

Adobe AIR SQLite: An Optimization Conversation

H. Paul Robertson
ActionScript Developer/Writer
Adobe Systems, Inc.

360Flex San Jose
August 19, 2008



Copyright 2008 Adobe Systems Incorporated. All rights reserved.



Welcome!

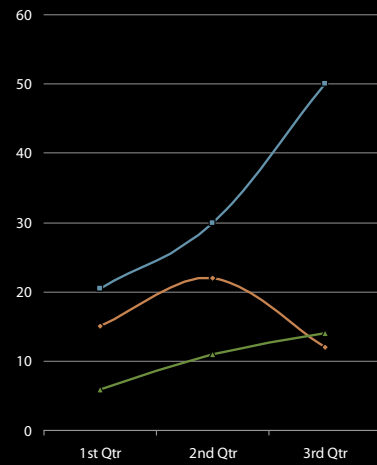
- Questions – please ask!
- My perspective
- Your experience – are you ready to share?

Background and concepts

- Synchronous/asynchronous
- Transaction
- Index
- “optimization”

Performance

(application performance)



Copyright 2008 Adobe Systems Incorporated. All rights reserved.



Application Performance

Actual performance

vs.

Perceived performance

(is there really a difference?)

Application Performance

Actual performance

Copyright 2008 Adobe Systems Incorporated. All rights reserved.



Application Performance > Actual performance

Write faster SQL

- Favor JOIN over subquery
- Avoid LIKE – especially LIKE ('%blah%')
- Avoid IN – use AND/OR
- Avoid JOINing to the same table multiple times (careful with views)
- Avoid needless lookups
 - Qualify table (and other object) names with database name (usually “main”)
 - Explicitly specify column names in SELECT and INSERT

Application Performance > Actual performance

Use (and re-use) prepared statements

Application Performance > Actual performance

Use (and re-use) prepared statements (bad example)

```
public function execute(conn:SQLConnection, firstName:String, lastName:String, phoneNumber):Number
{
    var stmt:SQLStatement = new SQLStatement();
    stmt.sqlConnection = conn;
    var sql:String = "" +
        "INSERT INTO customers (first_name, last_name) " +
        "VALUES (:firstName, :lastName)";
    stmt.text = sql;
    stmt.parameters[":firstName"] = firstName;
    stmt.parameters[":lastName"] = lastName;
    stmt.execute();

    var custId:Number = addCustomer.getResult().lastInsertRowID;
    sql = "" +
        "INSERT INTO phone_numbers (cust_id, phone_number) " +
        "VALUES (:custId, :phoneNumber)";
    stmt.text = sql;
    stmt.parameters[":custId"] = custId;
    stmt.parameters[":phoneNumber"] = phoneNumber;
    stmt.execute();

    return custId;
}
```

Application Performance > Actual performance

Use (and re-use) prepared statements (good example)

```
private var _addCustomer:SQLStatement;
private var _addPhone:SQLStatement;

public function AddCustomerCommand(conn:SQLConnection)
{
    _addCustomer = new SQLStatement();
    _addCustomer.sqlConnection = conn;
    var addCustomerSQL:String = "" +
        "INSERT INTO customers (first_name, last_name) " +
        "VALUES (:firstName, :lastName)";
    _addCustomer.text = addCustomerSQL;

    _addPhone = new SQLStatement();
    _addPhone.sqlConnection = conn;
    var addPhoneSQL:String = "" +
        "INSERT INTO phone_numbers (cust_id, phone_number) " +
        "VALUES (:custId, :phoneNumber)";
    _addPhone.text = addPhoneSQL;
}

public function execute(firstName:String, lastName:String, phoneNumber:String):Number
{
    _addCustomer.parameters[":firstName"] = firstName;
    _addCustomer.parameters[":lastName"] = lastName;
    _addCustomer.execute();

    var custId:Number = _addCustomer.getResult().lastInsertRowID;
    _addPhone.parameters[":custId"] = custId;
    _addPhone.parameters[":phoneNumber"] = phoneNumber;
    _addPhone.execute();

    return custId;
}
```

Copyright 2008 Adobe Systems Incorporated. All rights reserved.



