# Advanced Database Coding

Peter Haworth
www.lcsql.com
pete@lcsql.com

Materials are at www.lcsql.com/free-stuff.html

# Advanced Database Coding

## *Principal 1*

The performance and effectiveness of your application is directly related to the quality of your database design.

# Advanced Database Coding

## *Principal 2*

Your database is an extension of your code base.  Make the maximum use of SQL Data Design features to remove code from your application.

# Advanced Database Coding

## *Topics*

1. Opening A Database
2. Selecting Data
3. Changing Data
4. Using schema validation features
5. How to handle BLOB columns
6. Combining Databases and datagrids
7. Handling Unicode (Livecode v7)
8. Using Views

# Advanced Database Coding
## *Opening Your Database*

revOpenDatabase (SQLite):

- Creates a file if it doesn't exist
- Opens any file, not just an SQLite database file
- Foreign keys not enforced by default
- "binary" option (Livecode 6.6 onwards)

# Advanced Database Coding
## *Opening Your Database*

ATTACH command:

- Opens a database on the same connection created by revOpenDatabase
- Uses less memory than opening a second connection
- Allows JOIN statements across databases

```
revExecuteSQL gDBID,”ATTACH ‘/databases/addressdb.s3db’ AS Addressdb
SELECT City FROM Addressdb.Addresses
```

# Advanced Database Coding
## *Selecting Data*

*variableslist | arrayname*

- Available on many database functions
- Removes need to escape quotes
- Protects against SQL injection attacks
- Example:

```
function getCustomerName pcustid
   constant kSelect=”SELECT CustName FROM Customers WHERE CustID=:1”
   return revDataFromQuery(,,gDBID,kSelect,”pcustid”)
end getCustomerName
```

# Advanced Database Coding
## *Selecting Data*

- Don’t use * in SELECT statements; too reliant on database structure
- Increase code clarity by putting long/complex SELECT statements into your database as views.

# Advanced Database Coding
## *Selecting Data*

revDataFromQuery

- Only use for statements that return data (SELECT, PRAGMA)
- Don’t use if selection can include:
  - NULLs
  - BLOB Data
- Use alternative row and column delimiters if data can include return or tab characters

# Advanced Database Coding
## *Selecting Data*

revQueryDatabase

- Only use for statements that return data (SELECT, PRAGMA)
- Returns pointer to data (cursor) so less memory used
- Cursor manipulation functions return data.
- Cursor uses zero based record numbering.

# Advanced Database Coding
## *Changing The Database*

### Locking

For any series of logically related database change statements, issue:

- `BEGIN` before first one (revExecuteSQL)
- `COMMIT` after last one (revCommitDatabase)
- `ROLLBACK` if errors detected (revRollbackDatabase)

Protects the integrity of your database and greatly improves performance.

# Advanced Database Coding
## *Changing The Database*

revExecuteSQL

Use to issue SQL statements that do not return data from the database:

INSERT

UPDATE

DELETE.

# Advanced Database Coding
## *Schema Validation Features*

Define as much error checking as possible in your database schema

- Removes code from your scripts
- Ensures data integrity no matter what programs access the database
- No wasted application CPU cycles
- Available features:

```
NOT NULL

UNIQUE

CHECK

REFERENCES/FOREIGN KEY
```

# Advanced Database Coding
## *Schema Validation Features*

NOT NULL

```
..., CustomerName TEXT NOT NULL,...
```

- CustomerName column cannot have a NULL value. *Note: NULL is not the same as empty; this error occurs if you supply a NULL as the value for a column or don't name the column in your INSERT/UPDATE statement*
- Error message format:

  `NOT NULL constraint failed: <column name>`

- Easy to translate into user error message

# Advanced Database Coding
## *Schema Validation Features*

UNIQUE

- Column(s) cannot have the same value in any other row in the table

```
..., SocialSecurityNumber TEXT UNIQUE,...
...UNIQUE InvoiceNumber,InvoiceLineNumber
```

- Use `COLLATE NOCASE` to make the UNIQUE check case insensitive
- Error message format:

```
UNIQUE constraint failed: <columnnames>
```

- Easy to translate into user error message

# Advanced Database Coding
## *Schema Validation Features*

CHECK

- Use any SQL statement to validate the column value
- Define a constraint name for each CHECK statement to easily identify it in the SQL error message
- Value must be in a numeric range:

```
... CONSTRAINT InvalidSequenceNumber CHECK (SequenceNumber BETWEEN 1 AND 9999)
```

- Error Message Format:

```
CHECK constraint failed: <constraintname>
```

# Advanced Database Coding
## *Schema Validation Features*

REFERENCES/FOREIGN KEY

- Value must exist in another table

```
...InvoiceNumber REFERENCES InvoiceHeader.InvoiceNumber
... FOREIGN KEY AlbumID (Albums.AlbumID)
```

- Use option menu/scrolling list to present user with valid choices
- NOTE: Foreign keys are NOT enabled by default.

