

**SOLExpress for Xbase++**  
*industrial strength C/S development tool*

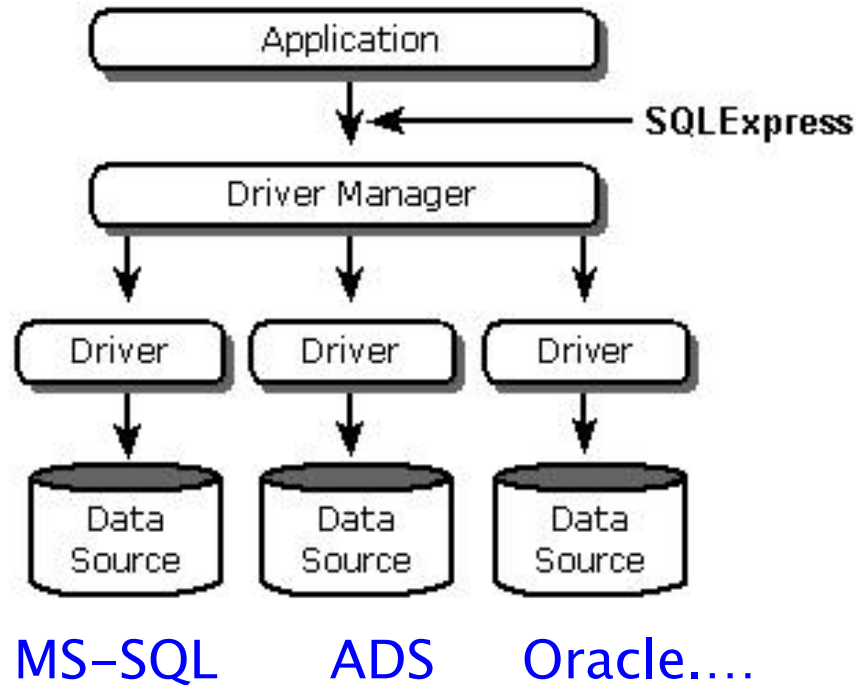
<http://www.sqlexpress.net>

- Object-oriented ODBC class library.
- ODBC: a vendor-independent API that uses SQL for accessing data.
- ODBC is a mature, widely supported standard.
- It's fast!

# SQLExpress for Xbase++

industrial strength C/S development tool

Architecture:



\* SQLExpress applications can simultaneously access data from more than one data source!

## Differences to DBF programming

- SQL is set based. Queries, updates and deletes typically have a WHERE clause that retrieve and/or modify zero or more records at one shot.
- SQL has no record number concept.
- DBF is record based. Always dealing with with one record at a time.
- SQL is transaction based. Concurrency is managed by the database server.
- With DBF's concurrency (record locking) must be managed by your application.
- With DBF you have to manage indexes in your app.
- Using SQL you don't need to worry about indexes.

## Clipper/Xbase syntax:

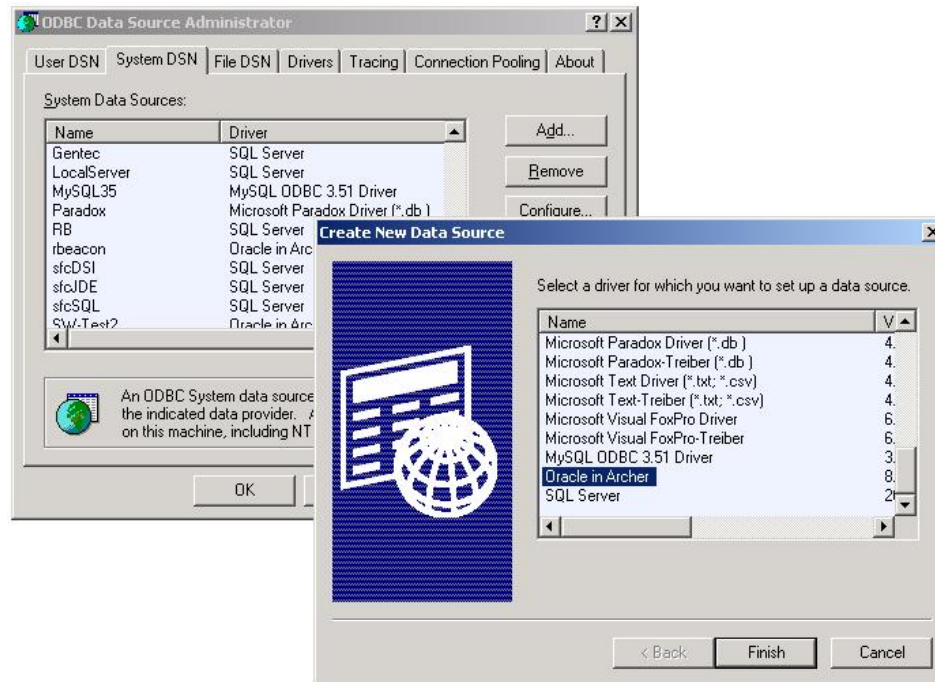
```
SET SOFTSEEK ON
USE ORDERS
SET INDEX TO ORDERS           // select index to use
...
...
SELECT ORDERS
SET ORDER TO TAG "PARTNO"     // index exp is: PARTNO+DTOS(DATE_REQ)
SEEK "123456"
WHILE !EOF() .AND. ORDERS->PARTNO == "123456"
    IF ORDERS->QTY_SHIPPED <> ORDERS->QTY_ORDERED
        ORDERS->RLock()         // you manage concurrency
        ORDERS->CANCELLED := .T. // one record at a time
        ORDERS->dbUnlock()
    ENDIF
SKIP
END
```

