Using DMSQL

Paul Kimpel

2006 UNITE Conference
Session MCP-3023

Wednesday, 11 October 2006, 2:45 p.m.

Paradigm Corporation

Presentation Topics

- Background
  - What is DMSQL?
  - The Relational Data Model and SQL
  - Mapping DMSII Concepts to Relational
- Installing and Configuring DMSQL
  - Software installation
  - Configuring Releases, Users, and Resources
  - Preparing a DMSII data base for DMSQL
  - Using the DMSQL client utilities
- Host program interfaces
  - Module Language (ModLang)
  - Call Level Interface (CLI)

What is DMSQL?

- Officially, the "SQL Query Processor for ClearPath MCP"
- Dual aspects
  - A relational data base system built on top of DMSII
  - An SQL engine for existing DMSII data bases
- Entirely MCP-based and DMSII-oriented
  - No separate Windows-based processor
  - Accessible directly from MCP applications
  - Accessible from external systems over TCP/IP
- A partial revival of the SQLDB / SIM / DMS.View product of the mid/late-1990s

DMSQL Features

- ANSI-92 SQL compliance
  - Entry level X3.135-1992 feature set
  - With some extensions
- Language support
  - COBOL-74, COBOL-85
  - Algol
  - Java (via JDBC)
- Native APIs
  - Module Language (ModLang)
  - Call Level Interface (CLI)
  - Type-2 JDBC driver for Java

Features – Data Base Definition

- SQL Data Definition Language (DDL)
  - Defines an MCP-based relational data base
  - Generates a standard description file & DMSUPPORT
  - Syntax extensions for DMSII physical options
  - DDL use is very different from other DB products
- Import existing DMSII data base
  - Creates a relational mapping from a DMSII description
  - Supports SQL query and update for most structures
  - Supports SQL **GRANT**/**REVOKE** security features
  - Supports SQL referential integrity
  - Supports assignment of aliases and date fields
  - Supports ability to hide some DMSII fields from SQL

Features – Data Query & Update

- SQL Data Manipulation Language (DML)
  - **SELECT**, **INSERT**, **UPDATE**, **DELETE** statements
  - Numeric, string, **CAST**, and aggregate functions
  - Date expressions and intervals
  - Nested queries
  - **BETWEEN**, **LIKE**, **IN**, **EXISTS**, **NULL** predicates
  - Cartesian and ANSI join syntax
  - **GROUP BY**, **HAVING**, and **ORDER BY** clauses
  - **UNION** and **UNION ALL** operators
- Full DMSII transaction support
  - Transaction **COMMIT** and **ROLLBACK**
  - Supports only manual stores to the Restart Data Set – no record area capture at Begin- or End-Transaction

Features – Java-based Utilities

- Relational Design Center (RDC)
  - Imports SQL DDL or DMSII schemas
  - Manages entire SQL schema update process
  - Establishes aliases and date items
  - Interfaces to MCP-resident DMSQL schema administration software
- Query Design Center (QDC)
  - Workstation tool to test and analyze DML syntax
  - Useful for ad-hoc query and update
  - Supports a "query-by-example" (QBE) mode
  - Can save as files both query text and result set data

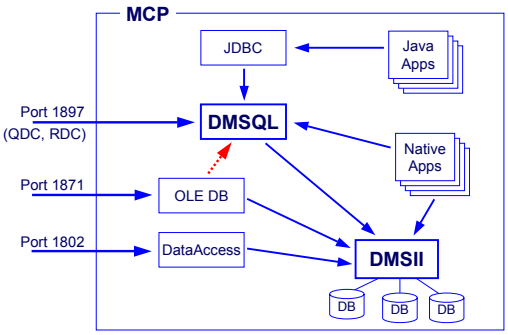
Features – Miscellaneous

- Internationalization (CCS)
- Isolation levels
  - Low (locking for updates, no read locking)
  - Medium (shared record locks for reads)
  - High (shared table locks for reads – serializable)

For Purpose of This Presentation...

- Will concentrate on DMSQL for the legacy environment (primarily COBOL)
  - Installing and configuring the software
  - Understanding relational mapping concepts
  - Preparing existing DMSII data bases for DMSQL
  - Using the new client tools (RDC, QDC)
  - Programming with the Module Language
  - Programming with the Call Level Interface
- Will mention, but otherwise largely ignore
  - Defining and maintaining SQL DDL data bases
  - Java and JDBC
  - The details of SQL query syntax

How DMSQL Relates to DMSII



The DMSII Data Model

- Loosely based on the CODASYL DBTG recommendations – early 1970s
- Primarily a hierarchical/network model
- Basically an extension of ISAM concepts
  - Very file/record oriented
  - Access to data requires procedural, record-at-a-time navigation
- Significant advantages over ISAM in
  - Centralized data structure (schema) definition
  - Record locking and deadlock detection
  - Data integrity and recoverability

The Relational Data Model

- Based on theoretical work of E.F. Codd
- Mathematically, a *relation* is a set of tuples
- A *tuple* is a finite sequence (ordered list)
  - Each object in the list is of a specific type
  - An n-tuple contains n objects
- A relation can be visualized as a table
  - Each row represents a tuple
  - Columns represent the objects of each tuple

The Relational Model (continued)

- Therefore, the relational model describes *all data* as two-dimensional tables
  - Rows (records)
  - Columns (fields, attributes, items, etc.)
- Rows must be unique (theoretically)
- Columns must be
  - Atomic (a single value – no arrays, no structures)
  - Of same "domain" (type) as all others in that column
- Data base structure (schema) is stored in a catalog that is part of the data base itself

