OO Programming with Client Access for OS/400 and ODBC using VisualAge for Smalltalk

August 1996
International Technical Support Organization

OO Programming with Client Access for OS/400 and ODBC using VisualAge for Smalltalk

August 1996
## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preface</td>
<td>vii</td>
</tr>
<tr>
<td>How This Redbook Is Organized</td>
<td>vii</td>
</tr>
<tr>
<td>The Team That Wrote This Redbook</td>
<td>viii</td>
</tr>
<tr>
<td>Comments Welcome</td>
<td>viii</td>
</tr>
<tr>
<td><strong>Chapter 1. Open Database Connectivity (ODBC) Interface</strong></td>
<td>1</td>
</tr>
<tr>
<td>1.1 ODBC Interface</td>
<td>1</td>
</tr>
<tr>
<td>1.2 ODBC Components</td>
<td>2</td>
</tr>
<tr>
<td>1.3 Types of ODBC Drivers</td>
<td>3</td>
</tr>
<tr>
<td>1.4 ODBC Conformance Levels</td>
<td>3</td>
</tr>
<tr>
<td>1.5 Client Access/400 V3R1 ODBC Driver</td>
<td>6</td>
</tr>
<tr>
<td>1.6 V3R1 ODBC Support</td>
<td>7</td>
</tr>
<tr>
<td>1.7 Calling ODBC Functions</td>
<td>7</td>
</tr>
<tr>
<td>1.8 Basic Application Steps</td>
<td>7</td>
</tr>
<tr>
<td><strong>Chapter 2. Introduction to the TPC-C Benchmark Application</strong></td>
<td>9</td>
</tr>
<tr>
<td>2.1 Requirements Specification of the TPC-C Benchmark Application</td>
<td>9</td>
</tr>
<tr>
<td>2.2 Phase 1 Object Analysis</td>
<td>9</td>
</tr>
<tr>
<td>2.3 TPC-C Benchmark Database Layout</td>
<td>10</td>
</tr>
<tr>
<td>2.3.1 District</td>
<td>11</td>
</tr>
<tr>
<td>2.3.2 Customer</td>
<td>11</td>
</tr>
<tr>
<td>2.3.3 Order</td>
<td>12</td>
</tr>
<tr>
<td>2.3.4 Order Line</td>
<td>12</td>
</tr>
<tr>
<td>2.3.5 Item (Catalog)</td>
<td>13</td>
</tr>
<tr>
<td>2.3.6 Stock</td>
<td>13</td>
</tr>
<tr>
<td>2.4 Database Terminology</td>
<td>13</td>
</tr>
<tr>
<td><strong>Chapter 3. Installation of VisualAge for Smalltalk with ODBC Support and Client Access/400</strong></td>
<td>15</td>
</tr>
<tr>
<td>3.1 Installation of VisualAge for Smalltalk with ODBC Support</td>
<td>15</td>
</tr>
<tr>
<td>3.2 Installation of Client Access/400</td>
<td>17</td>
</tr>
<tr>
<td>3.2.1 Installation of Client Access/400 Optimized for OS/2</td>
<td>17</td>
</tr>
<tr>
<td>3.2.2 Installation of Client Access/400 for Windows 3.1</td>
<td>19</td>
</tr>
<tr>
<td><strong>Chapter 4. Configuration of Client Access/400 ODBC Support in the VisualAge for Smalltalk Environment</strong></td>
<td>21</td>
</tr>
<tr>
<td>4.1 Configuration of Data Sources Using the Client Access/400 ODBC Driver</td>
<td>21</td>
</tr>
<tr>
<td>4.1.1 Configuration of the Client Access/400 Optimized for OS/2 ODBC Driver</td>
<td>21</td>
</tr>
<tr>
<td>4.1.2 Configuration of the Client Access/400 for Windows 3.1 ODBC Driver</td>
<td>27</td>
</tr>
<tr>
<td>4.2 Configuration of ODBC Data Sources in VisualAge for Smalltalk</td>
<td>35</td>
</tr>
<tr>
<td>4.2.1 VisualAge Database Access: Concepts</td>
<td>35</td>
</tr>
<tr>
<td>4.2.2 Setting up a Database Connection</td>
<td>37</td>
</tr>
<tr>
<td><strong>Chapter 5. Using the VisualAge Database Functions</strong></td>
<td>45</td>
</tr>
<tr>
<td>5.1 The Multi-Row Query Part</td>
<td>45</td>
</tr>
<tr>
<td>5.1.1 Defining a Multi-Row Query</td>
<td>45</td>
</tr>
<tr>
<td>5.1.2 Using the Results of a Multi-Row Query</td>
<td>53</td>
</tr>
<tr>
<td>5.1.3 Insert, Update, and Delete Using a Multi-row Query Part</td>
<td>60</td>
</tr>
<tr>
<td>5.2 The Single-Row Query Part</td>
<td>63</td>
</tr>
<tr>
<td>5.2.1 Defining a Single-Row Query</td>
<td>63</td>
</tr>
<tr>
<td>Section</td>
<td>Page</td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>How Customers Can Get ITSO Redbooks</td>
<td>128</td>
</tr>
<tr>
<td>IBM Redbook Order Form</td>
<td>129</td>
</tr>
<tr>
<td>List of Abbreviations</td>
<td>131</td>
</tr>
<tr>
<td>Index</td>
<td>133</td>
</tr>
</tbody>
</table>
Preface

This document contains detailed information about VisualAge for Smalltalk’s ODBC implementation using Client Access/400 to access an AS/400 database. Step-by-step instructions are included for the installation and configuration. How the VisualAge database parts can be used, and how ODBC database access to the AS/400 can be optimized is shown with the help of a real-life application.

This document was written for application developers, IBM Business Partners, and IBM technical representatives who are involved in developing client/server applications for AS/400 customers using VisualAge for Smalltalk.

Some knowledge of VisualAge for Smalltalk and the AS/400 database is assumed.

How This Redbook Is Organized

This redbook contains 134 pages. It is organized as follows:

- Chapter 1, “Open Database Connectivity (ODBC) Interface”
  This chapter provides an overview of Microsoft’s architected database access interface ODBC.

- Chapter 2, “Introduction to the TPC-C Benchmark Application”
  This chapter introduces you to the reduced version of the TPC-C benchmark application and the TPC-C database.

- Chapter 3, “Installation of VisualAge for Smalltalk with ODBC Support and Client Access/400”
  This chapter provides the installation of the ODBC support feature into your VisualAge for Smalltalk environment and the installation of the ODBC driver that comes with Client Access/400.

- Chapter 4, “Configuration of Client Access/400 ODBC Support in the VisualAge for Smalltalk Environment”
  This chapter covers the configuration of data sources using the ODBC driver of Client Access/400 for both OS/2 and Windows, as well as the configuration of the ODBC support of VisualAge for Smalltalk.

- Chapter 5, “Using the VisualAge Database Functions”
  This chapter covers the use of VisualAge database parts to access ODBC data sources.

- Chapter 6, “Real-life OO Application Design”
  This chapter discusses the implementation of the application used to access the AS/400 database using ODBC or DDM.

- Chapter 7, “Optimizing ODBC Access”
  This chapter provides you with hints and tips as a result of the development of the TPC-C application.

- Appendix A, “Installation of the Sample Application Diskette”
  This appendix describes the process to install the diskette contents onto your system.
• Appendix B, “ODBC Error Messages”

This appendix describes the format of ODBC Error Messages.

The Team That Wrote This Redbook

This redbook was produced by a team of specialists from around the world working at the International Technical Support Organization Rochester Center.

Ted Zonderland is an Advisory System Engineer and works in the International Technical Support Organization, Rochester Center. He writes extensively and teaches IBM classes worldwide on all areas of Application Development and Object Orientation. Before joining the ITSO 2 years ago, Ted worked in the Advanced Software Solutions Center in IBM Netherlands.

Peter Boonen is a Systems Engineer in IBM Belgium. He has 2 and a half years of experience in the Advanced Software Solutions Center in Belgium. The mission of the ASSC is to explore the possibilities of Client Server and Object Oriented Application Development in an AS/400 environment. Since the beginning of 1995, he is working with VisualAge for Smalltalk, especially the AS/400 feature.

Enrico Picinnin is a System Engineer in IBM Semea Milano. He has been working for 2 years in the AS/400 division, giving support to IBM Business Partners in the area of Object Technology. In particular, he has developed a deep knowledge and experience in VisualAge for Smalltalk in connection with the AS/400 system. He has also worked on various issues concerning the relationship between OO programming and relational database models.

Thanks to the following people for their invaluable contributions to this project:

Mike Jacobs
IBM Rochester Lab

Mike Koranda
IBM Rochester Lab

Kelly Keplinger
IBM Raleigh Lab

Bob Maatta
International Technical Support Organization, Rochester Center

Kris Peterson
International Technical Support Organization, Rochester Center

Comments Welcome

We want our redbooks to be as helpful as possible. Should you have any comments about this or other redbooks, please send us a note at the following address:

redbook@vnet.ibm.com

Your comments are important to us!
Chapter 1. Open Database Connectivity (ODBC) Interface

ODBC is a Microsoft architected database access interface that enables applications to access data using Structured Query Language (SQL) as a standard language.

ODBC provides a consistent set of APIs that permit a single application to access different database management systems.

The ODBC approach is:

• A program separate from the application to extract database information.
• A standard interface for applications to import the data.
• Database drivers, provided by the various database vendors or third parties:
  – Supplied as dynamic link libraries that an application can invoke.
  – Gain access to the database management system.
  – Client Access/400 provides drivers that can access the AS/400 database.

1.1 ODBC Interface

The ODBC interface defines a library of function calls that allow an application to:

• Connect to a DBMS.
• Execute SQL statements.
• Retrieve results.

The ODBC interface also provides for:

• SQL syntax.
• A standard set of error codes.
• A standard way to connect and log on to a DBMS.
• A standard representation for data types.
1.2 ODBC Components

The components of an ODBC application are:

1. Application
   - Requests a connection or session with a data source.
   - Sends SQL requests to the data source.
   - Defines storage areas and data formats for the results of SQL requests.
   - Requests results.
   - Retrieves result column data.
   - Processes errors.
   - Reports results back to user if necessary.
   - Requests commit or rollback operations for transaction control.
   - Terminates the connection to the data source.

2. Driver Manager
   - Uses ODBC.INI file to map a data source name to a specific driver dynamic link library (DLL).
   - Processes several ODBC initialization calls.
   - Provides entry points to ODBC functions for each driver function.
   - Provides parameter validation and sequence validation for ODBC calls.

3. Driver
   - Establishes a connection to the data source.
   - Submits requests to the data source.
   - Translates data to or from other formats, if requested by the application.
   - Returns results to the application.
   - Formats errors into standard error codes and returns them to the application.
   - Declares and manipulates cursors if necessary.
   - Initiates transactions if the data source requires explicit transaction initiation.

4. Data Source
   - The general features and functionality provided by an SQL database management system.
• A specific instance of a combination of a DBMS product, remote operating system, and networking necessary for access.
• Examples:
  – AS/400 DB2/400
  – Oracle DBMS running under OS/2
  – Tandem non-stop SQL DBMS running on a Guardian 90 operating system

### 1.3 Types of ODBC Drivers

ODBC drivers come in two basic types:

- **Single-tier**
  - Driver processes both ODBC calls and SQL statements.
  - Database file is processed directly by the driver.

- **Multiple-tier**
  - Driver processes ODBC calls and passes SQL statements to data source.
  - Can reside on a single system, most often divided across platforms.
    - Application, driver, driver manager on **Client**
    - Database, RDBMS on **Server**

  The Client Access/400 clients are multi-tier, client server implementations. The client driver DLL (EHNODBC3.DLL for the Windows 3.1 client, EHNODBC2.DLL for the Optimized for OS/2 client) interfaces with the host driver program (QZDAINIT) to provide the AS/400 ODBC support.

### 1.4 ODBC Conformance Levels

The ODBC standard allows for drivers to provide different levels of function. Conformance levels are used to define the function provided. These conformance levels cover both the API interface to ODBC and the SQL statements supported by the driver.

- **API conformance levels**
  - **Core API**
    - Allocate and free environment, connection, and statement handles.
    - Connect to data source, use multiple statements on connection.
    - Prepare and execute SQL statements, execute SQL statements immediately.
    - Assign storage for parameters in an SQL statement and result columns.
    - Retrieve data from a results set and about a results set.
    - Commit or rollback transactions.
    - Retrieve error information.
  - **Level 1 API**
    - Core API functionality.
    - Connect to data sources with driver specific dialog boxes.
    - Set and inquire values of statement and connection options.
    - Send all or part of a parameter value (useful for long data).
    - Retrieve all or part of a result column.
    - Retrieve catalog information (columns, special columns, and tables).
- Retrieve information about driver and data source capabilities.

- Level 2 API
  - Core and level 1 functionality.
  - Browse available connections and list available data sources.
  - Send arrays of parameter values.
  - Retrieve arrays of result column values.
  - Use a scrollable cursor.
  - Retrieve the native form of an SQL statement.
  - Retrieve catalog information (privileges, keys, and procedures).
  - Call a translation API.

- SQL conformance levels
  - Minimum SQL Grammar
    - Data Definition Language (DDL): CREATE TABLE and DROP TABLE
    - Data Manipulation Language (DML): simple SELECT, INSERT, UPDATE, DELETE, SEARCHED
    - Expressions: simple (A > B + C)
    - Data Types: CHAR
  - Core SQL Grammar
    - Minimum SQL grammar
    - DDL: ALTER TABLE, CREATE INDEX, DROP INDEX, CREATE VIEW, DROP VIEW, GRANT, and REVOKE
    - DML: full SELECT, positioned UPDATE, and positioned DELETE
    - Expressions: subquery, set functions such as SUM and MIN
    - Data types: VARCHAR, DECIMAL, NUMERIC, SMALLINT, INTEGER, REAL, FLOAT, DOUBLE PRECISION
  - Extended SQL Grammar
    - Minimum and Core SQL grammar
    - DML: outer joins
    - Expressions: scalar functions such as SUBSTRING and ABS, date, time and timestamp literals
    - Data types: LONG
    - VARCHAR, BIT, TINYINT, BIGINT, BINARY, VARBINARY, LONG
    - VARCHAR, DATE, TIME, TIMESTAMP
    - Batch SQL statements
    - Procedure calls

The conformance levels of the V3R1 Client Access/400 ODBC drivers are indicated in Table 1.