Application Performance > Actual performance

Use transactions for batch INSERT/UPDATE/DELETE operations

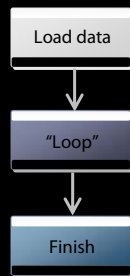
Application Performance > Actual performance

Use transactions for batch INSERT/UPDATE/DELETE operations

- Example: Inserting 85489 rows of data from a CSV file
 - No explicit transaction: 751510 ms (12.5 minutes)
 - Explicit transaction: 15662 ms (16 seconds – about 48x faster!)

Application Performance > Actual performance

Use transactions for batch INSERT/UPDATE/DELETE operations (bad example)

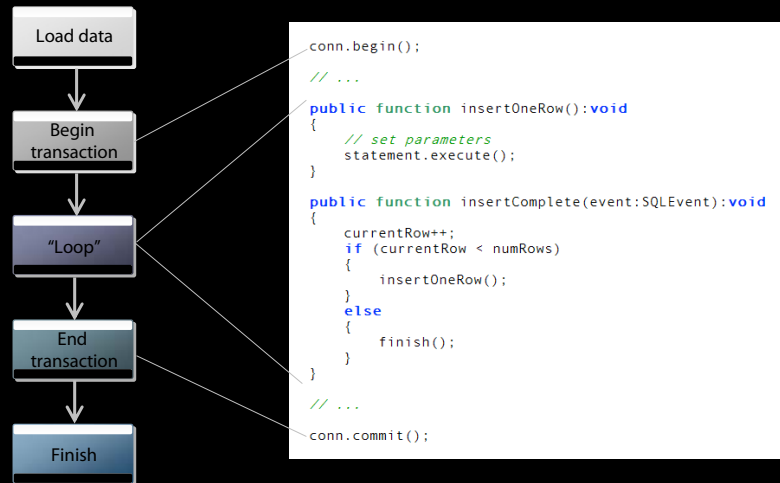


```
public function insertOneRow():void
{
    // set parameters
    statement.execute();
}

public function insertComplete(event:SQLEvent):void
{
    currentRow++;
    if (currentRow < numRows)
    {
        insertOneRow();
    }
    else
    {
        finish();
    }
}
```

Application Performance > Actual performance

Use transactions for batch INSERT/UPDATE/DELETE operations (good example)



Application Performance > Actual performance

Use indexes

- ...but use them wisely...
 - Index columns that are used in WHERE clause
 - Used together – index together
- ...and/or use analyze()

Used together – index together:

If a subset of the columns of a table are commonly used together in a WHERE clause and/or retrieved together, combine them in a single index.

Application Performance > Actual performance

Create table structure before adding data

Copyright 2008 Adobe Systems Incorporated. All rights reserved.



Create your table structure (tables, views, indexes, and triggers) before adding data to the database.

Application Performance > Actual performance

Create table structure before adding data

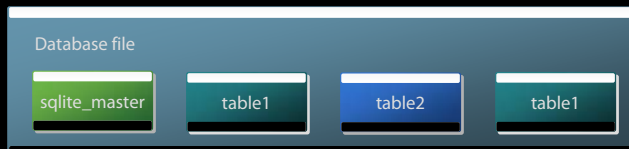
- On-disk, database file uses “pages”



Application Performance > Actual performance

Create table structure before adding data

- On-disk, database file uses “pages”



Copyright 2008 Adobe Systems Incorporated. All rights reserved.



Once the “table1” page fills up, a new page is created and used for additional table1 data

Application Performance > Actual performance

Create table structure before adding data

- On-disk, database file uses "pages"



Copyright 2008 Adobe Systems Incorporated. All rights reserved.



Like other tables, if you make enough schema changes that the `sqlite_master` page fills up, a new page is created for that table. This can negatively impact the startup time when connecting to a database.