Relational Access to Data

- Non-procedural
  - Uses a form of relational algebra
  - User describes the form and nature of the result
  - Not how to go about obtaining it
- Set oriented
  - Retrieval and update operate on groups of records
  - Not sequences of individual record retrievals
  - A single data base request generally operates on multiple "records" (rows), perhaps multiple tables
  - Result of a query is always another relation – a table with rows and columns

A Bit of History

- Codd's theoretical model published (1970)
- IBM's System R prototype (early 1970s)
  - Developed at IBM Research in San Jose, California
  - Introduced the SEQUEL language
  - Changed name to SQL after trademark dispute with Hawker-Siddeley Aircraft Company (1977)
- *Burroughs DMSII (1974)*
- Multics Relational Data Store (1976)
- Relational Software, Inc (1976)
- IBM SQL/DS and DB2 (early 1980s)

Why Use SQL?

- Provides a uniform, consistent way of thinking about data relationships
- Isolates data retrieval/update issues from the rest of the application
- Non-procedural data interface
  - Defines what the result is, not how to get it
  - Inherently multi-record
  - Adapts automatically to new indexes
  - Allows server to consider the whole request as a unit
  - Aids server-side optimization
  - Supports remote interfaces

Why Use SQL, continued

- Set-oriented approach to retrieval/update
- Absolutely great for
  - Ad-hoc query
  - Dynamic generation of a query based on parameters
  - Combining and retrieving data from multiple structures at the same time (joins, unions)
- It won the war of data base concepts
  - For better or for worse
  - SQL and the relational model are now the standard (and for some, only) way to think about and interface with data repositories

Issues with SQL and Relational Model

- It's almost more religion than technology
- A great concept, but not a universal one
- Works well for 90+% of business data
- Just about impossible for the rest of it
  - Variant data structures
  - Tree and network relationships
  - Non-scalar data types
- Lots of overhead for simple transactions – lookups and single-record updates

Mapping DMSII Concepts to Relational Concepts

Need for Mapping

- DMSQL requires a relational view of DMSII structures in order to work
- Mapping is the process of translating DMSII structures and behavior to a relational equivalent
- Not a perfect process
  - Some DMSII concepts map awkwardly to relational
  - A few DMSII concepts cannot be mapped at all

Disjoint Data Set Mapping

- Disjoint data sets are mapped as tables
- Global data items are mapped to a single-row table named GLOBALDATA
- Restart data set is mapped as a table
- Variable-format data sets
  - Each record type is mapped as a separate table
  - Tables named <data set name>`_FORMAT_`*n*
  - A view is generated for the union of all head fields
- Accesses for disjoint ordered data sets must have **`NO DUPLICATES`** specified

Embedded Data Set Mapping

- Modeled as separate table from master
- Limited to two levels of nesting
- Fields from the master's primary key are implicitly prepended to the embedded fields
  - Become part of the embedded table's primary key
  - Used by SQL to access the embedded record by implicitly accessing the master record first
- Embedded span sets cannot be modeled, but are used internally by DMSQL

Occurring Item/Group Mapping

- Mapped as separate tables, similar to embedded data sets
  - Fields from the master's primary key are implicitly pre-pended to the occurring fields
  - Primary key for mapped table consists of master's primary key fields + occurrence #
- Occurring items in global data are mapped as a fixed-length string
- Mapped table is (of course) fixed in size
  - Option to manage empty array slots automatically or manually on **`INSERT`** and **`DELETE`**
  - DMSQL handles the automatic slot management

Index Set Mapping

- Disjoint span sets and accesses are mapped as SQL indexes *and* SQL views
  - Keys are mapped as is
  - Key data is not mapped
  - Can have internationalized key values
  - Queries using the view force DMSQL to use that index
- Disjoint subsets are mapped as SQL views
  - Subset condition mapped as view **`WHERE`** clause
  - Cannot use these views for **`INSERT`** or **`UPDATE`**
- Not modeled – manual subsets, embedded sets, bit vectors, unordered lists

Data Item Mapping

DMSII Type	SQL Type	Comments
ALPHA(n)	char(n)	EBCDIC collation
ALPHA(n) SIZE VARYING	char varying(n)	EBCDIC collation
BOOLEAN	integer	never NULL
FIELD(n≤39)	integer	value verified as 0..2^n-1
FIELD(n>39)	real (sort of)	read-only
FIELD(of n BOOLEANs)	n integer fields	never NULL
FILLER	(not modeled)	
GROUP	(not modeled, unless occurring)	
NUMBER(n) or (Sn)	numeric(n)	always signed
NUMBER(n,d) or (Sn,d)	numeric(n,d)	always signed
REAL	real	
REAL(n) or (Sn)	integer	always signed
REAL(n,d) or (Sn,d)	numeric(n,d)	always signed

Control Item Mapping

DMSII Type	SQL Type	Comments
AGGREGATE(n) or (Sn)	numeric(n)	read-only, always signed
AGGREGATE(n,d) or (Sn,d)	numeric(n,d)	read-only, always signed
COUNT	integer	read-only
(All types of links)	(not modeled)	
POPULATION	integer	read-only
RECORD TYPE	integer	
RECORD SERIAL NUMBER	real	read-only, mapped only if declared in DASDL

