

# AppleScript-Verbindungen zu MySQL

---

Erstellt: Samstag, den 24. April 2010

Gedruckt: Donnerstag, den 19. August 2010

---

## Worum geht es?

Thema ist eine einfache AppleScript-Bibliothek mit etwas PHP-Scripting, die den Zugriff auf einen MySQL-Server einfach realisiert. Sie braucht nichts außer Dingen, die mit einer Standardinstallation von OS X kommen.

Vor langer Zeit brauchte ich eine Übertragung von Daten aus einer FileMaker-Datenbank zu einem MySQL Server. Unter Windows hatte ich mit ODBC experimentiert, aber die Probleme mit Codierung und Escape-Zeichen waren heftig. Außerdem bietet FileMaker mehr Möglichkeiten für einen AppleScript-Prozess auf einem Mac.

Als erstes hab ich eine kleine Bibliothek mit AppleScript-Anweisungen geschrieben, die einen einfachen Zugriff auf die MySQL-Datenbank brachte. Sie hat »do shell script« Befehle benutzt, die mit PHP-Skripts reden, die mit der Datenbank kommunizieren. PHP-Zugriff auf MySQL ist auf jeder Standard-Installation von OS X vorhanden, und ich wollte keine zusätzliche Erweiterung für das System installieren. Die Verbindung zweier Script-Sprachen ist langsamer als eine spezielle OSAX AppleScript-Erweiterung. Aber die meisten AppleScript-Lösungen verbrauchen mehr Zeit bei der Kommunikation mit anderen Programmen, deshalb verbessert ein schnellerer Datenbankzugriff die Gesamtperformance meist nur wenig.

Nachdem das Projekt mit FileMaker erledigt war, hab ich den Code für zukünftige Benutzung aufgeräumt. Das Ergebnis hat nichts mehr mit FileMaker zu tun und kann mit jedem scriptable Programm benutzt werden.

Änderungen in PHP haben meine erste Code-Version gebrochen und einige Änderungen wurden nötig. Da viele Dinge in AppleScript im Zusammenhang mit Seitengestaltung gemacht werden, wollte ich eine Möglichkeit haben, Bilder aus einer Datenbank zu laden. Diese Änderungen führten zur neuen Version.

## Voraussetzungen

Mac OS: Diese Scripts wurden unter Version 10.6.3 getestet. Ältere Versionen sollten funktionieren, ich habe aber nicht getestet.

MySQL-Server: Diese Scripts nehmen MySQL Version 4.1 oder neuer an. Diese Version war ein wesentlicher Schritt in Richtung angemessener Zeichencodierung, und dies nutzen diese Scripts.

# AppleScript-Verbindungen zu MySQL

Erstellt: Samstag, den 24. April 2010

Gedruckt: Donnerstag, den 19. August 2010

## Installation

Das Script »mysql\_objects.scptd« muss in den Ordner »/Library/Application Support/js\_aux\_scripts« kopiert werden. Legen Sie js\_aux\_scripts ggf. an.

## Erster Blick auf die Verwendung der MySQL Zugriffs-Scripts

### Eine einfache Situation

Nehmen wir folgende Situation für einen Textcode an.

Datenbank:

Der MySQL-Server läuft local, die IP-Nummer ist 127.0.0.1. Der Port ist 3306. An einer MySQL-Testinstallation wurde für dies Beispiel nichts geändert, es gibt den Benutzer »root« mit einem leeren Kennwort.

Die Datenbank heißt »test« mit einer Tabelle »humans«. Die Felder dieser Tabelle sind: id, firstname, secondname, street, city, zip, email und ein enter\_date. Das Feld »id« ist ein Wert mit autoincrement-Einstellung, das heißt: er zählt automatisch mit jedem neuen Datensatz hoch. Die übrigen Felder sind Zeichenketten, ausgenommen enter\_date. Das Feld ist ein MySQL datetime-Feld. Ein Benutzer mit vollem Zugriff zur Datenbank ist »test\_user« mit dem Kennwort »secret«.

## SQL Insert von Script ausführen

Zuerst muss die Bibliothek für die MySQL-Verbindung in das AppleScript geladen werden:

```
set library_path to "js_aux_scripts:mysql_objects.scptd"
set app_support to (path to application support folder) as string
set DB_Library to (load script file (app_support & library_path))
```

Dies legt die Bibliothek in einer Variable ab.

Zweitens braucht man eine Verbindung zur Datenbank:

```
set db_link to new_link("127.0.0.1", "root", "", "test") of
DB_Library
```

# AppleScript-Verbindungen zu MySQL

Erstellt: Samstag, den 24. April 2010

Gedruckt: Donnerstag, den 19. August 2010

Wählen wir eine einfache Einfügen-Anweisung in SQL:

```
INSERT INTO humans SET firstname='John', secondname='Nobody',email='nobody@bitbucket.com'
```

Dies hat nichts mit AppleScript zu tun, es ist einfaches SQL.