<table>
<thead>
<tr>
<th></th>
<th>Client Access/400 for Windows 3.1</th>
<th>Client Access/400 Optimized for OS/2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>V3R1M0</td>
<td>V3R1M1</td>
</tr>
<tr>
<td>ODBC Version</td>
<td>1.0</td>
<td>2.0</td>
</tr>
<tr>
<td>API Conformance Level</td>
<td>Level 2</td>
<td>Level 2</td>
</tr>
<tr>
<td>SQL Conformance Level</td>
<td>Minimum</td>
<td>Minimum</td>
</tr>
</tbody>
</table>
In V3R1, DB2/400 supports all of the core level of SQL grammar functions with the exception of ALTER TABLE add/drop column and some RI features. DB2/400 also supports many parts of the grammar that are classified as Extended SQL. These include:

- Outer Join.
- Parts of ALTER TABLE support.
- Positioned DELETE statement.
- Stored Procedures.
- SELECT with FOR UPDATE OF clause.
- Many of the extended elements used in SQL statements.

The following list describes special handling of certain ODBC APIs by the CA/400 ODBC driver.

- SQLExtendedFetch
  - Cursors opened for update cannot call SQLExtendedFetch to read rows but must use SQLFetch.
  - You cannot use SQLExtendedFetch in combination with SQLSetPos and SQLGetData.
  - SQL_FETCH_BOOKMARKS is not supported.
  - SQLExtendedFetch cannot be used to retrieve result sets for cataloging functions (SQLTables, SQLSpecialColumns, SQLStatistics, SQLColumns, SQLForeignKeys, SQLPrimaryKeys).

- SQLSetPos
  - SQL_UPDATE, SQL_DELETE, SQL_ADD options are not supported.
  - SQL_LOCK_EXCLUSIVE, SQL_LOCK_UNLOCK are not supported.

- SQLSetScrollOption
  - SQL_CONCUR_ROWVER, SQL_CONCUR_VALUES are not supported for the concurrency option.
  - SQL_SCROLL_KEYSET_DRIVEN is changed to SQL_SCROLL_DYNAMIC.

- SQLColumnPrivileges, SQITablePrivileges returns SQL_SUCCESS, then fetch returns SQL_NO_DATA_FOUND.

- SQLSetStmtOption
  - SQL_USE_BOOKMARKS is not supported.
  - SQL_RETRIEVE_DATA is always set to SQL_RD_ON (default).
  - SQL_SIMULATE_CURSOR is not supported.

- SQLSetConnectOption with SQL_TRANSLATE_DLL, SQL_TRANSLATE_OPTION is not supported.

Note

The same limitations that exist with AS/400 DB2/400 are imposed on the ODBC SQL grammar. For further information, refer to Appendix A of the DB2/400 SQL Reference, SC41-3162.

In order to claim a conformance level, all of the features for a particular level must be supported. The V3R1 ODBC driver is classified as minimum SQL Conformance level, but it supports almost all requirements for Core SQL Grammar and a large set of those required for Extended SQL Grammar.
ODBC has different versions. The version of ODBC defines which APIs are available for the application to use. In version 1.0, for instance, there are a set of APIs (functions) defined. In version 2.0, most of the APIs are the same as the version 1.0 APIs but there are also some that replace 1.0 APIs. This version is important because it defines the interfaces that the application uses, but it should not be confused with conformance levels.

With Client Access/400 for Windows 3.1 (V3R1M0) and the OS/2 Optimized client, an ODBC 1.0 driver is provided. Client Access/400 for Windows V3.1 (V3R1M1) provides an ODBC 2.0 driver. These drivers are provided to enable applications to access data in an AS/400 database through the Open Database Connectivity (ODBC) interface. Figure 2 shows the ODBC architecture for the Client Access/400 clients.

On the AS/400 system, ODBC requests are submitted to the QSERVER subsystem where QZDAINIT pre-started jobs run. Each request to connect a specific data source initiated by the Client Access/400 ODBC driver uses a QZDAINIT pre-started job, servicing the user profile specified in the connection string. If the DOS extended client or the NetSoft router is used, QZDAINIT runs in the QCMN subsystem.

Because the V3R1 ODBC driver accesses the new V3R1 Database server, it cannot be used with V2 systems.

---

**Figure 2. Application/Driver Architecture**
1.6 V3R1 ODBC Support

The V3R1 ODBC support has been greatly improved over the support provided in V2R3. The V3R1 ODBC driver performance improvements result because:

- The Remote SQL interface is no longer used.
- A new enhanced AS/400 data access server is provided.
- A new architected data stream to the server is used.
- Servers provide blocking and low-level fast path support for fetches.
- Previously PREPARED statements are stored in packages on the server for faster executions.
- It utilizes pre-started server tasks for faster connects.
- More ODBC functions are available such as extended fetch and stored procedures.

1.7 Calling ODBC Functions

ODBC API functions fall into several categories:

- Setting up the ODBC environment.
- Establishing connections to data sources.
- Executing SQL statements.
- Cleaning up the ODBC environment.

1.8 Basic Application Steps

An ODBC application needs to follow a basic set of steps in order to access a database server.

1. Connect to the data source.
2. Place the SQL statement string to be executed in a buffer. This is a text string.
3. Submit the statement for prepared or immediate execution.
   - Retrieve and process the results.
   - If there are errors, retrieve the error information from the driver.
4. End each transaction with a commit or rollback operation (if necessary).
5. Terminate the connection.
Chapter 2. Introduction to the TPC-C Benchmark Application

In the following chapters, we introduce various new concepts, VisualAge parts and programming techniques, that are illustrated with numerous examples. These examples are all based on a reduced version of the TPC-C benchmark application and the TPC-C database.

In the following section, we introduce the application. We discuss the requirements specifications and make a first phase object analysis. In another section, we provide you with the database layout.

2.1 Requirements Specification of the TPC-C Benchmark Application

Company EPY is a wholesale supplier with one warehouse and 10 districts. Each district serves a set of about 3000 customers. The warehouse maintains a stock for the 100,000 items sold by the company.

Company EPY currently runs its business on an AS/400 based solution, but wants to move towards a more advanced client/server and object-oriented approach. They have chosen VisualAge for Smalltalk as their new development environment, but want to keep the AS/400 system as the server of choice. In a first stage of the project, they developed a prototype of their order entry application. The district that has to test the prototype is District 1.

The best way to describe the order entry application is with the following scenario:

1. Customers call one of the 10 districts to place an order.
2. The district customer service representative answers the call and gathers the following information:
   - Customer number
   - Item numbers of the requested items
   - Required quantity of each item
3. The application performs the following tasks:
   a. Check whether the requested quantity is available in the stock for every requested item.
   b. Warn the representative and cancel the order if the result is negative.
   c. Write the new order and update the inventory in stock if the result is positive.

2.2 Phase 1 Object Analysis

The technique we adopt for the problem domain analysis is based on use cases and CRC cards (Class Responsibility Collaborators). We also keep in mind that the development process is an iterative process and that at all stages, decisions have to be made. These can always be changed later when they seem to be inappropriate.

In order to discover potential problem domain objects, we start reading the scenario (use case), described in 2.1, “Requirements Specification of the TPC-C
Benchmark Application™ and list all of the nouns that seem to relate to the application.

This results in the following list:

- Customer
- District
- Order
- Service representative
- Customer number
- Item number
- Item
- Quantity
- Stock

If we take a closer look at this list of potential objects, you can see that:

- Service representative does not belong to the scope of the application. It is the person that works with the application.
- Customer number, item number, and quantity are properties of one of the other potential objects.

When we remove these candidates, we end up with the following list of objects:

- Customer
- District
- Order
- Item
- Stock

Common sense tells us that this list is too limited. An order always contains multiple order lines and company EPY has a complete catalog of items to sell. Order line and catalog are added to the list of objects. This finally gives the following objects:

- Customer
- District
- Order
- Order line
- Item
- Stock
- Catalog

### 2.3 TPC-C Benchmark Database Layout

The sample application uses the following tables of the TPC-C benchmark database:

- District
- Customer
- Order
- Order line
- Stock
- Item (Catalog)

The following sections describe in detail the layout of these tables.
### 2.3.1 District

**Table 2. District Table Layout (DSTRCT)**

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Real Name</th>
<th>Type</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>DID</td>
<td>District ID</td>
<td>Decimal</td>
<td>3</td>
</tr>
<tr>
<td>DWID</td>
<td>Warehouse ID</td>
<td>Character</td>
<td>4</td>
</tr>
<tr>
<td>DNAME</td>
<td>District Name</td>
<td>Character</td>
<td>10</td>
</tr>
<tr>
<td>DADD1</td>
<td>Address Line 1</td>
<td>Character</td>
<td>20</td>
</tr>
<tr>
<td>DADD2</td>
<td>Address Line 2</td>
<td>Character</td>
<td>20</td>
</tr>
<tr>
<td>DCITY</td>
<td>City</td>
<td>Character</td>
<td>20</td>
</tr>
<tr>
<td>DSTATE</td>
<td>State</td>
<td>Character</td>
<td>2</td>
</tr>
<tr>
<td>DZIP</td>
<td>Zip Code</td>
<td>Character</td>
<td>10</td>
</tr>
<tr>
<td>DTAX</td>
<td>Tax</td>
<td>Decimal</td>
<td>5</td>
</tr>
<tr>
<td>DYTD</td>
<td>Year to Date Balance</td>
<td>Decimal</td>
<td>13</td>
</tr>
<tr>
<td>DNXTOR</td>
<td>Next Order Number</td>
<td>Decimal</td>
<td>9</td>
</tr>
</tbody>
</table>

**Note:** Primary Key: DID, DWID

### 2.3.2 Customer

**Table 3 (Page 1 of 2). Customer Table Layout (CSTMR)**

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Real Name</th>
<th>Type</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>CID</td>
<td>Customer ID</td>
<td>Character</td>
<td>4</td>
</tr>
<tr>
<td>CDID</td>
<td>District ID</td>
<td>Decimal</td>
<td>3</td>
</tr>
<tr>
<td>CWID</td>
<td>Warehouse ID</td>
<td>Character</td>
<td>4</td>
</tr>
<tr>
<td>CFIRST</td>
<td>First Name</td>
<td>Character</td>
<td>16</td>
</tr>
<tr>
<td>CINIT</td>
<td>Middle Initials</td>
<td>Character</td>
<td>2</td>
</tr>
<tr>
<td>CLAST</td>
<td>Last Name</td>
<td>Character</td>
<td>16</td>
</tr>
<tr>
<td>CLDATE</td>
<td>Date of Last Order</td>
<td>Numeric</td>
<td>8</td>
</tr>
<tr>
<td>CADD1</td>
<td>Address Line 1</td>
<td>Character</td>
<td>20</td>
</tr>
<tr>
<td>CCREDT</td>
<td>Credit Status</td>
<td>Character</td>
<td>2</td>
</tr>
<tr>
<td>CADD2</td>
<td>Address Line 2</td>
<td>Character</td>
<td>20</td>
</tr>
<tr>
<td>CDCT</td>
<td>Discount</td>
<td>Decimal</td>
<td>5</td>
</tr>
<tr>
<td>CCITY</td>
<td>City</td>
<td>Character</td>
<td>20</td>
</tr>
<tr>
<td>CSTATE</td>
<td>State</td>
<td>Character</td>
<td>2</td>
</tr>
<tr>
<td>CZIP</td>
<td>Zip Code</td>
<td>Character</td>
<td>10</td>
</tr>
<tr>
<td>CPHONE</td>
<td>Phone Number</td>
<td>Character</td>
<td>16</td>
</tr>
<tr>
<td>CBAL</td>
<td>Balance</td>
<td>Decimal</td>
<td>7</td>
</tr>
<tr>
<td>CCRDLM</td>
<td>Credit Limit</td>
<td>Decimal</td>
<td>7</td>
</tr>
<tr>
<td>CYTD</td>
<td>Year To Date</td>
<td>Decimal</td>
<td>13</td>
</tr>
</tbody>
</table>
Table 3 (Page 2 of 2). Customer Table Layout (CSTMR)

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Real Name</th>
<th>Type</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPAYCNT</td>
<td>Quantity</td>
<td>Decimal</td>
<td>5</td>
</tr>
<tr>
<td>CDELCNT</td>
<td>Quantity</td>
<td>Decimal</td>
<td>5</td>
</tr>
<tr>
<td>CLTIME</td>
<td>Time of Last Order</td>
<td>Numeric</td>
<td>6</td>
</tr>
<tr>
<td>CDATA</td>
<td>Customer Information</td>
<td>Character</td>
<td>500</td>
</tr>
</tbody>
</table>

Note: Primary Key: CID, CDID, CWID

2.3.3 Order

Table 4. Orders Table Layout (ORDERS)

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Real Name</th>
<th>Type</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>OWID</td>
<td>Warehouse ID</td>
<td>Character</td>
<td>4</td>
</tr>
<tr>
<td>ODID</td>
<td>District ID</td>
<td>Decimal</td>
<td>3</td>
</tr>
<tr>
<td>OCID</td>
<td>Customer ID</td>
<td>Character</td>
<td>4</td>
</tr>
<tr>
<td>OID</td>
<td>Order ID</td>
<td>Decimal</td>
<td>9</td>
</tr>
<tr>
<td>OENTDT</td>
<td>Order Date</td>
<td>Numeric</td>
<td>8</td>
</tr>
<tr>
<td>OENTTM</td>
<td>Order Time</td>
<td>Numeric</td>
<td>6</td>
</tr>
<tr>
<td>OCARID</td>
<td>Carrier Number</td>
<td>Character</td>
<td>2</td>
</tr>
<tr>
<td>O_LINES</td>
<td>Number of Order Lines</td>
<td>Decimal</td>
<td>3</td>
</tr>
<tr>
<td>OLOCAL</td>
<td>Local</td>
<td>Decimal</td>
<td>1</td>
</tr>
</tbody>
</table>

Note: Primary Key: OWID, ODID, OID

2.3.4 Order Line

Table 5. Order Line Table Layout (ORDLIN)