Application Performance > Actual performance

Create table structure before adding data

- On-disk, database file uses "pages"

...or use `SQLConnection.compact()`



- but be careful with `autoCompact`!

Copyright 2008 Adobe Systems Incorporated. All rights reserved.



Calling `SQLConnection.compact()` causes the tables to be rearranged in order. Note that this can take a notable amount of time, depending on the amount of data in the database.

Autocompact sounds like it does this, but it only does it partway. It doesn't rearrange the tables – it only removes pages that become completely empty (as data is deleted), shifts them to the end of the database, and deletes them. This keeps the db file size smaller, but requires more processing at the end of each commit.

Application Performance

Perceived performance

Copyright 2008 Adobe Systems Incorporated. All rights reserved.



Application Performance > Perceived performance

Bottleneck #1:

- “Batch” statements
 - Running multiple statements in sequence, such as a bulk INSERT
- Solution:
 - Use asynchronous execution mode

Application Performance > Perceived performance

Bottleneck #2:

- Large SELECT result set
- Solution:
 - Break it into chunks using execute() with prefetch parameter, and next()

```
statement.execute(20);  
  
// ...  
private function selectResult(event:SQLEvent):void  
{  
    var result:SQLResult = statement.getResult();  
    var rows:Array = result.data;  
  
    if (!result.complete)  
    {  
        statement.next(20);  
    }  
  
    // process this set of results  
}
```

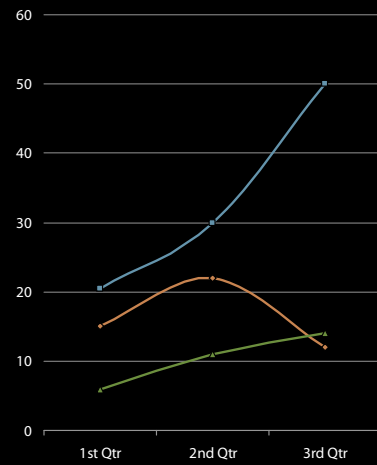
Application Performance > Perceived performance

Bottleneck #3:

- Fast query in the queue behind slow query
- Solution:
 - Use multiple SqlConnection objects (plan carefully and be prepared for errors)

Productivity

(developer productivity)



Copyright 2008 Adobe Systems Incorporated. All rights reserved.



Developer Productivity

Tools

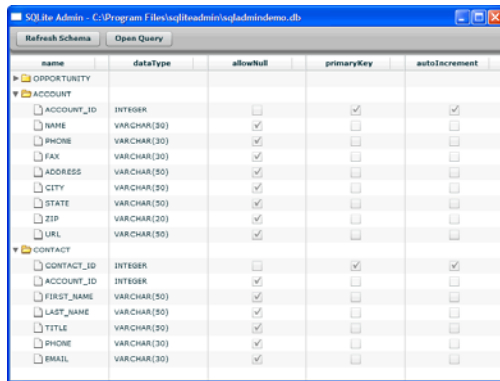
Copyright 2008 Adobe Systems Incorporated. All rights reserved.



Developer Productivity > Tools

Creating/modifying database structure

- Christophe Coenraets' "SQLite Admin"



The screenshot shows the SQLite Admin application window with the title bar "SQLite Admin - C:\Program Files\sqliteadmin\sqliteadmin.exe". The window has two tabs: "Refresh Schema" and "Open Query". The main area displays a tree view on the left with "OPPORTUNITY" expanded, showing "ACCOUNT" and "CONTACT" sub-entities. The "ACCOUNT" entity is further expanded, showing fields: ACCOUNT_ID, NAME, PHONE, FAX, ADDRESS, CITY, STATE, ZIP, and URL. The "CONTACT" entity is also expanded, showing fields: CONTACT_ID, ACCOUNT_ID, FIRST_NAME, LAST_NAME, TITLE, PHONE, and EMAIL. The main table lists these fields with their data types, whether they allow null values, and if they are primary keys or auto-incrementing.

