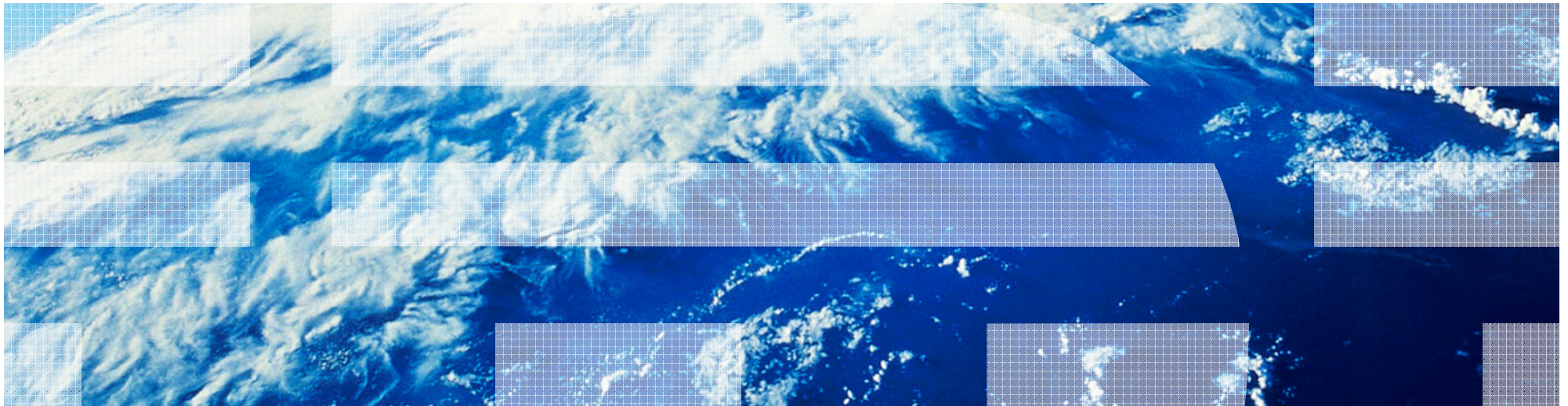


IS01 – The New Universal Database Connector (DBCLI)

Ingo Franzki

ifranzki@de.ibm.com



Trademarks

The following are trademarks of the International Business Machines Corporation in the United States, other countries, or both.

Not all common law marks used by IBM are listed on this page. Failure of a mark to appear does not mean that IBM does not use the mark nor does it mean that the product is not actively marketed or is not significant within its relevant market.

Those trademarks followed by ® are registered trademarks of IBM in the United States; all others are trademarks or common law marks of IBM in the United States.

For a complete list of IBM Trademarks, see www.ibm.com/legal/copytrade.shtml:

*, AS/400®, e business (logo)®, DBE, ESCO, eServer, FICON, IBM®, IBM (logo)®, iSeries®, MVS, OS/390®, pSeries®, RS/6000®, S/30, VM/ESA®, VSE/ESA, WebSphere®, xSeries®, z/OS®, zSeries®, z/VM®, System i, System i5, System p, System p5, System x, System z, System z9®, BladeCenter®

The following are trademarks or registered trademarks of other companies.

Adobe, the Adobe logo, PostScript, and the PostScript logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.

Cell Broadband Engine is a trademark of Sony Computer Entertainment, Inc. in the United States, other countries, or both and is used under license therefrom.

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Intel, Intel logo, Intel Inside, Intel Inside logo, Intel Centrino, Intel Centrino logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

ITIL is a registered trademark, and a registered community trademark of the Office of Government Commerce, and is registered in the U.S. Patent and Trademark Office.

IT Infrastructure Library is a registered trademark of the Central Computer and Telecommunications Agency, which is now part of the Office of Government Commerce.

* All other products may be trademarks or registered trademarks of their respective companies.

Notes:

Performance is in Internal Throughput Rate (ITR) ratio based on measurements and projections using standard IBM benchmarks in a controlled environment. The actual throughput that any user will experience will vary depending upon considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve throughput improvements equivalent to the performance ratios stated here.

IBM hardware products are manufactured from new parts, or new and serviceable used parts. Regardless, our warranty terms apply.

All customer examples cited or described in this presentation are presented as illustrations of the manner in which some customers have used IBM products and the results they may have achieved. Actual environmental costs and performance characteristics will vary depending on individual customer configurations and conditions.

This publication was produced in the United States. IBM may not offer the products, services or features discussed in this document in other countries, and the information may be subject to change without notice. Consult your local IBM business contact for information on the product or services available in your area.

All statements regarding IBM's future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only.

Information about non-IBM products is obtained from the manufacturers of those products or their published announcements. IBM has not tested those products and cannot confirm the performance, compatibility, or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