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Real Name</th>
<th>Type</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>OLOID</td>
<td>Order ID</td>
<td>Decimal</td>
<td>9</td>
</tr>
<tr>
<td>OLDID</td>
<td>District ID</td>
<td>Decimal</td>
<td>3</td>
</tr>
<tr>
<td>OLWID</td>
<td>Warehouse ID</td>
<td>Character</td>
<td>4</td>
</tr>
<tr>
<td>OLNBR</td>
<td>Order Line Number</td>
<td>Decimal</td>
<td>3</td>
</tr>
<tr>
<td>OlSPWH</td>
<td>Supply Warehouse</td>
<td>Character</td>
<td>4</td>
</tr>
<tr>
<td>OLIID</td>
<td>Item ID</td>
<td>Character</td>
<td>6</td>
</tr>
<tr>
<td>OLOQTY</td>
<td>Quantity Ordered</td>
<td>Numeric</td>
<td>3</td>
</tr>
<tr>
<td>OLMAMNT</td>
<td>Amount</td>
<td>Numeric</td>
<td>7</td>
</tr>
<tr>
<td>OLDLVD</td>
<td>Delivery Date</td>
<td>Numeric</td>
<td>8</td>
</tr>
<tr>
<td>OLDLVT</td>
<td>Delivery Time</td>
<td>Numeric</td>
<td>6</td>
</tr>
<tr>
<td>OLDSTI</td>
<td>District Information</td>
<td>Character</td>
<td>24</td>
</tr>
</tbody>
</table>

Note: Primary Key: OLWID, OLDID, OLOID, OLNBR
2.3.5 Item (Catalog)

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Real Name</th>
<th>Type</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>IID</td>
<td>Item ID</td>
<td>Character</td>
<td>6</td>
</tr>
<tr>
<td>INAME</td>
<td>Item Name</td>
<td>Character</td>
<td>24</td>
</tr>
<tr>
<td>IPRICE</td>
<td>Price</td>
<td>Decimal</td>
<td>5</td>
</tr>
<tr>
<td>IDATA</td>
<td>Item Information</td>
<td>Character</td>
<td>50</td>
</tr>
</tbody>
</table>

Note: Primary Key: IID

2.3.6 Stock

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Real Name</th>
<th>Type</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>STWID</td>
<td>Warehouse ID</td>
<td>Character</td>
<td>4</td>
</tr>
<tr>
<td>STIID</td>
<td>Item ID</td>
<td>Character</td>
<td>6</td>
</tr>
<tr>
<td>STQTY</td>
<td>Quantity in Stock</td>
<td>Decimal</td>
<td>5</td>
</tr>
<tr>
<td>STDI01</td>
<td>District Information</td>
<td>Character</td>
<td>24</td>
</tr>
<tr>
<td>STDI02</td>
<td>District Information</td>
<td>Character</td>
<td>24</td>
</tr>
<tr>
<td>STDI03</td>
<td>District Information</td>
<td>Character</td>
<td>24</td>
</tr>
<tr>
<td>STDI04</td>
<td>District Information</td>
<td>Character</td>
<td>24</td>
</tr>
<tr>
<td>STDI05</td>
<td>District Information</td>
<td>Character</td>
<td>24</td>
</tr>
<tr>
<td>STDI06</td>
<td>District Information</td>
<td>Character</td>
<td>24</td>
</tr>
<tr>
<td>STDI07</td>
<td>District Information</td>
<td>Character</td>
<td>24</td>
</tr>
<tr>
<td>STDI08</td>
<td>District Information</td>
<td>Character</td>
<td>24</td>
</tr>
<tr>
<td>STDI09</td>
<td>District Information</td>
<td>Character</td>
<td>24</td>
</tr>
<tr>
<td>STDI10</td>
<td>District Information</td>
<td>Character</td>
<td>24</td>
</tr>
<tr>
<td>STYTD</td>
<td>Year To Date</td>
<td>Decimal</td>
<td>9</td>
</tr>
<tr>
<td>STORDRS</td>
<td>Quantity</td>
<td>Decimal</td>
<td>5</td>
</tr>
<tr>
<td>STREMORD</td>
<td>Quantity</td>
<td>Decimal</td>
<td>5</td>
</tr>
<tr>
<td>STDATA</td>
<td>Item Information</td>
<td>Character</td>
<td>50</td>
</tr>
</tbody>
</table>

Note: Primary Key: STWID, STIID

2.4 Database Terminology

This redbook concentrates on the use of the AS/400 system as the database server in an environment where you develop applications with VisualAge for Smalltalk. The focal point of this publication is the use of ODBC and SQL to access the data in the database. However, we also make a comparison between the ODBC access and native access to the data using the AS/400 Connection, which is a separately purchasable feature of VisualAge for Smalltalk.
The terminology used for the database access is different in both cases. In Table 8 on page 14, you find the correspondence between the different terms.

<table>
<thead>
<tr>
<th>Table 8. Database Terminology</th>
</tr>
</thead>
<tbody>
<tr>
<td>AS/400 Native</td>
</tr>
<tr>
<td>Library</td>
</tr>
<tr>
<td>Physical File</td>
</tr>
<tr>
<td>Field</td>
</tr>
<tr>
<td>Record</td>
</tr>
<tr>
<td>Logical File</td>
</tr>
</tbody>
</table>
Chapter 3. Installation of VisualAge for Smalltalk with ODBC Support and Client Access/400

This chapter covers the installation of the ODBC Support feature into your VisualAge for Smalltalk environment and the installation of the ODBC driver that comes with Client Access/400.

3.1 Installation of VisualAge for Smalltalk with ODBC Support

In VisualAge for Smalltalk Version 3, the ODBC support is part of the base product. This means that you have to specify at the moment that you install VisualAge for Smalltalk whether you want this feature to be installed on your system or not. When you start the installation program and choose to install the standard or professional version of VisualAge for Smalltalk, you are shown with a list of components that you can install. Choose from this list at least VisualAge Base and ODBC Support. It is up to you to install additional components, but the two components mentioned are necessary when you want to develop VisualAge applications with ODBC support.

![Figure 3. Install VisualAge for Smalltalk on OS/2](image)

The bytes needed number as shown in Figure 3 is for a Professional Standalone installation, for a Professional Client installation the number is approximately 30 MB.

With VisualAge for Smalltalk Version 3.0a the installation process of VisualAge is no longer identical for OS/2 and Windows. VisualAge for OS/2 uses a front end installation program called CD Showcase, while VisualAge for Windows uses a front end installation program called Install Shield.
If you installed the base product without choosing to install ODBC support, you will need to run the installation utility again. This time you choose only ODBC Support from the list of components.

This completes the first step of the installation. Once you have the ODBC Support installed on your system, you have to load it into your manager and/or image. Go to the VisualAge Organizer and select Options from the menu bar. From the drop-down menu, select Load Features.... This opens a window with a list of features that are installed on your system and that can be loaded into your manager and/or image. Select ODBC Database from this list and optionally select ODBC Database VA Samples.

**Note:** The only time you will actually be loading a feature into the “manager”, is the very first time the first person connected to the manager loads the feature. Any time after that the feature is requested to be loaded, it is already loaded into the manager and would only be loaded into the “image”.

*Figure 4. Selecting Components and Features from Windows Install Shield*
3.2 Installation of Client Access/400

This section covers the installation of Client Access/400. The installation path to follow is determined by the operating system that you are using. When you are an OS/2 user, you have to install the Client Access/400 Optimized for OS/2 client. When you are installing in a Windows 3.1 or 3.11 environment, you need to install Client Access/400 for Windows 3.1.

3.2.1 Installation of Client Access/400 Optimized for OS/2

For the installation of Client Access/400 Optimized for OS/2, we want to reference Client Access/400 Optimized for OS/2 - Getting Started Version 3, SC41-3510, and Inside Client Access/400 Optimized for OS/2, SG24-2587. In these publications, you find all of the information that you need to install this product.

You also need to install the following PTFs on Client Access/400 (product number 5763XG1 version V3R1M1):

- SF28583
- SF28840

In this section, we want to concentrate on the known problems with the ODBC driver supplied with Client Access/400 Optimized for OS/2 and how to overcome those problems. There are two major problems with this ODBC driver:

1. In order to use the Client Access/400 ODBC driver, it is necessary to have an ODBC driver manager and an administration program. These programs are not included in Client Access/400 Optimized for OS/2.

2. The ODBC driver returns an error code if the ODBC driver’s MRI file, EHNODBC2.MRI0, is not in a subdirectory underneath the directory from which the executing program is run.

In the following sections, we explain how to fix these problems.
3.2.1.1 ODBC Driver Manager and Administration Programs

These programs are not included in Client Access/400. They are available from companies such as Visigenic Software (http://www.visigenic.com) and Intersolv (http://www.intersolv.com).

They are also included in the VisualAge for Smalltalk ODBC feature. You can find an ODBC administrator program ODBCADM.EXE in the DLL subdirectory of the directory where you installed VisualAge.

You might consider creating a program object for this administrator. Use the Program template in the Templates folder. Drag this template onto your desktop. This opens a settings notebook. On the first page, you have to specify the path and file name of the program: C:\VISUALAG\DLL\ODBCADM.EXE, where C:\VISUALAG is the directory where you installed VisualAge. On the last page of the notebook, you can change the name of the program. Change it into ODBC Administrator. You can now close the notebook. On your desktop, you find an icon for the ODBC Administrator.

![Figure 6. ODBC Administrator Settings Notebook](image)

3.2.1.2 Missing EHNODBC2.MR0 File

You can find this file in a subdirectory of the directory where you installed Client Access/400 Optimized for OS/2. Let’s consider that Client Access/400 is installed in the directory C:\CAOS2. In this directory, you can find a subdirectory named MRIxxxx, where xxxx is the identifier of the locale that you used to install Client Access/400. When you installed a U.S. English version of Client Access/400, this identifier is equal to 2924. This is the directory that contains the EHNODBC2.MR0 file.

To solve the problem with the ODBC driver, you have to create a subdirectory MRIxxxx underneath the directory from which the program that is using the ODBC driver is running, and copy the EHNODBC2.MR0 file to this directory. In our case, this means that you have to create a MRIxxxx subdirectory underneath the directory where you installed VisualAge. Let’s assume that you installed
VisualAge in the directory C:\VISUALAG; then you can use the following OS/2 commands to create the necessary directory and copy the MRI file to it.

C:\>cd visualag
C:\VISUALAG>md mri2924
C:\VISUALAG>cd mri2924
C:\VISUALAG\MRI2924>copy c:\caos2\mri2924\ehnodbc2.mr0

You have to go through this operation for every installation of VisualAge where you are going to use ODBC support as well as for every installation of the applications that you developed with VisualAge and that use the ODBC driver of Client Access/400 Optimized for OS/2.

3.2.2 Installation of Client Access/400 for Windows 3.1

For the installation of Client Access/400 for Windows 3.1, we want to reference Client Access/400 for Windows 3.1 - Getting Started Version 3, SC41-3530, Client Access/400 for Windows 3.1 - Migration, Installation and Configuration, GG24-2571, and Inside Client Access/400 for Windows 3.1 Version 1 Release 1 Modification 1, SG24-4429. In these publications, you can find all of the information that you need to install this product.

On completion of this installation, you should find the ODBC driver, the driver manager, and the ODBC administrator installed on your system. No further action is necessary to start using the ODBC Support of Client Access/400 for Windows 3.1.
This chapter covers the configuration of data sources using the ODBC driver of Client Access/400 for both OS/2 and Windows. In a second part, we cover the configuration of the ODBC support of VisualAge for Smalltalk and how to use these data sources.

4.1 Configuration of Data Sources Using the Client Access/400 ODBC Driver

This section covers the definition of data sources using the ODBC drivers supplied by Client Access/400. The configuration schedule to follow is determined by the operating system that you are running. We describe configuration schedules for both OS/2 and Windows.

4.1.1 Configuration of the Client Access/400 Optimized for OS/2 ODBC Driver

Client Access/400 Optimized for OS/2 only supplies you with an ODBC driver. As we discussed in 3.2.1, "Installation of Client Access/400 Optimized for OS/2" on page 17, the ODBC driver manager and ODBC administrator are not part of the product. We use the ODBC driver manager and administrator supplied by VisualAge.

To define a data source, you must go through the following steps:

1. Start the ODBC administrator by double-clicking the ODBC Administrator icon that you created in 3.2.1, "Installation of Client Access/400 Optimized for OS/2" on page 17, or by typing ODBCADM in the DLL subdirectory of the directory where you installed VisualAge. This opens the panel shown in Figure 7 showing the data sources already configured on your PC (the list is empty if you have not yet configured a data source). You may have data sources configured to ODBC drivers of other manufacturers in addition to the ones of Client Access/400.

![Figure 7. ODBC Data Source Setup - Displaying Installed Data Sources](image)

This panel enables you to configure all of the ODBC drivers that you have installed. When you install the drivers supplied by VisualAge, you have drivers to access (for example: DB2/2, dBase, Oracle, Sybase, or Text).

- To see how many ODBC drivers are already in your PC, click on the Drivers... button.
From this panel, you can add ODBC drivers from other sources by clicking on the **Add** button, or use the **Delete** button to remove existing ODBC drivers from your PC.

- To see information about a specific ODBC driver, click on the **About** button. The panel shown in Figure 9 is an example of the Client Access/400 Optimized for OS/2 ODBC driver information panel.

Click **OK** and **Close** to go back to the Data Sources panel.

2. From the Data Sources panel, click on the **Add** button to go to the Add Data Source panel shown in Figure 10 on page 23.
3. Highlight the ODBC driver name you want to configure and click on **OK**. You see the panel shown in Figure 11.

4. Give the data source a meaningful name. In this example, we use **Client Server Performance Data** as the data source name.

5. Select the system name from the drop-down list, of the AS/400 system that you want to connect to its database.

6. Click on the **Options...** button to display the Options panel where you can specify more parameters. This opens the panel shown in Figure 12 on page 24. Replace the default libraries (QGPL and QTEMP) with library **CSDB**. Leave the other parameters unchanged.
7. Select **OK**. You see the data source that you have just defined in the Data Sources panel.

8. Select **Close** when done.

You have now defined a data source that consists of all of the data in the CSDB and QIWS libraries on your AS/400 system (QIWS gets added automatically). This data source can be used by any ODBC-compliant OS/2 application. In the following chapters, we show how you can create VisualAge for Smalltalk applications that retrieve their information from this data source.

### 4.1.1.1 Description of Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Data Source Name</strong></td>
<td>This is the name by which the data source is known by applications that use the ODBC driver and can be up to 32 characters.</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>Use this to describe the contents of the data source (for example, <em>Client Server Performance Data</em>). Maximum length is 80 characters.</td>
</tr>
<tr>
<td><strong>System</strong></td>
<td>The name of the AS/400 system in which the data is found. This value can only be set by choosing a system name from the drop-down list.</td>
</tr>
<tr>
<td><strong>Commit Mode</strong></td>
<td>This is the SQL data access commit mode you want to use with this data source.</td>
</tr>
</tbody>
</table>
Note

If you are going to access an AS/400 physical file that is not journaled, you must use Commit Mode = *NONE. Use the other options only when you are going to access either SQL tables or journaled AS/400 physical files.