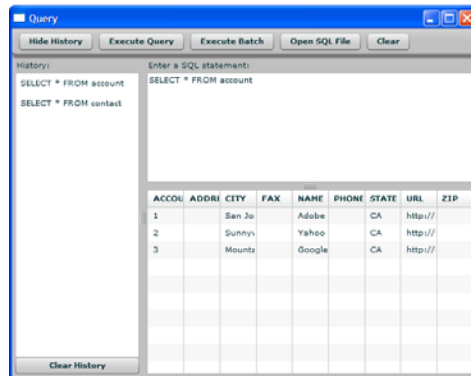
name	dataType	allowNull	primaryKey	autoIncrement
OPPORTUNITY				
ACCOUNT				
ACCOUNT_ID	INTEGER	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
NAME	VARCHAR(50)	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
PHONE	VARCHAR(30)	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
FAX	VARCHAR(30)	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
ADDRESS	VARCHAR(50)	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
CITY	VARCHAR(50)	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
STATE	VARCHAR(50)	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
ZIP	VARCHAR(20)	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
URL	VARCHAR(50)	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
CONTACT				
CONTACT_ID	INTEGER	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
ACCOUNT_ID	INTEGER	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
FIRST_NAME	VARCHAR(50)	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
LAST_NAME	VARCHAR(50)	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
TITLE	VARCHAR(20)	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
PHONE	VARCHAR(30)	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
EMAIL	VARCHAR(30)	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

<http://coenraets.org/blog/2008/02/air-based-sqlite-admin-updated-for-beta-3/>

Developer Productivity > Tools

Prototyping/testing SQL statements #1

- Christophe Coenraets' "SQLite Admin" (again)

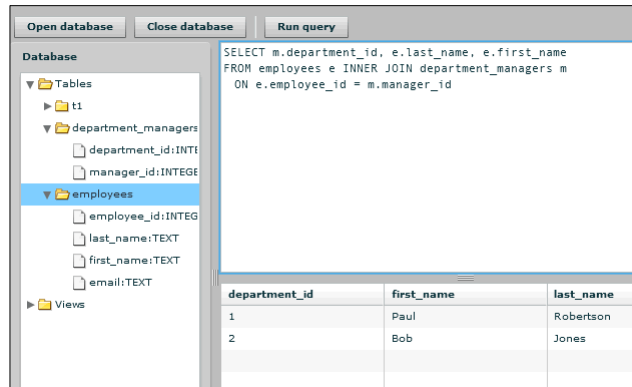


<http://coenraets.org/blog/2008/02/air-based-sqlite-admin-updated-for-beta-3/>

Developer Productivity > Tools

Prototyping/testing SQL statements #2

- My tools (in progress =)



<http://probertson.com/projects/doppler-air-sql-admin-tool/>

Developer Productivity

Application architecture

Copyright 2008 Adobe Systems Incorporated. All rights reserved.



Developer Productivity > Application architecture

Goals:

- Minimize duplicate/boilerplate code
- Reduce number of event handlers needed for asynchronous execution
 - Data binding
 - Abstraction layer
- Easily reuse `SQLStatement` objects
- Easily “chain” dependent database operations
- Queue up multiple instances of the same statement
- Execute multiple independent statements “simultaneously”

Solution #1: SQLite MXML wrapper classes (Peter Elst)

```
<sql:SQLite id="contacts_db"
  file="contacts.db"
  open="allcontacts_query.execute()" />
<sql:Query id="allcontacts_query"
  connection="{contacts_db.connection}"
  sql="SELECT * FROM contacts"
  result="contacts_dg.dataProvider = allcontacts_query.data" />

<mx:DataGrid id="contacts_dg" left="0" right="0" top="0" bottom="0"/>
```

