Chapter 1: Introduction to SQL and SQLite

```
SELECT parameter1, STTDEV(parameter2) FROM Table1 Group by parameter1 HAVING parameter1 > MAX(parameter3)
```

```
let testdb = SQLiteDB.sharedInstance()
```

```
var theresult = testdb.query("select * from people where county = 'Berks'", parameters: nil)
for row in result
{
   println(row["name"]!.asString())
}
```

```
testdb.execute("delete from people where county = 'Bucks' ", parameters: nil)
```

Chapter 2: Database Design Concepts

```
CREATE table database-name. table-name( column1 datatype, column2 datatype, column3, datatype, PRIMARY KEY column1);
```

```
INSERT into table-name(column1,column2,column3) VALUES(variable1,variable2,variable3);
```

```
UPDATE table-name SET column1=variable1, column2=variable2, column3=variable3) [where variable4 = 10];
```

```
SELECT column1, column2, column3 FROM table-name WHERE column1 > 10;
```

```
DELETE from table-name where column1 >10;
```

```
sqlite3 aFile.db "create table aTable(field1 int); drop table aTable;"
```

```
# example of using reset - START
db1= open('property.db')
sql_statement= db1.prepare('insert into property_info(id,property_id,desc) values(:id,:pr_id,:desc)')
sql_statement.bind('id','100')
sql_statement.bind('property_id','1')
sql_statement.bind('desc','this is a test')
sql_statement.step()

# Reuse existing compiled parameters
sql_statement.reset()
sql_statement.bind('id','200')
sql_statement.bind('property_id','2')
sql_statement.bind('desc','this is a test again')

# End
```

```
statement_sql.finalize()
db1.close()
```

```
db1= open('property.db')
sql_statement= db1.exec("insert into property_info(id,property_id,desc) values(1,2,'Property Description 1')")
sql_statement= db1.exec("insert into property_info(id,property_id,desc) values(2,2,'Property Description 2')")
```

```
SELECT * from property where property_name='%s';
```

```
void test_function(sqlite3_content* tmp_value, int tmp_assign, sqlite3_value** values)
{
    /* Respond back Text or reply */
    const char *tmp_string ="Test String - Hello World";
    /* Set value to be returned */
    sqlite3_result_text(tmp_value,tmp_string,strlen(tmp_string),SQL_STATIC);
}
```

Execute it by creating function using - sqlite3_create_function(db1,"test_function", 0,test_function);

Chapter 3: Administering the Database

```
$ sqlite3 testdatabase.db
```

Chapter 4: Essentials of SQL

```
SQLite> Insert into Salary values (Select id, name, salary from salary_import where name='Smith');
SQLite> Select * from Salary where name like '%smith%';
```

```
SQLite> Update salary = 15000 Where name='John Smith';
SQLite> Select * from Salary where name like '%smith%';
```

```
SQLite> INSERT into salary (name, salary, bonus) values ('John Smith',15000,2000); sqlite> SELECT * FROM salary;
```

```
CREATE Table Salary (id integer primary key, name text, salary};
```

```
CREATE INDEX table_index_name ON customer;
```

```
CREATE INDEX table_index_salary ON customer (salary);
```

```
CREATE INDEX table_index_salary ON customer (salary, bonus);
```

```
sqlite> .indices customer;
```

```
sqlite> select sql from sqlite_master where name='update_customer_trigger';
```

```
CREATE TRIGGER update_customer_trigger UPDATE OF tel_no ON customers BEGIN UPDATE orders SET tel_no =
new.tel_no WHERE customer_name = old.name; END
```

```
FMDatabase *db = [FMDatabase databaseWithPath:@"/tmp/atmp.db"];
```

```
if (![db open]) { [db release]; return; }
```

```
FMResultSet *s = [db executeQuery:@"SELECT * FROM aTable"]; while ([s next]) { //retrieve values for each record }
```

```
FMResultSet *s = [db executeQuery:@"SELECT COUNT(*) FROM aTable"]; if([s next]) { int totalCount = [s
intForColumnIndex:0]; }
```