Im dritten Schritt muss die Anweisung in einer Variable abgelegt werden:

```
set command to "INSERT INTO humans SET firstname='John',  
secondname='Nobody',email='nobody@bitbucket.com'"
```

Viertens führen wir die Anweisung aus:

```
tell db_link to do_query(command)
```

Unser Testscript hat jetzt 6 Zeilen. Führen Sie das Script im Script Editor aus. Schauen Sie auf die »Ergebnis«-Tafel: Der Wert der letzten Zeile des Scripts ist ein AppleScript Record: {rows:1, insert\_id:1, error\_code:0, error\_string:""}. Nehmen Sie irgendein MySQL Hilfsprogramm wie SEQUEL Pro zum prüfen der Datenbank: Die Daten wurden in die Tabelle »humans« eingesetzt.

Wiederholen wir das Experiment mit kleiner Änderung: Ändern Sie die Zeile, die die Anweisung festlegt (Zeile 5), damit etwas schief geht. Ändern Sie den Feldnamen »secondname« in »lastname«. So ein Feld existiert nicht in der Datenbanktabelle. Lassen Sie das Script wieder laufen.

In der Ergebnistafel steht wieder ein Record: {rows:-1, insert\_id:0, error\_code:1054, error\_string:"Unknown column 'lastname' in 'field list'"}.

Damit wir klar über Zeilen sprechen können, hier noch einmal das Script:

```
Zeile 1  set library_path to "js_aux_scripts:mysql_objects.scptd"  
Zeile 2  set app_support to (path to application support folder) as  
         string  
Zeile 3  set DB_Library to (load script file (app_support &  
         library_path))  
Zeile 4  set db_link to new_link("127.0.0.1", "root", "", "test")  
         of DB_Library  
Zeile 5  set command to "INSERT INTO humans SET firstname='John',  
         secondname='Nobody',email='nobody@bitbucket.com'"  
Zeile 6  tell db_link to do_query(command)
```

# AppleScript-Verbindungen zu MySQL

Erstellt: Samstag, den 24. April 2010

Gedruckt: Donnerstag, den 19. August 2010

## Hinweise zum ersten Experiment

Fassen wir zusammen, was bisher passiert ist:

Die ersten drei Zeilen laden schlicht das Helfer-AppleScript, das die MySQL-Kommunikation abwickelt, in eine Variable. Wie jedes Script in einer Variable können seine Handler mit »tell« oder »of« aufgerufen werden. Die vierte Zeile ruft den Handler »new\_link« des Scripts auf und speichert es in Variable »db\_link«. Der Handler wird mit vier Argumenten aufgerufen: IP-Adresse des Servers (oder sein Name im domain name system), dem Benutzernamen, dem Benutzerkennwort und dem Namen der benutzten Datenbank.

Der Handler »new\_link« liefert ein anderes Script. Es kennt die Verbindungsparameter zum MySQL-Server. Wieder hat es Handler, die im Script benutzt werden können.

Die Zeile »tell db\_link to do\_query(command)« ist so ein Aufruf des Handlers des Scripts in »db\_link«. Sie schickt die Anweisung an den MySQL-Server und liefert einen Record mit einiger Information zum Vorgang.

»rows« ist die Anzahl der Zeilen, die die Anweisung betroffen hat. Da wir einen Datensatz eingefügt haben, ist der Wert 1 im ersten Versuch. Im zweiten Fall gab es einen Fehler, er ist -1. Er ist in allen Fällen -1, in denen ein Fehler auftritt.

»insert\_id« ist nur ein nützlicher Wert im Fall von insert-Anweisungen. Es ist der Wert des autoincrement-Felds der Tabelle, »id« in unserem Fall.

»error\_code« ist der Fehler-Code, den der MySQL-Server geliefert hat. Er ist 0, wenn kein Fehler auftrat.

»error\_string« ist die Textnachricht vom Server geliefert. Sie hilft oft, Fehler zu finden, wie der falsche Spalten- (oder Feld-) Name im zweiten Experiment.

## Sicherheitshinweis

Leider werden Dinge komplizierter in realistischen Fällen. Werte in eine Datenbank einzusetzen kann verheerend sein, wenn die Werte einige illegale Zeichen enthalten. Spätere Beispiele zeigen, wie man Werte für SQL-Verwendung sicher macht. Suchen Sie im Web nach »SQL Injektion« um Details zu möglichen Problemen zu finden.

