As Promised, here is the SQLite Cheat Sheet. It list all the SQLite functions and methods you've used in the Using SQLite workshops.

# Common SQLite Functions

This table list common functions of the SQLite engine for preparing, executing, and finalizing SQL statements.

| |
|---|
| This function open a database file. If the file doesn't exists, it is created first.<br><br>sqlite3_open([self.dbFile UTF8String], &dbHandle); |
| This function close the database file the sqlite3_open() function opened.<br><br>sqlite3_close(dbHandle); |
| This is wrapper function does the work of three functions listed below.<br><br>sqlite3_exec(dbHandle, sql, NULL, NULL, NULL); |
| These functions prepare an SQL statement, executed the prepared statement, and finalize the prepared statement.<br><br>sqlite3_prepare_v2(dbHandle, sqlStatement, -1, &compiledStatement, NULL);<br>sqlite3_step(compiledStatement);<br>sqlite3_finalize(compiledStatement); |
| These bind functions bind values to a prepared statement's place holder which is a ?. The first parameter is a pointer to the sqlite3_stmt, the second parameter is the index of the SQL parameter to be set. The third parameter is the value to bind to the prepared statement's place holder. As you can see, the last two bind functions take a fourth and fifth parameter.<br><br>sqlite3_bind_int(compiledStatement, 0, 34);<br>sqlite3_bind_double(compiledStatement, 1, 999.99);<br>sqlite3_bind_text(compiledStatement, 2, [itemName UTF8String], -1, NULL);<br>sqlite3_bind_blob(compiledStatement, 3, [imageData bytes], [imageData length], NULL); |
| These sqlite3_column_xxx functions return a column's data for the current row. The first parameter is a pointer to the prepared statement that is being evaluated. The second parameter is the index number of the row's column and it is zero based. This mean the first column's index number is 0, the second column index number is 1, etc.<br><br>sqlite3_column_double(compiledStatement, 2);<br>sqlite3_column_int(compliedStatement, 3);<br>sqlite3_column_text(compiledStatement, 0);<br>sqlite3_column_blob(compiledStatement, 1); |

# Custom Database Methods

Here is a list iOS methods you can use in an iOS application to perform specific database operation, using above mentioned SQLite functions.

*This method fetch and return the full path to the SQLite database file residing in the Documents folder of the app's sandbox.*

```
- (NSString *)docPath
{
  NSArray *paths = NSSearchPathForDirectoriesInDomains(NSDocumentDirectory,
NSUserDomainMask, YES);
  NSString *docsFolder = [paths objectAtIndex:0];
  return [docsFolder stringByAppendingPathComponent:@"database.sqlite"];
}
```

*This method opens the SQLite database file. If it doesn't already exists, the method create it first.*

```
- (void)openDatabase
{
  // Get full path to the database file
  NSString *dbFile = [self docPath];

  // Check to see if the database file exists in the Documents folder
  BOOL dbExist = [[NSFileManager defaultManager] fileExistsAtPath:dbFile];

  if(!dbExist) {
    // It doesn't, so create the database in the sandbox's Documents folder, then open it
    int result = sqlite3_open([dbFile UTF8String], &dbHandle);

    if (result != SQLITE_OK) {
      // Unable to open the database file, so close it
      [self closeDatabase];

      // Display the error message
      NSLog(@"openDatabase Error:\n%s",sqlite3_errmsg(dbHandle));
    }
  }
  sqlite3_open([dbFile UTF8String], &dbHandle);
}
```

*This method close the SQLite database file the openDatabase method opened.*

```
- (void)closeDatabase
{
  // Close the SQLite db file
  int result = sqlite3_close(dbHandle);

  if (result != SQLITE_OK) {
    // Unable to close the db file, so display the error message
    NSLog(@"closeDatabase Error:\n'%s'", sqlite3_errmsg(dbHandle));
  }
}
```

*This method create a table in the database file.*

```
- (void)createTable
{
  const char *sql = "CREATE TABLE IF NOT EXISTS possessions(id INTEGER PRIMARY KEY
AUTOINCREMENT, name TEXT, description TEXT, photo BLOB)";

  // Create the table in the database file
  int result = sqlite3_exec(dbHandle, sql, NULL, NULL, NULL) != SQLITE_OK;

  if (result != SQLITE_OK) {
    // Query failed, display the error message
    NSLog(@"createTable Error:\n'%s'", sqlite3_errmsg(dbHandle));
  }
}
```