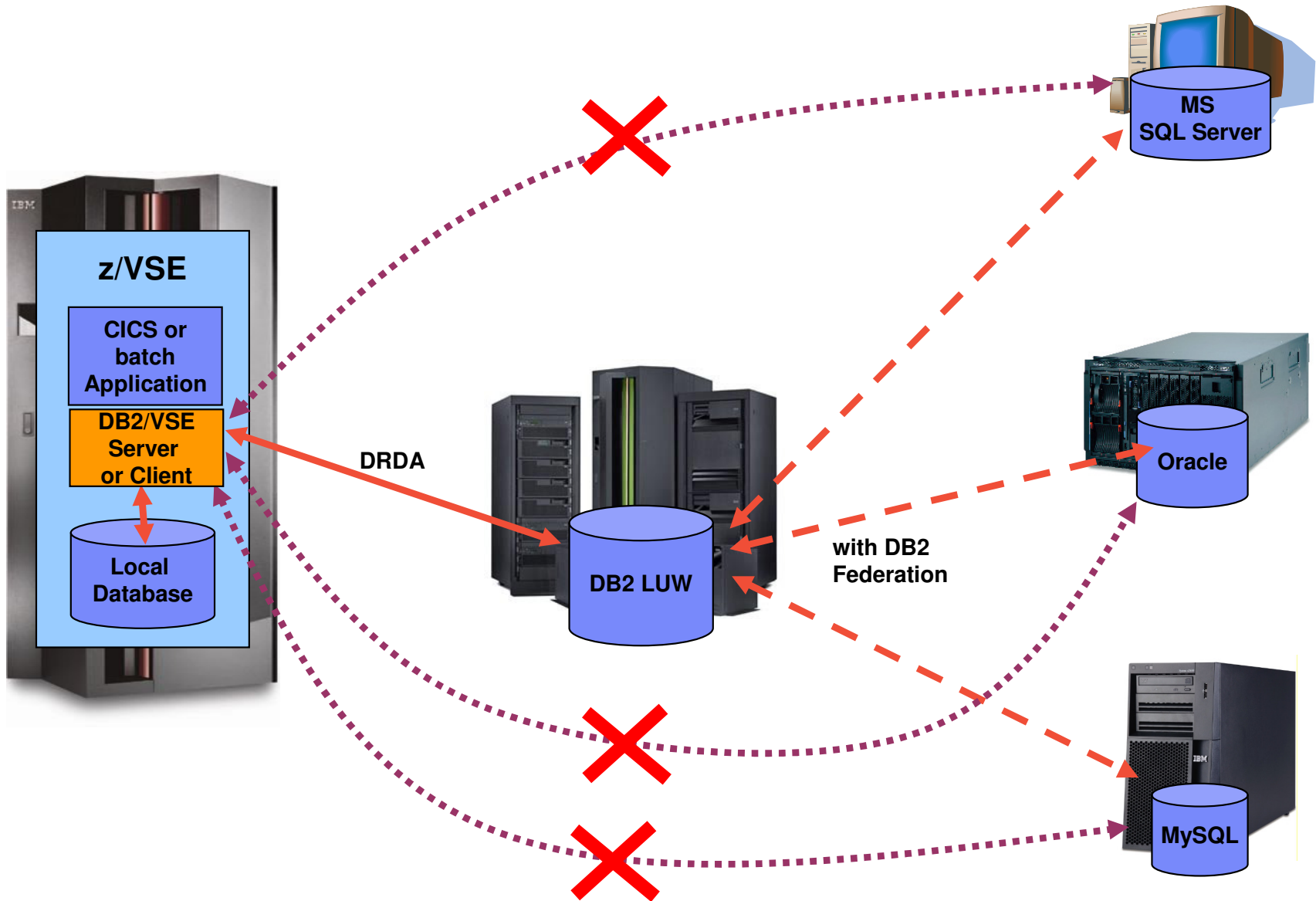
Prices subject to change without notice. Contact your IBM representative or Business Partner for the most current pricing in your geography.

Agenda

- Options for using Databases with z/VSE applications
- z/VSE Database Call Level Interface (DBCLI)
- DBCLI Concepts
- COBOL Example
- Hints & Tips
- Summary



z/VSE applications accessing Databases



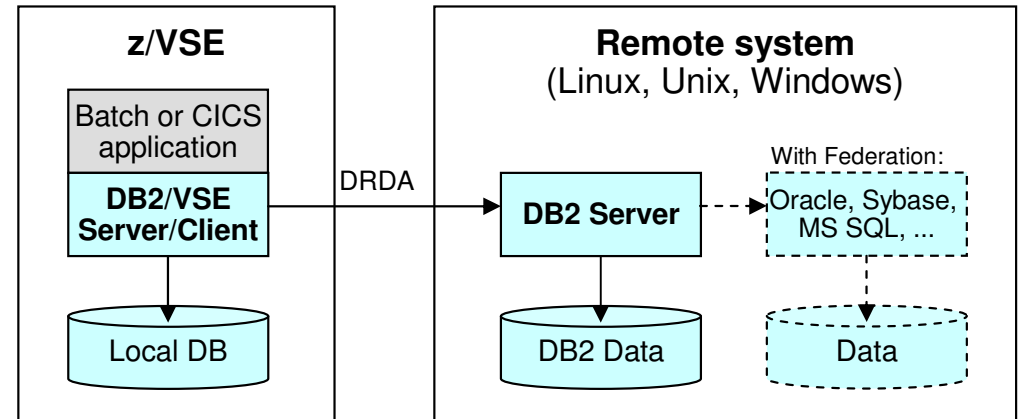
Options for using Databases with z/VSE applications

▪ DB2/VSE or DB2/VM Server

- Local database residing in z/VSE or z/VM
- Lacks support of modern SQL functionality
- Only quite old SQL level supported

▪ DB2/VSE Client Edition

- Remote database (on Linux, Windows, Unix)
- Communication via DRDA protocol
- Same old SQL level supported as DB2/VSE Server
- Can not use modern SQL functionality provided by DB2 LUW
- Can only access remote DB2 databases
 - Other databases (e.g. MS SQL Server, Oracle, etc) can only be accessed through IBM InfoSphere Federation Server



▪ VSAM Redirector

- Primarily used to keep Databases in sync with VSAM data
- Also allows migration from VSAM to database

▪ **New:** z/VSE Database Call Level Interface

- Allows z/VSE applications to access a relational database on any suitable database server
 - IBM DB2, IBM Informix, Oracle, MS SQL Server, MySQL, etc.
- Utilize advanced database functions and use SQL statements provided by modern database products