DASDL Item Attributes	SQL Attributes	Comments
REQUIRED	NOT NULL	
INITIALVALUE	DEFAULT	value on INSERT

Referential Integrity

- Establishes a record-level constraint
  - Specified columns in one row of a table must identify equivalent columns in another row
  - Rows can be in the same or different tables
- Referring table defines a "foreign key"
- Constraints can be added for existing DMSII data bases without affecting non-DMSQL applications
  - Integrity only checked on **`INSERT`**, **`UPDATE`**, and **`DELETE`** through DMSQL processor
  - Can result in consistency problems if updates are also done by native applications

Mapping Dates

- ANSI SQL has built-in support for dates, but DMSII does not support dates
- DMSQL permits mapping 1-3 DMSII fields into a virtual date column
- Single-field date mappings
  - `YYYYMMDD`, `YYMMDD`, `YYYYDDD`, `YYDDD`
  - `MMDDYY`, `MMDDYYYY`
  - `DDMMYY`, `DDMMYYYY`, `DDDYYYY`, `DDDYY`
- Multi-field date mappings
  - Year + day of year
  - Year + month + day

SQL Views

- A view defines a new table composed of rows and columns from other tables
  - The new table does not physically exist
  - Only the specification for the view is stored
  - Basically, a view is a stored query
- Views can be used in SQL statements in most places where tables can be used
- Separate security can be assigned to restrict the rows and/or columns a user can access

SQL Security

- ANSI SQL defines a "grant" mechanism to control access to data
- Grants can be based on
  - Users
  - Tables and views
  - Columns (for restricting update only)
  - Statements: `SELECT`, `UPDATE`, `INSERT`, `DELETE`
- Grants are only effective within the DMSQL environment
- All standard DMSII security (guardfiles, etc.) also applies to DMSQL access

Installing and Configuring the DMSQL Software

System Requirements

- MCP 10.1 with…
  - DALICENSESUPPORT 051.1A.3 or later
  - DMSII 051.1A.10 or later (or 050.1A.44 or later)
  - DMSQL 051.1A.9 or later
  - MCP Java 1.4.2A.30 or later
  - SIMPLEINSTALL 051.1A.6 or later
- MCP 11.0 or later
- Strongly recommend you use SIMPLEINSTALL or Install Center

DMSQL DSS Commands

- **`NA DMSQL HELP`** [<command>]
- Controlling the DMSQL server
  - **`NA DMSQL +`**
  - **`NA DMSQL -`**
  - **`NA DMSQL QUIT NOW`**
- Diagnostic commands
  - **`NA DMSQL STATUS`** [<worker id>]
  - **`NA DMSQL LOG`** [**`RELEASE`**]
  - **`NA DMSQL TRACE`**
    - [**`+`** / **`ON`** / **`-`** / **`OFF`** / **`RELEASE`**]
    - [<worker ID>]

Client Software Installation

- Two utilities for Windows environments
  - Relational Design Center (**RDC**)
  - Query Design Center (**QDC**)
- Requires Java 1.4 or higher on workstation
- Install from the MCP Installs share or download from support.unisys.com
  - Standard Windows MSI file
  - By default, only QDC is installed
  - Must explicitly select RDC to install it
  - Installation Assistant will automatically install the Java JRE if version 1.4 or higher if not already present
  - Recommend using MCP 11 versions with MCP 10.1

DMSQL Configuration Concepts

- Releases
  - Support for multiple releases of DMSQL running on the same server at the same time
  - Optionally override standard names for SL libraries
  - Up to 255 releases can be defined
- Users
  - Assigned to releases
  - Limits on number of data base operations per query
- Resources
  - DMSQL connects to *resources*, not *data bases*
  - One resource consists of 1–5 DMSII data bases
  - Can be a mix of read-only or updatable access modes

*DMSQL/CONFIG File

- Global configuration for DMSQL
  - Must be under same usercode and family as `SYSTEM/SQL/CONFIG` codefile
  - CANDE `SEQDATA` format
- Configuration commands
  - `RELEASE`
  - `USER`
  - `DEFAULT`
  - `PROGRAM`
  - `RESOURCE`

Sample Release Specs

```
RELEASE MCPSTD (PACK=DISK)


RELEASE MCP11FIXED = MCPSTD (PACK=TEMP)
  SL DMSQLPARSER =       *SYSTEM/SQL/011-01-4/PARSER


RELEASE MCP11TEST (PACK=MCPTEST)
  SL DMSQLDRIVER =       *SYSTEM/SQL/DRIVER
  SL DMSQLSUPPORT =      *SYSTEM/SQL/SUPPORT
  SL DMSQLFILESUPPORT =  *SYSTEM/SQL/FILESUPPORT
  SL DMSQLDMSIIMAPPER =  *SYSTEM/SQL/DMSIIMAPPER
  SL DMSQLCOBOLDESC =    *SYSTEM/SQL/COBOLDESC
  SL DMSQLADMIN =        *SYSTEM/SQL/ADMIN
  SL DMSQLSCODESUPPORT = *SYSTEM/SQL/SCODESUPPORT
  SL DMSQLPARSER =       *SYSTEM/SQL/PARSER
  SL DMSQLCATALOG =      *DESCRIPTION/SQLDIR/DMSQL-CATALOG
  SL DMSQLSCODE =        DMSQLSCODE
```


