Tutorial: SQLite and Processing Part II Files

--D. Thiebaut (talk) 17:27, 31 July 2013 (EDT)

Contents

sqlite-jdbc-3.7.2.jar

- The sqlite-jdbc-3.7.2.jar file is available here (http://cs.smith.edu/dftwiki/images/sqlite-jdbc-3.7.2.jar) for convenience.

SQL.java

```
//package de.bezier.data.sql;
import processing.core.*;
//import de.bezier.data.sql.mapper.*;
import mapper.*
import java.io.*;
import java.sql.*;
import java.lang.reflect.*;
import java.util.HashMap;
import java.util.Map;
import java.util.ArrayList;

/**
 *              <h1>SQL library for Processing 2+</h1>
 *
 *              Since v 0.2.0 it has some ORM like features, see
 *              <ul>
 *                      <li><a href="#setFromRow(java.lang.Object)">setFromRow(Object)</a></li>
 *                      <li><a href="#saveToDatabase(java.lang.Object)">saveToDatabase(Object)</a></li>
 *                      <li><a href="#insertUpdateInDatabase(java.lang.String, java.lang.String[], java.lang.Object[])">insertUpdateIntoDatabase(String,Objec
 *              </ul>
 *
 *              Links:<ul>
 *              <li>http://www.mysql.com/products/connector/j/</li>
 *              <li>http://java.sun.com/products/jdbc/</li>
 *              <li>http://www.toxi.co.uk/blog/2007/07/using-javadb-and-db4o-in-processing.htm</li>
 *              <li>http://www.tom-carden.co.uk/2007/07/30/a-quick-note-on-using-sqlite-in-processing/</li>
 *              </ul>
 *
 *              @author                 Florian Jenett - mail@florianjenett.de
 *
 *              created:                07.05.2005 - 12:46 Uhr
 *              modified:               fjenett 20121217
 */

abstract public class SQL
{
        PApplet papplet;

        public String server;
        public String database;
        public String url;
        public String user;
        protected String pass;
        public String driver = "";
        public String type = "";

        private int driverMinorVersion = -1;
        private int driverMajorVersion = -1;

        public java.sql.Connection connection;
        public String previousQuery;

        public java.sql.Statement statement;
        public java.sql.ResultSet result;

        private boolean DEBUG = true;
        private HashMap<ResultSet,String[]> columnNamesCache;
        private NameMapper mapper;
        private HashMap<Class,String> classToTableMap;
        protected ArrayList<String> tableNames;

        /**
         *      Do not use this contructor.
         */
        public SQL ()
        {
                System.out.println(
                        "SQL(): Please use this constructor\n"+
                        "\tSQL ( String _serv, String _db, String _u, String _p, PApplet _pa )"
                );

                //mapper = new de.bezier.data.sql.mapper.UnderScoreToCamelCaseMapper();
                mapper = new UnderScoreToCamelCaseMapper();
        }


        /**
         *      You should not directly use the SQL.class instead use the classes for your database type.
         */
        public SQL ( PApplet _pa, String _db )
```

```
{
        this.user = "";
        this.pass = "";
        this.server = "";

        String f = _pa.dataPath(_db);
        File ff = new File(f);
        if ( !ff.exists() || !ff.canRead() )
        {
                f = _pa.sketchPath( _db );
                ff = new File(f);

                if ( !ff.exists() || !ff.canRead() )
                {
                        f = _db;
                        ff = new File(f);

                        if ( !ff.exists() || !ff.canRead() )
                        {
                                System.err.println(
                                        "Sorry can't find any file named " + _db +
                                        " make sure it exists and the path is correct."
                                );
                        }
                }
        }

        _pa.println( "Using this database: " + f );

        this.database = f;

        this.url = "jdbc:" + type + ":" + database;

        this.papplet = _pa;
        papplet.registerDispose( this );
        //mapper = new de.bezier.data.sql.mapper.UnderScoreToCamelCaseMapper();
        mapper = new UnderScoreToCamelCaseMapper();
}


/**
 *      You should not directly use the SQL.class instead use the classes for your database type.
 */
public SQL ( PApplet _pa, String _serv, String _db, String _u, String _p )
{
        this.server = _serv;
        this.database = _db;

        this.url = "jdbc:" + type + "://" + server +  "/" + database;

        this.user = _u;
        this.pass = _p;

        this.papplet = _pa;
        papplet.registerDispose( this );
        //mapper = new de.bezier.data.sql.mapper.UnderScoreToCamelCaseMapper();
        mapper = new UnderScoreToCamelCaseMapper();
}


/**
 *      Turn some debugging on/off.
 *
 *      @param  yesNo   Turn it on or off
 */
public void setDebug ( boolean yesNo )
{
        DEBUG = yesNo;
}


/**
 *      Get current debugging setting
 *
 *      @param  yesNo   Turn it on or off
 */
public boolean getDebug ()
{
        return DEBUG;
}


/**
 *      Open the database connection with the parameters given in the contructor.
 */
public boolean connect()
{
        if ( driver == null || driver.equals("") ||
                 type == null || type.equals("") )
        {
                System.out.println( "SQL.connect(): You have to set a driver and type first." );
                return false;
        }

        // TODO: need to add mechanisms for different connection types and parameters, see:
        // http://jdbc.postgresql.org/documentation/83/connect.html

        try
        {
                Class.forName(driver);
                connection = java.sql.DriverManager.getConnection(url, user, pass);

        }
        catch (ClassNotFoundException e)
        {
                System.out.println( "SQL.connect(): Could not find the database driver ( "+driver+" ).\r" );
                if (DEBUG) e.printStackTrace();
                return false;
```

```
		}
		catch (java.sql.SQLException e)
		{
			System.out.println( "SQL.connect(): Could not connect to the database ( "+url+" ).\r" );
			if (DEBUG) e.printStackTrace();
			return false;

		}

		getTableNames();

		try {
			Driver jdbcDriver = java.sql.DriverManager.getDriver( url );
			if ( jdbcDriver != null ) {
				driverMinorVersion = jdbcDriver.getMinorVersion();
				driverMajorVersion = jdbcDriver.getMajorVersion();

				if ( DEBUG ) System.out.println( "Using driver " + getDriverVersion() );
			}
		} catch ( SQLException sqle ) {
			sqle.printStackTrace();
		}

		return true;
	}


	/**
	 *	Return the version of the currently active JDBC driver
	 *
	 *	@return String The version of the current driver
	 */
	public String getDriverVersion ()
	{
		if ( connection == null ) {
			System.out.println( "SQL.getDriverVersion(): you need to connect() first" );
			return null;
		} else if ( driver == null || driver.equals("") ) {
			System.out.println( "SQL.getDriverVersion(): no driver specified ... or it is null" );
			return null;
		}
		return driver + " " + driverMajorVersion + "." + driverMinorVersion;
	}

	private void preQueryOrExecute ()
	{
		result = null;
	}


	/**
	 *	Execute a SQL command on the open database connection.