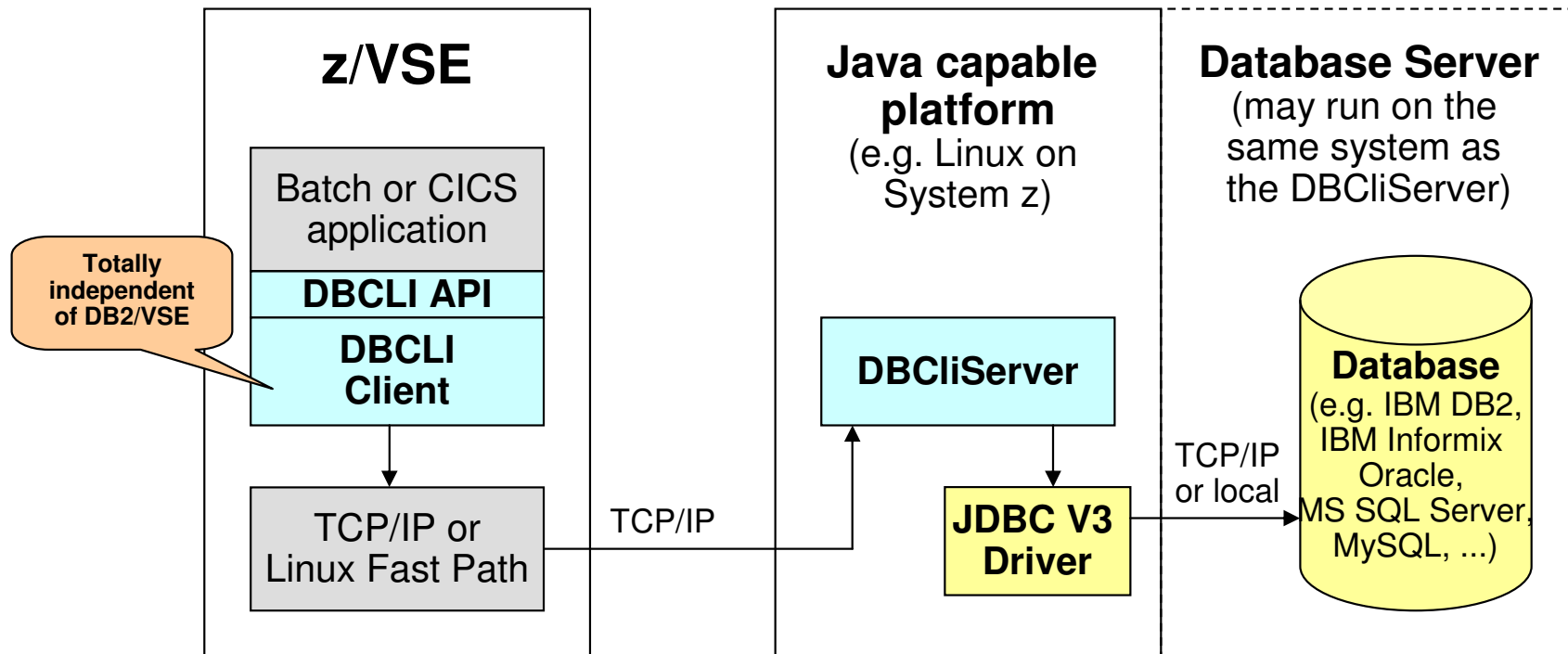


z/VSE Database Call Level Interface (DBCLI)



- **Allows z/VSE applications to access a relational database on any suitable database server**
 - IBM DB2, IBM Informix, Oracle, MS SQL Server, MySQL, etc.
 - The database product must provide a **JDBC driver that supports JDBC V3.0** or later
- **Utilize advanced database functions and use SQL statements provided by modern database products**

Requires z/VSE 5.1 plus PTFs (UK78892 and UK78893)



z/VSE Database Call Level Interface (DBCLI)

- **The z/VSE Database Call Level Interface (DBCLI) provides a programming interface (API)**
 - Call interface for use with COBOL, PL/1, Assembler, C and REXX
 - Can be used in Batch applications as well as in CICS TS applications
 - Supports LE enabled as well as non-LE environments (Assembler, REXX)
- **It provides callable functions for**
 - Initializing and Terminating the API Environment
 - Connecting and Disconnecting to/from the DBCLI Server and the Database
 - Executing SQL Statements
 - Retrieving query results through cursors
 - Handling of Logical Units of Work (Transactions)
 - Retrieving Database Meta Data
- **DBCLI can only support what the underlying Database supports**



z/VSE Database Call Level Interface (DBCLI)

- **The DBCLI API is not compatible with DB2/VSE's EXEC DB2 preprocessor interface**
 - It provides similar functions and concepts
 - The API is similar to the **ODBC programming interface** known from distributed platforms (ODBC = Open Data Base Connectivity)

- **No preprocessor is needed, instead you code the **CALL statements** directly in your program**
 - **Using DBCLI in COBOL:**
 - The COBOL copybook IESDBCOB contains common declarations

```
CALL 'IESDBCLI' USING FUNCTION ENV-HANDLE parm1 parm2...parmN RETCODE.
```

- **A **COBOL example** is provided to show how DBCLI can be used in your applications**

- **Documentation** is provided in the updated manual “z/VSE V5R1 e-business Connectors User's Guide” - SC34-2629-01
 - Chapter 9 and 22
 - Available on z/VSE web page: <http://ibm.com/zvse/documentation/#conn>

Using the DBCLI API in your applications

▪ Using DBCLI in COBOL:

- The COBOL copybook IESDBCOB contains common declarations

```
CALL 'IESDBCLI' USING FUNCTION ENV-HANDLE parm1 parm2 ... parmN RETCODE.
```

▪ Using DBCLI in PL/1

- The PL/1 copybook IESDBPL1 contains common declarations

```
CALL IESDBCLI (FUNCTION, ENV_HANDLE, parm1, parm2, ..., parmN, RETCODE) ;
```

▪ Using DBCLI in C

- The C header file IESDBC.h contains common declarations

```
IESDBCLI (function, &env_handle, &parm1, &parm2, ..., &parmN, &retcode) ;
```

▪ Using DBCLI in Assembler

- The Assembler macro IESDBASM contains common declarations

```
CALL IESDBCLI, (FUNCTION, ENV_HANDLE, parm1, parm2, ..., parmN, RETCODE), VL
```

- The following register conventions apply:

- Register 0, 1, 14, and 15 are used by the interface and must be, if necessary, saved prior to invocation
- Register 13 must point to a 72-byte save area provided by the caller

▪ Using DBCLI in REXX

```
ADDRESS LINKPGM "IESDBCLA FUNCTION ENV_HANDLE parm1 parm2 ... parmN RETCODE"
```

- All parameters must be initialized with a value of the appropriate length before calling the DBCLI API. This is especially true for output parameters.
- Fullword binary variables must be initialized to contain 4 bytes (for example, VARIABLE = D2C(0,4))
- Since the variable is expected to contain a value in binary representation, you must convert the value from the REXX string representation into the binary representation and vice versa using the REXX functions C2S and D2C



DBCLI Concepts: Initializing and terminating the environment

When using the API provided by the DBCLI client, you must:

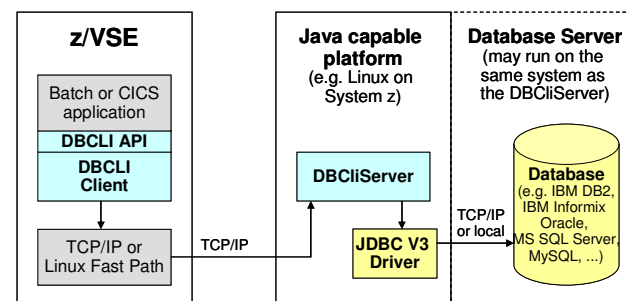
- **Initialize** the API environment by calling the **INITENV** function before calling any other function
 - The INITENV function allocates an **environment handle** that you must pass to all subsequent functions
 - You can have only one active environment at a time in your program
- **Terminate** the API environment (at the end of your program) by calling the **TERMENV** function
 - The TERMENV function frees all resources allocated by the DBCLI code
 - The TERMENV function will also close any "left over" connections or statements
 - After the TERMENV function, the environment handle is no longer valid
- **You can set and get various attributes on the environment level**
 - You do so by calling the **SETENVATTR** or **GETENVATTR** function



DBCLI Concepts: Connecting to the DBCLI Server and Database

To access a Database, you must connect to the DBCLI server and the Vendor database

- You connect to the DBCLI server (DBCliServer) and the database by calling the **CONNECT** function
- You must supply the:
 - IP address or hostname of DBCliServer
 - Alias name of the database or the JDBC URL to which you wish to connect
 - User-ID and Password to authenticate with the database
- The **CONNECT** function allocates a **connection handle** that you must pass to all subsequent functions that require a connection
 - You can have multiple connections to the same or different DBCLI servers and databases at a time
 - Each connection is represented by its own connection handle
- When you are finished working with a database, you must disconnect from the database and the DBCLI server (DBCliServer) by calling the **DISCONNECT** function
 - The DISCONNECT function frees the connection handle and all left over statements (if any) that you have allocated using this connection



DBCLI Concepts: Logical Units of Work (Transactions)



Per default, a connection operates in **transaction mode**:

- Any database updates that you perform are contained in a **logical unit of work**
- You can **end a logical unit of work** by calling the COMMIT or ROLLBACK functions:
 - The **COMMIT** function commits all changes done since the beginning of the logical unit of work and starts a new logical unit of work
 - The **ROLLBACK** function rolls back (reverts) all changes since the beginning of the logical unit of work or up to a savepoint

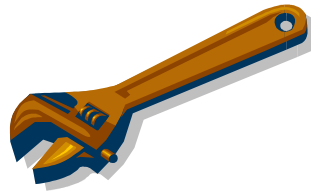
- Usually, you should **explicitly call the COMMIT function at the end of the program**.
- If you do not call the COMMIT function, DBCLI Server will **automatically commit** all changes
 - **if you gracefully close the connection** by calling the **DISCONNECT** function
- If the **connection is dropped** (for example, because the program abends), the DBCLI server **rolls back all changes** done since the beginning of the last logical unit of work

- You can set a connection into **auto-commit mode**
 - In auto-commit mode, every SQL statement is treated as **its own logical unit of work** and is **committed automatically** when the statement execution is complete.
 - Therefore, you do not have to call the COMMIT or ROLLBACK functions.
 - You set a connection into auto-commit mode by calling the **SETCONNATTR** function to set the **CONNATTR-AUTO-COMMIT** attribute to TRUE

DBCLI Concepts: Preparing SQL Statements

In order to execute an SQL statement, you must first prepare the SQL statement

- During preparation, the database will **pre-compile the SQL statement** and create an **access plan** for the statement
 - The access plan is kept as long as the statement exists
 - You can then **execute** the statement **as many times** as you want
- The **PREPARESTATEMENT** function prepares an SQL statement for execution
 - It allocates a **statement handle** that represents the statement



SQL statements may contain parameters that are evaluated at execution time

- Parameters are marked by a **question mark (?)** within the SQL statement

```
SELECT * FROM EMPLOYEE WHERE EMPNO>? AND SALARY>?
```

Parameter 1 Parameter 2

- The parameters are **numbered in order of appearance**, starting with 1
- When using DB2/VSE preprocessor, above statement would look like:

```
SELECT * FROM EMPLOYEE WHERE EMPNO>:empno AND SALARY>:salary
```

- The application **binds host variables to the parameters** using the **BINDPARAMETER** function
 - When the statement is later executed, the **content of the host variables is used** and sent to the database
 - You also specify the **data type** and **length** of the variable with the **BINDPARAMETER** call
 - **Indicator variables** are used to determine if the parameter value is **NULL**

DBCLI Concepts: Executing SQL Statements



To execute a prepared statement, you must call the EXECUTE function

- If the statement was an SQL **update statement**, you can retrieve the number of rows updated using the **GETUPDATECOUNT** function or the **UPDATE-COUNT** parameter at the EXECUTE function

- If the statement was a SQL **query statement**, you can **use a cursor** to retrieve (fetch) the result rows and columns
 - A statement can provide multiple results (mostly stored procedures)
 - To retrieve the additional results you must call the **GETMORERESULTS** function
 - The GETMORERESULTS function will move to the next available cursor or update count

- If the statement was a stored procedure call, **output parameters** are updated with the data passed back by the stored procedure

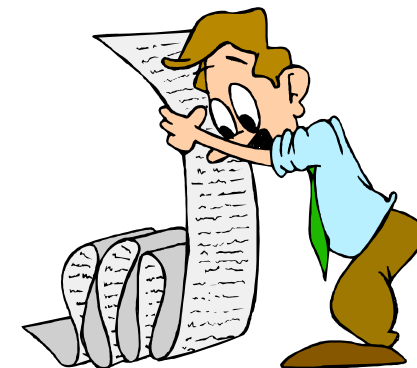
- When you no longer need a statement, you must close it by calling the **CLOSESTATEMENT** function:
 - The CLOSESTATEMENT function frees the statement handle and closes all cursors (if any) that may still be open from the last statement execution