# AppleScript-Verbindungen zu MySQL

Erstellt: Samstag, den 24. April 2010

Gedruckt: Donnerstag, den 19. August 2010

## Daten abrufen

Fügen Sie der Datenbank einige weitere Zeilen zu, indem Sie das Script wieder aufrufen, mit richtigem Spaltennamen und neuen Werten, damit es mehr Einträge in der Tabelle gibt.

Eine SQL-Anweisung, alle Werte abzufragen, ist »SELECT \* from humans«.

Ändern Sie die Variable command (Zeile 5):

```
set command to "SELECT * FROM humans"
```

Lassen Sie das Script wieder laufen. Sie erhalten einen Record wie folgt: {rows:3, insert\_id:0, error\_code:0, error\_string:""}. Der Wert für die Zeilen mag bei Ihnen abweichen: Es ist einfach die Anzahl der Zeilen in Ihrer Tabelle. Ich habe davon drei, deshalb gibt es 3 betroffene Zeilen. Dies mag interessant sein, aber Sie brauchen die Daten. Das Script in »db\_link« hat dafür einen Handler, »do\_select«. Ändern Sie deshalb die letzte Zeile (Zeile 6) des Scripts zu

```
tell db_link to do_select(command)
```

und lassen Sie das Script wieder laufen. Jetzt sieht der abgelieferte Record ganz anders aus. Bei mir ist das Ergebnis:

```
{rows:3, cols:7, record_data:{{"6", "John", "Nobody", "", "", "", "nobody@bitbucket.com"}, {"7", "Joan", "Sample", "", "", "", "sample@bitbucket.com"}, {"8", "Joseph", "Tester", "", "", "", "tester@bitbucket.com"}}
```

Wieder gibt es den »rows«-Wert. Zusätzlich aber den Wert »cols« und einen für »record\_data«, eine Liste von Listen. »rows« ist die Anzahl der Zeilen (oder Datensätze), die die Abfrage liefert. »cols« ist die Anzahl der Spalten in jedem Datensatz. »record\_data« ist eine Liste mit einer Liste für jeden Datensatz.

Lassen Sie uns einen Fehler machen, wie beim insert-Experiment, ein falscher Name für die Datenbank-Tabelle. Ändern Sie die command-Variable (Zeile 5):

```
set command to "SELECT * FROM persons"
```

und lassen Sie das Script laufen. Es liefert: {rows:-1, cols:1, record\_data:{{1146}, {"Table 'test.persons' doesn't exist"}}}.

Wieder ist die Zeilenanzahl -1 um den Fehler anzuzeigen. »record\_data« enthält jetzt die Fehlernummer und die Textnachricht des SQL-Servers.

Sie mögen das Konzept negativer Zeilenzahlen für Fehler nicht? Direkt nachdem der »db\_link« angelegt wird (zwischen Zeile 4 und 5) fügen Sie ein:

```
set raise_errors of db_link to true
```

# AppleScript-Verbindungen zu MySQL

Erstellt: Samstag, den 24. April 2010

Gedruckt: Donnerstag, den 19. August 2010

Starten Sie das Script. Eine normale AppleScript-Fehlermeldung erscheint. Beachten Sie: Fehler-Codes sind die, die der Server liefert. In einem »try – on error«-Konstrukt dürfen sie nicht als AppleScript-Fehlercodes behandelt werden. Es sind MySQL-Fehlercodes.

## Unterwegs zur wirklichen Welt

Nehmen wir an, Ihr Script bekommt die Daten, die in den Server sollen, von einem anderen Programm, FileMaker Pro zum Beispiel. Dieselben Zeilen wie zuvor laden die Bibliothek und erzeugen db\_link.

Erzeugen wir eine Liste mit den Feldnamen in MySQL. Wir ignorieren das enter\_date-Feld im Augenblick.

```
set field_names to {"firstname", "secondname", "email"}
```

Wir simulieren die Werteliste, die von FileMaker kommt, mit der Zeile

```
set value_list to {"Liza", "O'Connor", "oconnor@bitbucket.com"}
```

In einem ersten Schritt machen wir die Werteliste sicher für SQL-Gebrauch:

```
set value_list to escape_string(value_list) of db_link
```

Um die fertige insert-Anweisung zu erzeugen, enthalten Konverter einige nützliche Funktionen. Wir erzeugen einen:

```
set converter to new_converter() of db_link
```

Damit sind die meisten Elemente beisammen, nun die insert-Anweisung:

```
set command to "INSERT INTO humans SET " &  
merge_names_values(field_names, value_list) of converter
```

Starten Sie das Script und schauen Sie sich das Ergebnis an:

```
"INSERT INTO humans SET `firstname` = 'Liza', `secondname` = 'O\\  
'Connor', `email` = 'oconnor@bitbucket.com'"
```

Sie sehen die beiden Backslashes in "...= 'O\\'Connor'..."? Hier wird die Darstellung unübersichtlich. Sie sehen den String mit den Escape-Zeichen für SQL wiederum in der Escape-Darstellung von AppleScript. Um den wirklichen String zu sehen, ergänzen Sie

```
display dialog command
```

am Ende. Starten Sie das Script, erscheint ein Dialog mit »INSERT INTO humans SET `firstname` = 'Liza', `secondname` = 'O\\'Connor', `email` = 'oconnor@bitbucket.com'«. Dies ist der wirkliche String.