	 *
	 *	@param	_sql	The SQL command to execute
	 */
	public void execute ( String _sql )
	{
		preQueryOrExecute();

		query( _sql, false );
	}


	/**
	 *	Execute a SQL command on the open database connection.
	 *	Arguments are passed to String.format() first.
	 *
	 *	@param	_sql	SQL command as pattern for String.format()
	 *	@param	args	Zero or more objects to be passed to String.format()
	 *
	 *  @see <a href="http://docs.oracle.com/javase/1.5.0/docs/api/java/util/Formatter.html#syntax">Format syntax</a>
	 *	@see java.lang.String#format(java.lang.String,java.lang.Object...)
	 */
	public void execute ( String _sql, Object ... args )
	{
		preQueryOrExecute();

		if ( args == null || args.length == 0 ) queryOrExecute( _sql, false );

		Method meth = null;
		try {
			meth = String.class.getMethod(
				"format",
				String.class,
				java.lang.reflect.Array.newInstance(Object.class,0).getClass()
			);
		} catch ( Exception ex ) {
			ex.printStackTrace();
			return;
		}
		// Object[] args2 = new Object[args.length+1];
		// args2[0] = _sql;
		// System.arraycopy( args, 0, args2, 1, args.length );
		String sql2 = null;
		try {
			sql2 = (String)meth.invoke( null, _sql, args );
		} catch ( Exception ex ) {
			if (DEBUG) ex.printStackTrace();
		}

		queryOrExecute( sql2, false );
	}


	/**
	 *	Issue a query on the open database connection
	 *
```

```
	 *	@param  _sql	SQL command to execute for the query
	 */
	public void query ( String _sql )
	{
		preQueryOrExecute();

		queryOrExecute( _sql, true );
	}


	/**
	 *	Issue a query on the open database connection.
	 *	Arguments are passed to String.format() first.
	 *
	 *	@param _sql	SQL command as pattern for String.format()
	 *  @param args Zero or more objects to be passed to String.format()
	 *
	 *  @see <a href="http://docs.oracle.com/javase/1.5.0/docs/api/java/util/Formatter.html#syntax">Format syntax</a>
	 *	@see java.lang.String#format(java.lang.String,java.lang.Object...)
	 */
	public void query ( String _sql, Object ... args )
	{
		preQueryOrExecute();

		if ( args == null || args.length == 0 ) queryOrExecute( _sql, true );

		Method meth = null;
		try {
			meth = String.class.getMethod(
				"format",
				String.class,
				java.lang.reflect.Array.newInstance(Object.class,0).getClass()
			);
		} catch ( Exception ex ) {
			ex.printStackTrace();
			return;
		}

		String sql2 = null;
		try {
			sql2 = (String)meth.invoke( null, _sql, args );
		} catch ( Exception ex ) {
			if (DEBUG) ex.printStackTrace();
		}

		queryOrExecute( sql2, true );
	}


	/**
	 *	Query implemenbtation called by execute() / query()
	 */
	private void queryOrExecute ( String _sql, boolean keep )
	{
		if ( connection == null )
		{
			System.out.println( "SQL.query(): You need to connect() first." );
			return;
		}

		previousQuery = _sql;

		try
		{
			if ( statement == null )
			{
				statement = connection.createStatement();
			}

			boolean hasResults = statement.execute( _sql );

			if ( keep && hasResults )
			{
				this.result = statement.getResultSet();
			}
		}
		catch ( java.sql.SQLException e )
		{
			System.out.println( "SQL.query(): java.sql.SQLException.\r" );
			if (DEBUG) {
				System.out.println( _sql );
				e.printStackTrace();
			}
		}
	}


	/**
	 *	Check if more results (rows) are available. This needs to be called before any results can be retrieved.
	 *
	 *	@return boolean true if more results are available, false otherwise
	 */
	public boolean next ()
	{
		if ( result == null )
		{
			System.out.println( "SQL.next(): You need to query() something first." );
			return false;
		}

		try
		{
			return result.next();
		}
		catch ( java.sql.SQLException e )
		{
			System.out.println( "SQL.next(): java.sql.SQLException.\r" );
			if ( DEBUG ) e.printStackTrace();
```

```
		}
		return false;
	}


	/**
	 *	Get names of available tables in active database,
	 *	needs to be implemented per db adapter.
	 *
	 * @return String[] The table names
	 */
	abstract public String[] getTableNames ();


	/**
	 *	Returns an array with the column names of the last request.
	 *
	 *	@return String[] the column names of last result or null
	 */
	public String[] getColumnNames ()
	{
		String[] colNames = null;

		if ( result == null )
		{
			System.out.println( "SQL.getColumnNames(): You need to query() something first." );
			return null;
		}

		// if ( columnNamesCache == null )
		//	columnNamesCache = new HashMap<ResultSet,String[]>();
		//
		// colNames = columnNamesCache.get( result );
		// if ( colNames != null ) return colNames;

		java.sql.ResultSetMetaData meta = null;
		try {
			meta = result.getMetaData();
		} catch ( SQLException sqle ) {
			if (DEBUG) sqle.printStackTrace();
			return null;
		}

		if ( meta != null )
		{
			try {
				colNames = new String[ meta.getColumnCount() ];
				for ( int i = 1, k = meta.getColumnCount(); i <= k; i++ )
				{
					colNames[i-1] = meta.getColumnName( i );
				}
			} catch ( SQLException sqle ) {
				if (DEBUG) sqle.printStackTrace();
				return null;
			}
		}

		// columnNamesCache.clear();
		// columnNamesCache.put( result, colNames );

		return colNames;
	}


	/**
	 *	Get connection. ... in case you want to do JDBC stuff directly.