- The statement handle is no longer valid after the CLOSESTATEMENT function

DBCLI Concepts: Result sets and Cursors

The execution on an **SQL query** returns a result in form of a **cursor**

- A cursor allows you to retrieve (fetch) the result rows and columns
 - You can use the **GETNUMCOLUMNS** and **GETCOLUMNINFO** functions to obtain detailed information about the cursor's columns
 - The **columns are numbered** in order of appearance, starting at 1
- To fetch the result rows using the cursor, you must first **bind host variables to the columns** of interest
 - You bind host variables to the columns of interest by calling the **BINDCOLUMN** function
 - If the **FETCH** function is called later on, the host variables will be updated with the contents of the column in the row that has been fetched
- Per default, the **FETCH** function processes the cursor **from the beginning to the end**
 - You may **reposition with a cursor**
 - Providing the database supports this and you have created the statement using the appropriate type (**CURSOR-TYPE-SCROLL-INSENSITIVE** or **CURSOR-TYPE-SCROLL-SENSITIVE**)
- Repositioning can be performed using either the:
 - **FETCH** function with operations **FETCH-PREVIOUS**, **FETCH-FIRST**, **FETCH-LAST**, **FETCH-ABSOLUTE** or **FETCH-RELATIVE**.
 - **SETPOS** function



DBCLI Concepts: Database Meta Data

The DBCLI interface allows you to retrieve meta data from the database

- This includes functions to get a **list of tables**, indexes, keys, **columns of a table**, and so on
- This information is typically stored in system catalog tables in the database.
 - You can also execute regular SELECT statements against the system catalog tables, but this requires that you know which database system and vendor you are using
 - System catalog tables are vendor- and database-specific
- The DBCLI interface provides a **set of database independent functions** to retrieve meta data information.
 - These functions are prefixed with 'DB'
 - The function DBTABLES for example retrieves a list of tables available in the database
- Please note that some databases may not support all of the meta data functions



DBBESTROWIDENT
 DBCATALOGS
 DBCOLUMNPRIV
 DBCOLUMNS
 DBCROSSREFERENCE
 DBEXPORTEDKEYS
 DBIMPORTEDKEYS
 DBINDEXINFO
 DBPRIMARYKEYS
 DBPROCEDURECOLS

DBPROCEDURES
 DBCHEMAS
 DBSUPERTABLES
 DBSUPERTYPES
 DBTABLEPRIV
 DBTABLES
 DBTABLETYPES
 DBTYPEINFO
 DBUDTS
 DBVERSIONCOLS

COBOL Example

```
PROCEDURE DIVISION.  
MAIN-PROGRAM.  
    DISPLAY 'COBSAMPL STARTED'.  
*  
* Perform the INITENV call  
*  
    MOVE 'SOCKET00' TO TCPNAME.  
    MOVE 'EZASOH99' TO ADSNAME.  
    CALL 'IESDBCLI' USING FUNC-INITENV ENV-HANDLE  
        TCPNAME ADSNAME RETCODE.  
    DISPLAY 'RETCODE OF INITENV IS ' RETCODE.  
    IF RETCODE > EOK THEN  
        PERFORM CHECK-ERROR  
    END-IF.
```

← Initialize the environment

COBOL Example

```
PROCEDURE DIVISION.  
MAIN-PROGRAM.  
    DISPLAY 'COBSAMPL STARTED'.  
*  
* * Connect to the DBCLI server and the database  
*  
    MOVE '9.152.2.70' TO SERVER.  
    MOVE 10 TO SERVER-LEN.  
    MOVE 16178 TO PORT.  
    MOVE 'SAMPLE' TO DBNAME.  
    MOVE 6 TO DBNAME-LEN.  
    MOVE 'dbuserid' TO USERID.  
    MOVE 8 TO USERID-LEN.  
    MOVE 'password' TO PASSWD.  
    MOVE 8 TO PASSWD-LEN.  
    CALL 'IESDBCLI' USING FUNC-CONNECT ENV-HANDLE CON-HANDLE  
        SERVER SERVER-LEN PORT DBNAME DBNAME-LEN  
        USERID USERID-LEN PASSWD PASSWD-LEN  
        RETCODE.  
    DISPLAY 'RETCODE OF CONNECT IS ' RETCODE.  
    IF RETCODE > EOK THEN  
        PERFORM CHECK-ERROR  
    END-IF.
```

IP or hostname of
DBCLI Server

Database alias name

User-ID & Password

Connect to the
DBCLI Server
and the Database

COBOL Example

```
PROCEDURE DIVISION.  
MAIN-PROGRAM.  
    DISPLAY 'COBSAMPL STARTED'.  
*  
*  
*   * Connect to the DBCLI server and the database  
*  
    MOVE '9.152.2.70' TO SERVER.  
    MOVE 10 TO SERVER-LEN.  
    MOVE 16178 TO PORT.  
    MOVE 'SAMPLE' TO DBNAME.  
    MOVE 6 TO DBNAME-LEN.  
*  
*   * Prepare an SQL statement for later execution  
*  
    MOVE 'SELECT * FROM EMPLOYEE WHERE EMPNO>? AND SALARY>?'  
      TO SQL.  
    MOVE LENGTH OF SQL TO SQL-LEN.  
    CALL 'IESDBCLI' USING FUNC-PREPARESTATEMENT ENV-HANDLE  
      CON-HANDLE STMT-HANDLE SQL SQL-LEN  
      CURSOR-TYPE-SCROLL-INSENSITIVE CURSOR-CONCUR-READ-ONLY  
      HOLD-CURSORS-OVER-COMMIT RETCODE.  
    DISPLAY 'RETCODE OF PREPARESTATEMENT IS ' RETCODE.  
    IF RETCODE > EOK THEN  
        PERFORM CHECK-ERROR  
    END-IF.
```

SQL Statement
Containing Parameter
Markers ('?')

Prepare an
SQL Statement
for later execution

COBOL Example

```

PROCEDURE DIVISION.
MAIN-PROGRAM.
  DISPLAY 'COBSAMPL STARTED'.
*
*
* * Connect to the DBCLI server and the database
*
  MOVE '9.152.2.70' TO SERVER.
  MOVE 10 TO SERVER-LEN.
  MOVE 16178 TO PORT.
  MOVE 'SAMPLE' TO DBNAME.
  MOVE 6 TO DBNAME-LEN.

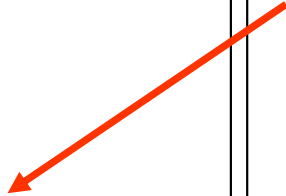
```

```

*
* Prepare SQL statement for later execution
*
* Bind the EMPNO host variable (Text) to parameter 1.
* Here we specify the optional codepage parameter to
* send the text data in the desired codepage.
*
  MOVE 1 TO PARM-IDX.
  MOVE LENGTH OF EMPNO TO EMPNO-LEN.
  MOVE 'CP1047' TO CODEPAGE.
  MOVE LENGTH OF CODEPAGE TO CODEPAGE-LEN.
  CALL 'IESDBCLI' USING FUNC-BINDPARAMETER ENV-HANDLE
    STMT-HANDLE PARM-IDX NATIVE-TYPE-STRING
    EMPNO EMPNO-LEN EMPNO-IND
    CODEPAGE CODEPAGE-LEN RETCODE.
  DISPLAY 'RETCODE OF BINDPARAMETER IS ' RETCODE.
  IF RETCODE > EOK THEN
    PERFORM CHECK-ERROR
  END-IF.

