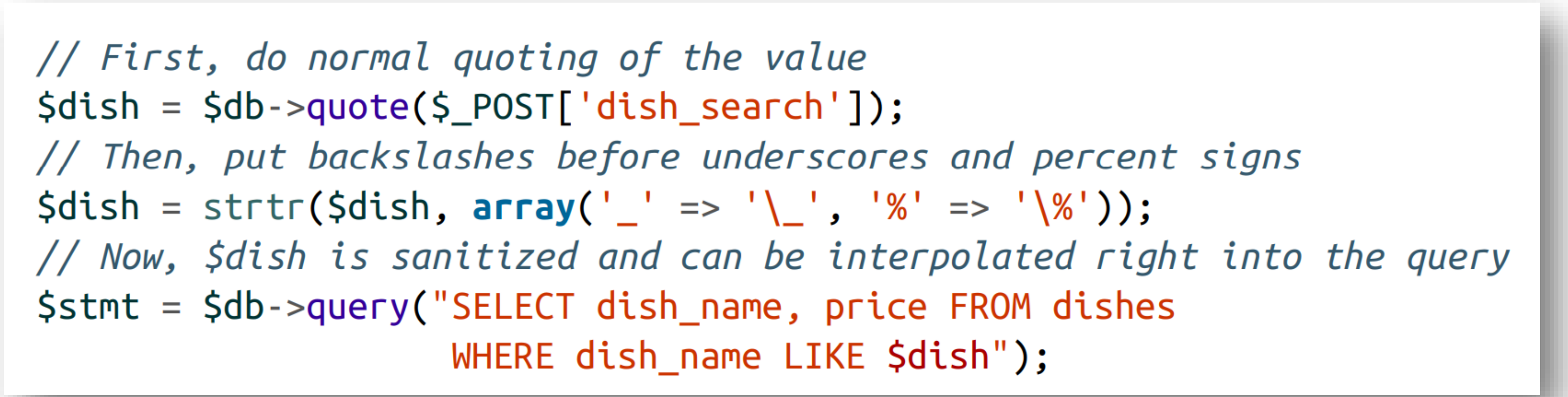


```
$stmt = $db->prepare('SELECT dish_name, price FROM dishes
                      WHERE dish_name LIKE ?');
$stmt->execute(array($_POST['dish_search']));
while ($row = $stmt->fetch()) {
    // ... do something with $row ...
}
```

- if the user typed in %chicken% Then, the query becomes SELECT dish_name, price FROM dishes WHERE dish_name LIKE '%chicken%'.
- To prevent SQL wildcards in form data, use quote() and strtr() function.

Retrieving Form Data Safely

- quote() adds quote around the input string
- strtr() replaces a string with another string. Use it to backslash-escape the SQL wildcards % and _.



```
// First, do normal quoting of the value
$dish = $db->quote($_POST['dish_search']);
// Then, put backslashes before underscores and percent signs
$dish = strtr($dish, array('_' => '\_', '%' => '\%'));
// Now, $dish is sanitized and can be interpolated right into the query
$stmt = $db->query("SELECT dish_name, price FROM dishes
                    WHERE dish_name LIKE $dish");
```

- NOTE: Not quoting wildcard characters has an even more drastic effect in the WHERE clause of an UPDATE or DELETE statements!

Exercises

The following exercises use a database table called `dishes` with the following structure:

```
CREATE TABLE dishes (
   dish_id     INT,
   dish_name   VARCHAR(255),
   price       DECIMAL(4,2),
   is_spicy    INT
)
```

Here is some sample data to put into the `dishes` table:

```
INSERT INTO dishes VALUES (1,'Walnut Bun',1.00,0)
INSERT INTO dishes VALUES (2,'Cashew Nuts and White Mushrooms',4.95,0)
INSERT INTO dishes VALUES (3,'Dried Mulberries',3.00,0)
INSERT INTO dishes VALUES (4,'Eggplant with Chili Sauce',6.50,1)
INSERT INTO dishes VALUES (5,'Red Bean Bun',1.00,0)
INSERT INTO dishes VALUES (6,'General Tso''s Chicken',5.50,1)
```

1. Write a program that lists all of the dishes in the table, sorted by price.
2. Write a program that displays a form asking for a price. When the form is submitted, the program should print out the names and prices of the dishes whose price is at least the submitted price. Don't retrieve from the database any rows or columns that aren't printed in the table.

Exercises

3. Write a program that displays a form with a `<select>` menu of dish names. Create the dish names to display by retrieving them from the database. When the form is submitted, the program should print out all of the information in the table (ID, name, price, and spiciness) for the selected dish.
4. Create a new table that holds information about restaurant customers. The table should store the following information about each customer: customer ID, name, phone number, and the ID of the customer's favorite dish. Write a program that displays a form for putting a new customer into the table. The part of the form for entering the customer's favorite dish should be a `<select>` menu of dish names. The customer's ID should be generated by your program, not entered in the form.