	 *
	 *	@return java.sql.Connection The connection
	 */
	public java.sql.Connection getConnection ()
	{
		return connection;
	}


	/**
	 *	Read an integer value from the specified field.
	 *	Represents an INT / INTEGER type:
	 *	http://java.sun.com/j2se/1.3/docs/guide/jdbc/getstart/mapping.html
	 *	"8.9.6  Conversions by ResultSet.getXXX Methods"
	 *
	 *	@param  _field  The name of the field
	 *	@return int     Value of the field or 0
	 */
	public int getInt ( String _field )
	{
		// TODO: 0 does not seem to be a good return value for a numeric field to indicate failure
		// same goes for other numeric fields

		if ( result == null )
		{
			System.out.println( "SQL.getInt(): You need to query() something first." );
			return 0;
		}

		try
		{
			return result.getInt( _field );
		}
		catch ( java.sql.SQLException e )
		{
			System.out.println( "SQL.getInt(): java.sql.SQLException.\r" );
			if (DEBUG) e.printStackTrace();
		}
		return 0;
	}

	/**
```

```
	 *	Read an integer value from the specified field.
	 *	Represents an INT / INTEGER type:
	 *	http://java.sun.com/j2se/1.3/docs/guide/jdbc/getstart/mapping.html
	 *	"8.9.6  Conversions by ResultSet.getXXX Methods"
	 *
	 *	@param  _column The column index of the field to read
	 *	@return int     Value of the field or 0
	 */
	public int getInt ( int _column )
	{
		if ( result == null )
		{
			System.out.println( "SQL.getInt(): You need to query() something first." );
			return 0;
		}

		try
		{
			return result.getInt( _column );
		}
		catch ( java.sql.SQLException e )
		{
			System.out.println( "SQL.getInt(): java.sql.SQLException.\r" );
			if (DEBUG) e.printStackTrace();
		}
		return 0;
	}


	/**
	 *	Read a long value from the specified field.
	 *	Represents a BIGINT type:
	 *	http://java.sun.com/j2se/1.3/docs/guide/jdbc/getstart/mapping.html
	 *	"8.9.6  Conversions by ResultSet.getXXX Methods"
	 *
	 *	@param  _field  The name of the field
	 *	@return long    Value of the field or 0
	 */
	public long getLong ( String _field )
	{
		if ( result == null )
		{
			System.out.println( "SQL.getLong(): You need to query() something first." );
			return 0;
		}

		try
		{
			return result.getLong( _field );
		}
		catch ( java.sql.SQLException e )
		{
			System.out.println( "SQL.getLong(): java.sql.SQLException.\r" );
			if (DEBUG) e.printStackTrace();
		}
		return 0;
	}


	/**
	 *	Read a long value from the specified field.
	 *	Represents a BIGINT type:
	 *	http://java.sun.com/j2se/1.3/docs/guide/jdbc/getstart/mapping.html
	 *	"8.9.6  Conversions by ResultSet.getXXX Methods"
	 *
	 *	@param  _column The column index of the field
	 *	@return long    Value of the field or 0
	 */
	public long getLong ( int _column )
	{
		if ( result == null )
		{
			System.out.println( "SQL.getLong(): You need to query() something first." );
			return 0;
		}

		try
		{
			return result.getLong( _column );
		}
		catch ( java.sql.SQLException e )
		{
			System.out.println( "SQL.getLong(): java.sql.SQLException.\r" );
			if (DEBUG) e.printStackTrace();
		}
		return 0;
	}


	/**
	 *	Read a float value from the specified field.
	 *	Represents a REAL type:
	 *	http://java.sun.com/j2se/1.3/docs/guide/jdbc/getstart/mapping.html
	 *	"8.9.6  Conversions by ResultSet.getXXX Methods"
	 *
	 *	@param  _field  The name of the field
	 *	@return float   Value of the field or 0
	 */
	public float getFloat ( String _field )
	{
		if ( result == null )
		{
			System.out.println( "SQL.getFloat(): You need to query() something first." );
			return 0.0f;
		}

		try
		{
```

```
                return result.getFloat( _field );
        }
        catch ( java.sql.SQLException e )
        {
                System.out.println( "SQL.getFloat(): java.sql.SQLException.\r" );
                if (DEBUG) e.printStackTrace();
        }
        return 0.0f;
}


/**
 *      Read a fLoat value from the specified field.
 *      Represents a REAL type:
 *      http://java.sun.com/j2se/1.3/docs/guide/jdbc/getstart/mapping.html
 *      "8.9.6  Conversions by ResultSet.getXXX Methods"
 *
 *      @param  _column The index of the column of the field
 *      @return float   Value of the field or 0
 */
public float getFloat ( int _column )
{
        if ( result == null )
        {
                System.out.println( "SQL.getFloat(): You need to query() something first." );
                return 0.0f;
        }

        try
        {
                return result.getFloat( _column );
        }
        catch ( java.sql.SQLException e )
        {
                System.out.println( "SQL.getFloat(): java.sql.SQLException.\r" );
                if (DEBUG) e.printStackTrace();
        }
        return 0.0f;
}


/**
 *      Read a double value from the specified field.
 *      Represents FLOAT and DOUBLE types:
 *      http://java.sun.com/j2se/1.3/docs/guide/jdbc/getstart/mapping.html
 *      "8.9.6  Conversions by ResultSet.getXXX Methods"
 *
 *      @param  _field  The name of the field
 *      @return double  Value of the field or 0
 */
public double getDouble ( String _field )
{
        if ( result == null )
        {
                System.out.println( "SQL.getDouble(): You need to query() something first." );
                return 0.0;
        }

        try
        {
                return result.getDouble( _field );
        }
        catch ( java.sql.SQLException e )
        {
                System.out.println( "SQL.getDouble(): java.sql.SQLException.\r" );
                if (DEBUG) e.printStackTrace();
        }
        return 0.0;
}


/**
 *      Read a double value from the specified field.