```

Bind host variable
"EMPNO"
to parameter
number 1
as STRING



COBOL Example

```
PROCEDURE DIVISION.
```

```
MAIN-PROGRAM.
```

```
    DISPLAY 'COBSAMPL STARTED'.
```

```
*
*
* * Connect to the DBCLI server and the database
*
```

```
    MOVE '9.152.2.70' TO SERVER.
```

```
    MOVE 10 TO SERVER-LEN.
```

```
    MOVE 16178 TO PORT.
```

```
    MOVE 'SAMPLE' TO DBNAME.
```

```
    MOVE 6 TO DBNAME-LEN.
```

```
* Prepare SQL statement for table connection
```

```
*
* Bind the EMPNO host variable (Text) to parameter 1.
* Here we specify the optional codepage parameter to
* send the text data in the desired codepage.
```

```
*
* Bind the SALARY host variable (packed decimal) to parameter 2.
* Here we specify the decpos parameter to indicate that we
* want to send the numeric data with 2 implied decimal places.
```

```
    MOVE 2 TO PARM-IDX.
```

```
    MOVE LENGTH OF SALARY TO SALARY-LEN.
```

```
    MOVE 2 TO DECPOS.
```

```
    CALL 'IESDBCLI' USING FUNC-BINDPARAMETER ENV-HANDLE
        STMT-HANDLE PARM-IDX NATIVE-TYPE-PACKED-SIGNED
        SALARY SALARY-LEN SALARY-IND
        DECPOS RETCODE.
```

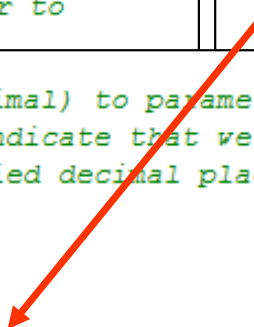
```
    DISPLAY 'RETCODE OF BINDPARAMETER IS ' RETCODE.
```

```
    IF RETCODE > EOK THEN
```

```
        PERFORM CHECK-ERROR
```

```
    END-IF.
```

Bind host variable
"SALARY"
to parameter
number 2
as PACKED decimal



COBOL Example

```

PROCEDURE DIVISION.
MAIN-PROGRAM.
    DISPLAY 'COBSAMPL STARTED'.
*
*
* * Connect to the DBCLI server and the
*
    MOVE '9.152.2.70' TO SERVER.
    MOVE 10 TO SERVER-LEN.
    MOVE 16178 TO PORT.
    MOVE 'SAMPLE' TO DBNAME.
    MOVE 6 TO DBNAME-LEN.
*
* Prepare SQL statement
*
* Bind the EMPNO host variable
* Here we specify the options
* send the text data in the
*
* Bind the SALARY host variable
* Here we specify the options
* want to send the number
*
    MOVE 2 TO PARM-IDX.
    MOVE LENGTH OF SALARY TO SALARY-LEN.
    MOVE 2 TO DECPOS.
    CALL 'IESDBCLI' USING FUNC-BINDPARAMETER ENV-HANDLE
        STMT-HANDLE PARM-IDX NATIVE-TYPE-PACKED-SIGNED
        SALARY SALARY-LEN SALARY-IND
        DECPOS RETCODE.
    DISPLAY 'RETCODE OF BINDPARAMETER IS ' RETCODE.
    IF RETCODE > EOK THEN
        PERFORM CHECK-ERROR
    END-IF.
*
* Set the host variables values and corresponding indicator
* variables:
*
    MOVE '000030' TO EMPNO.
    MOVE INDICATE-NOTNULL TO EMPNO-IND.
    MOVE 01000.00 TO SALARY.
    MOVE INDICATE-NOTNULL TO SALARY-IND.
*
* Execute the statement. This will use the values of the
* host variables for the parameters.
*
    CALL 'IESDBCLI' USING FUNC-EXECUTE ENV-HANDLE
        STMT-HANDLE RETCODE.
    IF RETCODE > EOK THEN
        PERFORM CHECK-ERROR
    END-IF.
    DISPLAY 'RETCODE OF EXECUTE IS ' RETCODE.
    IF RETCODE > EOK THEN
        PERFORM CHECK-ERROR
    END-IF.

```

Assign values

Execute the statement

COBOL Example

```

PROCEDURE DIVISION.
MAIN-PROGRAM.
    DISPLAY 'COBSAMPL STARTED'.
*
*
* * Connect to the DBCLI server and the
*
    MOVE '9.152.2.70' TO SERVER.
    MOVE 10 TO SERVER-LEN.
    MOVE 16178 TO PORT.
    MOVE 'SAMPLE' TO DBNAME.
    MOVE 6 TO DBNAME-LEN.
*
* Prepare SQL statement
*
* Bind the EMPNO host variable
* Here we specify the option
* send the text data in the
*
* Bind the SALARY host
* Here we specify the c
* want to send the nume
*
    MOVE 2 TO PARM-IDX.
    MOVE LENGTH OF SALARY TO SALARY-LEN.
    MOVE 2 TO DECPOS.
    CALL 'IESDBCLI' USING FUNC-BINDPARAMETER ENV-HANDLE
        STMT-HANDLE PARM-IDX NATIVE-TYPE-PACKED-SIGNED
        SALARY SALARY-LEN SALARY-IND
        DECPOS RETCODE.
    DISPLAY 'RETCODE OF BINDPARAMETER IS ' RETCODE.
    IF RETCODE > EOK THEN
        PERFORM CHECK-ERROR
    END-IF.
*
* Set the host variables values and corresponding indicator
* variables:
*
* Bind host variable FIRSTNAME (text) to the column 2.
* Here we do not specify the codepage parameter so we
* receive the text data in the default codepage.
*
    MOVE 2 TO COL-IDX.
    MOVE LENGTH OF FIRSTNAME TO FIRSTNAME-LEN.
    CALL 'IESDBCLI' USING FUNC-BINDCOLUMN ENV-HANDLE
        STMT-HANDLE COL-IDX NATIVE-TYPE-STRING
        FIRSTNAME FIRSTNAME-LEN FIRSTNAME-IND
        RETCODE.
    DISPLAY 'RETCODE OF BINDCOLUMN IS ' RETCODE.
    IF RETCODE > EOK THEN
        PERFORM CHECK-ERROR
    END-IF.