Configuring User & Program Access

- **USER**
  - Specifies MCP user capabilities within DMSQL
  - Users optionally are associated with a Release
  - Users optionally have a limit on data base operations
- **DEFAULT** specifies a Release and limit for all users not otherwise mentioned
- **PROGRAM**
  - Specifies MCP codefile data base operation limits
  - Overrides any applicable **USER** limits when this program is used

Sample USER & PROGRAM Specs

```
USER PAUL = MCP11TEST

USER FRED LIMIT 1500 = MCPSTD

USER PROD, SYSTEMS, MSMITH
    LIMIT UNLIMITED = MCPTEST

USER DEMO LIMIT 500 = MCPSTD


DEFAULT LIMIT 5000 = MCPSTD


PROGRAM (PROD)OBJECT/GL/EXTRACT ON FINANCE
    LIMIT=UNLIMITED

PROGRAM (TEST)OBJECT/SQL/DEMO ON WORKPACK
    LIMIT=2000
```

Configuring Resources

- Resources define the entities DMSQL can open and access
- Each resource represents 1–5 DMSII or SQL data bases
- Optionally specifies a CCS for the resource
- Each data base in a resource can have a separate access mode
  - **INQUIRY**
  - **UPDATE**

Sample RESOURCE Specs

```
RESOURCE AVRDB =
    (ONLINE)AVRDB ON OPS


RESOURCE TESTDB = CCS SPANISH
    (BETA)FINDB ON WORKPACK (MODE=UPDATE)


RESOURCE BUNCH =
    (PROD)GLDB ON GLPACK (MODE=INQUIRY),
    (PROD)ARDB ON AR01 (MODE=UPDATE),
    (PROD)APDB ON FINANCE (MODE=UPDATE)
```


Establishing the Configuration

- After updating ***DMSQL/CONFIG**, must inform DMSQL the file has changed
- From CANDE or MARC
  - Run ***SYSTEM/SQL/CONFIG**
  - Program verifies **CONFIG** file syntax
  - Writes **CONFIG** file name and status to a remote file
- Changes take effect the next time a connection is established
- Changes are not visible to any currently-connected users

Preparing a Data Base for DMSQL

- Two alternatives
  - Define a new relational schema using SQL DDL
  - Create a schema from an existing DMSII data base
- Steps:
  - Update Resources and Users in **`*DMSQL/CONFIG`**
  - Use RDC to establish relational schema and catalog
  - As necessary, use RDC to
    - Assign alias names to tables, indexes, columns
    - Suppress columns, map columns to dates
    - Add relational integrity constraints for tables
    - Establish **`GRANT`** security
    - Define SQL views

Defining a Relational Data Base

- Prepare SQL DDL statements in a Windows text file
- Run the Relational Design Center
  - Add a server, if necessary
  - Right-click on server, select "**Create SQL Database**"
  - Fill in
    - Data base title (DMSII control file location)
    - DDL text file name (click Browse if necessary)
    - MCP pack family
    - MCP usercode and password
  - Click "**Create**"
  - Any errors will display in the Output window

Creating a Schema from DMSII (1 of 3)

- Run the Relational Design Center
  - Add a server, if necessary
  - Right-click server, select "**Import Relational Schema**"
  - Fill in
    - Data base name (i.e., title – control file location)
    - Usercode and password
    - Description file title [optional]
    - Access control – check box if is to be read-only
    - Click "**Import**"
  - Errors are reported in the "Output" window
- The default schema is now established

Creating from DMSII (2 of 3)

- If desired, modify the default schema
  - Assign name aliases to tables, indexes, columns
  - Hide tables, indexes, columns
  - Create date columns from other columns
  - Add referential integrity constraints for tables
  - Add **GRANT** security
  - Create views
- When finished with schema updates
  - Right-click data base and select "**Apply Schema Modifications**"
  - Relational mapping will be stored on MCP host as **DESCRIPTION/**<dbname>**/DMSQL-CATALOG**

Creating from DMSII (3 of 3)

- Mapping dates creates virtual columns
  - Date columns can be updated
  - Original columns are marked read-only for DMSQL
  - Can hide original columns if desired
- Notes
  - DASDL must specify **DMSUPPORT**, **ACR**, and **DMRECOVERY** titles with explicit usercode and family
  - DASDL must specify **INDEPENDENTTRANS** option
  - If using guardfiles, give **SYSTEM/SQL/WORKER** appropriate access to data base files
  - If no **GRANT**s are specified, all users get full access
  - Schema modifications can be saved in Windows and reloaded later, without applying them to the MCP host

An Alternate to the RDC

- RDC is just a GUI front end to schema utilities that run in the MCP environment
- SQL data bases and DMSII schema imports can also be maintained using ***SYSTEM/SQL/ADMIN**
  - Not as easy or convenient as the RDC
  - Documented in the DMSQL *Installation and Operations Guide* and *Programming Guide*

Using the Query Design Center (QDC)

Query Design Center (QDC)

- Java GUI to develop and test SQL DML
  - Somewhat like Microsoft's Query Analyzer for SQL Server or MSQRY32.exe for ODBC
  - Can connect to multiple resources simultaneously
  - Supports multiple simultaneous query windows
  - Supports **`SELECT`**, **`UPDATE`**, **`INSERT`**, **`DELETE`**
  - Supports **`COMMIT`**/**`ROLLBACK`** and isolation levels
- Three main modes
  - **Analyze Query** – enter SQL directly and execute
  - **Design Query** – build SQL using QBE wizard
  - **View Catalog** – browse relational schema