*This method fetch the database table names (including the sqlite_sequence table) and return it to the caller of the method.*

```
- (NSString*)getDatabaseTableNames
{
  NSString *tableNames = nil;
  NSMutableArray *dbRecords = [[NSMutableArray alloc] init];

  const char *sql = "SELECT tbl_name FROM sqlite_master";
  sqlite3_stmt *compiledStatement;

  int result = sqlite3_prepare_v2(dbHandle, sql, -1, &compiledStatement, NULL);
  if (result != SQLITE_OK) {
    // Query failed, display the error message
    NSLog(@"createTable Error:\n'%s'", sqlite3_errmsg(dbHandle));
  }

  while(sqlite3_step(compiledStatement) == SQLITE_ROW) {
    // Put records the sql query returned in the dbRecords array
    NSString *currentRecord = [NSString stringWithUTF8String:
    (char *)sqlite3_column_text(compiledStatement, 0)];
    [dbRecords addObject:currentRecord];
  }
```

```
  sqlite3_finalize(compiledStatement);

  // Convert the mutable array to an array
  NSArray *mutableArray = [dbRecords mutableCopy];

  // Convert the array elements to string objects before assigning them to the tableNames variable
  tableNames = [mutableArray componentsJoinedByString:@"\n"];

  return tableNames;
}
```

*This method create a record in a the database's table.*

```
- (void) createRecord:(Item *)formData
{
  // Set up the INSERT string
  const char *sqlStatement = "INSERT INTO possessions (name, description, photo) VALUES
(?, ?, ?)";
  sqlite3_stmt *compiledStatement;

  // Prepare the sqlStatement
  int result = sqlite3_prepare_v2(dbHandle, sqlStatement, -1, &compiledStatement, NULL);

  // Bind the sqlStatement parameters
  sqlite3_bind_text(compiledStatement, 1, [formData.itemName UTF8String], -1, NULL);
  sqlite3_bind_text(compiledStatement, 2, [formData.itemDescription UTF8String], -1, NULL);
  NSData *imageData = UIImagePNGRepresentation(formData.itemImage);
  sqlite3_bind_blob(compiledStatement, 3, [imageData bytes], [imageData length], NULL);

  if(result != SQLITE_OK) {
    NSLog(@"createRecord Error:\n%s", sqlite3_errmsg(dbHandle));
  }

  // Execute the prepared statement (Insert a record in the database file)
  result = sqlite3_step(compiledStatement);

  // Finalize the prepared statement
  sqlite3_finalize(compiledStatement);
}
```

*This method retrieve one or more records from a single table of the database file, put the result set in a mutable array, then return the array to the caller of the method.*

```
- (NSMutableArray *) retrievePossessions
{
  // This array is for holding records fetched from the database's possessions table
  NSMutableArray *dbRecords = [[NSMutableArray alloc] init];
  NSData *dbImage = [[NSData alloc] init];

  const char *sql = "SELECT name, description, photo FROM possessions";
  sqlite3_stmt *compiledStatement;

  //1. Prepare the sql statement
```

```
  int result = sqlite3_prepare_v2(dbHandle, sql, -1, &compiledStatement, NULL);

  if(result != SQLITE_OK) {
    NSLog(@"retrievePossessions Error:\n%s", sqlite3_errmsg(dbHandle));
  } else {
    //2. Execute the prepared statement
    while(sqlite3_step(compiledStatement) == SQLITE_ROW) {
      // Convert the record's columns to Objective-C objects before assigning them to object variables
      NSString *itemName = [NSString stringWithUTF8String:(const char
*)sqlite3_column_text(compiledStatement, 0)];
      NSString *itemDescription = [NSString stringWithUTF8String:(const char
*)sqlite3_column_text(compiledStatement, 1)];

      const char *binaryData = sqlite3_column_blob(compiledStatement, 2);
      int dataLength = sqlite3_column_bytes(compiledStatement, 2);
      dbImage = [NSData dataWithBytes:binaryData length:dataLength];

      // Instantiate an object from the Item class
      Item *record = [[Item alloc] init];

      // Initialize the object's properties
      record.itemName = itemName;
      record.itemDescription = itemDescription;
      record.itemImage = [UIImage imageWithData:dbImage];

      // Add the object in the mutable array
      [dbRecords addObject:record];
    }
  }
  //3. Finalize the prepared statement
  sqlite3_finalize(compiledStatement);
  return dbRecords;
}
```

*This method Check to see a record already exists in the database's table that satisfy the select statement's WHERE condition. If the statement return any results, the method return the Boolean value YES to the caller of the method; otherwise the boolean value NO is returned.*

```
- (BOOL) checkForDuplicateEntry:(NSString *)itemName
{
  const char *sql = "SELECT name FROM possessions WHERE name = ?";
  sqlite3_stmt *compiledStatement;
  BOOL recordFound = NO;

  //1. Prepare the sql statement
  int result = sqlite3_prepare_v2(dbHandle, sql, -1, &compiledStatement, NULL);

  // 2. Bind the sqlStatement parameters to a value
  sqlite3_bind_text(compiledStatement, 1, [itemName UTF8String], -1, NULL);

  if(result != SQLITE_OK) {
    NSLog(@"checkForDuplicateEntry Error:\n%s", sqlite3_errmsg(dbHandle));
  }
```

```
  //3. Execute the prepared statement
  if (sqlite3_step(compiledStatement) == SQLITE_ROW) {
    recordFound = YES;
  }

  //4. Finalize the prepared statement
  sqlite3_finalize(compiledStatement);
  return recordFound;
}
```