```

Bind host variable
"FIRSTNAME" to
result set column
number 2

COBOL Example

```

PROCEDURE DIVISION.
MAIN-PROGRAM.
    DISPLAY 'COBSAMPL STARTED'.
*
*
* * Connect to the DBCLI server and the
*
    MOVE '9.152.2.70' TO SERVER.
    MOVE 10 TO SERVER-LEN.
    MOVE 16178 TO PORT.
*
*
* * Bind the EMPNO host variable
* * Here we specify the options
* * send the text data in the
*
* * Bind the SALARY host
* * Here we specify the column
* * want to send the number
*
    MOVE 2 TO PARM-IDX.
    MOVE LENGTH OF SALARY TO
    MOVE 2 TO DECPOS.
    CALL 'IESDBCLI' USING FUNC-FETCH ENV-HANDLE
    STMT-HANDLE PARM-IDX
    SALARY SALARY-LEN SA
    DECPOS RETCODE.
    DISPLAY 'RETCODE OF BINDE
    IF RETCODE > EOK THEN
        PERFORM CHECK-ERROR
    END-IF.
*
* * Set the host variables values and corresponding indicator
* * variables:
*
* * Bind host variable FIRSTNAME (text) to the column 2.
*
* * Fetch all available rows and display the data.
* * Since columns may be NULL we check the indicator variables.
* * FETCH without an operation argument means FETCH NEXT.
*
    PERFORM WITH TEST AFTER UNTIL RETCODE > EOK
    CALL 'IESDBCLI' USING FUNC-FETCH ENV-HANDLE
    STMT-HANDLE RETCODE
    DISPLAY 'RETCODE OF FETCH IS ' RETCODE
    IF RETCODE > EOK AND RETCODE NOT = ENOMOREDATA THEN
        PERFORM CHECK-ERROR
    END-IF
    IF RETCODE = EOK THEN
        DISPLAY 'ROW DATA INFO FOR ROW NUMBER ' ROW-NUMBER
        IF EMPNO-IND = INDICATE-NULL THEN
            DISPLAY ' EMPNO IS NULL'
        ELSE
            DISPLAY ' EMPNO IS ' EMPNO
        END-IF
        IF FIRSTNAME-IND = INDICATE-NULL THEN
            DISPLAY ' FIRSTNAME IS NULL'
        ELSE
            DISPLAY ' FIRSTNAME IS ' FIRSTNAME
        END-IF
    END-IF

```

Fetch all rows

Summary: Connecting to a database



- **The following steps need to be performed to connect to a database**

- Initialize the environment using **INITENV**
- [optional] Set environment attributes (if needed) using **SETENVATTR**
- Connect to the DBCLI server and database using **CONNECT**
- [optional] Set connection attributes (if needed) using **SETCONATTR**

- ...

- Commit (or rollback) the changes using **COMMIT** (or **ROLLBACK**)
- Disconnect from the database using **DISCONNECT**
- Terminate the environment using **TERMENV**

Summary: Executing SQL statements



- **The following steps need to be performed to execute SQL statements**
 - Prepare the SQL statement using **PREPARESTATEMENT**
 - Statement may contain parameter markers
 - [optional] Set statement attributes (if needed) using **SETSTMTATTR**
 - Bind statement parameters to host variables using **BINDPARAMETER**
 - Set the content of the host variables to be used for the execution
 - Execute the statement using **EXECUTE**

- **If the statement was a query (e.g. SELECT), retrieve the result set via a cursor:**
 - Bind result set columns of interest to host variables using **BINDCOLUMN**
 - Fetch the result rows using **FETCH**
 - You can reposition within the result set via fetch operation or **SETPOS**
 - Close the cursor using **CLOSECURSOR**
 - You may re-execute the statement as many times as you want
 - Close the statement using **CLOSESTATEMET**

Summary: Executing stored procedures



▪ The following steps need to be performed to execute stored procedures

- Prepare the SQL statement using **PREPARECALL**
 - Call statement may contain parameter markers for input and/or output parameters
- [optional] Set statement attributes (if needed) using **SETSTMTATTR**
- Bind statement parameters to host variables using **BINDPARAMETER**
- Set the content of the host variables to be used for the execution
- Execute the call statement using **EXECUTE**

▪ If the call returned a result set, retrieve the result set via a cursor:

- Bind result set columns of interest to host variables using **BINDCOLUMN**
- Fetch the result rows using **FETCH**
 - You can reposition within the result set via fetch operation or **SETPOS**
- Close the cursor using **CLOSECURSOR**
- Move on to the next result set using **GETMORERESULTS**
 - This will set the all statement's output parameters (if any)
- Close the statement using **CLOSESTATEMET**

Differences between DBCLI and embedded SQL

- **An application that uses an embedded SQL interface requires a precompiler**
 - To convert the SQL statements into code, which is then compiled, bound to the data source, and executed
- **In contrast, a DBCLI application does not have to be precompiled or bound**
 - Instead uses a set of functions to execute SQL statements and related services at run time
- **This difference is important because precompilers are specific to the database product used**
 - This ties your applications to a specific database product and vendor
- **DBCLI enables you to write applications that are independent of any particular database product or vendor**
- **Further differences:**
 - DBCLI does not require the explicit declaration of cursors, they are generated as needed. The application can then use the generated cursor to fetch the result rows
 - A COMMIT or ROLLBACK in DBCLI is issued using the COMMIT or ROLLBACK functions calls rather than by passing it as an SQL statement
 - DBCLI manages statement related information on behalf of the application, and provides a **statement handle** to refer to it as an abstract object. This handle eliminates the need for the application to use product specific data structures
 - Similar to the statement handle, the **environment handle** and **connection handle** provide a means to refer to all global variables and connection specific information