QDC Analyze Query Mode

- Select File>Connect>Analyze Query…
  - Fill in server, user, and resource info
  - Opens a two-pane sub-window
  - Enter SQL text (or highlight part of it) in top window
  - Press F5 to execute, F4 to parse ("prepare") only
  - View tabular result set in bottom window
- Options
  - Save and reload SQL query text
  - Save result set as a columnar Windows text file
  - Select debugging output (Qgraphs/Qdumps)
  - Select query isolation level
- Demo

QDC Design Query Mode

- Select File>Connect>Design Query…
  - Fill in server, user, and resource info
  - Opens a four-pane sub-window
    - Table and join diagram
    - QBE column/sort/condition grid
    - Generated SQL text
    - Tabular result set
  - Press F5 to execute or Alt+P to parse
- Save/reload options same as Analyze Query – no debug or isolation-level options
- Demo

QDC View Catalog

- Select View>Catalog…
  - Enter server and usercode/password info
  - Opens a two-pane browse sub-window
- Basics
  - Click on items in the left-hand tree pane
  - View item details in right-hand pane
- Advanced
  - ***Right***-click on tables in the left-hand pane
  - View more detailed information on columns, primary keys, foreign keys, statistics in a separate window
- Demo

Transactions in the QDC

- By default, each **`UPDATE`**, **`INSERT`**, or **`DELETE`** is a separate transaction
- Select Options>Manual Commit Mode…
  - Supports multiple SQL statements as one transaction
  - Enables **`COMMIT`** and **`ABORT`**/**`ROLLBACK`** statements
  - Enables creation of and rollback to "savepoints" within a larger transaction
- Transaction control commands are also available by right-clicking in the QDC SQL pane

QDC Hints

- Query isolation level
  - Default is **MEDIUM** (locks rows on read)
  - Usually much more efficient if set to **LOW** (dirty reads)
- **qdc.properties** file
  - Controls GUI configuration, including display font
  - Usually stored under
    C:\Documents and Settings\<user name>\ .dmsqlprovider\qdc.properties
- No way to stop a query once started, except by File>Disconnect
- No way to limit the number of rows

Alternative to the QDC

- Run ***SYSTEM/SQL/DMQUERY** from CANDE or MARC
  - Must be run from the MCP host having the data base
  - Uses a remote file interface
  - Documented in *Installation and Operations Guide*
- Open the data base:

```
OPEN DMDEMO (USER=DEMO, FAMILY=TEST,
   MODE=UPDATE)
```

- Commands
  - **SELECT**, **UPDATE**, **INSERT**, **DELETE**
  - Transaction control: **COMMIT**, **ROLLBACK**, …
  - **DEFINE**, **DO**, **SET** options, a few others

Using the Module Language

Invoking SQL From an MCP Program

- The DMSII API is easy and convenient
  - Record-at-a-time retrieval
  - Record and field layouts are known to the compiler
- SQL results sets are more difficult to use
  - Rows and columns are determined by the query
  - Fields are not known to the compiler in advance
  - SQL values must be converted to host language types
  - A more dynamic programming interface is needed
- Three SQL APIs for MCP host languages
  - Module language [ModLang] (COBOL, Algol)
  - Call Level Interface [CLI] (COBOL, Algol)
  - JDBC (Java only)

The Module Language (ModLang)

- Based on ANSI SQL-92 Module Language
  - Standard COBOL module type
  - Unisys-specific Algol module type
- Modules define two types of entities
  - Cursors
  - Procedures
- Each module compiles to an MCP server library program
  - Generates a wrapper for lower-level calls to DMSQL
  - Host language programs call procedures as normal library subroutines, passing parameters

Structure of a Module

- Module header
  - $ options – **`ANSI`**, **`ISOLATION LEVEL`**
  - Optional module name
  - Language specification – **COBOL** or **ALGOL**
  - Optional data base declarations
  - Authorization (MCP "owner" usercode)
- Module body
  - **`CURSOR`** declarations
  - **`PROCEDURE`** declarations
  - Declarations can be intermixed
  - Cursors must be defined before procedures that reference them

Sample Module Header

```
$ RESET ANSI SET ISOLATION LEVEL LOW
MODULE TEST_QUERY
LANGUAGE COBOL
DATABASE (DEMO)TESTDB ON WORKPACK
AUTHORIZATION DEMO
```

Optional

Owner – usercode under which module will be compiled

ModLang Cursors

- Define multi-row result sets for queries
  - Support sequential processing of result rows
  - Allow record-at-a-time processing, similar to DMSII **FIND NEXT**
- Cursor functions
  - **DECLARE** (specifies a **SELECT** statement)
  - **OPEN** (optionally passes parameters to query)
  - **FETCH** (reads the next row)
  - **UPDATE** (the current row)
  - **DELETE** (the current row)
  - **CLOSE**

Sample Cursor Declarations

- Invariant query

```
DECLARE CURSOR C1 FOR
  SELECT NAME, DOB, ZIPCODE FROM PERSONS
  WHERE STATUS='P' AND GENDER IS NULL
  ORDER BY ZIPCODE, NAME
```

- Parameterized query

```
DECLARE CURSOR C2 FOR
  SELECT NAME, DOB, ZIPCODE FROM PERSONS
  WHERE STATUS=:QSTAT AND GENDER IS NULL
  ORDER BY ZIPCODE,NAME
```