 *      Represents FLOAT and DOUBLE types:
 *      http://java.sun.com/j2se/1.3/docs/guide/jdbc/getstart/mapping.html
 *      "8.9.6  Conversions by ResultSet.getXXX Methods"
 *
 *      @param  _column The column index of the field
 *      @return double  Value of the field or 0
 */
public double getDouble ( int _column )
{
        if ( result == null )
        {
                System.out.println( "SQL.getDouble(): You need to query() something first." );
                return 0.0;
        }

        try
        {
                return result.getDouble( _column );
        }
        catch ( java.sql.SQLException e )
        {
                System.out.println( "SQL.getDouble(): java.sql.SQLException.\r" );
                if (DEBUG) e.printStackTrace();
        }
        return 0.0;
}


/**
 *      Read a java.math.BigDecimal value from the specified field.
 *      Represents DECIMAL and NUMERIC types:
 *      http://java.sun.com/j2se/1.3/docs/guide/jdbc/getstart/mapping.html
 *      "8.9.6  Conversions by ResultSet.getXXX Methods"
 *
 *      @param  _field  The name of the field
```

```
	 *	@return java.math.BigDecimal	Value of the field or null
	 */
	public java.math.BigDecimal getBigDecimal ( String _field )
	{
		if ( result == null )
		{
			System.out.println( "SQL.getBigDecimal(): You need to query() something first." );
			return null;
		}

		try
		{
			return result.getBigDecimal( _field );
		}
		catch ( java.sql.SQLException e )
		{
			System.out.println( "SQL.getBigDecimal(): java.sql.SQLException.\r" );
			if (DEBUG) e.printStackTrace();
		}
		return null;
	}


	/**
	 *	Read a java.math.BigDecimal value from the specified field.
	 *	Represents DECIMAL and NUMERIC types:
	 *	http://java.sun.com/j2se/1.3/docs/guide/jdbc/getstart/mapping.html
	 *	"8.9.6  Conversions by ResultSet.getXXX Methods"
	 *
	 *	@param  _column The column index of the field
	 *	@return java.math.BigDecimal	Value of the field or null
	 */
	public java.math.BigDecimal getBigDecimal ( int _column )
	{
		if ( result == null )
		{
			System.out.println( "SQL.getBigDecimal(): You need to query() something first." );
			return null;
		}

		try
		{
			return result.getBigDecimal( _column );
		}
		catch ( java.sql.SQLException e )
		{
			System.out.println( "SQL.getBigDecimal(): java.sql.SQLException.\r" );
			if (DEBUG) e.printStackTrace();
		}
		return null;
	}


	/**
	 *	Read a boolean value from the specified field.
	 *	Represents BIT type:
	 *	http://java.sun.com/j2se/1.3/docs/guide/jdbc/getstart/mapping.html
	 *	"8.9.6  Conversions by ResultSet.getXXX Methods"
	 *
	 *	@param  _field  The name of the field
	 *	@return boolean Value of the field or 0
	 */
	public boolean getBoolean ( String _field )
	{
		if ( result == null )
		{
			System.out.println( "SQL.getBoolean(): You need to query() something first." );
			return false;
		}

		try
		{
			return result.getBoolean( _field );
		}
		catch ( java.sql.SQLException e )
		{
			System.out.println( "SQL.getBoolean(): java.sql.SQLException.\r" );
			if (DEBUG) e.printStackTrace();
		}
		return false;
	}


	/**
	 *	Read a boolean value from the specified field.
	 *	Represents BIT type:
	 *	http://java.sun.com/j2se/1.3/docs/guide/jdbc/getstart/mapping.html
	 *	"8.9.6  Conversions by ResultSet.getXXX Methods"
	 *
	 *	@param  _column The column index of the field
	 *	@return boolean Value of the field or 0
	 */
	public boolean getBoolean ( int _column )
	{
		if ( result == null )
		{
			System.out.println( "SQL.getBoolean(): You need to query() something first." );
			return false;
		}

		try
		{
			return result.getBoolean( _column );
		}
		catch ( java.sql.SQLException e )
		{
			System.out.println( "SQL.getBoolean(): java.sql.SQLException.\r" );
			if (DEBUG) e.printStackTrace();
```

```
        }
        return false;
    }


    /**
     *      Read a String value from the specified field.
     *      Represents VARCHAR and CHAR types:
     *      http://java.sun.com/j2se/1.3/docs/guide/jdbc/getstart/mapping.html
     *      "8.9.6  Conversions by ResultSet.getXXX Methods"
     *
     *      @param  _field  The name of the field
     *      @return String  Value of the field or null
     */
    public String getString ( String _field )
    {
        if ( result == null )
        {
            System.out.println( "SQL.getString(): You need to query() something first." );
            return null;
        }

        try
        {
            return result.getString( _field );
        }
        catch ( java.sql.SQLException e )
        {
            System.out.println( "SQL.getString(): java.sql.SQLException.\r" );
            if (DEBUG) e.printStackTrace();
        }
        return null;
    }

    /**
     *      Read a String value from the specified field.
     *      Represents VARCHAR and CHAR types:
     *      http://java.sun.com/j2se/1.3/docs/guide/jdbc/getstart/mapping.html
     *      "8.9.6  Conversions by ResultSet.getXXX Methods"
     *
     *      @param  _column The column index of the field
     *      @return String  Value of the field or null
     */
    public String getString ( int _column )
    {
        if ( result == null )
        {
            System.out.println( "SQL.getString(): You need to query() something first." );
            return null;
        }

        try
        {
            return result.getString( _column );
        }
        catch ( java.sql.SQLException e )
        {
            System.out.println( "SQL.getString(): java.sql.SQLException.\r" );
            if (DEBUG) e.printStackTrace();
        }
        return null;
    }

    /**
     *      Read a java.sql.Date value from the specified field.