Default Libraries

These are the names of the libraries that the Client Access/400 ODBC driver searches. You can enter multiple libraries separated by commas or spaces in this text box. Always use capital letters.

Libraries can either be added to the current server job’s library list or replace the list entirely. To replace the list, simply specify a list of library names. To add to the existing list, you need to add a special entry, *USRLIBL, to the list of libraries. All libraries listed before *USRLIBL are added to the front of the library list and all libraries listed after *USRLIBL are added to the end of the list. The first library in the list is used as the default collection. If you do not want a default collection, start the list with a comma.

Library QIWS gets added automatically.

Naming Convention

This determines whether the AS/400 naming convention is used (using a / between library and filename) or the SQL naming convention (where a period is used as the delimiter).

Decimal Format

User-defined decimal format.

Date Format

User-defined date format.

Date Separator

User-defined date separator.

Time Format

User-defined time format.

Time Separator

User-defined time separator.

4.1.1.2 Tracing ODBC Calls

The VisualAge ODBC administrator allows you to trace ODBC calls. This option can be used as a problem analysis tool if you encounter errors with a database query through ODBC. Or, you can just use it to record streams of SQL statements to study the syntax and data stream flow used for performance analysis. The trace records SQL streams into an ASCII text file, by default called SQL.LOG.

Follow these steps to enable the ODBC trace option:

1. From the Data Sources panel in Figure 7 on page 21, click on the Options... button to invoke the panel in Figure 13 on page 26.
2. Click on the check boxes to:

   - Start the **Trace ODBC Calls** option.
   - **Stop Tracing Automatically**. If this box is checked, tracing runs only once. Tracing stops when the application that uses ODBC stops. You need to check the **Trace ODBC Calls** box again to restart tracing.

   You can see the name of the current trace log file to be used in the Trace field.

3. If you want to use a different trace log file, click on the **Select File** button.

4. Click on **OK** when done.

You are now able to trace the ODBC calls between your PC and the AS/400 system that holds the data source.

**Sample ODBC Trace Log:** This is an example of the content of an ODBC trace log file.

```
T4 SQLCONNECT(hdbc C6064C, szDSN "Tpecs\00bd\00bcsDataSource\006M1", cbDSN -3, szUID "A960209C", cbUID -3, szAuthStr "PETER", cbAuthStr -3)
T1 SQLGETINFO(hdbc C6064C, fInfoType 46, rgbInfoValue BA1180, cbInfoValueMax 4, pcbInfoValue BA1190)
T1 SQLSETCONNECTOPTION(hdbc C6064C, fOption 102, vParam 0)
T1 SQLSETCONNECTOPTION(hdbc C6064C, fOption 1041, vParam 201326592)
T1 SQLERROR(henv C600D0, hdbc C6064C, hstmt 0, szSqlState "", pfNativeError BA22AC, szErrorMsg "", cbErrorMsgMax 511, pcbErrorMsg 8A24C8)
T1 SQLERROR(henv C600D0, hdbc C6064C, hstmt 0, szSqlState "", pfNativeError BAA0A4, szErrorMsg "", cbErrorMsgMax 511, pcbErrorMsg 8A2C20)
T1 SQLALLOCSTMT(hdbc C6064C, phstmt 8A4DA8)
T1 SQLPREPARE(hstmt C62180, szSqlStr "SELECT * FROM CSTMR 
WHERE (((CSTMR.CWID = '0001') 
AND (CSTMR.CDID = 1)) 
AND (CSTMR.CID = ?))", cbSqlStr -3)
T1 SQLBINDPARAMETER(hstmt C62180, ipar 1, fCType 1, fSqlType 1, cbColDef 4, ibScale 0, rgbValue 1120000, pcbValue 1130000)
T1 SQLSETPARAM(hstmt C62180, ipar 1, fCType 1, fSqlType 1, cbColDef 4, ibScale 0, rgbValue 1120000, pcbValue 1130000)
T1 SQLNUMRESULTCOLS(hstmt C62180, pccol 8A578C)
T1 SQLCOLATTRIBUTES(hstmt C62180, icol 1, fDescType 1, rgbDesc BA5968, cbDescMax 32, pcbDesc BA5994, pfDesc 0)
T1 SQLCOLATTRIBUTES(hstmt C62180, icol 1, fDescType 2, rgbDesc BA600C, cbDescMax 32, pcbDesc BA6038, pfDesc 0)
T1 SQLCOLATTRIBUTES(hstmt C62180, icol 1, fDescType 3, rgbDesc 0, cbDescMax 0, pcbDesc 0, pfDesc BA658B)
T1 SQLCOLATTRIBUTES(hstmt C62180, icol 1, fDescType 4, rgbDesc 0, cbDescMax 0, pcbDesc 0, pfDesc BA677C)
T1 SQLCOLATTRIBUTES(hstmt C62180, icol 1, fDescType 5, rgbDesc 0, cbDescMax 0, pcbDesc 0, pfDesc BA6760)
T1 SQLCOLATTRIBUTES(hstmt C62180, icol 1, fDescType 6, rgbDesc 0, cbDescMax 0, pcbDesc 0, pfDesc BA67C0)
T1 SQLCOLATTRIBUTES(hstmt C62180, icol 1, fDescType 7, rgbDesc 0, cbDescMax 0, pcbDesc 0, pfDesc BA6744)
T1 SQLCOLATTRIBUTES(hstmt C62180, icol 1, fDescType 8, rgbDesc 0, cbDescMax 0, pcbDesc 0, pfDesc BA671A)
```

26  OO Programming with CA/400 and ODBC using VA for ST
4.1.1.3 Configuration file

The Client Access/400 Optimized for OS/2 ODBC driver uses the ODBC.INI file in the OS2 directory. This file is created and updated by the ODBC Administrator that is accessed through the program object created in 3.2.1, “Installation of Client Access/400 Optimized for OS/2” on page 17, or by typing ODBCADM in the DLL subdirectory of the directory where you installed VisualAge. This file is a binary file that is only accessed with appropriate tools as, for example, an INI viewer.

Usually ODBC.INI should be updated using the configuration program described in 4.1.1, “Configuration of the Client Access/400 Optimized for OS/2 ODBC Driver” on page 21.

4.1.2 Configuration of the Client Access/400 for Windows 3.1 ODBC Driver

To define a data source with the Client Access/400 for Windows 3.1 ODBC driver, you must go through the following steps:

1. To start ODBC driver configuration, select the ODBC Driver icon from the Client Access/400 for Windows 3.1 group. The panel shown in Figure 14 on page 28 is displayed showing the data sources already set up on your PC (there can be none if you have not yet configured any). You may have data sources configured to ODBC drivers of other manufacturers in addition to the ones of Client Access/400.
This panel enables you to configure all of the ODBC drivers that you have installed. When you install the drivers supplied by VisualAge, you have drivers to access (for example: DB2/2, dBase, Oracle, Sybase, or Text).

- To see how many ODBC drivers are already in your PC, click on the Drivers... button.

From this panel, you can add ODBC drivers from other sources by clicking on the Add... button, or use the Delete button to remove existing ODBC drivers from your PC.

- To see information about a specific ODBC driver, click on the About... button. The panel shown in Figure 16 is an example of the Client Access/400 for Windows 3.1 ODBC driver information panel.

Click OK and Close to go back to the Data Sources panel.
2. From the Data Sources panel, click on the Add... button to go to the Add Data Source panel shown in Figure 17 on page 29.

3. Highlight the ODBC driver name you want to configure and click on OK. You see the panel shown in Figure 18.

4. Give the data source a meaningful name. In this example, we use Client Server Performance Data as the data source name.

5. Type in or search for the system name of the AS/400 system that you want to connect to its database by clicking on the drop-down button.

6. Add the CSDB library to the default library group. Keep in mind that the fewer libraries used, the better performance is in bringing up lists of libraries and files in those libraries.

7. Click on the Options>> button to specify other parameters for Format Options, Performance Options, and Miscellaneous Options as needed.
Figure 19. ODBC Data Source Setup - Format Options

Figure 20. ODBC Data Source Setup - Performance Options
8. Select **OK**. You see the data source that you have just defined in the Data Sources panel.

9. Select **Close** when done.

You have now defined a data source that consists of all data in the QGPL, CSDB, and QIWS libraries on your AS/400 system (QIWS gets added automatically). This data source can be used by any ODBC-compliant Windows application. In the following chapters, we show how you can create VisualAge for Smalltalk applications that retrieve their information from this data source.

### 4.1.2.1 Description of Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Data Source Name</strong></td>
<td>This is the name by which the data source is known by applications that use the ODBC driver and can be up to 32 characters.</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>Use this to describe the contents of the data source (for example, <em>Client Server Performance Data</em>). Maximum length is 80 characters.</td>
</tr>
<tr>
<td><strong>System</strong></td>
<td>The name of the AS/400 system in which the data is found. This value can only be set by choosing a system name from the drop-down list.</td>
</tr>
<tr>
<td><strong>Commit Mode</strong></td>
<td>This is the SQL data access commit mode you want to use with this data source.</td>
</tr>
</tbody>
</table>
If you are going to access an AS/400 physical file that is not journaled, you must use Commit Mode = *NONE. Use the other options only when you are going to access either SQL tables or journaled AS/400 physical files.

**Default Libraries**
These are the names of the libraries that the Client Access/400 ODBC driver searches. You can enter multiple libraries separated by commas or spaces in this text box. Always use capital letters.

Libraries can either be added to the current server job’s library list or replace the list entirely. To replace the list, simply specify a list of library names. To add to the existing list, you need to add a special entry, *USRLIBL, to the list of libraries. All libraries listed before *USRLIBL are added to the front of the library list and all libraries listed after *USRLIBL are added to the end of the list. The first library in the list is used as the default collection. If you do not want a default collection, start the list with a comma.

Library QIWS gets added automatically.

**Naming Convention**
This determines whether the AS/400 naming convention is used (using a / between library and filename, for example) or the SQL naming convention (where a period is used as the delimiter).

**Date Format**
User-defined date format.

**Date Separator**
User-defined date separator.

**Time Format**
User-defined time format.

**Time Separator**
User-defined time separator.

**Decimal Format**
User-defined decimal format.

### 4.1.2.2 Tracing ODBC Calls
The ODBC administrator allows you to trace ODBC calls. This option can be used as a problem analysis tool if you encounter errors with a database query through ODBC. Or, you can just use it to record streams of SQL statements to study the syntax and data stream flow used for performance analysis. The trace records SQL streams into an ASCII text file, by default called SQL.LOG.

Follow these steps to enable the ODBC trace option:

1. From the Data Sources panel in Figure 14 on page 28, click on the **Options** button to invoke the panel in Figure 22 on page 33.
2. Click on the check boxes to:
   • Start the **Trace ODBC Calls** option.
   • **Stop Tracing Automatically**. If this box is checked, tracing runs only once. Tracing stops when the application that uses ODBC stops. You need to check the **Trace ODBC Calls** box again to restart tracing.

   You can see the name of the current trace log file to be used in the **Trace File** field.

3. If you want to use a different trace log file, click on the **Select File...** button.

4. Click on **OK** when done.

You are now able to trace the ODBC calls between your PC and the AS/400 system that holds the data source.

**Sample ODBC Trace Log:** This is an example of the content of an ODBC trace log file.

```c
SQLAllocEnv(phenv2FD70000);  
SQLAllocConnect(hdbc390F0000, phdbc390F0000);  
SQLConnect(hdbc390F0000, "TucksOrderEntryTestDSM01", -3, "A90209C", -3, "******", -3);  
SQLGetInfo(hdbc390F0000, 46, rgbInfoValue, 4, pcbInfoValue);  
SQLSetConnectOption(hdbc390F0000, 102, 00000000);  
SQLSetConnectOption(hdbc390F0000, 1041, 3A5F0010);  
SQLError(henv2FD70000, hdbc390F0000, hstmt00000000, szSqlState, pfNativeError, szErrorMsg, 511, pcbErrorMsg);  
SQLPrepare(hstmt3A670000, "SELECT * FROM CSTMR WHERE (((CSTMR.CWID = #0001#) AND (CSTMR.CDID = 1)) AND (CSTMR.CID = ?))", -3);  
```

**Chapter 4. Configuration of Client Access/400 ODBC Support in the VisualAge for Smalltalk Environment**
4.1.2.3 Configuration file

The Client Access/400 for Windows 3.1 ODBC driver uses the ODBC.INI file in the Windows directory. This file is created and updated by the ODBC Administrator that is accessed through the ODBC Driver icon in the Client Access/400 for Windows 3.1 group (see 4.1.2, "Configuration of the Client Access/400 for Windows 3.1 ODBC Driver" on page 27 for details on using the Administrator).

Figure 23 on page 35 shows an example ODBC.INI file. In this example, the driver has been configured for a data source called Client Server Performance Data. You can also see that the Trace ODBC Calls and Stop Tracing Automatically options are enabled.

Usually, ODBC.INI should be updated using the configuration program described in 4.1.2, "Configuration of the Client Access/400 for Windows 3.1 ODBC Driver" on page 27.
WARNING: Do not make changes to this file without using either the ODBC Setup program, the ODBC Administrator, or other utilities provided for maintaining installed drivers and data sources.

Changing these files directly could result in a loss of information required by ODBC to execute.

[ODBC Data Sources]
Client Server Performance Data=Client Access/400 ODBC Driver

[ODBC]
TraceAutoStop=1
Trace=1
TraceFile=C:\VISUAL\AG\ODBC\TRAC.LOG

[Client Server Performance Data]
Driver=c:\cawin\ehnodbc3.d1l
Description=Client Access/400 ODBC Driver
System=ODBCM01
UserID=
CommitMode=0 ; *NONE (Commit Immediate)
DefaultLibraries=QGPL, CSDB
Naming=0 ; *SQL (SQL Naming Convention .)
DateFormat=5 ; *ISO (date format yyyy-mm-dd)
DateSeparator=1 ; *Date separator -(dash)
TimeFormat=0 ; *HMS (time format hh:mm:ss)
TimeSeparator=0 ; *Time separator :(colon)
Decimal=0 ; *Decimal format .(period)
ExtendedDynamic=1 ; *Extended dynamic enabled
RecordBlocking=2 ; *Block except for FOR UPDATE OF specified
BlockSizeKB=32 ; *Blocking size from 1 to 512 KBytes
LazyClose=1 ; *LazyClose enabled
LibraryView=0 ; *Library list
ForceTranslation=0 ; *No translate for CCSID 65535
ODBCRemarks=0 ; *OS/400 object description
AlwaysScrollable=0 ; *No scrollable cursor if row set is 1

Figure 23. Sample ODBC.INI File

4.2 Configuration of ODBC Data Sources in VisualAge for Smalltalk

In this section, we introduce some new concepts about database access in VisualAge for Smalltalk. This introduction is followed by step-by-step instructions on how to set up your environment to use the ODBC data sources that were created in 4.1, "Configuration of Data Sources Using the Client Access/400 ODBC Driver" on page 21.