Parameter

ModLang Procedures

- Encapsulate SQL statements
  - Callable directly by the host language program
  - Parameters map between SQL data types and host language data types
  - Special parameters for
    - Error results
    - NULL value indicators
    - COMS input/output headers
- Only one SQL statement per procedure
  - SQL DML statements (**SELECT**, **INSERT**, etc.)
  - Cursor operations
  - Transaction control (**COMMIT**, **ROLLBACK**)

Procedure Parameters

- Parameter types
  - :<parameter name> <data type>
  - **SQLCODE**
  - **SQLSTATE**
  - **COMS_IN**
  - **COMS_OUT**
- Each procedure must use *either* **SQLCODE** *or* **SQLSTATE** for result reporting
- **COMS_IN**, **COMS_OUT**
  - COMS input and output headers
  - Must be first parameter passed to a procedure
  - Used only with synchronized recovery transactions

Host Language Data Types

- COBOL modules
  - **NUMERIC**
  - **NUMERIC**(n)
  - **NUMERIC**(n,d)
  - **CHARACTER**(n) or **CHAR**(n)
- Algol modules
  - **NUMERIC**
  - **CHARACTER**
  - **INTEGER**
  - **SMALLINT**
  - **REAL**
  - **DOUBLE**

COBOL Data Type Conversion

- NUMERIC (all forms)
  - `PIC S9(...)`
    - Must be `USAGE DISPLAY` (the default)
    - Must have `SIGN LEADING SEPARATE`
  - Always signed
  - Precision and scale must match COBOL picture
  - `NUMERIC` without "`()`" implies `PIC S9(11)`
- CHARACTER(n)
  - `PIC X(n) USAGE DISPLAY`
- SQLCODE – `PIC S9(9) COMP`
- SQLSTATE – `PIC X(5)`

Error Parameters

- SQLCODE
  - Numeric result code for SQL statement
  - 0 = completed successfully
  - 100 = warning (no data or no more data)
  - <0 = failure
- SQLSTATE
  - 5-character result string
  - `SQLSTATE (1:2)` = error category
  - `SQLSTATE (3:3)` = error subcategory
- Call `RETURN_ERROR_MESSAGE` in module library to get a formatted message

ModLang Procedure Statements

- Cursor manipulation
  - `OPEN` (passes parameters to cursor)
  - `FETCH` (retrieves current row values)
  - `UPDATE` (current cursor row only)
  - `DELETE` (current cursor row only)
  - `CLOSE`
- Non-cursor DML statements
  - `SELECT` (single row result only)
  - `INSERT` (single or multiple row insertion)
  - `UPDATE` (multiple rows can be affected)
  - `DELETE` (multiple rows can be affected)
- Transaction control statements
  - `COMMIT WORK`
  - `ROLLBACK WORK`
  - `SAVEPOINT`

Sample ModLang Procedures

```
PROCEDURE C2_OPEN (
      :STAT     CHAR(1),
      SQLCODE);
   OPEN C2;

PROCEDURE C2_FETCH (
      :NAME     CHAR(30),
      :DOB      NUMERIC(8),
      :ZIP      CHAR(10),
      :NOZIP    NUMERIC,
      SQLCODE);
   FETCH C2 INTO :NAME, :DOB, :ZIP:NOZIP;

PROCEDURE C2_CLOSE (SQLCODE);
   CLOSE C2;
```

An "indicator" – indicates if NULL

Sample Procedures (continued)

```
PROCEDURE C2_FIND_NAME (
      :ID       NUMERIC(9),
      :NAME     CHAR(30),
      SQLCODE);
   SELECT NAME:NAME FROM PERSONS
   WHERE PERSONID=:ID;

PROCEDURE C2_UPDATE_DOB (
      :DOB      NUMERIC(8),
      SQLCODE);
   UPDATE PERSONS SET DOB=:DOB
   WHERE CURRENT OF C2;

PROCEDURE C2_COMMIT (SQLCODE);
   COMMIT WORK;
```


ModLang Transactions

- A transaction begins implicitly
- Continues until procedure is called for
  - **COMMIT WORK**
  - **ROLLBACK WORK** without the **SAVEPOINT** option
- Transaction must be *explicitly* ended or an *implicit* rollback will occur at end of task
- Savepoints
  - Creates a checkpoint within a transaction
  - Analogous to a DMSII **SAVE TRANSACTION POINT**
  - **ROLLBACK WORK TO SAVEPOINT** is analogous to DMSII **CANCEL TRANSACTION POINT**

Compiling Modules

- Run ***SYSTEM/SQL/MODLANG("...")**
  - Parses module source code
  - Initiates DMALGOL compile to generate the library
- String parameter
  - List of data base titles used by the module
  - Multiple titles must be separated by ";"
  - Must pass an empty string if **DATABASE** is used instead in the module header
- File assignments
  - **INFILE** module source (**SEQDATA**)
  - **ERRORFILE** defaults to a remote file
  - **CODE** title of resulting library codefile

Calling ModLang Libraries

- Use standard library declaration and call syntax in the host language
- Each ModLang procedure becomes an entry point in the library
  - Parameters are 1-for-1
  - Error results are returned via the required **SQLCODE** or **SQLSTATE** parameter
- Cursors must be explicitly opened & closed
- Otherwise, no explicit initialization or termination library calls are required