# Advanced Database Coding
## *BLOB Data*

BLOB = Binary Large OBject, a way to store binary data in your database.

*Prior to version 6.6, Livecode encoded/decoded binary data when inserting/selecting from database. This data will not be accessible by anything other than Livecode programs.*

*V6.6 includes an option to not do this encoding with revOpenDatabase.*

# Advanced Database Coding
## *BLOB Data*

- Use revQueryDatabase to SELECT BLOB columns
- Use cursor manipulation functions to put BLOB data into variables
- Use the *varslist*|*arrayname* parameter of revExecuteSQL to INSERT/UPDATE BLOB columns. The variable names or array keys must be preceded by "*b":

```
revExecuteSQL gDBID,tSelect,"*btVar"
put tBinaryData into tArray[*b1]
revexecuteSQL gDBID,tSelect,"tArray"
```

# Advanced Database Coding
## *Databases and Datagrids*

dgText

Use to load tab delimited data from revDataFromQuery

```
put revDataFromQuery(,,gDBID,tSelect) into tData
if not (tData begins with “revdberr”) then
   set the dgText of group “myDatagrid” to tData
else
   ....
end if
```

# Advanced Database Coding
## *Databases and Datagrids*

dgData

Use to load an array keyed by datagrid line number/column name into a datagrid.

Create an array from a database cursor:

```
put revQueryDatabase(gDBID,tSelect) into tCursor
if tCursor is an integer then
   set the dgData of group “myDatagrid” to lcsql_CreateArrayFromCursor(tCursor)
else
   ....
end if
```

# Advanced Database Coding
## *Databases and Datagrids*

dgNumberOfRecords

Improves performance when loading large amounts of data into a datagrid.

*Many of the datagrid features (dgText, dgData, column sorting, etc) are not available when using this method and must be handled by your scripts.*

1. Use revDataFromQuery to get the primary key values of the required rows and store as a custom property of the datagrid.
2. Set dgNumberofRecords to the number of rows selected.
3. Write a handler named GetDataForLine in the datagrid script. Called by the datagrid when it needs another row of data to display.

# Advanced Database Coding
## *Unicode (Livecode v7+)*

- Most SQL databases include the ability to set the encoding when the database is created and to discover the encoding programmatically.  For example “PRAGMA encoding” for SQLite.

- Encoding is not changeable once data has been added to the database

# Advanced Database Coding
## *Unicode (Livecode v7+)*

- Use textDecode when selecting data

```
put textDecode(revDataFromQuery(,,gDBID,tSelect),”UTF8”) into tData
OR
put revQueryDatabase(gDBID,tSelect) into tCursor
put textDecode(revDatabaseColumnNamed(tCursor,”Name”),”UTF8”) into tName
```

- Use textEncode with INSERT/UPDATE

```
revExecuteSQL gDBID,”UPDATE Customers SET Custname=” & makeString (textEncode(field “Name”,”UTF8”))
```

- *varslist* | *arrayname* values must be textEncoded

```
put textEncode(field “Name”,”UTF8”) into tName
revExecuteSQL gDBID,”UPDATE Customers SET CustName=:1”,”tName”
```

# Advanced Database Coding
## *Using Views*

Use views to keep your scripts tidy by defining long, complex SELECT statements in your database.

- To create view:

```
CREATE VIEW vInvoiceData AS SELECT CustomerName,CustomerAddress,CustomerCity,
CustomerState,CustomerZIP,InvoiceHeaders.InvoiceNumber,strftime('%m/%d/%Y',
InvoiceHeaders.InvoiceDate),InvoiceLines.LineNumber,InvoiceLines.LineQty,Products.
ProductName,InvoiceLines.LinePrice, sum(InvoiceLines.LineValue)AS InvTotal FROM
Customers LEFT JOIN InvoiceHeaders ON InvoiceHeaders.CustomerID=Customers.
CustomerID LEFT JOIN InvoiceLines ON InvoiceLines.InvoiceNumber=InvoiceHeader.
InvoiceNumber LEFT JOIN Products ON Products.ProductID=InvoiceLines.ProductID ORDER
BY Customers.CustomerName,InvoiceHeader.InvoiceDate,InvoiceLines.LineNumber
```

- To use view in your script:

```
put revDataFromQuery(,,gDBID,"SELECT * FROM vInvoiceData WHERE InvoiceNumber=35")
```