4.2.1 VisualAge Database Access: Concepts

Before you can start with setting up a connection to an ODBC data source, you have to be familiar with the following concepts:

- Database manager
- Database connection
- Connection specification
- Alias
• Access set

4.2.1.1 Database Manager

A database manager is the interface between VisualAge and a DBMS. When you use the database support of VisualAge, you connect to a database through a VisualAge database manager. VisualAge provides the following database managers:

IBM Database 2  The interface between VisualAge and the IBM DB2/2 DBMS. This is available when you install and load the IBM Database feature (part of the VisualAge for Smalltalk base product).

DB2 Call Level Interface  This database manager is available on all platforms. Window 95 and Windows NT users must use this database manager to access DB2. With this Call Level Interface, multiple concurrent database connections and distributed unit of work are supported.

ODBC  The ODBC interface between VisualAge and your ODBC data source. This is available when you install and load the ODBC Support feature (part of the VisualAge for Smalltalk base product).

Oracle - Native  The native interface between VisualAge and your Oracle DBMS. This is available when you install and load the Native Oracle Database feature (separately purchasable feature).

In this publication, we concentrate on the ODBC database manager, but the use of the other database managers is similar.

4.2.1.2 Database Connection, Connection Specification, and Alias

A database connection is a connection from VisualAge to a data source through a database manager. With the ODBC database manager, you can establish multiple connections with the same database manager to the same data source if the driver supports this.

To keep track of the database connections, VisualAge uses connection specifications. A connection specification associates an alias, which is a name that is used by VisualAge to refer to a data source with a Database Manager and a data source. You might use the alias ClientServerPerformanceData to refer to the Client Server Performance Data data source. A connection specification has several purposes:

• It points to the database manager and the data source that you want to use for database operations.

• It opens the possibility to develop applications independent of a specific data source. This independence implies that you can use different data sources for development and testing of an application than you use in a production environment.

• It keeps track of the connection status.

A connection specification is not restricted to one VisualAge application. You define it once and you can reuse it for every application.
If you change the data source that an alias points to, all of the parts that use this alias are updated automatically (for example, when you have the same database on two different AS/400 systems and you have two data sources that point to the database on the different machines). Let’s suppose that data source 1 points to AS/400 system 1 and data source 2 points to AS/400 system 2. The alias that your application uses points currently to data source 1. When you want your alias to point to data source 2 instead of data source 1, you only have to change the connection specification. You do not have to change the parts in your application. They adopt the changes automatically.

4.2.1.3 Access Set
An access set is a special part that you define and package with your application to store connection specifications, database query specifications and stored procedure definitions. An access set tells your application which connection specifications, queries, and stored procedures it can use.

Note
VisualAge stores all specifications in the access set with method categories that are used by VisualAge. Do not change a method category for a connection, a query, or a stored procedure specification. You can add the selector to other categories if you want.

4.2.2 Setting up a Database Connection
To define a database connection, follow these steps:
1. Define an access set for your application.
2. Create a connection specification for your data source and add the connection specification to the access set.
3. Activate the database connection using this connection specification.

4.2.2.1 Defining an Access Set
VisualAge saves connection specifications, queries, and stored procedures in access sets. An access set can be packaged with your application, such as the classes that define your visual and non-visual parts. An access set can hold many connection specifications, queries and stored procedures, and can be shared by different applications.

To define an access set, follow these steps. Make sure the application where you want to add the ODBC support is currently selected in the VisualAge Organizer, and that it is an edition.
1. From the Parts menu in the VisualAge Organizer, select New.... This opens the panel shown in Figure 24 on page 38.
2. In the **Part class** field, type a name for your access set, for example, `OdbcAccessSet`.

3. From the **Part type** drop-down list, select `Database access set`.

4. Select **OK**.

The new access set is added to the list of parts for your application.

If you use only one database manager, you can use the same access set for storing all of the connection specifications, queries, and stored procedures in an application. If you use more than one database manager (for example, ODBC and DB2/2), create an access set for each database manager. Though using separate access sets is not a requirement, it helps you to determine appropriate prerequisites when you package your application. VisualAge attempts to maintain prerequisites according to the connection specifications defined in an application.

### 4.2.2.2 Creating a Connection Specification

To define a connection specification, you must go through the following steps:

1. From the **Options** menu in the VisualAge Organizer, select **Database Connections...**. This opens the **Database connection specifications** panel in Figure 25 on page 39.
This panel is used to define, update, or delete connection specifications, and to add existing connection specifications to an application. The top portion of the panel contains a list of connection specifications currently defined in your image. The push button right below this list shows the name of the application that is currently selected in the VisualAge Organizer.

The bottom portion of the panel contains all connection specifications that have already been added to the selected application. The push buttons below this list let you create new connection specifications, update the selected one, or delete the selected one. Each time you change a connection specification, VisualAge changes the prerequisites according to the connection specification currently defined for the application.

2. To define a new connection specification, select the New... push button. The New database connection specification panel in Figure 26 on page 40 is displayed. You use this panel to associate an alias with a database manager and a data source.
3. In the **Connection alias** field, type an alias for the database manager and data source. The alias must be a valid method name (for example, `ClientServerPerformanceData`).

4. From the **Access set** drop-down list, select the access set to store the connection specification in (for example, `OdbcAccessSet`).

5. From the **Database manager** drop-down list, select the appropriate database manager. In our case, that is `ODBC`.

   **Note**

   If no database managers appear in the drop-down list, then you need to load a database support feature as we discussed in 3.1, “Installation of VisualAge for Smalltalk with ODBC Support” on page 15.

6. From the **Data source name** drop-down list, select the name of the data source that you want to associate with the alias (for example `Client Server Performance Data`).

7. To prompt users for their database logon ID and password, select the **Prompt for logon information** check box.

   When you have checked this check box, you are prompted with a logon panel as shown in Figure 27 on page 41.
You have to enter your userid, password, and the data source that you want to connect to. This means that at the moment of the logon, you can decide with which data source you want to associate the alias of the connection specification that you are activating.

**Note**

When you are using the ODBC driver of Client Access/400, be sure to enter USERID and PASSWORD in capitals. Otherwise, the connection attempt fails.

If you did not check this check box, you are prompted with the panel in Figure 28, displaying all the SQL data sources that are configured on your workstation. You have to choose at that moment with which data source you want to associate the alias of the connection specification that you are activating.

When the user chooses to connect to a Client Access/400 data source, the USERID and password that were used to start the router are used to activate the database connection to the data source.

8. When you have completed the fields, select **OK**. The connection specification is saved and you are returned to the Database connection specifications panel. Your new connection specification appears in both lists in the Database connection specifications panel and the prerequisites for the application are updated.
4.2.2.3 Adding a Connection Specification to an Access Set

When you create a new connection specification as part of your application development process, VisualAge adds the connection specification to your access set for you. But you can also use connection specifications that were created outside of your application and add them to your access set.

After you have created an access set and a connection specification, putting them together is easy. Simply select the connection specification that you want to use in the top portion of the Database connection specifications panel and then select the Add to YourApp push button, where YourApp is the name of your application. If you have only one access set in the application, the connection specification is added to this access set. If you have more than one access set, selecting the Add to YourApp push button opens the Update database connection specifications panel in Figure 29.

![Update database connection specification](image)

Figure 29. Update Database Connection Specifications

Choose from the Access set drop-down list, the access set where you want to add the connection specification, and from the Data source name drop-down list the data source that you want to associate with the alias.

4.2.2.4 Establishing a Database Connection

To connect to a database manager, follow these steps:

1. Select a connection specification from the Database connection specifications panel.

2. Click mouse button 2 to display the connection menu and select Connect.

If you have checked the Prompt for logon information check box, the Logon prompt is displayed. Specify the name of the data source that you want to connect to and the USERID and password to connect to that data source.

**Note**

When you are connecting to a data source that uses the ODBC driver of Client Access/400, be sure to enter USERID and password in capitals.
If you did not check the **Prompt for logon information** check box, you have to choose from the SQL Data Sources panel to which data source you want to connect.

3. When the connection is established, the **No** in the **Active?** column for that connection specification has turned in to **Yes**.
Chapter 5. Using the VisualAge Database Functions

In this chapter, you learn how to add database logic using the VisualAge database parts to access ODBC data sources. You can use the following parts from the Database Functions category on the parts palette to add ODBC support to an application:

- **Multi-row Query**: Provides access to SQL for retrieving large blocks of data.
- **Single-Row Query**: Provides access to SQL for retrieving a single row of data.
- **SQL Statement**: Provides access to SQL for updating, inserting, and deleting data.
- **Stored Procedure**: Provides access to a program that is accessible through the database.

This chapter focuses on the use of the Query and SQL statement parts. You learn how to use these parts by building sample visual parts that focus on the use of one of the database parts. In the next chapter, you learn how you can use these database parts in real-life application design. In that chapter, the database parts are used to provide an order-entry application with the data it needs.

The sample visual parts that are built throughout this chapter are parts of a new VisualAge application, TpcWsSampleApp. This application has an access set called TpcWsOdbcAccessSet that contains one connection specification which connects alias ClientServerPerformanceData to the ODBC data source Client Server Performance Data. In every section of this chapter, you add one or more visual parts to this application that focuses on the functionality of the different database parts.

### 5.1 The Multi-Row Query Part

We cover the functionality of the multi-row query, look at all the settings, show you how to use the results of the query, and finally show you how to insert, update, and delete rows in a table.

The query that we use selects all of the rows and columns in the DSTRCT table and looks as follows:

```
SELECT * FROM DSTRCT
```

Before you start creating a query, create a new visual part for the sample application. Give it a meaningful name, for example, TpcWsMultiRowQueryView. You are now ready to start the definition of a multi-row query for this visual part.

### 5.1.1 Defining a Multi-Row Query

Add a Multi-row Query part from the Database Functions category to the free-form surface of the Composition Editor. Open the settings of the part by double-clicking it or by selecting Open Settings from the part’s pop-up menu. This opens the Multi-row Query Settings panel in Figure 30 on page 46.
Figure 30. Multi-row Query Settings - Notebook

This panel contains a notebook with three main sections:

- Query specifications
- Fetch options
- Update options

We discuss these sections now more in detail.

5.1.1.1 Query Specifications - Page 1

The first page of this section of the settings notebook enables you to specify the query that you want to associate with the multi-row query part.

1. From the Connection alias drop-down list, select a connection alias, for example, ClientServerPerformanceData.

   **Note**
   You can check the status of the connection specification by selecting the Specs... push button next to the Connection alias drop-down list. This opens the Database connection specifications panel as shown in Figure 25 on page 39. If the value in the Active? column is Yes, then you are connected to the database manager. If the value is No, then you can connect to the database manager by selecting the Connect button.

2. From the Access set drop-down list, select an access set, for example, TpcWsOdBCAccessSet.
3. Select a query name from the **Query name** drop-down list. This list is empty if you have not created queries for that access set yet. To create a new query, edit, or delete an existing one, use the **Create**, **Edit**, or **Delete** buttons at the bottom of this notebook page.

In case you select the **Create** or **Edit** push button, the *interactive query editor* in Figure 31 is displayed.

The interactive query editor helps you create a SELECT statement as follows:

a. Type a name for the query in the **Query name** entry field:

   A query requires a name that is unique within a particular access set. It cannot contain double-byte characters or spaces. For the sample application, type, for example, *DistrictQuery*.

b. Type a description for the query in the **description** entry field:

   The description is optional. Type, for example, *Select all rows and columns from the DSTRCT table*.

   This concludes the identification steps. The following steps actually define the query.

c. Select *DSTRCT* from the **Tables/views** list:

   This list contains all the tables and views that exist in the data source that is linked to the alias that you selected in the **Connection alias** drop-down list in Figure 30 on page 46.
The names of the columns of the DSTRCT table are displayed in the Columns list.

d. To retrieve all of the columns in the DSTRCT table, select the Select * check box. An asterisk is added to the Column sequence list.

Note

Any time you want to see the text of the SQL statement, select the Show SQL... push button. This opens a panel that displays the text of your SQL statement as it is being defined. You can find an example of this panel in Figure 32.

![SQL Statement]

SELECT *
FROM DSTRCT

Figure 32. Interactive Query Editor - SQL Statement

e. To save your query, select the Apply push button.

Your query is saved in the access set and you are returned to the Multi-row Query Settings panel. The name and description of your query is displayed in the Query name drop-down list and Description field, respectively.

5.1.1.2 Query Specifications - Page 2

The second page of the Query Specifications section of the multi-row query settings notebook (Figure 33 on page 49) contains three main groups of controls. These controls allow you to manipulate the query statement itself.
High-level qualifiers  When you use a data source that uses the Client
Access/400 ODBC driver, selecting this check box results
in tables and views that are qualified with the name of the
library that they are in.

For example, the DSTRCT table belongs to the library
CSDB. When you use high-level qualifiers, this table
appears in the Tables/views list in Figure 31 on page 47
as CSDB.DSTRCT, and the query statement looks similar
to the following:

SELECT * FROM CSDB.DSTRCT

A drawback of this approach is that your application is
linked to that specific library. When you do not use
high-level qualifiers, you only have to make sure that the
table is available in one of the libraries that you specified
in your data source.

SQL statement  This group of controls enables you to manually create or
modify the query associated with this part, or import or
export the queries from or to a file.

Note

When you manually create or edit the query, or import
it from a file, you are not able to edit the query with the
interactive query editor again.
Host variables

The **Host variables**... push button opens a panel that displays all host variables used in the query and enables you to modify the type of these variables. We cover the use of host variables later on in this chapter.

### 5.1.1.3 Fetch Options

One of the benefits of the multi-row query part is that it enables you to set fetch and update policies for queries. Its fetching capabilities enables you to limit the number of rows that are returned in the result table and to set the size of blocks that are retrieved. If you compose a query that returns thousands of rows, for example, and you know that you are interested in only the first 200, then you can limit the number of rows returned to 200. Packeting can improve your application’s performance by fetching only a portion of the rows returned as your application needs them. The default block size is 25. This means that the query retrieves the result table rows in blocks of 25 at a time.

You can manipulate the fetch policy of a query with the settings that are available on the first page of the **Fetch Options** section of the multi-row query settings notebook that is displayed in Figure 34.

![Figure 34. Multi-row Query Settings - Fetch Policy](image)

- **Maximum number of rows to fetch**
  - The number of rows to be returned by the query.