Sample ModLang Calls

```
01  W-SQLCODE     PIC S9(9) COMP.
01  W-STATUS      PIC X(1).
01  W-NAME        PIC X(30).
01  W-DOB         PIC S9(8) SIGN LEADING SEPARATE.
01  W-ZIP         PIC X(10).
01  W-ZIP-NULL    PIC S9(11) SIGN LEADING SEPARATE.

CHANGE ATTRIBUTE TITLE OF "MODLIB" TO
    "(DEMO)OBJECT/MODULE/TEST ON TEMP.".
MOVE "A" TO W-STATUS.
CALL "C2_OPEN IN MODLIB" USING
    W-STATUS, W-SQLCODE.
CALL "C2_FETCH IN MODLIB" USING
    W-NAME, W-DOB, W-ZIP, W-ZIP-NULL,
    W-SQLCODE.
CALL "C2_CLOSE IN MODLIB" USING W-SQLCODE.
```

Issues with the Module Language

- Efficient
- Much easier to use than CLI
- Much less flexible than CLI
  - Queries must be specified and compiled in advance
  - Code must be maintained in more than one place
  - Parameter mapping can be tedious and error-prone, especially for long parameter lists
- Problems on MCP 10.1 and 11.0
  - **`CHARACTER`** data type not recognized for COBOL
  - Missing **`SYMBOL/SIM/PROPERTIES`** file (apparently can use **`SYMBOL/SQL/PROPERTIES`** instead)
  - Missing 10.1 library codefiles at run time

Using the Call Level Interface (CLI)

The DMSQL Call Level Interface

- Very flexible, dynamic, and general purpose interface to DMSQL
  - Everything can be configured at run time
    - No pre-compiling or static definitions
    - Works with any DMSQL-capable data base
  - Catalog (schema discovery) capabilities
  - Suitable for building tools and ad hoc interfaces
  - Should work with any library-capable language
  - Low-level API – a bit of a challenge to use
- Based on ANSI SQL-92 CLI specification
  - Significant differences in parameter types
  - Uses offsets within buffers rather than C-like pointers

Call Level Interface API

- Environment Control
  - SQLAllocConnect
  - SQLAllocEnv
  - SQLAllocStmt
  - SQLDriverConnect
  - SQLGetConnectOption
  - SQLSetConnectOption
  - SQLDisconnect
  - SQLFreeConnect
  - SQLFreeEnv
  - SQLFreeStmt
- Query Configuration
  - SQLPrepare
  - SQLBindParameter
  - SQLNumParams
  - SQLDescribeParam
  - SQLBindCol
  - SQLNumResultCols
  - SQLDescribeCol
  - SQLColAttributes
- Miscellaneous
  - SQLError
  - SQLMemberOf
  - SQLDiagnose

Call Level Interface API (continued)

- Query Execution
  - SQLExecDirect
  - SQLExecDirectWith-Parameters
  - SQLExecute
  - SQLExecuteWith-Parameters
  - SQLRowCount
- Result-set Control
  - SQLFetch
  - SQLFetchBoundCol
  - SQLGetData
  - SQLGetLength
  - SQLGetSubstring
- Transaction Control
  - SQLTransact
  - SQLSavepoint
  - SQLRollbackToSavepoint
- Catalog Routines
  - SQLTables
  - SQLColumns
  - SQLForeignKeys
  - SQLPrimaryKeys
  - SQLStatistics
  - SQLGetInfo
  - SQLGetInfoInt

CLI Concepts and Objects

- Environment object
  - The global data object for the CLI
  - Only one allowed per task
- Connection object
  - Defines access to a DMSQL *resource*
  - Can be opened or closed at will
  - Only one *active* connection allowed per Environment
- Statement object
  - The data object supporting an SQL command
  - Supports preparation (pre-compiling) and cursors
  - Can have multiple statements per Connection
  - Can be reused and dynamically reconfigured

CLI Concepts (continued)

- Supplied include files
  - `*SYMBOL/SQL/CLI/PROPERTIES/ALGOL`
  - `*SYMBOL/SQL/CLI/PROPERTIES/COBOL`
- All parameters are passed *by reference*
  - Must always pass variables
  - No literals, no expressions (sorry, Algol programmers)
- All SQL **`SELECT`** queries generate a cursor, even if result has only one row
- All others (**`INSERT`**, **`UPDATE`**, **`DELETE`**) generate only a count of rows affected

Programming for the CLI (1 of 3)

- Global initialization
  - Connect to "**`DMSQLCLI`**" library by function
  - Allocate an Environment
  - Allocate a Connection for the Environment
  - Open the Connection (by DMSQL resource name)
- Query preparation
  - Allocate one or more Statements for the Connection
  - Specify a query
    - Construct the SQL command text
    - Bind parameters, if any
    - Optionally "prepare" query prior to execution
    - Optionally bind result-set columns

Programming for the CLI (2 of 3)

- Query Execution
  - Two methods:
    - Execute directly (no "prepare" first)
    - Execute a prepared query
  - Separate API calls for parameterized and non-parameterized queries
- Process query results
  - For non-cursor operations, optionally call **`SQLRowCount`** for the number of rows affected
  - For cursor-based results, two options:
    - Fetch rows, then get result columns individually
    - Bind columns before execute, then fetch a complete, formatted record area for each row

Programming for the CLI (3 of 3)

- After processing the query results
  - Can re-execute the query
    - Modify parameter values first, if necessary
    - Prepared queries can be efficiently re-executed
    - Directly-executed queries must be re-parsed first
  - Can rebind parameters and result columns
  - Can reuse a Statement object and modify the query
  - Can free (deallocate) the Statement object
- At termination
  - Free all Statement objects
  - Close the Connection
  - Free the Connection object
  - Free the Environment object