     *      Represents DATE type:
     *      http://java.sun.com/j2se/1.3/docs/guide/jdbc/getstart/mapping.html
     *      "8.9.6  Conversions by ResultSet.getXXX Methods"
     *
     *      @param  _field  The name of the field
     *      @return java.sql.Date   Value of the field or null
     */
    public java.sql.Date getDate ( String _field )
    {
        if ( result == null )
        {
            System.out.println( "SQL.getDate(): You need to query() something first." );
            return null;
        }

        try
        {
            return result.getDate( _field );
        }
        catch ( java.sql.SQLException e )
        {
            System.out.println( "SQL.getDate(): java.sql.SQLException.\r" );
            if (DEBUG) e.printStackTrace();
        }
        return null;
    }


    /**
     *      Read a java.sql.Date value from the specified field.
     *      Represents DATE type:
     *      http://java.sun.com/j2se/1.3/docs/guide/jdbc/getstart/mapping.html
     *      "8.9.6  Conversions by ResultSet.getXXX Methods"
     *
     *      @param  _column The column index of the field
     *      @return java.sql.Date   Value of the field or null
     */
    public java.sql.Date getDate ( int _column )
    {
        if ( result == null )
        {
            System.out.println( "SQL.getDate(): You need to query() something first." );
            return null;
```

```
        }

        try
        {
                return result.getDate( _column );
        }
        catch ( java.sql.SQLException e )
        {
                System.out.println( "SQL.getDate(): java.sql.SQLException.\r" );
                if (DEBUG) e.printStackTrace();
        }
        return null;
}


/**
 *      Read a java.sql.Time value from the specified field.
 *      Represents TIME type:
 *      http://java.sun.com/j2se/1.3/docs/guide/jdbc/getstart/mapping.html
 *      "8.9.6  Conversions by ResultSet.getXXX Methods"
 *
 *      @param  _field  The name of the field
 *      @return java.sql.Time   Value of the field or null
 */
public java.sql.Time getTime ( String _field )
{
        if ( result == null )
        {
                System.out.println( "SQL.getTime(): You need to query() something first." );
                return null;
        }

        try
        {
                return result.getTime( _field );
        }
        catch ( java.sql.SQLException e )
        {
                System.out.println( "SQL.getTime(): java.sql.SQLException.\r" );
                if (DEBUG) e.printStackTrace();
        }
        return null;
}


/**
 *      Read a java.sql.Time value from the specified field.
 *      Represents TIME type:
 *      http://java.sun.com/j2se/1.3/docs/guide/jdbc/getstart/mapping.html
 *      "8.9.6  Conversions by ResultSet.getXXX Methods"
 *
 *      @param  _column The column index of the field
 *      @return java.sql.Time   Value of the field or null
 */
public java.sql.Time getTime ( int _column )
{
        if ( result == null )
        {
                System.out.println( "SQL.getTime(): You need to query() something first." );
                return null;
        }

        try
        {
                return result.getTime( _column );
        }
        catch ( java.sql.SQLException e )
        {
                System.out.println( "SQL.getTime(): java.sql.SQLException.\r" );
                if (DEBUG) e.printStackTrace();
        }
        return null;
}


/**
 *      Read a java.sql.Timestamp value from the specified field.
 *      Represents TIMESTAMP type:
 *      http://java.sun.com/j2se/1.3/docs/guide/jdbc/getstart/mapping.html
 *      "8.9.6  Conversions by ResultSet.getXXX Methods"
 *
 *      @param  _field  The name of the field
 *      @return java.sql.Timestamp      Value of the field or null
 */
public java.sql.Timestamp getTimestamp ( String _field )
{
        if ( result == null )
        {
                System.out.println( "SQL.getTimestamp(): You need to query() something first." );
                return null;
        }

        try
        {
                return result.getTimestamp( _field );
        }
        catch ( java.sql.SQLException e )
        {
                System.out.println( "SQL.getTimestamp(): java.sql.SQLException.\r" );
                if (DEBUG) e.printStackTrace();
        }
        return null;
}


/**
 *      Read a java.sql.Timestamp value from the specified field.
 *      Represents TIMESTAMP type:
```

```
     *      http://java.sun.com/j2se/1.3/docs/guide/jdbc/getstart/mapping.html
     *      "8.9.6  Conversions by ResultSet.getXXX Methods"
     *
     *      @param  _column The column index of the field
     *      @return java.sql.Timestamp      Value of the field or null
     */
    public java.sql.Timestamp getTimestamp ( int _column )
    {
            if ( result == null )
            {
                    System.out.println( "SQL.getTimestamp(): You need to query() something first." );
                    return null;
            }

            try
            {
                    return result.getTimestamp( _column );
            }
            catch ( java.sql.SQLException e )
            {
                    System.out.println( "SQL.getTimestamp(): java.sql.SQLException.\r" );
                    if (DEBUG) e.printStackTrace();
            }
            return null;
    }


    /**
     *      Read a value from the specified field to hav it returned as an object.
     *
     *      @param  _field  The name of the field
     *      @return Object  Value of the field or null
     */
    public Object getObject ( String _field )
    {
            if ( result == null )
            {
                    System.out.println( "SQL.getObject(): You need to query() something first." );
                    return null;
            }

            try
            {
                    return result.getObject( _field );
            }
            catch ( java.sql.SQLException e )
            {
                    System.out.println( "SQL.getObject(): java.sql.SQLException.\r" );
                    if (DEBUG) e.printStackTrace();
            }
            return null;
    }


    /**
     *      Read a value from the specified field to hav it returned as an object.