Chapter 5: Exposing the C API

```
var db1 = SQLiteDatabase();
db1.open("/path/to/database1.sqlite");
```

```
let datadocuments = NSSearchPathForDirectoriesInDomains(.DocumentDirectory, .UserDomainMask, true)[0] as String
let databasepath = documents.stringByAppendingPathComponent("tester.sqlite")
// open the database
var databasedb: DBPointer = nil
if sqlite3_open(path, &databasedb) != SQLITE_OK
{
    println("error opening database")
}
```

```
if sqlite3_exec(databasedb, "create table if not exists test table (id integer primary key autoincrement, name2 text)", nil, nil, nil)!=
SQLITE_OK {
    let errmsg = String.fromCString(sqlite3_errmsg(db))
    println("error creating new table: \(errmsg)")

}
```

```
var statement: DBPointer = nil
if sqlite3_prepare_v2(databasedb, "insert into test (name) values (?)", -1, &statement, nil) != SQLITE_OK
{
    let errmsg = String.fromCString(sqlite3_errmsg(databasedb))
    println("error preparing insert: \(errmsg)")
}
if sqlite3_bind_text(statement, 1, "data", -1, SQLITE_TRANSIENT) != SQLITE_OK
{
    let errmsg = String.fromCString(sqlite3_errmsg(databasedb))
    println("failure binding record data: \(errmsg)")
}
if sqlite3_step(statement) != SQLITE_DONE
{
    let errmsg = String.fromCString(sqlite3_errmsg(databasedb))
    println("failure inserting record data: \(errmsg)")
}
```

```
let SQLITE_STATIC = sqlite3_destructor_type(DBPointer(bitPattern: 0))
let SQLITE_TRANSIENT = sqlite3_destructor_type(DBPointer(bitPattern: -1))
```

```
var statement: DBPointer = nil
if sqlite3_prepare_v2(databasedb, "insert into testtable (name) values (?)", -1, &statement, nil) != SQLITE_OK
{
   let errmsg = String.fromCString(sqlite3_errmsg(db))
   println("error preparing the insert: \(errmsg)")
}
if sqlite3_bind_text(statement, 1, "Bind1", -1, SQLITE_TRANSIENT) != SQLITE_OK
{
   let errmsg = String.fromCString(sqlite3_errmsg(db))
   println("failure binding this statement: \(errmsg)")
}
if sqlite3_step(statement) != SQLITE_DONE
{
   let errmsg = String.fromCString(sqlite3_errmsg(db))
   println("failure on inserting data : \(errmsg)")

}
```

```
#define SQLITE_STATIC((sqlite3_destructor_type)0)
#define SQLITE_TRANSIENT((sqlite3_destructor_type)-1)4
```

```
if sqlite3_reset(statement) != SQLITE_OK
{
   let errmsg = String.fromCString(sqlite3_errmsg(databasedb))
   println("error resetting prepared statement: \(errmsg)")
}
if sqlite3_bind_null(statement, 1) != SQLITE_OK

{
   let errmsg = String.fromCString(sqlite3_errmsg(databasedb))
   println("failure binding the null value: \(errmsg)")
}
```

```
if sqlite3_step(statement) != SQLITE_DONE
{
    let errmsg = String.fromCString(sqlite3_errmsg(databasedb))
    println("failure inserting null: \(errmsg)")
}
```

```
if sqlite3_close(databasedb) != SQLITE_OK
{
    println("error closing the database")
}databasedb = nil
```

Chapter 6: Using Swift with iOS and SQLite

```
class Mortgage_data: NSObject {
    var mortgage_rollno: String = String()
    var mortgage_name: String = String()
}
```

```
class func copyFile(fileName:NSString){
    var database_path:NSString=getPath(fileName)
    var MortgageManager=NSFileManager.defaultManager()
    if !MortageManager.fileExistsAtPath(database_path){
        var fromthePath:NSString= NSBundle.mainBundle().resourcePath.stringByAppendingPathComponent(fileName)
        MortgageManager.copyItemAtPath(fromPath,toPath:database_path,error: nil)
    }
}
```