Primary Query Processing Routines

- Query preparation
  - `SQLPrepare`
  - `SQLBindParameter`
  - `SQLBindCol`
- Query execution
  - Direct execution – no prior query preparation
    - `SQLExecDirect`
    - `SQLExecDirectWithParameters`
  - Prepared query
    - `SQLExecute`
    - `SQLExecuteWithParameters`

Primary Routines (continued)

- Results retrieval
  - `SQLFetch`
  - `SQLFetchBoundCol`
  - `SQLGetData`
  - `SQLGetRowCount`
- Transaction control
  - Begin-Transaction is implicit
  - `SQLTransact`
    - `COMMIT` option
    - `ROLLBACK` option
  - `SQLSavePoint`
  - `SQLRollBackToSavePoint`

Query Parameters

- Queries optionally can have parameters
  - Designated by a "?" in the SQL query text
  - Must "bind" values to parameters prior to execution
- Values and "indicators" must be stored in a single record area (Algol array row)
  - Binding calls specify the data type, size, and 0-relative offset of the value and indicator within the record area
  - Record area with parameter values and indicators is passed in the **`ExecuteWithParameters`** call
- Indicators ("pcbValues")
  - 6-byte integer word – COBOL **`PIC S9(11) BINARY`**
  - Indicates length of value (>=0) or that value is null (-1)

Query Parameters (continued)

- Some indicators can be optional
  - Indicator and value length required for character data
  - Optional for other types – can pass **`-1`** for the offset
- Example (other layouts are possible)

```
01 WPA-PARAM-AREA.
   02 WPA-INDICATORS.
      03 WPA-PARAM1-IND PIC S9(11) BINARY.
      03 WPA-PARAM2-IND PIC S9(11) BINARY.
   02 WPA-VALUES.
      03 WPA-PARAM1     PIC X(8).
      03 WPA-PARAM2     PIC S9(5)  COMP.
```


Retrieving Result Set Columns

- Two ways to retrieve values from a cursor
  - Use column-at-a-time retrieval
    - Call **`SQLFetch`** to get the next row
    - For each column in the row, call **`SQLGetData`**
  - Use bound columns
    - Specify offsets to columns and indicators in a record area beforehand
    - Similar to binding of parameters
    - Call **`SQLFetchBoundCol`** to get the next row
    - CLI will automatically format all bound values into the record area passed in the fetch call
  - Can mix both methods for the same query
- Binding is usually much more efficient

Parameter and Column Data Types

- SQL data types are somewhat different from host language data types
- CLI performs data conversion for parameter and column values
- Data types and sizes are specified in
  - `SQLBindParameter` calls
  - `SQLBindCol` calls for result sets
  - `SQLGetData` calls
- Indicators for bound result set columns
  - Report length of data available in the cursor
  - May be more or less than amount formatted in record

CLI Data Conversions

	COBOL Data Type				
SQL Type	**PIC X(...)**	**COMP**	**BINARY**	**REAL**	**DOUBLE**
Char	X				
VarChar	X				
Date	X				
Decimal	X	X			
Numeric	X	X			
Smallint	X	X	X		
Integer	X	X	X		
Float					X
Real				X	X
Double					X

From DMSQL Programming Guide, Section 9

Additional CLI Capabilities

- Obtain parameter info
  - `SQLNumParams`
  - `SQLDescribeParam`
- Obtain result set column info
  - `SQLNumResultCols`
  - `SQLDescribeCol`
  - `SQLColAttributes`
- BLOB and CLOB support
  - `SQLGetLength`
  - `SQLGetSubstring`

Additional Capabilities (continued)

- Obtain data base catalog info
  - `SQLTables`
  - `SQLColumns`
  - `SQLPrimaryKeys`
  - `SQLForeignKeys`
  - `SQLGetInfo`, `SQLGetInfoInt`
  - `SQLStatistics`
- Errors and Diagnostics
  - `SQLError` – obtain text error messages
  - `SQLDiagnose` – obtain diagnostic info

Example CLI Programs

- CLI coding is far too complex and voluminous to attempt to show here
- Check out some examples
  - `*EXAMPLE/SQL/CLI/=` on the release media
  - Printed examples in the DMSQL *Programming Guide*
  - `WASPDB` example on our web site
    `http://www.digm.com/UNITE/2006`

A Critique of DMSQL

- General design and functional scope appears to be good
- For the initial MCP 10/11 releases, this is an immature product
  - Java-based GUIs are nice, but could use refinement
  - Module Language is currently unusable
  - SQL DDL capability is very weak
  - Documentation needs *a lot* of work
  - Performance is okay, but needs to get better
- All together, a reasonable first effort

Things I Would Like to See

- A case or conditional expression
- Derived tables

  `SELECT … FROM (SELECT … FROM … ) …`
- Set-oriented, multi-row update

  `UPDATE … FROM` <table expression>
- In QDC
  - Auto-width sizing of result set columns
  - Ability to limit the number of rows returned
  - Ability to gracefully terminate a query
  - Better visual signals that a query is running
  - Save result sets to formats other than columnar text

References

- *SQL Query Processor for ClearPath MCP Installation and Operations Guide* (3850 8206-000)
- *SQL Query Processor for ClearPath MCP Programming Guide* (3850 8214-000)
- This presentation and examples

  `http://www.digm.com/UNITE/2006`

END

Using DMSQL

2006 UNITE Conference

Session MCP-3023