     *
     *      @param  _column The column index of the field
     *      @return Object  Value of the field or null
     */
    public Object getObject ( int _column )
    {
            if ( result == null )
            {
                    System.out.println( "SQL.getObject(): You need to query() something first." );
                    return null;
            }

            try
            {
                    return result.getObject( _column );
            }
            catch ( java.sql.SQLException e )
            {
                    System.out.println( "SQL.getObject(): java.sql.SQLException.\r" );
                    if (DEBUG) e.printStackTrace();
            }
            return null;
    }


    /**
     *      Close the database connection
     */
    public void close()
    {
            dispose();
    }


    /**
     *      Callback function for PApplet.registerDispose()
     *
     *      @see processing.core.PApplet.registerDispose(java.lang.Object)
     */
    public void dispose ()
    {
            if ( result != null )
            {
                    try
                    {
                            result.close();
                    }
                    catch ( java.sql.SQLException e ) { ; }

                    result = null;
            }
```

```
		if ( statement != null )
		{
			try
			{
				statement.close();
			}
			catch ( java.sql.SQLException e ) { ; }

			statement = null;
		}

		if ( connection != null )
		{
			try
			{
				connection.close();
			}
			catch ( java.sql.SQLException e ) { ; }

			connection = null;
		}
	}


	/**
	 *	Generate an escaped String for a given Object
	 *
	 *	@param object the Object to escape
	 *	@return String the ecaped String representation of the Object
	 */
	public String escape ( Object o )
	{
		return "\"" + o.toString().replaceAll("\"","\\\"") + "\"";
	}


	/**
	 *	Set the current NameMapper
	 *
	 *	@param mapper the name mapper
	 *  @see de.bezier.data.sql.mapper.NameMapper
	 */
	public void setNameMapper ( NameMapper mapper )
	{
		this.mapper = mapper;
	}


	/**
	 *	Get the current NameMapper
	 *
	 *  @see de.bezier.data.sql.mapper.NameMapper
	 */
	public NameMapper getNameMapper ()
	{
		return mapper;
	}


	/**
	 *	<p>Highly experimental ...<br />
	 *	tries to map column names to public fields or setter methods
	 *	in the given object.</p>
	 *
	 *	<p>Use like so:
	 *  <pre>
	 *	db.query("SELECT name, id, sometime FROM table");
	 *
	 *	while ( db.next() ) {
	 *		SomeObject obj = new SomeObject();
	 *		db.setFromRow(obj);
	 *		// obj.name is now same as db.getString("name"), etc.
	 *  }
	 *  </pre></p>
	 *
	 *	<p>SomeObject might look like:
	 *	<pre>
	 *	class SomeObject {
	 *		public String name;
	 *		public int id;
	 *		Date sometime;
	 *	}
	 *	</pre></p>
	 *
	 *	@param object The object to populate from the currently selected row
	 */
	public void setFromRow ( Object object )
	{
		if ( object == null ) {
			System.err.println( "SQL.rowToObject(): Handing in null won't cut it." );
			return;
		}

		if ( result == null ) {
			System.err.println( "SQL.rowToObject(): You need to query() something first!" );
			return;
		}

		String[] colNames = getColumnNames();
		if ( colNames == null )
		{
			System.err.println(
				"SQL.rowToObject(): uh-oh something went wrong: unable to get column names." );
			return;
		}

		if ( colNames.length > 0 )
		{
```

```
                Class klass = null;
                try {
                        klass = Class.forName("DeBezierDataSQL");
                } catch ( Exception e ) {
                        if (DEBUG) e.printStackTrace();
                }
                if ( klass != null ) {
                        Method meth = null;
                        try {
                                meth = klass.getMethod(
                                        "setFromRow",
                                        new Class[]{ SQL.class, Object.class }
                                );
                        } catch ( Exception e ) {
                                if (DEBUG) e.printStackTrace();
                        }
                        // System.out.println( meth );
                        // System.out.println( meth.getParameterTypes() );
                        if ( meth != null ) {
                                try {
                                        meth.invoke( null, this, object );
                                } catch ( Exception e ) {
                                        if (DEBUG) e.printStackTrace();
                                }
                        }
                }
        }
}


/**
 *      Convert a field name to a setter name: fieldName -> setFieldName().
 */
public String nameToSetter ( String name )
{
        if ( name == null ) return name;
        if ( name.length() == 0 ) return null;
        return "set" + name.substring(0,1).toUpperCase() + name.substring(1);
}


/**
 *      Convert a field name to a getter name: fieldName -> getFieldName().
 */
public String nameToGetter ( String name )
{
        if ( name == null ) return name;
        if ( name.length() == 0 ) return null;
        return "get" + name.substring(0,1).toUpperCase() + name.substring(1);
}


/**
 *      Set a table name for a class.
 */
public void registerTableNameForClass ( String name, Object classOrObject )
{
        if ( name == null || name.equals("") || classOrObject == null ) return;

        Class klass = null;
        if ( classOrObject.getClass() != Class.class )
                klass = classOrObject.getClass();
        else
                klass = (Class)classOrObject;

        if ( classToTableMap == null )
                classToTableMap = new HashMap<Class,String>();

        classToTableMap.put( klass, name );
        if (DEBUG) System.out.println( String.format(
                "Class \"%s\" is now mapped to table \"%s\"", klass.getName(), name
        ));
}


/**
 *      Take an object, try to find table name from object name (or look it up),
 *      get fields and getters from object and pass that on to
 *      insertUpdateIntoDatabase(table, columns, values).
 *
 *      @param object Object The object to save to db
 *
 *  @see #insertUpdateInDatabase(java.lang.String, java.lang.String[], java.lang.Object[])
 */
public void saveToDatabase ( Object object )
{
        if ( object == null ) return;

        // Find the table name

        String tableName = null;

        if ( classToTableMap == null )
                classToTableMap = new HashMap<Class,String>();

        tableName = classToTableMap.get(object.getClass());

        if ( tableName != null )
                saveToDatabase( tableName, object );
        else
        {
                Class klass = object.getClass();
                tableName = klass.getName();

                for ( char c : new char[]{'$','.'} )
                {
                        int indx = tableName.lastIndexOf(c);
                        if ( indx >= 0 ) {
```

```
                                tableName = tableName.substring(indx+1);
                        }
                }

                if ( mapper != null ) {
                        tableName = mapper.backward(tableName);
                }

                registerTableNameForClass( tableName, klass );

                saveToDatabase( tableName, object );
        }
}


/**
 *      Takes a table name and an object and tries to construct a set of
 *      columns names from fields and getters found in the object. After
 *      the values are fetched from the object all is passed to
 *      insertUpdateIntoDatabase().