- **Enable packeting**
  - Enables the container details part (the quick form for multi-row query result tables) to request the result table rows as the application needs them. This option can speed the display of result table data by loading into memory only those rows that can be displayed in the container details part (plus a number of buffered rows to make scrolling through the result table more smooth).

- **Enable blocked fetches**
  - Enables you to set a blocked fetch size.
Blocked fetch size  The number of rows to fetch in a block.

The second page of the Fetch Options section of the multi-row query settings notebook only applies to IBM DB2/2 V2.1. When using another database manager, this page is disabled.

For the sample application, use the default settings for the fetch policy.

5.1.1.4 Update Options
The Update Options section of the multi-row query settings notebook enables you to set the update policy of a query.

The settings on the first page of this section that is displayed in Figure 35 determine the following:

- When updates to a result table are applied to the data source.
- When updates to the data source are committed.
- When VisualAge obtains a lock on a row.

![Figure 35. Multi-row Query Settings - Update Policy (page 1)](image)

Read only  Prohibits updates to the query result table. When this option is selected, the remaining options on this page are disabled.

Defer updates  Determines when changes made to a result table are applied to the data source. When you select this setting, VisualAge holds all changes to the result table until the user performs a specific action (such as selecting a push button) to apply or cancel them. After the user applies the changes, VisualAge obtains a lock on the rows that have been changed. When this option is selected, the Lock row on edit option is disabled.

When this setting is not selected, all changes are applied to the data source immediately. They are not committed, however, unless you enable the Auto commit option.
**Lock row on edit**  Obtains a lock on the row being edited so that no other users can edit it. VisualAge obtains locks by opening a cursor on the row. When a different row is edited, the previous row is updated if it has changed. Changing to a different row causes an update lock and the cursor is closed.

**Auto commit**  Commits all changes to the result table without requiring the user to perform a specific action. The effect of this option varies according to the Defer updates setting:

- When Auto commit and Defer updates are both selected, changes are committed when the apply action is triggered.
- When Auto commit is selected but Defer updates is not, changes are committed when the user selects another cell or row from the result table.

The settings on the second page of this section that is displayed in Figure 36 determine the following:

- Which columns can be updated.
- Which columns to use as index.

![Multi-row Query Settings](image)

**Columns to update**  Displays the columns to be returned in the result table and enables you to select the columns that the user can update.
Columns to use as index
Displays all columns to be returned in the result table and enables you to select the columns to use as index. These are the columns used to lock the row. The columns you select must uniquely identify a row.

Available
Displays a list of all indexes available in the data source and enables you to select one to use as an update index.

For the sample application, use the following update policy:

- Defer updates and no auto commit.
- Columns to update: DNAME, DADDR1, DADDR2, DCITY, DSTATE, DZIP.
- Columns to use as index: DID, DWID.

To save the settings of the multi-row query part and close the Multi-row Query Settings panel, select OK.

5.1.2 Using the Results of a Multi-Row Query
This section covers the use of the results of a database query. You learn how to tear off results, use quick forms, and work with a Container Details part. You continue working with the visual part you created in the previous section, and when you finish, you have a visual part that displays in a tabular form all of the columns and rows of the DSTRCT table in your data source.

5.1.2.1 Tearing-off Results
A VisualAge multi-row query part holds several attributes, including the name of the data source, the name of the query, and the results of that query. To expose the results of a query as a stand-alone part, you tear-off the result table attribute as follows:

1. From the multi-row query part’s pop-up menu, select Tear-Off Attribute...:
   This loads the cursor with a new part that represents the query’s result table.

2. Drop the new part on the free-form surface:
   The new part is a resultTable part that holds the result rows of the query after the query has been executed.

3. Using the resultTable part, perform a similar tear-off action to expose the currentRow attribute of the result table:
   This part holds the current row of the result table part.

Your Composition Editor now looks the same as the Composition Editor in Figure 37 on page 54.
Before continuing with the following section, take a look at the connection pop-up menu of each of the torn-off parts. The current row part’s public interface has attributes that match each of the column names in the database table. VisualAge dynamically builds this part’s public interface based on the actual result columns involved in the query. If you tore-off the currentRow attribute of a different query’s result table, its public interface is different.

5.1.2.2 Using Quick Forms

Most attributes of VisualAge parts have quick forms associated with them. In many cases, the quickest way to construct a user interface is to begin with a part’s quick form. You can use quick forms of database query parts to display an entire table of information or just a single value. Adding a visual representation for the result table of your multi-row query part can be done as follows:

1. From the pop-up menu of the multi-row query part, select Quick Form... and then resultTable (AbtMultiRowResultTable).

   **Note**

   If you change the settings of a query part, you need to generate the quick form again for the changes to take effect.

2. Move the mouse pointer over the panel and click mouse button 1. VisualAge tears off the result table, and a default visual representation of form of the result table appears. The quick form for a result table is the Container Details part.

   When creating a quick form, VisualAge provides the connections necessary to keep the non-visual part synchronized with the visual part. For a result
table, VisualAge creates three connections between the result table part and the container details part in the window:

- The result table’s rows event to the container details part’s forcePacketRequest action.
- The container details part’s self attribute to the result table’s widget attribute.
- The result table’s selectedRowIndex attribute to the container details part’s selectionIndices attribute.

Another example of a quick form is the following:

1. Select Quick Form... from the current row’s pop-up menu and select DNAME (String).
2. Move the cursor loaded with this attribute over the panel and click mouse button 1.

A text part, the quick form for a string, and a label part appear on the panel. Notice the connection that synchronizes the current row’s DNAME attribute with the text part’s object attribute.

**Note**

To get the quick form of the entire current row, select its self attribute.

3. Change the label to a more meaningful name, for example, District Name.

After performing these two quick forms, the Composition Editor that you opened on the visual part looks the same as the Composition Editor in Figure 38.
5.1.2.3 Running a Query
You have now defined a multi-row query and built a visual part to display the results of the query. The only missing component in this part is a connection between an event and an action that activates the execution of the query. There are several ways to do this. You can, for example, execute the query when you open the panel:

1. Connect the panel’s `aboutToOpenWidget` event to the multi-row query part’s `executeQuery` action. This executes the query and returns the result before the panel opens.

   **Note**
   If you enabled packeting in the settings of the multi-row query part, use the `openedWidget` event to trigger the query. The result table is filled just after opening the panel.

2. Test your visual part by selecting 🔄 in the upper left corner of the Composition Editor.

3. Play with the panel to see how the query that you defined is executed and how the results are displayed in the result table.
   - Try selecting a cell in a column that you selected for update and one that is read-only. Notice how selecting a read-only cell differs from selecting an updatable cell.
   - When you select a row, the district name of the selected row appears in the `District Name` field.
   - Experiment with scrolling through the rows with the vertical scroll bar and scrolling through the columns with the horizontal scroll bar.

4. Close the panel when you finish testing.

5.1.2.4 Working with the Container Details Part
The container details part not only provides a convenient way to display data, but also provides a working form that enables the user to edit and update data in the data source.

**Formatting the Container Details Part:** You can add some finishing touches to the visual part:
- You can change the column headings of a container details part by direct-editing. Hold down the Alt key and click mouse button 1 on the column heading that you want to edit.
- To size the column in the container details part, select the column and then hold down mouse button 1 or 2 to drag one of the selection handles to the left or right.
- To adjust the position or order of columns, select the column and then hold down mouse button 2 to drag the column to its new position.

   **Note**
   For more information on using container details parts, refer to chapter 14 of *VisualAge for Smalltalk User’s Guide Version 3, Release 0*, SC34-4518.
**Applying Changes to the Data Source:** You can also add push buttons to the visual part for saving or discarding changes that the user makes to the container details part.

1. Add two push buttons labeled **Apply** and **Cancel**.

2. Make the following connections for the Apply push button:
   a. Connect the **clicked** event to the **apply** action of the result table part.
   b. Connect the **clicked** event to the **commitTransaction** action of the multi-row query part.

3. Make the following connections for the Cancel push button:
   a. Connect the **clicked** event to the **cancel** action of the result table part.
   b. Connect the **clicked** event to the **rollbackTransaction** action of the multi-row query part.

Be sure to make the connections in this order.

4. Select ![Select Icon] to test the visual part.

When you change the rows in the container details part and then apply the changes, VisualAge performs all of the SQL statements that are needed to update the rows you changed in the data source. Similarly, if you use the **newRow** or **deleteRow** actions to insert or delete rows, VisualAge performs the necessary insert or delete SQL statements. When you use the cancel action instead, VisualAge discards the changes made.

---

**Note**

The **commitTransaction** action makes the changes permanent in the data source. If your application has more than one query, be certain that you perform all of the apply actions for the various queries first, and perform the **commitTransaction** action last, since it commits all of the work for all the transactions. Conversely, the **rollbackTransaction** action discards all of the work for all uncommitted transactions.

The **commitTransaction** and **rollbackTransaction** actions are executed against a database connection. If your application has more than one database connection, make sure you commit or rollback against the correct connection.

For the sample visual part, the update policy set for the multi-row query part (Defer updates, no Auto commit) requires that the user specifically applies and commits the changes that are made to the result table. If you edit one cell in the result table and then move on to another cell, you can still roll back your changes to the first cell until you select the **Apply** push button.

If the update policy for a query uses the Auto commit setting but not the Defer updates setting, then changes to the result table are committed immediately and the **Apply** and **Cancel** push buttons are not required.
5.1.2.5 Using Host Variables

A host variable is a variable whose value can be substituted into a query when the query is executed as a way of passing a parameter to the SQL statement.

Host variable names usually match a column name in your query, but are prefixed by a colon (:). For example, a host variable for the DID column might be :DID. The host variable can have any name you choose, but it cannot contain double-byte characters nor blanks.

To use a host variable in the sample application, edit the DistrictQuery query that you created in 5.1.1.1, “Query Specifications - Page 1” on page 46 and add the following WHERE clause:

WHERE (DSTRCT.DID = :districtID) AND (DSTRCT.DWID = :warehouseID)

Follow these steps to add the WHERE clause with the interactive query editor:

1. Open the Multi-row Query settings panel and select the Edit push button. This opens the interactive query editor on the query.
2. From the Clause menu, select WHERE....
3. From the Left operand list, select DSTRCT.DID.
4. From the Operator list, select =.
5. In the Right operand field, type districtID.
6. From the Unary operator menu, select Right operand, and from the cascaded menu, select Host variable :x.
7. Add this expression to the operand lists by selecting the Add to operand lists push button.
8. From the Left operand list, select DSTRCT.DWID.
9. From the Operator list, select =.
10. In the Right operand field, type warehouseID.
11. From the Unary operator menu, select Right operand, and from the cascaded menu, select Host variable :x.
12. Add this expression to the operand lists by selecting the Add to operand lists push button.
13. From the Left operand list, select (DSTRCT.DID = :districtID).
14. From the Operator list, select AND.
15. From the Right operand list, select (DSTRCT.DWID = :warehouseID).
16. Select Apply on the WHERE Details and the SELECT Details panels to save the query.

You have now defined a query with two host variables. VisualAge provides a host variable editor to change the type of these host variables.

17. On page 2 of the Query Specifications section of the Multi-row Query Settings Notebook, select the Host variables... push button.

This opens the Host variables for ‘DistrictQuery’ panel in Figure 39 on page 59. This panel lists all host variables in the query.
18. To edit a host variable’s data type, select the host variable from the list and select the **Edit** push button.

The *Edit host variable* panel in Figure 40 is displayed.

19. You can select a new data type from the drop-down list. The fields below the list change according to the data type selected.

20. Close these panels without changing the default data type for the host variables and then select **OK** on the *Multi-row Query Settings* Notebook.

21. Display the connect pop-up menu for the multi-row query part. It now includes `districtID` and `warehouseID`, the host variables used in the `WHERE` clause.

### 5.1.2.6 Running a Query with Host Variables

In 5.1.2.3, “Running a Query” on page 56, you ran the query before the window opened. Because the query uses host variables, you cannot run the query before the window opens because the variables would be nil. To run a query with host variables, do the following:

1. Remove the `districtID` label and text part, the `currentRow` part, and the `aboutToOpenWidget-to-executeQuery` connection.
2. The query expects a districtID and warehouseID as input. This means that you have to provide text fields in your visual part to type values for these variables. The easiest way to add labels and text fields for the host variables is using quick form.

From the multi-row query part’s pop-up menu, select Quick Form... and then districtID (Integer). Move the cursor over the panel and click mouse button 1. This adds a text part and a label part for the host variable to your panel.

Notice the connection from the object attribute of the text part to the districtID attribute of the multi-row query part.

Perform the same operations for host variable warehouseID.

3. Add a new Find push button and connect its clicked event to the executeQuery action of the multi-row query part.

The Composition Editor opened on the visual part now looks the same as the Composition Editor in Figure 41.

![Figure 41. Composition Editor - Query with Host Variables](image)

4. Test this visual part by selecting . Try typing districtID number 1 through 10, and warehouseID '0001'.

5.1.3 Insert, Update, and Delete Using a Multi-row Query Part

In the previous sections, we explained how to define a multi-row query part and how to use the results of the query. The query statements that you can execute with this part are only SELECT statements. It is not possible to define INSERT, UPDATE, or DELETE statements as the part’s query. However, the result table gives you the possibility to execute these statements indirectly without the use of database parts. In the following sections, you learn how to:
- Insert a row in the table.
- Update a row in the table.
- Delete a row from the table.

We use a new visual part for these sections. Name it, for example, 
TpcWsManipulateMultiRowQueryView, and add a multi-row query part to the 
free-form surface. Create a new query SelectAllDistricts that selects all rows 
and columns from the DSTRCT table. Uncheck all of the options on the first 
page of the Update Policy section of the Multi-row Query Settings notebook. 
Quick form the resultTable attribute from the multi-row query part. Perform the 
executeQuery action before opening the panel (aboutToOpenWidget event). The 
Composition Editor that you opened on this visual part looks the same as the 
Composition Editor in Figure 42.

![Figure 42. Composition Editor - Select All Rows and Columns](image)

**Note**

We assume in these sections that you are using a data source that has no 
commitment control configured. This means that all changes made to the 
resultTable are automatically committed to the data source. If you did 
configure commitment control, you have to execute a commitTransaction 
command before the changes take effect in your data source.
5.1.3.1 Inserting a Row in a Table

To add the functionality to add a row to your table, follow these steps:

1. Add a push button to the visual part and label it Insert Row.
2. Connect the clicked event of the Insert Row push button to the newRow action of the result table part.
3. Test this visual part by selecting . Add a new row to the table by selecting Insert Row. A new row is added before the selected row. Type information in this empty row. With every key stroke, the table on the AS/400 system is modified. No specific action is necessary to apply the changes to the AS/400 system.