Advantages of using DBCI instead of embedded SQL

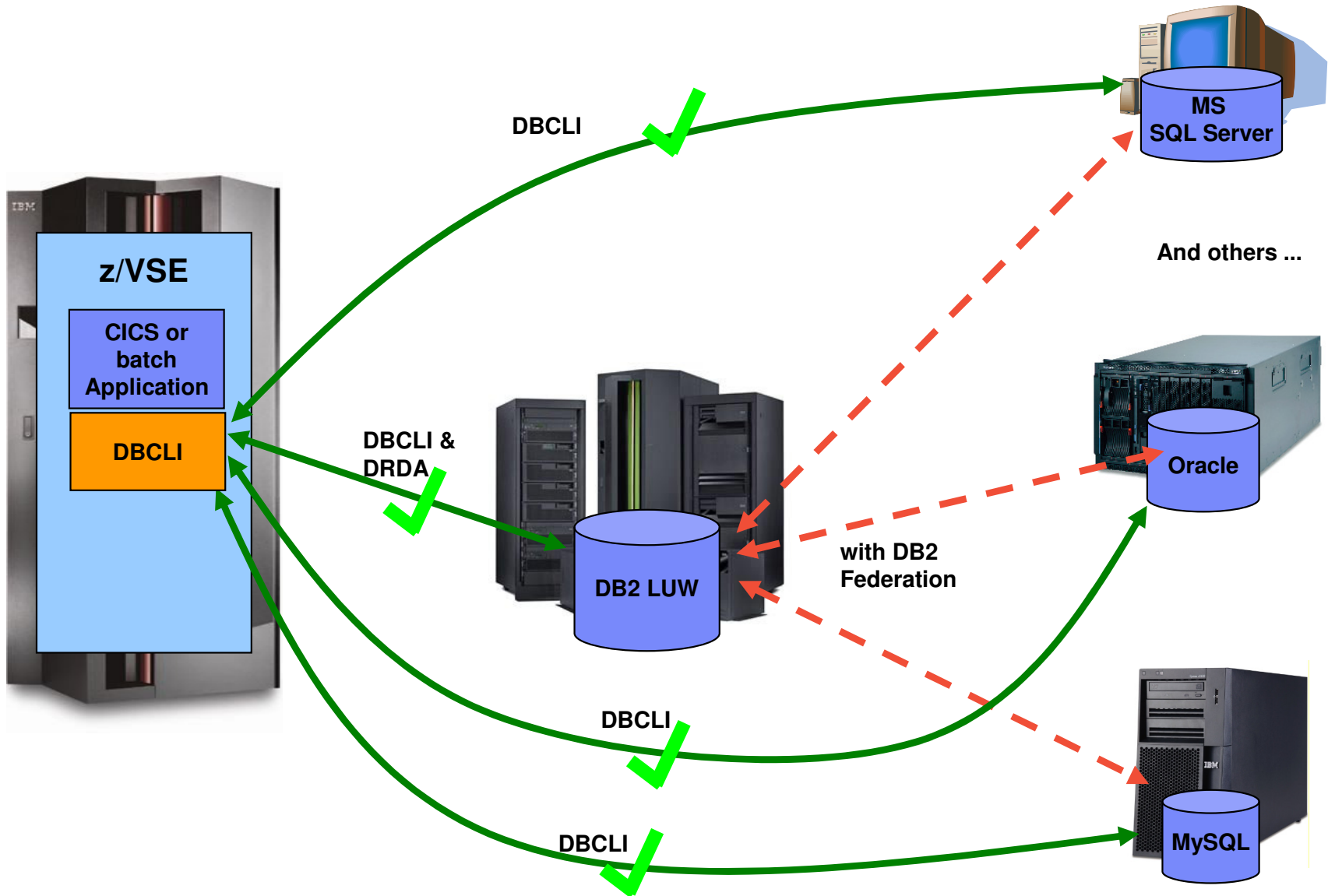
- **Ideally suits the client-server environment in which the target data source is unknown when the application is built**
 - It provides a consistent interface for executing SQL statements, regardless of which database server the application connects to
- **Lets you write applications that are independent of any particular database product**
 - DBCLI applications do not have to be recompiled or rebound to access different database. Instead they connect to the appropriate database at run time.
- **Lets applications connect to multiple data sources from the same application**
- **Allocates and controls data structures, and provides a handle for the application to refer to them**
 - Applications do not have to control complex global data areas such as the SQLDA and SQLCA
- **Lets you retrieve multiple rows and result sets generated from a call to a stored procedure**
- **Provides a consistent interface to query catalog information that is contained in various database management system catalog tables**
 - The result sets that are returned are consistent across database management systems. Application programmers can avoid writing version-specific and server-specific catalog queries
- **Programming interface is very similar to the accepted industry standard of ODBC/JDBC**
- **Allows application developers to apply their knowledge of industry standards directly to DBCLI**
 - The interface is intuitive for programmers who are familiar with function libraries but know little about product specific methods of embedding SQL statements into a host language

Hints & Tips



- **The DBCLI code is CICS-aware**
 - If running under CICS, any memory allocations are performed using EXEC CICS GETMAIN instead of using the GETVIS macro
- **When using the DBCLI API in CICS transactions while CICS operates with storage protection, all programs using the DBCLI API need to be defined with EXECKEY(CICS)**
 - This is also true for those programs that link to these programs
 - TASKDATAKEY(CICS) for the transaction definition is NOT required.
- **When using the DBCLI API in CICS transactions, the EZA "task-related-user-exit" (TRUE) has to be activated before these transactions can be run**
 - For details on how to activate this TRUE, refer to "CICS Considerations for the EZA Interfaces" in the z/VSE TCP/IP Support, SC34-2640
- **Most JDBC drivers will only accept pure SQL statements**
 - They will not accept SQL preprocessor statements that are used for DB2 Server for VSE applications
 - Basically DBCLI can execute any SQL statement that can be prepared dynamically in embedded SQL
- **The call to the IESDBCLI function must be a static CALL in COBOL**
 - Do not use the DYNAM compiler option

z/VSE applications accessing Databases



Questions ?