 *
 * @param tableName String The name of the table
 * @param object Object The object to look at
 *
 * @see #insertUpdateInDatabase(java.lang.String, java.lang.String[], java.lang.Object[])
 */
public void saveToDatabase ( String tableName, Object object )
{
        if ( object == null ) return;

        String[] tableNames = getTableNames();
        if ( !java.util.Arrays.asList(tableNames).contains(tableName) ) {
                System.err.println(String.format(
                        "saveToDatabase(): table '%s' not found in database '%s'", tableName, database
                ));
                return;
        }

        String[] colNames = getColumnNames();
        String[] fieldNames = new String[colNames.length];

        if ( mapper != null )
        {
                for ( int i = 0; i < colNames.length; i++ )
                {
                        //System.out.println(colNames[i]);
                        fieldNames[i] = mapper.forward(colNames[i]);
                        //System.out.println(fieldNames[i]);
                }
        }

        Class klass = object.getClass();
        Field[] fields = new Field[colNames.length];
        Method[] getters = new Method[colNames.length];

        for ( int i = 0; i < colNames.length; i++ )
        {
                String fieldName = fieldNames[i];
                String colName = colNames[i];

                Field f = null;
                try {
                        f = klass.getField(fieldName);
                        if ( f == null ) {
                                f = klass.getField(colName);
                        }
                } catch ( Exception e ) {
                        if (DEBUG) e.printStackTrace();
                }
                if ( f != null ) {
                        // try {
                        //      values[i] = f.get(object);
                        // } catch ( Exception e ) {
                        //      if (DEBUG) e.printStackTrace();
                        // }
                        fields[i] = f;
                }
                else
                {
                        if (DEBUG) System.out.println( "Field not found, trying setter method" );

                        String getterName = nameToGetter(fieldName);
                        Method getter = null;
                        try {
                                getter = klass.getMethod(
                                        getterName, new Class[0]
                                );
                        } catch ( Exception e ) {
                                if (DEBUG) e.printStackTrace();
                        }
                        // try {
                        //      values[i] = getter.invoke(object);
                        // } catch ( Exception e ) {
                        //      if (DEBUG) e.printStackTrace();
                        // }
                        getters[i] = getter;
                }
                if ( fields[i] == null && getters[i] == null ) {
                        System.err.println(String.format(
                                "Unable to get a field or getter for column '%s'", colName
                        ));
                        return;
                }
        }

        Object[] values = null;

        Class clazz = null;
```

```
		try {
			clazz = Class.forName("DeBezierDataSQL");
		} catch ( Exception e ) {
			if (DEBUG) e.printStackTrace();
		}

		if ( klass != null ) {
			Method meth = null;
			try {
				meth = clazz.getMethod(
					"getValuesFromObject",
					new Class[]{ SQL.class, Field[].class, Method[].class, Object.class }
				);
			} catch ( Exception e ) {
				if (DEBUG) e.printStackTrace();
			}
			// System.out.println( meth );
			// System.out.println( meth.getParameterTypes() );
			if ( meth != null ) {
				try {
					values = (Object[])meth.invoke( null, this, fields, getters, object );
				} catch ( Exception e ) {
					if (DEBUG) e.printStackTrace();
				}
			}
		}

		if ( values != null )
		{
			insertUpdateInDatabase( tableName, colNames, values );
		}
		else
		{
			System.err.println("saveToDatabase() : trouble, trouble!!");
		}
	}


	/**
	 *	Insert or update a bunch of values in the database. If the given table has a
	 *	primary key the entry will be updated if it already existed.
	 *
	 *	@param tableName String The name of the table
	 *	@param columnNames String[] The names of the columns to fill or update
	 *	@param values Object[] The values to instert or update
	 */
	public void insertUpdateInDatabase ( String tableName, String[] columnNames, Object[] values )
	{
		HashMap<Object, Object> valuesKeys = new HashMap<Object,Object>();
		for ( int i = 0; i < values.length; i++ )
		{
			valuesKeys.put(columnNames[i], values[i]);
		}

		HashMap<String, Object> primaryKeys = null;
		try {
			DatabaseMetaData meta = connection.getMetaData();
			ResultSet rs = meta.getPrimaryKeys(null, null, tableName);

		    while ( rs.next() )
			{
				if ( primaryKeys == null )
				primaryKeys = new HashMap<String, Object>();

				String columnName = rs.getString("COLUMN_NAME");
			primaryKeys.put(columnName, valuesKeys.get(columnName));
				valuesKeys.remove(columnName);
			}

		} catch ( SQLException sqle ) {
			sqle.printStackTrace();
		}

		//System.out.println(valuesKeys);
		//System.out.println(primaryKeys);

		String cols = "";
		String patt = "";
		HashMap<Object, Integer> valueIndices = new HashMap<Object, Integer>();
		int i = 1;
		for ( Map.Entry e : valuesKeys.entrySet() )
		{
			cols += ( i > 1 ? " , " : "" ) + e.getKey();
			patt += ( i > 1 ? " , " : "" ) + "?";
			valueIndices.put( e.getKey(), i );
			i++;
		}

		String sql = null, opts = null;
		HashMap<Object, Integer> primaryIndices = null;
		if ( primaryKeys == null || primaryKeys.size() == 0 ) {
			sql = "INSERT INTO " + tableName + " ( " + cols + " ) VALUES ( " + patt + " )";
		} else {
			primaryIndices = new HashMap<Object, Integer>();
			opts = " WHERE ";
			int p = 1;
			for ( Map.Entry e : primaryKeys.entrySet() ) {
				opts += (p > 1 ? " , " : "") + e.getKey() + " = ? ";
				primaryIndices.put( e.getKey(), p );
				p++;
			}
			//System.out.println( opts );
			String sqlFind = "SELECT * FROM "+tableName+" "+opts;
			//System.out.println( sqlFind );
			try {
				PreparedStatement psFind = connection.prepareStatement( sqlFind );
				for ( Map.Entry e : primaryKeys.entrySet() ) {
					psFind.setString( primaryIndices.get(e.getKey()), e.getValue().toString() );
```

```
                                }
                                result = psFind.executeQuery();
                                boolean found = next();
                                psFind.close();
                                if ( !found ) {
                                        if (DEBUG) System.out.println(String.format(
                                                "No entry with %s found in table '%s'", primaryKeys.toString(), tableName
                                        ));
                                        int k = 1, m = valueIndices.size();
                                        for ( Map.Entry e : primaryKeys.entrySet() ) {
                                                cols += ( m > 0 ? " , " : "" ) + e.getKey();
                                                patt += ( m > 0 ? " , " : "" ) + "?";
                                                valuesKeys.put( e.getKey(), e.getValue() );
                                                valueIndices.put( e.getKey(), m+k );
                                                k++;
                                        }
                                        sql = "INSERT INTO " + tableName + " ( " + cols + " ) VALUES ( " + patt + " )";
                                        primaryKeys = null;
                                }
                                else
                                {
                                        cols = "";
                                        for ( Map.Entry e : valuesKeys.entrySet() )
                                        {
                                                cols += ( cols.equals("") ? "" : " , " ) + e.getKey() + " = " + "?";
                                        }
                                        sql = "UPDATE "+tableName+" SET " + cols + " " + opts;
                                }
                        } catch ( java.sql.SQLException sqle ) {
                                sqle.printStackTrace();
                        }
                }

                if (DEBUG) System.out.println( sql );

                try {
                        PreparedStatement ps = connection.prepareStatement( sql );

                        for ( Map.Entry e : valuesKeys.entrySet() )
                        {
                                ps.setString( valueIndices.get(e.getKey()), e.getValue()+"" );
                        }

                        if ( primaryKeys != null )
                        {
                                for ( Map.Entry e : primaryKeys.entrySet() )
                                {
                                        ps.setString( valueIndices.size()+primaryIndices.get(e.getKey()), e.getValue()+"" );
                                }
                        }

                        ps.executeUpdate();
                        ps.close();
                } catch ( java.sql.SQLException sqle ) {
                        sqle.printStackTrace();
                }
        }
}
```

SQLite.java

```
//package de.bezier.data.sql;

import processing.core.*;
import java.util.ArrayList;

/**
 *              SQLite wrapper for SQL library for Processing 2+
 *              <p>
 *              A wrapper around some of sun's java.sql.* classes
 *              and the pure java "org.sqlite.JDBC" driver of the Xerial project (Apache 2 license).