## SQLExpress syntax (simpler + faster):

```
// update many records with one statement
// don't care about indexes
// don't care about record locking
nRowsUpdated := oConnection:Execute(
    "UPDATE ORDERS SET CANCELLED=1 "+
    "WHERE PARTNO='123456' AND QTY_SHIPPED<>QTY_ORDERED")
```

## Connecting to data source using ODBC DSN:

Create ODBC DSN (control panel | administrative tools | data sources)



In Xbase++ program use DSN name to create connection.  
You may need a User ID and password to log into DBMS:

```
oConn := SQLConnection():New("MyDSN", "MyUID", "MyPWD")
```

## Connecting to data source without DSN:

1. Select ODBC driver to use. :GetDrivers() provides a list of all ODBC drivers available on client machine:

```
aDriverList := SQLConnection():GetDrivers()
```

2. Get name of desired ODBC driver (eg. "SQL Server") and create initial connect string. User will be prompted for rest:

```
cConnectString := "DRIVER=SQL Server;"
```

```
oConn := SQLConnection():New()
```

```
if oConn:DriverConnect(SetAppWindow():getHWND(), ;
```

```
    @cConnectString, SQL_DRIVER_PROMPT)
```

```
    ? "Full connect string is:", oConn:ConnectionString
```

```
else
```

```
    ? "unable to create ODBC connection"
```

```
endif
```

3. Now that you have a complete connect string, save it so that it can be used later to connect without prompting:

```
oConn := SQLConnection():New()
```

```
oConn:DriverConnect(nil, cConnectString)
```

## Now that you have a connection...

- Use [SQLStatement](#) to execute an SQL statement
- Use [SQLSelect](#) or [SQLDataSet](#) to execute an SQL query
- Query the DBMS catalog (data dictionary):
  - [SQLListColumnPrivileges](#)
  - [SQLListColumns](#)
  - [SQLListForeignKeys](#)
  - [SQLListPrimaryKeys](#)
  - [SQLListProcedureColumns](#)
  - [SQLListProcedures](#)
  - [SQLListSpecialColumns](#)
  - [SQLListStatistics](#)
  - [SQLListTablePrivileges](#)
  - [SQLListTables](#)
  - [SQLListTypeInfo](#)

## SQLStatement class

used for executing SQL statements that do not generate a result set. Eg: CREATE, DROP, ALTER, GRANT, UPDATE, DELETE, INSERT, etc...

```
TEXT INTO cSQL TRIMMED
```

```
CREATE TABLE Security (EmpNumber real not null,  
EmpShortName char(10), IsActive bit, JobFunction  
tinyint, DateCreate smalldatetime, ReportsTo real)  
ENDTEXT
```

```
oStmt := SQLStatement():new(cSQL, oConnection)  
if oStmt:execute() = SQL_XPP_ERROR  
    MsgBox("Error creating SECURITY table")  
    oStmt:destroy() // don't forget to do this!  
    Return  
endif
```

## SQLSelect class

- Inherited from SQLStatement.
- Used to manage a result set generating stored procedure or an SQL "SELECT" statement that retrieves a set of rows from a data source (a cursor).
- Use familiar Xbase commands to navigate through the result set, eg. :Skip, :GoTop, :GoBottom, :GoTo, :Bof, :Eof, :RecNo, :FieldName, :Seek, :Locate
- Use familiar Xbase commands to modify result set, eg. :FieldGet, :FieldPut, :Append, :Delete
- SQLSelect generates server side cursors (cursors that are managed by the data source). Application must be connected to the data source in order to navigate and modify the cursor.
- Cursors consume server resources. Try to create small cursors and don't keep them open for too long.

