PLDatabase

Project Information

Download

- 1.2.1 (Binary)
- 1.2.1 (Source)
- 2.0-beta2 (Binary)
- 2.0-beta2 (Source)
- Previous Releases

Documentation

- 1.2.1 (Stable Release)
- 1.1.1 (Previous Release)

Mailing Lists

- User Mailing List
- Commits List

Development

- Source Repository
- Issue Tracker
- Release Checklist

• License (3-Clause BSD)

Introduction

PLDatabase provides a SQL database access library for Objective-C, focused on SQLite as an application database. The library supports both Mac OS X and iPhone development.

Plausible Database is provided free of charge under the BSD license, and may be freely integrated with any application.

Building

To build your own release binary, build the 'Disk Image' target:

```
user@max:~/pldatabase-1.2> xcodebuild -configuration Release -target 'Disk Image'
```

This will output a new release disk image containing an embeddable Mac OS X framework and a static iPhone framework in build/Release/Plausible Database-{version}.dmg

---

Basic Usage

Refer to the API Documentation for details.

Creating a connection

Open a connection to a database file:

Creating a connection

```
PLSqliteDatabase *db = [[PLSqliteDatabase alloc] initWithPath:  @"/path/to/database"];
if (![db openAndReturnError: &error])
    NSLog(@"Could not open database");
```

Update Statements

Update statements can be executed directly via -[PLDatabase executeUpdateAndReturnError:statement:...]

```
if (![db executeUpdateAndReturnError: &error statement: @"CREATE TABLE example (id
INTEGER)"])
    NSLog(@"Table creation failed");
if (![db executeQueryAndReturnError: &error statement: @"INSERT INTO example (id)
VALUES (?)", [NSNumber numberWithInteger: 42]])
    NSLog(@"Data insert failed");
```

Query Statements

Queries can be executed using -[PLDatabase executeQueryAndReturnError:statement:...]. To iterate over the returned results, a instance conforming to the PLResultSet protocol will be returned.

```
id<PLResultSet> results = [db executeQueryAndReturnError: &error statement: @"SELECT
id FROM example WHERE id = ?", [NSNumber numberWithInteger: 42]];
PLResultSetStatus rss;
while ((rss = [results nextAndReturnError: &error]) == PLResultSetStatusRow) {
    NSLog(@"Value of column id is %d", [results intForColumn: @"id"]);
}

if (rss != PLResultSetStatusDone)
    NSLog(@"Iterating results failed");

// Failure to close the result set will not leak memory, but may
// retain database resources until the instance is deallocated.
[results close];
```

Prepared Statements

Pre-compilation of SQL statements and advanced parameter binding are supported by PLPreparedStatement. A prepared statement can be constructed using -[PLDatabase prepareStatement:error:].

```
id<PLPreparedStatement> stmt = [db prepareStatement: @"INSERT INTO example (name,
color) VALUES (?, ?)" error: &error];

// Bind the parameters
[stmt bindParameters: @["Widget", @"Blue"]];

// Execute the INSERT
if ([stmt executeUpdateAndReturnError: &error] == NO)
    NSLog(@"INSERT failed");
```

Name-based Parameter Binding

Name-based parameter binding is also supported:

```
// Prepare the statement
id<PLPreparedStatement> stmt = [db prepareStatement: @"INSERT INTO test (name, color)
VALUES (:name, :color)" error: &error];

// Bind the parameters using a dictionary
[stmt bindParameterDictionary: @{ @"name" : @"Widget", @"color" : @"Blue" }];

// Execute the INSERT
if ([stmt executeUpdateAndReturnError: &error] == NO)
    NSLog(@"INSERT failed");
```