*This method update a record in the database's table that satisfy the update statement's WHERE condition.*

```
- (void)updatePossession:(NSString *)oldItemName field2:(Item *)formData
{
  const char *sql = "UPDATE possessions SET name = ?, description = ?, photo = ? WHERE name = ?";
  sqlite3_stmt *compiledStatement;

  //1. Prepare the sql statement
  int result = sqlite3_prepare_v2(dbHandle, sql, -1, &compiledStatement, NULL);

  //2. Bind the method's parameters
  sqlite3_bind_text(compiledStatement, 1, [formData.itemName UTF8String], -1, NULL);
  sqlite3_bind_text(compiledStatement, 2, [formData.itemDescription UTF8String], - 1, NULL);
  NSData *imageData = UIImagePNGRepresentation(formData.itemImage);
  sqlite3_bind_blob(compiledStatement, 3, [imageData bytes], [imageData length], NULL);
  sqlite3_bind_text(compiledStatement, 4, [oldItemName UTF8String], -1, NULL);

  if(result != SQLITE_OK) {
    NSLog(@"updatePossession method error:\n%s", sqlite3_errmsg(dbHandle));
  }

  //3. Execute the sql statement
  result = sqlite3_step(compiledStatement);
  if (result != SQLITE_DONE) {
    NSLog(@"updatePossession method error:\n%s", sqlite3_errmsg(dbHandle));
  }

  //3. Finalize the prepared statement
  sqlite3_finalize(compiledStatement);
}
```

*This method delete a record from the database's table that satisfy the delete statement's WHERE condition.*

```
- (void) deletePossession:(NSString *)itemName
{
  const char *sqlStatement = "DELETE FROM possessions WHERE name = ?";
  sqlite3_stmt *compiledStatement;

  //1. Prepare the sql statement
  int result = sqlite3_prepare_v2(dbHandle, sqlStatement, -1, &compiledStatement, NULL);

  if(result != SQLITE_OK) {
```

```
    NSLog(@"There's an error in the delete statement:\n%s", sqlite3_errmsg(dbHandle));
  }

  //2. Bind the method's parameter value to the sqlStatement's placeholder ?
  sqlite3_bind_text(compiledStatement, 1, [itemName UTF8String], -1, NULL);

  //3. Execute the prepared statement
  sqlite3_step(compiledStatement);

  if (result != SQLITE_DONE) {
    NSLog(@"The deletePossession method error:\n%s", sqlite3_errmsg(dbHandle));
  }

  //4. Finalize the prepared statement
  sqlite3_finalize(compiledStatement);
}
```

*This method perform two queries. The first one gets the total number of records the albums table contain. The second query perform a INNER JOIN query, which fetch records from two tables. Now, if the first query returned any rows, the second query rows are added in a mutable array and it is returned to the sender of the method. If the first query returned no rows, an empty mutable array is returned to the caller of the method.*

```
- (NSMutableArray *)joinQuery {
  NSMutableArray *dbRows = [[NSMutableArray alloc] init];
  int totalRows = 0;

  // This query returns the total number of links the album contain
  const char *queryOne = "SELECT count(photos.pid) AS totalRows FROM albums INNER JOIN photos
ON albums.aid=photos.aid WHERE albums.albumName = \"Fathers Day\"";
  sqlite3_stmt *compiledStatement1;

  // This query fetch an album's photoUrl
  const char *queryTwo = "SELECT photos.photoUrl FROM albums INNER JOIN photos ON
albums.aid=photos.aid WHERE albums.albumName = \"Fathers Day\"";
  sqlite3_stmt *compiledStatement2;

  // Prepare both statements and display errors, if any
  int resultOne = sqlite3_prepare_v2(dbHandle, queryOne, -1, &compiledStatement1, NULL);
  int resultTwo = sqlite3_prepare_v2(dbHandle, queryTwo, -1, &compiledStatement2, NULL);

  if(resultOne != SQLITE_OK) {
    NSLog(@"joinQuery error:\n%s", sqlite3_errmsg(dbHandle));
  } else if (resultTwo != SQLITE_OK){
    NSLog(@"joinQuery error:\n%s", sqlite3_errmsg(dbHandle));
  }

  // Both queries returned no errors, so execute the queryOne
  resultOne = sqlite3_step(compiledStatement1);

  if (resultOne == SQLITE_ROW) {
    // Place the single column queryOne returned in a variable
    totalRows = sqlite3_column_int(compiledStatement1, 0);
  }
```

```
  // Validate the totalRows variable
  if (totalRows > 0) {
    // Execute queryTwo in a while loop
    while(sqlite3_step(compiledStatement2) == SQLITE_ROW) {
      // Convert queryTwo's column values from C data types to objective-C data types,
      // before assigning them to Objective-C object variable
      NSString *photoUrls = [NSString stringWithUTF8String:
      (char *)sqlite3_column_text(compiledStatement2, 0)];

      // Add each database row in the mutable array
      [dbRows addObject:photoUrls];
    }
  }
  // Finalize both prepared statements
  sqlite3_finalize(compiledStatement1);
  sqlite3_finalize(compiledStatement2);

  // return an empty array or rows returned from queryTwo
  return dbRows;
}
```