let mortgage_instance=**ModelManager**()

```
var database:FMDatabase?= nil
class var instance:ModelManager{
    mortgage_instance.database=FMDatabase(path:Util.getPath("Mortgagedata.sqlite"))
    var Mydatapath=Util.getPath("Mortgagedata.sqlite")
    println("The Current Path is> : \(Mydatapath)")
    return mortgage_instance
}
```

```
func addMortageData(Mortage_Data:Mortgage_data)-> Bool {
    mortgage_instance.database!.open()
    let Mortage_Inserted= Mortgage_instance.database!.executeUpdate("INSERT INTO Mortgage_Data (mortgage_rollno,
mortgage_name) VALUES (?, ?)",withArgumentsInArray:[Mortgage_data.mortgage_rollno,Mortgage_data.mortgage_name])
mortgage_instance.database!.close()
    return isInserted
}
```

```
@IBAction func btnInsertClicked(sender: AnyObject) {
    var mortgage_data: Mortgage_data = Mortgage_data()
    Mortgage_data.mortgage_rollno = tmp_ Mortgage_data.mortgage_rollno.text
    Mortgage_data.studentName = Mortgage_data.mortgage_name.text
    var Mortgage_insert = ModelManager.instance.MortgageData(Mortgage)
    if Mortgage_insert {
        Util.invokeAlertMethod("", MortgageBody: " Data Inserted ", delegate:nil)
    } else {

        Util.invokeAlertMethod("", MortgageBody: "Error in inserting data", delegate: nil)
    }
    Mortgage_data.tmp_rollno.text = ""
    Mortgage_data.tmp_name.text = ""
    Mortgage_data.tmp_rollno =.becomeFirstResponder()
}
```

```
func Mortgage_Updatedata(Mortage_data: Mortgage_Data) -> Bool {
    ModelManager.instance.database!.open()
    let Mortgage_Info_Updated {= sharedInstance.database!.executeUpdate("UPDATE Mortgage_data SET Mortgage_name=?
WHERE Mortgage_rollno=?",withArgumentsInArray:[Mortgage_data.Name, Mortgage_data.rollno])
    Mortgageinstance.database!.close()
    return Mortgage_Info_Updated
}
```

```
@IBActionfuncMortgage_UpdateClicked(sender:AnyObject){
    var Mortgage_data:Mortgage_data=Mortgage_data()
    Mortgage_data.mortgage_rollno =tmp_mortgage_rollno.text
    Mortgage_data.mortgage_name=tmp_mortgage_name.text
    var Mortgage_Data:Mortgage_data=Mortgage_data()
    var tmp_roll_no: String ="mortgage_rollno"
    var tmp_name: String ="mortgage_name"
    var Mortgage_Info_Updated = ModelManager.instance.updateStudentData(Mortgage_data)
    if Mortgage_Info_Updated {
        Util.invokeAlertMethod("", strBody: "Mortgage Record has been updated", delegate: nil)
    } else {
        Util.invokeAlertMethod("", strBody: "Error in updating the Mortgage record", delegate: nil)
    }
    Mortgage_data.tmp_rollno.text=""
    Mortgage_data.tmp_name.text=""
    Mortgage_data.tmp_rollno=.becomeFirstResponder()
}
```

```
func deleteStudentData(Mortgage_data:Mortgage_Data)-> Bool {
    Mortgageinstance.database!.open()
    let Mortgage_isDeleted_var= Mortgageinstance.database!.executeUpdate("DELETE FROM Mortgage_data WHERE
Mortgage_data_rollno=?", withArgumentsInArray:[Mortgagedata.name])
    sharedInstance.database!.close()
    return Mortgage_isDeleted
}
```

```
@IBAction func btnDeleteClicked(sender:AnyObject){
    var Mortgage_data:Mortgage_data=Mortgage_data()
    Mortgage_data.mortgage_rollno =tmp_mortgage_rollno.text
    Mortgage_data.mortgage_name=tmp_mortgage_name.text
    var isDeleted_var=ModelManager.instance.deleteStudentData(studentInfo)
    if isDeleted_var{
        Util.invokeAlertMethod("",strBody:"Record Deleted", delegate: nil)
        }else{
        Util.invokeAlertMethod("",strBody:"Error- On Deleting Record", delegate: nil)
    }
    Mortgage_data.tmp_rollno.text=""
    Mortgage_data.tmp_name.text=""
    Mortgage_data.tmp_rollno=.becomeFirstResponder()
}
```

```
func SelectMortgageData (){
    Mortgageinstance.database!.open()
    var mortgage_resultSet:FMResultSet!= Mortgageinstance.database!.executeQuery("SELECT * FROM
Mortgage_data",withArgumentsInArray: nil)
    var tmp_roll_no: String ="mortgage_rollno"
    var tmp_name: String ="mortgage_name"
    if resultSet{
        while mortgage_resultSet.next(){
            println("roll no data is : \(mortgage_resultSet. stringForColumn(tmp_roll_no))")
            println("name data is : \(mortgage_resultSet.stringForColumn(tmp_name))"
        }
    }Mortgageinstance.database!.close()
}
```

```
@IBAction func btnDisplayRecordClicked(sender:AnyObject){
    ModelManager.instance.MortgageData()
}
```

```
@IBAction func SaveMortgageData(sender: AnyObject) {
    let Mortgage_data_save = FMDatabase(path: Database_path as String)
    if Mortgage_data_save.open() {
        let insertdata = "INSERT INTO Mortgage_data (mortgage_rollno, mortgage_name) VALUES
('\(tmp_mortgage_rollno.text)', '\(tmp_mortgage_name.text)')"
        let mortgage_result = Mortgage_data_save.executeUpdate(insertdata, withArgumentsInArray: nil)
        if !mortgage_result {
            Msg_info.text = "Error inserting Mortgage Details"
            println("Error: \(Mortgage_data_save.Mortgage_ErrorMessage())")
        } else {
        Msg_info.text = "Mortgage Details inserted to system"
        tmp_mortgage_rollno.text = ""
        tmp_mortgage_name.text = ""
        }
    } else {
        println("Error: \(Mortgage_data_save.Mortgage_ErrorMessage())")
    }
}
```