 *              </p>
 *              see:<ul>
 *                      <li>http://www.xerial.org/trac/Xerial/wiki/SQLiteJDBC</li>
 *                      <li>http://www.xerial.org/maven/repository/site/xerial-project/sqlite-jdbc/apidocs/index.html?index-all.html</li>
 *                      <li>http://java.sun.com/products/jdbc/</li>
 *              </ul>
 *
 *
 *              @author                 Florian Jenett - mail@florianjenett.de
 *
 *              created:                2008-11-29 12:15:15 - fjenett
 *              modified:               fjenett 20121217
 *
 */

//public class SQLite extends de.bezier.data.sql.SQL
public class SQLite extends SQL
{
        /**
         *      Creates a new SQLite connection.
         *
         *      @param  _papplet        Your sketch, pass "this" in here
         *      @param  _database       Path to the database file, if this is just a name the data and sketch folders are searched for the file
         */

        public SQLite ( PApplet _papplet, String _database  )
        {
                super( _papplet, _database );
                init();
        }

        /**
```

```
         *      Creates a new SQLite connection, same as SQLite( PApplet, String )
         *
         *      @param  _papplet        Your sketch, pass "this" in here
         *      @param  _server         Ignored
         *      @param  _database       Path to the database file, if this is just a name the data and sketch folders are searched for the file
         *      @param  _user           Ignored
         *      @param  _pass           Ignored
         */

        public SQLite (  PApplet _papplet, String _server, String _database, String _user, String _pass )
        {
                this( _papplet, _database );
        }


        private void init ()
        {
                this.driver = "org.sqlite.JDBC";
                this.type = "sqlite";

                this.url = "jdbc:" + type + ":" + database;
        }

        public String[] getTableNames ()
        {
                if ( tableNames == null )
                {
                        tableNames = new ArrayList<String>();
                query( "SELECT name AS 'table_name' FROM SQLITE_MASTER WHERE type=\"table\"" );
                        while ( next() ) {
                                tableNames.add( getObject("table_name").toString() );
                        }
                }
                return tableNames.toArray(new String[0]);
        }
}
```

UnderScoreToCamelCaseMapper.java

```
//package de.bezier.data.sql.mapper;

/**
 *      UnderScoreToCamelCaseMapper, does as it says.
 */

public class UnderScoreToCamelCaseMapper implements NameMapper {
        public String backward ( String name )  {
                String newName = "";
                for ( int i = 0, k = name.length(); i < k; i++ )                {
                        String c = name.charAt(i) + "";
                        if ( c.toUpperCase().equals(c) && !c.toLowerCase().equals(c) ) {
                                if ( i > 0 && i < k-1 )
                                        c = "_" + c.toLowerCase();
                                else
                                        c = c.toLowerCase();
                        }
                        newName += c;
                }
                return newName;
        }

        public String forward ( String name )   {
                String[] pieces = name.split("_");
                String newName = pieces[0];
                for ( int i = 1; i < pieces.length; i++ ) {
                        if ( pieces[i] != null && pieces[1].length() > 0 ) {
                                newName += pieces[i].substring(0,1).toUpperCase() +
                                                pieces[i].substring(1);
                        }
                }
                return newName;
        }

        public static void main ( String ... args ) {
                String[] test = new String[]{
                        "created_at",
                        "created_at_and_else",
                        "rank_1",
                        "_rank_abc",
                        "rank_abc_",
                        "camelCase_and_more"
                };
                UnderScoreToCamelCaseMapper mapper = new UnderScoreToCamelCaseMapper();
                for ( String t : test ) {
                        String fwd = mapper.forward(t);
                        System.out.println( t + " -> " + fwd );
                        String back = mapper.backward(fwd);
                        System.out.println( fwd + " -> " + back );
                        System.out.println( t.equals(back) );
                        System.out.println( "" );
                }
        }
}
```

NameMapper.java

```
//package de.bezier.data.sql.mapper;

/**
```

```
 *      NameMapper is used to map database names to instance names
 *      When setting objects from objects with SQL.setFromRow(). *
 *      This is just an interface and only one implementation is
 *      provided in form of the default UnderScoreToCamelCaseMapper
 *      which does: field_name -> fieldName and vv.
 */

public interface NameMapper {
        /**
         *      Maps a database name to an object name, typically
         *      this might look like: field_name -> fieldName.
         */
        public String forward ( String name );

        /**
         *      Reverse of forward, maps object names to database
         *      names like: fieldName -> field_name.
         */
        public String backward ( String name );
}
```