<http://www.peterelst.com/blog/2008/04/07/introduction-to-sqlite-in-adobe-air/>

Solution #2: Data access layer (Brandon Ellis)

```
<mx:Script>
  <![CDATA[
    private var da:DataAccess;

    private function init():void
    {
      var databasePath:String = /* your db path */;
      da = new DataAccess(databasePath);
      da.openConnection("SELECT * FROM friends");
    }

    private function dbSelect():void
    {
      var sql:String = "SELECT * FROM friends";
      da.DataAccessSelect(sql);
    }

    // and similar methods for INSERT, UPDATE, etc.

    da.DataAccessInsert(sql);
    da.DataAccessUpdate(sql);
  ]]>
</mx:Script>

<mx:DataGrid id="friends_dg" dataProvider="{da.dbResult}" ... />
```

<http://www.brandonellis.org/?p=49>

Solution #3: asqlib SQL statement generator (Miran Loncaric)

```
var vo:PersonVO = new PersonVO();  
  
var create:SQLCreateTable = new SQLCreateTable(vo);  
trace(create.statement);  
  
var select:SQLSelect = new SQLSelect(vo);  
select.conditions = conds;  
var cond:SQLCondition = new SQLCondition("id", "=", "1");  
var conds:SQLConditions = new SQLConditions(cond);  
select.conditions = conds;  
  
var insert:SQLInsert = new SQLInsert(vo);  
  
var update:SQLUpdate = new SQLUpdate(vo);  
update.conditions = conds;  
trace(insert.statement);  
  
var delete:SQLDelete = new SQLDelete(vo);
```

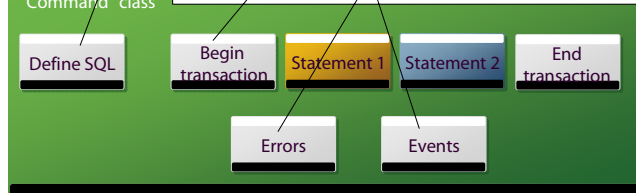
```
[Bindable]  
[Table(name="user")]  
public class PersonVO  
{  
    [Id]  
    public var id:int;  
    public var first_name:String;  
    public var last_name:String;  
    public var age:uint;  
    public var isMarried:Boolean;  
}
```

<http://code.google.com/p/asqlib/>

Solution #4: "command" classes (me)

```
var addCmd:AddCommand = new AddCommand();  
// ...  
var contactToAdd:Contact = new Contact("Bob", "Jones", "925-555-0000");  
addCmd.contact = contactToAdd;  
  
addCmd.addEventListener(Event.COMPLETE, completeHandler);  
addCmd.addEventListener(SQLErrorEvent.ERROR, errorHandler);  
addCmd.execute();  
  
private function completeHandler(event:Event):void {}  
private function errorHandler(event:SQLErrorEvent):void {}
```

"Command" class



<http://probertson.com/projects/addressbook/>

Developer Productivity > Application architecture

Solution #5: AIR ActiveRecord (Jacob Wright)

<http://code.google.com/p/air-activerecord/>

<http://jacwright.com/blog/79/air-activerecord-is-open-source/>



Solution #6: Cairngorm services (Eric Feminella)

- AIRServiceLocator
- SQLService
- ISQLResponder
- SQLStatementHelper

- Plus code generation

<http://www.ericfeminella.com/blog/2008/06/22/air-cairngorm-20/>

Solution #7: Pools (Daniel Rinehart)

```
private var _connectionPool:ConnectionPool;
private var _statementPool:StatementPool;
private static const LOAD_CONTACT_SQL:String = "" +
    "SELECT last_name, first_name, email " +
    "FROM contacts " +
    "WHERE contact_id = :id";

// ...

_connectionPool = new ConnectionPool(databaseFile);
_statementPool = new StatementPool(_connectionPool);

// ...

public function loadContact(contactId:uint):void
{
    _statementPool.execute(LOAD_CONTACT_SQL, {id: contactId}, resultHandler);
}

private function resultHandler(result:SQLResult):void {}
```

http://www.adobe.com/devnet/air/flex/articles/air_sql_operations.html

Thanks!

Thanks for listening and sharing!

H. Paul Robertson

<http://probertson.com/>

Copyright 2008 Adobe Systems Incorporated. All rights reserved.



Better by Adobe.™

Copyright 2008 Adobe Systems Incorporated. All rights reserved.