Beachten Sie zwei Dinge: »...= 'O\\'Connor',...« hat den Backslash vor dem »'«. Dies ist notwendig, damit die SQL-Anweisung nicht bricht. Die escape\_string-

# AppleScript-Verbindungen zu MySQL

Erstellt: Samstag, den 24. April 2010

Gedruckt: Donnerstag, den 19. August 2010

Funktion hat dies erledigt.

Alle Feldnamen sind von »backquotes« begrenzt, die Feldwerte von einfachen Anführungszeichen. Dies ist der zuverlässigste Weg, eine gültige Anweisung für MySQL zu erzeugen. Der Konverter merge\_names\_values hat dies erledigt.

Nun das INSERT:

```
do_query(command) of db_link
```

(Denken Sie daran, die display dialog- Anweisung zu entfernen!)

Lassen Sie das Script laufen.

## In Kürze

Hier das ganze Script soweit:

```
set library_path to "js_aux_scripts:mysql_objects.scptd"
set app_support to (path to application support folder) as string
set DB_Library to (load script file (app_support & library_path))
```

```
set db_link to new_link("127.0.0.1", "root", "", "test") of
DB_Library
```

```
-- create field names list
set field_names to {"firstname", "secondname", "email"}
```

```
-- simulating some FileMaker result
set value_list to {"Liza", "O'Connor", "oconnor@bitbucket.com"}
```

```
set converter to new_converter() of db_link
```

```
-- make values SQL safe. Really important for security!!!
set value_list to escape_string(value_list) of db_link
```

```
set command to "INSERT INTO humans SET " &
merge_names_values(field_names, value_list) of converter
```

```
do_query(command) of db_link
```

Ohne die Kommentare sind das 10 Zeilen. Beginnend mit zwei Listen, eine für SQL-Feldnamen, eine für Werte, erzeugen sie eine sichere INSERT-Anweisung und führen sie aus.

Eine knappe Skizze der Grundstrategie:

# AppleScript-Verbindungen zu MySQL

Erstellt: Samstag, den 24. April 2010

Gedruckt: Donnerstag, den 19. August 2010

- Erzeugen je einer Liste für Feldnamen und Werte.
- Mit `escape_string` werden die Werte abgesichert
- Die `INSERT`-Anweisung wird mit Hilfe von `merge_names_values` aus Feldnamen und Werten aufgebaut.
- Die Anweisung wird mit `do_query` ausgeführt.

## Konverter benutzen

Unser ursprünglicher Vorschlag für die Tabelle enthielt ein Feld namens `enter_date`. Ergänzen wir es im nächsten Versuch.

Erweitern Sie die Feldliste um `enter_date`. Ändern Sie dafür die Zeile, die die Liste erzeugt zu:

```
set field_names to {"firstname", "secondname", "email",  
"enter_date"}
```

Nach der Zeile, die den Konverter erzeugt, ergänzen Sie folgenden Code:

```
set my_now to current date  
set now_string to SQLdatetime(my_now) of converter  
copy now_string to end of value_list
```

Damit wird ein MySQL-kompatibler Wertestring für aktuelles Datum und Zeit an die Liste angehängt. Die resultierende SQL-Anweisung sieht etwa wie folgt aus (abhängig von Datum und Uhrzeit natürlich):

```
INSERT INTO humans SET `firstname` = 'Liza', `secondname` = 'O\  
'Connor', `email` = 'oconnor@bitbucket.com', `enter_date` = '2010-  
04-17 22:14:52'
```

Das zeigt grundsätzlich, wie Konverter benutzt werden. Es gibt eine Gruppe mit Grundfunktion in einem Konverter-Objekt. `SQLdatetime` ist eine davon. Sie nimmt ein AppleScript Datumsobjekt und liefert einen String, den MySQL versteht. Weitere sind:

- `float2mysql(Zahl)`: Konvertiert eine Zahl in einen String, mit einem Punkt als Dezimaltrenner, auch wenn Ihr gewähltes Lokale ein Komma nimmt.
- `SQLdate` (Datumsobjekt): Liefert einen Datumsstring, ohne Zeitanteil.
- `SQLtime` (Datumsobjekt): Liefert einen Zeitstring, ohne Datumsteil.