Chapter 7: iOS Development with PhoneGap and HTML5

```
$cd ~/Documents
$cordova create hello com.example.helloHelloWorld
```

```
$ cd hello
$ cordova platform add ios
```

```
$ cordova platforms ls

Installed platforms: ios 3.7.0
Available platforms: amazon-fireos, android, blackberry10, browser, firefoxos
```

```
$ sudo npm install –g phonegap
```

```
<!DOCTYPE HTML>
<html>
   <head>
      <script type="text/javascript">
      var db = openDatabase('testdb', '1.0', 'Test DB', 2 * 1024 *1024);
      var msg;
      db.transaction(function (tx) {
         tx.executeSql('CREATE TABLE IF NOT EXISTS BLOGS (id unique, log)');
         tx.executeSql('INSERT INTO BLOGS (id, log) VALUES (1, "This is test blog 1")');
         tx.executeSql('INSERT INTO BLOGS (id, log) VALUES (2, "This is test blog 2")');
         msg = '<p>Blog message created and row inserted.</p>';
         document.querySelector('#status').innerHTML = msg;
      });
      db.transaction(function (tx) {
         tx.executeSql('SELECT * FROM BLOGS', [], function (tx, results) {
            var len = results.rows.length, i;
```

```
                msg = "<p>Found rows: " + len + "</p>";
                document.querySelector('#status').innerHTML += msg;
                for (i = 0; i < len; i++){
                    msg = "<p><b>" + results.rows.item(i).log + "</b></p>";
                    document.querySelector('#status').innerHTML += msg;
                }
            }, null);
        });
        </script>
    </head>
    <body>
        <div id="status" name="status">Status Message</div>
    </body>
</html>
```

Chapter 8: More Features and Advances in SQLite

```
<gap:plugin name="com.phonegap.plugins.example" version="0.3.3" source="plugins.cordova.io" />
// Wait for Cordova html5 plugin to load document
    addEventListener("deviceready", onDeviceReady, false); var db;
    function onDeviceReady() { db1 = window.sqlitePlugin.openDatabase({name: "DB"}); }
```

```
// Wait for Cordova to load document.addEventListener("deviceready", onDeviceReady, false);
// Cordova is ready function onDeviceReady()
{
    var db = window.sqlitePlugin.openDatabase({name: "DB"
}
); // ... }
```

```
Db.transaction(function(Tx1) {
    Tx1.executeSql("Create table if not exists" + " test(id integer primary key asc, newcolumn text, []);
});
}
```

```
sqlite> CREATE TABLE one (y, z);
sqlite> CREATE TABLE two (a, b);
sqlite> EXPLAIN QUERY PLAN SELECT A FROM one JOIN two ON one.z = two.b WHERE y = 30;
```

```
func UnloadData() throws { }
func zTest() {
    do {
        try UnloadData()
    } catch {
        print(error)
    }
}
```