## SQLDataSet class

- Static client side cursor. Can be transmitted via Xb2.NET
- Provides same methods as SQLSelect class for navigation.
- Retrieves result set and immediately closes server side cursor.
- Once data is retrieved, application can disconnect from data source, navigate through the result set and modify data as required.
- Allows relations to be built between other SQLDataSet objects that may be connected to different data sources.
- Allows creating ad-hock, disconnected datasets, eg:

```
oData := SQLDataSet():new()  
oData:AddColumn("MyField1", {||""})  
oData:AddColumn("MyField2", {||0})  
oData:Append()  
oData:FieldPut("MyField1", "ABCdefg")  
oData:FieldPut("MyField1", 12345.67)  
// etc...
```

## Cursor Types

### FORWARD ONLY

- Non scrollable (can't move backwards).
- Fastest type of cursor, consumes least amount of server resources.
- Server does not build result set – data is fresh as new rows are fetched.

### STATIC

- Scrollable cursor containing a snapshot of result set.
- Does not reflect changes of underlying data after cursor is created.
- Server typically builds a temp table to hold result set – slow.

### KEYSET DRIVEN

- Scrollable cursor where data in result set is dynamic but rows are static.
- Can detect updated and deleted rows but not newly inserted ones.
- Server only builds a result set of key values – faster than static cursor.

### DYNAMIC

- Scrollable cursor capable of detecting rows updated, deleted, or inserted at the database by other users.
- Server does not build temporary result set.
- Returns first row faster than static or keyset cursor, but consumes more server resources when skipping through the result set.

## Concurrency

The manner in which a driver or data source checks for changes to a record made by others in order to maintain database integrity.

### SQL\_CONCUR\_READ\_ONLY

You cannot change the data.

### SQL\_CONCUR\_LOCK

Lock the current record or set of records in the result set so no one else can modify the data.

### SQL\_CONCUR\_ROWVER

Each row has a special time stamp column that is automatically updated when the row is changed. The system checks the timestamp column to see if changes were made by another user.

### SQL\_CONCUR\_VALUES

The current and cached values for all the columns are compared. If they are the same, then the row is modified.

## Transactions

Here's a simple way of implementing transactions with SQLExpress:

```
oConn:onError := {||break()}
BEGIN SEQUENCE
    oConn:BeginTransaction()
    oCursor := oConn:Cursor("select ...")
    oCursor:Execute()
    oCursor:FieldPut(...)
    oCursor:UpdateRow()
    //...
    // other database updates that must be done as a single "transaction"
    //...
    oCursor:Destroy()
    oCursor:CommitTransaction()
RECOVER
    oCursor:Destroy()
    oConn:RollbackTransaction()
END SEQUENCE
oConn:EndTransaction()
```

## SQL Tips

### Know your tools

Learn what can/can't be done

### Don't open large recordsets

Don't browse table with millions of records... change the user interface

### Don't SELECT \*

Specify only the columns you need

Reduce memory consumption & network bandwidth

Give query optimizer a chance to read columns from indexes

### Avoid use of cursors

Cursors consume memory, lock tables in weird ways and are slow, use:

INSERT INTO... SELECT FROM...

DELETE/UPDATE ... WHERE ... IN (SELECT...)

UPDATE... SET XXX=(SELECT...)

## SQL Tips

### Normalize tables

DBMSs were designed to be used with normalized databases

### Avoid NULL(able) columns (provide default values)

Consume an extra byte in each row and have more overhead when querying

SQL is harder to code, need to check if column is NULL whenever accessed

### Use referential integrity

Saves programming time since validation is done on server

### Don't create an index on the 'sex' column

Design indexes with the most sparse columns first and the least sparse columns last, e.g, Name + Province + Sex.

### Choose indexes wisely

Adding index speeds up SELECT but slows down UPDATE, INSERT, DELETE

Locks are held longer while indexes are updated

Create a clustered index for the most often used queries

## SQL Tips

### Use transactions but keep them short

Saves data integrity in case of app crash or failed statement

### Use parametrized queries / statements

data is safer

Allows DBMS to cache similar queries

### Beware of deadlocks

Always 'lock' tables in same order

Create separate read-only connection with TransactionIsolation level set to SQL\_TXN\_READ\_UNCOMMITTED

Consider adding "WITH (NOLOCK)" hint to SELECT statements so that shared locks are not issued

### Don't do SELECT max(ID) from Master when inserting in a Detail table

Will fail when two users are inserting data at the same time

Use SCOPE\_IDENTITY or IDENT\_CURRENT

## Simplify your life...

SQL is cleaner, easier and much more powerful than record based DBF programming :-)

<http://www.sqlexpress.net>