Note

This is only true when you configured your data source to commit changes immediately, and unchecked all the options on the first page of the Update Policy section of the Multi-row Query Settings notebook.

When you checked, for example, the Defer updates in the update policy, you have to make the changes persistent with an apply action on the resultTable part.

If your data source uses commitment control, you have to perform an additional commitTransaction action on the resultTable part or on the multi-row query part. This action commits all database transactions that were performed since the last commit point.

5.1.3.2 Updating a Row in a Table

Updating a row in a table using a multi-row query part is even easier. To update a row, edit the fields that you want to change. The changes are automatically made in your data source.

Note: The same remarks as with the insertion of a row apply here.

5.1.3.3 Deleting a Row in a Table

Deleting a row from the table works in a similar way as inserting a row. Follow these steps:

1. Add a push button to the visual part and label it Delete Row.
2. Connect the clicked event of the Delete Row push button to the deleteRow action of the result table part. This action removes the current row from the table.
3. Test this visual part by selecting . Select one of the rows that you added in 5.1.3.1, “Inserting a Row in a Table” and choose Delete Row. This removes the selected row from the table. Check on the AS/400 system whether the record is deleted or not.

Note: The same remarks as with the insertion of a row apply here.
5.2 The Single-Row Query Part

In 5.1, “The Multi-Row Query Part” on page 45, we discussed the functionality of the multi-row query part. This part enables you to retrieve multiple rows. It gives you the possibility to impose your own fetch and update policy. It offers you techniques for inserting new rows, updating existing rows, or even deleting rows.

In some cases, you are not interested in all of this functionality, for example, when you want to retrieve one single row. For these cases, VisualAge provides another part: the single-row query part. This part can only execute queries that return exactly one row.

This section covers three main issues:

- Defining a single-row query.
- Using the results of a single-row query.
- Using the single-row query for insert, update, or delete.

5.2.1 Defining a Single-Row Query

Create a new visual part and name it, for example, TpcWsSingleRowQueryView. Add a single-row query part to the free-form surface of the Composition Editor. Open the settings of the single-row query part. This opens the Database Query - Settings panel in Figure 43.

![Database Query - Settings](image)

Figure 43. Database Query - Settings

1. From the Connection alias drop-down list, select a connection alias, for example, ClientServerPerformanceData.

2. From the Access set drop-down list, select an access set, for example, TpcWsOdbcAccessSet.
3. Select a query name from the **Query name** drop-down list. The queries that are shown in this list are all the queries that exist for that access set. The single-row query can only execute queries that return a single result. We have not created queries so far that return a single result so we need to create a new one. From the **Query** menu, select **Create...**

4. This opens the *interactive query editor* in Figure 44. This is the same editor as for a multi-row query. 

![Interactive Query Editor - SELECT Details](image)

**Figure 44. Interactive Query Editor - SELECT Details**

5. Create the following query:

   ```sql
   SELECT *
   FROM DSTRCT
   WHERE ((DSTRCT.DID = :districtID) AND (DSTRCT.DWID = :warehouseID))
   ```

   **Note**
   
   This query looks similar to the query that you built in 5.1.2.5, “Using Host Variables” on page 58. You can always go back to this section if you have a problem creating the query.

6. The **Database Query - Settings** panel offers, the same as the **Multi-row Query Settings** notebook, the possibility to:
   - Create and edit a query (with the interactive query editor).
   - Import and export a query.
   - Manually create and edit a query.
   - Work with host variables.
   - Delete a query.
   - Set high-level qualifiers.
Only the organization of the Database Query - Settings panel is different. All of this functionality is available from the menu bar. In the Multi-row Query Settings notebook, all of these functionalities are available through push buttons on one of the two pages of the Query Specifications section.

7. Close the Database Query - Settings panel by selecting OK.

5.2.2 Using the Results of a Single-Row Query

The query that we defined uses two host variables. This means that before you can execute the query, these variables need to be set to a value different than nil. You can either do this with a script, or you can provide entry fields in the user interface, let the user type in values for the host variables, and then execute the query.

We chose the last solution. Follow these steps to create the user interface:

1. Drag two group boxes on the panel of your visual part. You use one box to group the host variables, and the other to display the result of the query. Give the boxes meaningful names, for example, Host variables and Result.

2. Quick form labels and text fields for the attributes districtID and warehouseID of the single-row query part. Add these controls to the group box Host variables.

3. Tear-off the resultRow attribute of the single-row query part. Quick form labels and text fields for the DNAME, DADDR1, and DADDR2 and add these to the Result group box.

4. Connect the object attribute of the districtID and warehouseID text field to the DID and DWID attributes of the result row, respectively.

5. Add a push button labeled Execute to execute the query.

Your Composition Editor should now look the same as the Composition Editor in Figure 45 on page 66.
6. Test this visual part by selecting . Type values for the two host variables and execute the query by selecting \textit{Execute}.

5.2.3 Using the Single-Row Query for Insert, Update, or Delete

You can use the single-row query part to insert new rows in to your table, update, or delete existing rows. In this section, you make a few modifications to your visual part. After completing this section, the Composition Editor on this part looks similar to the one in Figure 46 on page 67.
5.2.3.1 Inserting a Row in a Table
To add the possibility to insert a new row in your table using the single-row query part, follow these steps:

1. Add a push button labeled New Row to your panel.
2. Connect the clicked event of New Row push button to the newRow action of the single-row query part.
3. Add a push button labeled Update Row to your panel.
4. Connect the clicked event of Update Row push button to the updateRow action of the single-row query part.
5. Test this visual part by selecting . Select New Row to create an empty row. Type information in the text fields. Make sure you type unique values in the districtID and warehouseID fields because these fields are the key for this table.

Select Update Row. This updates the row in the table or adds the row to the database if it is not already in the database.

5.2.3.2 Updating a Row in a Table
To update a row in a table, select the row by specifying its key values in the districtID and warehouseID fields. Execute the query by selecting Execute. Make some modifications to the row and select Update Row. The row is updated with the new values.
5.2.3.3 Deleting a Row in a Table

Make the following modifications to your part:

1. Add a push button labeled Delete Row to your panel.
2. Connect the clicked event of Delete Row push button to the deleteRow action of the single-row query part.
3. Test this visual part by selecting . Execute the query by specifying key values in the districtID and warehouseID fields and selecting Execute. To delete the row, select Delete Row. This removes the row from the table. Check on the AS/400 system that the row has been removed from the table.

5.3 The SQL Statement Part

As we have seen in the previous sections, 5.1, “The Multi-Row Query Part” on page 45 and 5.2, “The Single-Row Query Part” on page 63, you can use the multi-row query and single-row query parts for selecting, updating, inserting, and deleting database information. Sometimes though, you need to create database queries specifically for updating, inserting, or deleting information in a table. The SQL Statement part enables you to create UPDATE, INSERT, and DELETE SQL statements.

5.3.1 The UPDATE Statement

UPDATE statements are used to update existing data in a database.

In this section, we use the SQL statement part and the interactive query editor to create the following UPDATE statement:

```
UPDATE DSTRCT
    SET DTAX=(DTAX + (DTAX * 0.1))
```

1. Create a new visual part, TpcWsUpdateStatementView. Add an SQL statement part from the Database Functions category to the free-form surface.

2. Open the settings of the SQL statement part. This opens the SQL Statement - Settings panel in Figure 47 on page 69. This panel looks exactly the same as the Database Query - Settings panel in Figure 43 on page 63. The main difference between the panels is that the Database Query - Settings panel only gives you the possibility to create SELECT statements where the SQL Statement - Settings panel can help you create INSERT, UPDATE, and DELETE statements but no SELECT statements.
3. From the **Connection alias** drop-down list, select a connection alias, for example, *ClientServerPerformanceData*.

4. From the **Access set** drop-down list, select an access set, for example, *TpcWsOdbcAccessSet*.

5. Select a query name from the **Query name** drop-down list. The queries that are shown in this list are all of the queries that exist for that access set. We have only created SELECT statements so we have to create a new one. From the **Query** menu, select **Create**. Choose **UPDATE...** from the cascaded menu.

   This opens the *interactive query editor* for an UPDATE statement and this panel looks the same as the panel in Figure 48 on page 70.
a. Give the query a meaningful name and description, for example, `UpdateDistrictTable` and `Update the DTAX column with 10 percent`.

b. Select the `DISTRICT` table from the `Table/view` drop-down list.
   All of the columns in this table are automatically displayed in the `Columns` list.

c. Select the `DTAX` column from the `Columns` list.
   After selecting this column, it is displayed in the `Set columns` list and given a default value in the `Column value` text field. The default value is a host variable with the same name as the column and can be changed by typing in the text field or by using the `Column value` menu. The type of data that the column can hold is also displayed for you under the `Column value` text field.

d. From the `Column value` menu, select `Specify expression...`. This opens the `Expression Details` panel in Figure 49 on page 71.
e. From the **Left operand** list, select DTAX. From the **Operator** list, select *.
   And finally, type 0.1 in the **Right operand** text field.
   Add this expression to the operand lists by selecting the **Add to operand lists** push button.

f. Select this new expression from the **Left operand** list, + from the **Operator** list, and DTAX from the **Right operand** list.
   This is the expression that adds 10 percent to the current values in the DTAX column in the DSTRCT table.

g. Select **Apply** on the **Expression Details** panel and on the **UPDATE Details** panel to save the query into the access set that you specified on the **SQL Statement - Settings** panel.

6. Select **OK** on the **SQL Statement - Settings** panel to apply your changes and close the settings panel.

7. Add a push button labeled **Execute Query** to the visual part, and connect its **clicked** event to the **executeQuery** action of the SQL statement part.

   Run the visual part and execute the query by selecting the **Execute Query** push button. Check what happens on the AS/400 system.

5.3.2 The INSERT Statement

INSERT statements are used to add new data to a database.

In this section, we use the SQL statement part and the interactive query editor to create the following INSERT statement:

```sql
INSERT INTO DSTRCT (DADDR1, DADDR2, DCITY, DID, DNAME, DNXTOR, DSTATE, DTAX, DWID, DYTD, DZIP)
VALUES (:DADDR1, :DADDR2, :DCITY, :DID, :DNAME, 0, :DSTATE, 0, :DWID, 0, :DZIP)
```
1. Create a new visual part, TpcWsInsertStatementView. Add an SQL statement part from the Database Functions category to the free-form surface.

2. Open the settings of the SQL statement part. This opens the SQL Statement - Settings panel in Figure 47 on page 69.

3. From the Connection alias drop-down list, select a connection alias, for example, ClientServerPerformanceData.

4. From the Access set drop-down list, select an access set, for example, TpcWsOdbcAccessSet.

5. Select a query name from the Query name drop-down list. The queries that are shown in this list are all of the queries that exist for that access set. We have not created INSERT queries yet, so we create a new one. From the Query menu, select Create. Choose INSERT... from the cascaded menu.

This opens the interactive query editor for an INSERT statement and this panel looks the same as the panel in Figure 50.

![Figure 50. Interactive Query Editor - INSERT Details](image)

- a. Give the query a meaningful name and description, for example, InsertDistrictRow, and Insert a row in the DSTRCT table.
- b. Select the DSTRCT table from the Table/view drop-down list.

All of the columns in this table are automatically displayed in the Columns list.

- c. Select all of the columns from the Columns list. You can use the Select all option from the pop-up menu of the list for this purpose.

All of the selected columns are displayed in the Set columns list and given a default value in the Column value text field. The default value is a host variable with the same name as the column and can be changed.
by typing in the text field. The type of data the column can hold is also displayed for you under the **Column value** text field.

In our example, we use the default values (host variables) for all of the columns except DNXTOR, DTAX, and DYTD. Set the value for these values to be 0.

d. Select **Apply** on the **INSERT Details** panel to save the query into the access set that you specified on the **SQL Statement - Settings** panel.

6. Select **OK** on the **SQL Statement - Settings** panel to apply your changes and close the settings panel.

7. Quick form labels and text fields for the host variables.

8. Add a push button labeled **Insert Row** to the visual part, and connect its **clicked** event to the **executeQuery** action of the SQL statement part.

The Composition Editor opened on the visual part should now look the same as the Composition Editor in Figure 51.

![Composition Editor - Insert Row in Table](image)

Run the visual part. Type values for the different host variables and execute the query by selecting the **Insert Row** push button. Check what happens on the AS/400 system.

### 5.3.3 The DELETE Statement

DELETE statements are used to delete *existing* data in a database.

In this section, we use the SQL statement part and the interactive query editor to create the following DELETE statement:
DELETE FROM DSTRCT
WHERE ((DID = :districtID) AND (DWID = :warehouseID))

1. Create a new visual part, TpcWsDeleteStatementView. Add an SQL statement part from the Database Functions category to the free-form surface.

2. Open the settings of the SQL statement part. This opens the SQL Statement - Settings panel in Figure 47 on page 69.

3. From the Connection alias drop-down list, select a connection alias, for example, ClientServerPerformanceData.

4. From the Access set drop-down list, select an access set, for example, TpcWsOdbcAccessSet.

5. Select a query name from the Query name drop-down list. The queries that are shown in this list are all of the queries that exist for that access set. We have not created DELETE queries so far, so we create a new one. From the Query menu, select Create. Choose DELETE... from the cascaded menu.

   This opens the interactive query editor for a DELETE statement and this panel looks the same as the panel in Figure 52.

   ![Interactive Query Editor - DELETE Details](image)

   Figure 52. Interactive Query Editor - DELETE Details

   a. Give the query a meaningful name and description, for example, DeleteDistrictRow, and Delete a row in the DSTRCT table.

   b. Select the DSTRCT table from the Table/view drop-down list.

   c. To specify which rows you want to delete, you have to build a WHERE clause. Choose the WHERE clause... push button to open the WHERE Details panel of the interactive query editor displayed in Figure 53 on page 75.
d. Build the following WHERE clause:

\[
((\text{DID} = \text{:districtID}) \text{ AND } (\text{DWID} = \text{:warehouseID}))
\]

e. Select Apply on the WHERE Details panel to apply the changes made to the WHERE clause.

f. Select Apply on the DELETE Details panel to save the query into the access set that you specified on the SQL Statement - Settings panel.

6. Select OK on the SQL Statement - Settings panel to apply your changes and close the settings panel.

7. Quick form labels and text fields for the districtID and warehouseID host variables.

8. Add a push button labeled Delete Row to the visual part, and connect its clicked event to the executeQuery action of the SQL statement part.

The Composition Editor opened on the visual part should now look the same as the Composition Editor in Figure 54 on page 76.
Figure 54. Composition Editor - Delete Row in Table

Run the visual part. Type values for the different host variables and execute the query by selecting the Delete Row push button. Check what happens on the AS/400 system.
Chapter 6. Real-life OO Application Design

In the previous chapters, you learned how to install, configure, and use the ODBC support in your VisualAge for Smalltalk environment. The techniques that we describe in these chapters are very useful when you work on small projects. In real-life object-oriented application design, you might consider not using the visual parts. In some cases, for example, you want your application to be independent from the way you access the database system. In this case, you can create several classes that all use a different approach to access the DBMS, but that all share a common interface for the domain objects to access them. In this case, changing the access method only implies swapping these parts in your application. Instead of placing a multi-row query part on your composition editor, you can add one of these database access classes, and then changing from access method only has the consequence of replacing the database access parts in your application by the corresponding database access class.

In this chapter, we discuss the implementation of the application that is described in Chapter 2, “Introduction to the TPC-C Benchmark Application” on page 9. The implementation of the domain objects and the domain logic is independent of the database that you use and the way you access it. The database used is the TPC-C database residing in an AS/400 library. The way the application accesses this database is either by using ODBC or DDM.

6.1 Object Design

In 2.2, “Phase 1 Object Analysis” on page 9, we did a first phase analysis. The result of this analysis was the following list of objects:

- Customer
- District
- Order
- Order line
- Item
- Stock
- Catalog

In this section, we elaborate on a more detailed object design.

6.1.1 A Design Pattern

In 2.2, “Phase 1 Object Analysis” on page 9, we already mentioned that company EPY has a complete catalog of items to sell. Any item is listed in the catalog, and the catalog knows all of the items. If someone wants more information about a specific item, the catalog provides this information. In other words, through a catalog, you can manage the item related information.

The relationship between an item and the catalog is a relationship of the type Manager - Managed Object. This kind of relationship is very common in commercial application environments.

Many types of objects live in a proper environment and obey specific environmental rules. For instance, a student usually studies in a school. In this case, you can see the school as the manager and the student as the managed object.
The first important role of the manager is that it knows the managed objects. If you want to know which students are enrolled in a specific class, you have to ask the school for this information. The same is true when you want to know which foreign students have applied for the next year. We can conclude that the manager knows its managed objects and that the manager has the responsibility to return information about these objects when it is asked for.

Another important role of managers is that they know the rules that apply to the managed objects in order for them to belong to a certain environment and they enforce these rules. A school, for example, does not accept students younger than five years old. This rule is known by the school and the school knows that it has to check the students’ ages before admitting them.

Let’s have a closer look at the manager. Let’s suppose that we have to write an application that prints letters for the students enrolled in a specific class. In an object-oriented environment, the application asks the school first for all of the students enrolled for this specific class. The application then asks the students for their mailing address, and finally it prepares the letter based on this address.

As we already pointed out, the school (manager) knows the students (managed object) and returns them on request. There are two possible scenarios to handle such a request:

- The requested object already exists.
- The requested object has to be created.

Performance considerations usually lead us to the second choice, that is, creating the objects only when they are actually needed. If the number of managed objects is small, you might consider keeping these objects permanently alive. In this case, the manager can reach the objects immediately. In case of a large amount of managed objects, you may want to choose the second possibility and create the managed objects only when the manager needs them.

For example, when the school has 1000 students enrolled and another 1800 students have applied for the next year, you choose the second possibility. It is impossible to manage all of these instances permanently in our application. When another object asks for the foreign students that have applied for the next year, the school creates these instances at that moment.

In both cases (managed objects always existing, or created on request), the manager must be able to retrieve the information necessary to build the managed objects. In most cases, this information comes from a database. This means that the manager is responsible for accessing the database. And this is the third important role of the manager.

Summarizing these considerations, you can say that the manager has to play the following roles:

1. Creating, deleting, returning, and accepting managed objects (in other words, managing the managed objects).
2. Enforcing environmental rules.
3. Accessing the database.

Figure 55 on page 79 illustrates the different roles of the manager.
We try to apply to this design pattern whenever it makes sense in order to keep a consistent design. On the other hand, we also keep in mind that this is not a mandatory rule, and that we can choose to do things different if we think this is more appropriate.

**6.1.2 Object Model Design**

The previous section helps us to elaborate on a more complete list of objects than the list we came up with in Section 2.2, “Phase 1 Object Analysis” on page 9. These objects communicate with each other and the rules of this communication are fixed in the object model.

The objects that we add to our list of objects are two Managers. We add a Manager for the Customers and one for the Orders. We call these objects the *Customer Portfolio* and the *Order Portfolio*. This gives the following final list of domain specific objects:

- Customer Portfolio
- Customer
- District
- Order Portfolio
- Order
- Order line
- Item
- Stock
- Catalog

**Note**

Naming of classes and methods is very important in OO programming. The objects in your design represent, in many cases, real-life objects. To make the design easier to understand, it is a good practice to give your application objects the same name as they have in real life. That is why we chose Catalog instead of Item Manager.
Taking a closer look at this list, you see that Order is a sort of Manager because one Order manages various Order Lines. You can say that Stock is also a Manager, more precisely a Manager of Items, but these objects are already managed by the Catalog.

The static object model looks the same as the object model in Figure 56.

![Object Diagram](image)

**Figure 56. Object Model**

### 6.1.3 Composition of Objects

As was mentioned in the introduction of this chapter, we want our design to be independent of the way the DBMS is accessed. This means that we have to find a general and easy-to-configure mechanism to deal with different DBMSs.

The technique that we use to solve this problem is one of the most powerful OO techniques and is called **Composition**. This technique allows you to build complex objects using more simple components. The task that has to be performed by the complex object can be divided into smaller sub-tasks, and each component is delegated to perform the sub-tasks that fall in its competences.

In our design, we delegate the responsibility of the **Object Manager** to access the database to another object that we call **Persistency Manager**. For every DBMS, we create such a Persistency Manager. Composition allows us to easily change from one DBMS to another by replacing only the Persistency Manager. We only have to provide a protocol that all Persistency Managers have to obey to enable the composition.

In the TPC-C application, we create a Persistency Manager to access an AS/400 database using DDM and one to access the same database using ODBC.
In order to be able to use composition, your design has to apply the following rules:

1. The Object Manager talks to the Persistency Manager in a precise and well-defined language that we call the Persistency Manager protocol. This protocol can be defined by an abstract class, Abstract Persistency Manager.

   The Abstract Persistency Manager has the role to define the methods that each of the concrete Persistency Managers must implement. In the Abstract Persistency Manager, there is no code that can be performed, only method definitions.

   All of the concrete Persistency Managers are subclasses of the Abstract Persistency Manager. This means that they inherit all of the methods defined for this class (the protocol). Their role is to fill the methods with code necessary to access the corresponding DBMS and to return the required information. Figure 57 illustrates the relationship between the Abstract Persistency Manager and the concrete Persistency Managers.

![Persistency Manager Hierarchy](image)

*Figure 57. Persistency Manager Hierarchy*

Let’s illustrate these concepts with a small example. The Abstract Persistency Manager defines a method:

```java
getRecordFromId;
```

No code is written for this method.

The AS/400 DDM File Persistency Manager inherits the `getRecordFromId` method and defines the code necessary to reach DB2/400 using a DDM File part, opening the file, read the record with the specified ID as key, return the record, deal with exceptions, and so on.

The ODBC Persistency Manager performs similar actions in order to prepare the SQL statement, open the cursor, fetch the data, and so on.

2. As a consequence, the Object Manager has to use methods defined at the Abstract Persistency Manager level when it sends messages to one of the Persistency Managers. If we enforce this rule, we can be sure that we can switch from one Persistency Manager to the other without much effort. This way of programming is also known as programming at an abstract interface level.

3. The information exchanged between the Object Manager and the Persistency Managers are records and keys. The interface between these classes is on a record level. This is illustrated in Figure 58 on page 82.
The record management is hidden in the Object Manager. If an application object asks the Object Manager for a certain object, it receives the object and not the record. The record manipulation is hidden in the Object Manager. The interface between application objects and the Object Manager is on an object level. Figure 59 illustrates this concept.

6.2 Implementation of the Persistency Managers

In this section, we focus on the implementation of the different Persistency Managers. The database that these Persistency Managers access is the TPC-C database residing on an AS/400 system. We take a close look at two different Persistency Managers. One accesses the database through DDM (Distributed Data Management). This Persistency Manager uses the file parts of the VisualAge for Smalltalk AS/400 Connection feature. The other Persistency Manager uses ODBC to access the data.

In a first part, we focus on the protocol defined by the Abstract Persistency Manager that all of the different implementations have to follow. In the following parts, we focus on the details of the two different approaches.

6.2.1 The Abstract Persistency Manager

The Abstract Persistency Manager has the role to define the methods that each of the concrete Persistency Managers must implement. This is the interface that the Object Managers use to retrieve information from the database. As long as the Object Managers only use these methods, the Persistency Managers can be exchanged easily.
In our implementation of the application, there is one main VisualAge application called *TpcWsLabApp*. This application has four sub-applications:

**TpcWsBaseClassesExtensionApp**
Contains extensions made to existing classes to give them the behavior needed in the application.

**TpcWsDomainClassesApp**
Contains all of the classes that correspond with the domain objects. These classes implement the actual domain logic.

**TpcWsPersistencyMgmClassesApp**
This subapplication only contains the Abstract Persistency Manager. The different concrete Persistency Managers are implemented in separate applications that are prerequisites of our application. Depending on the database you use or the way you want to access the database, you can change the prerequisites. This results in smaller runtime images.

**TpcWsViewsApp**
This subapplication contains the different visual parts of the application.

For every implementation of the Persistency Manager, we create a new VisualAge application. These applications are all prerequisites for the *TpcWsLabApp*. When you compose your final application, you can choose one or more of the Persistency Managers. When you have made this choice, remove all of the Persistency Manager applications that you do not use from the prerequisites. This should result in smaller runtime images.

This section concentrates on the *TpcWsPersistencyMgmClassesApp* and more specifically on the one class that the subapplication contains, *TpcAbstractPM*. This class has nine public instance methods:

- **addRecord**: Adds the parameter (aRecord) to the database.
- **beginTransaction**: Starts commitment control.
- **commit**: Commits the transaction.
- **getRecordFrom**: Gets the record with the parameter (aKey) as key.
- **record**: Returns an empty record.
- **rollback**: Rolls back the transaction.
- **updateRecord**: Updates the parameter (aRecord) in the database.

This protocol works for all Object Managers except for the Order Portfolio Manager. For this class, the protocol has to be extended with additional methods. The reason for this is that all of the domain objects have more or less the same structure and correspond with only one file/table in the database. Order has a different structure. In fact, Order is a managed object and a manager at the same time. Order manages Order Lines. The object corresponds with two files/tables in the database; an order header file/table and an order detail file/table. The protocol that we describe in this section is not sufficient to make these objects persistent.

This has the consequence of having to define different protocols for different kinds of objects. In this application, there are only two kinds of objects that have
to be made persistent. The techniques to implement these protocols are the same. We concentrate on the preceding protocol.

Figure 60 shows an hierarchy browser opened on the TpcAbstractPM class. In the method list, you see all of the methods defined for this class. In the class list, you see that TpcAbstractPM subclasses AbtAppBldrPart and that it has two subclasses, TpcASFilePM and TpcODBCSqlTablePM. We discuss the implementation of these subclasses in more detail in the following sections.

6.2.2 The ODBC Persistency Manager

In this section, we discuss how you can implement the protocol that the Abstract Persistency Manager defines using ODBC to access the database. The techniques that are used are very similar to the techniques used in Chapter 5, “Using the VisualAge Database Functions” on page 45. But where we used visual parts in that chapter, we mainly use scripts in this chapter. This allows us to create more generic parts.

All of the classes that implement this Persistency Manager are contained in the VisualAge application TpcWsODBCApp. This application has two sub-applications:

**TpcODBCClassesExtensions**
This subapplication holds one extension to the class AbtRow.

**TpcODBCPM**
This subapplication contains all of the Persistency Manager classes and the access set class that holds all of the definitions of connection specifications and queries.
TpcODBCPM holds a class *TpcODBCSqlTablePM* that is the implementation of the Abstract Persistency Manager. The other classes in this application are an access set and several subclasses of TpcODBCSqlTablePM.

### 6.2.2.1 Database Connection

TpcODBCSqlTablePM has an instance variable `database` that holds an instance of *AbtOdbcDatabaseConnection*. This is the connection to the database. All of the methods of the class use this database connection to access the database.

The database instance variable has to be instantiated before any database access can be performed. The instance method `database` uses a *connection specification* defined in the access set *TpcODBCAccessSet* to connect to a data source. This makes you independent of the data source that you finally use. The only thing you have to impose is that the alias that you refer to is defined in the access set.

In the application, we use the same alias as in Chapter 5, “Using the VisualAge Database Functions” on page 45, `ClientServerPerformanceData`. You can add this alias to your application by selecting TpcODBCPM in the VisualAge Organizer and opening the *Database connection specifications* panel, selecting the alias, and then choosing the **Add to TpcODBCPM** push button.

The code to instantiate the database instance variable looks similar to this:

```smalltalk
database
  " returns the active database.
  If nil, initializes the connection "

database isNil ifTrue:
  [database := TpcODBCAccessSet ClientServerPerformanceData
   connectIfError: [:error | ^CwMessagePrompter
     errorMessage: 'Error connecting to a database',
     error errorText].
  ].
^database
```

### 6.2.2.2 Protocol Implementation

The implementation of the protocol is completely handled by this class except for the implementation of the methods that return the actual *query specifications* that must be executed. These queries are specific for the table that has to be accessed. For each table, you have to subclass *TpcODBCSqlTablePM* and implement these methods.

In this section, we concentrate on the implementation of the different protocol methods. We leave the implementation of the methods that return the query specifications and discuss those in the next section.

**addRecord:**

```smalltalk
addRecord: anAbtRecord
  " adds record to the database.
  Uses Smalltalk to issue SQL commands"

  self database executeQuerySpec: (self staticInsertStatement)
    withValues: anAbtRecord
    ifError: [:error | ^CwMessagePrompter
      errorMessage: 'Error adding a row',
      error errorText]
```

Chapter 6. Real-life OO Application Design  85
This method sends the database instance variable the message

```executeQuerySpec:withValues:ifError:`

The database connection executes the query that corresponds with the first parameter with the values in the second parameter. If an error occurs while executing the query, the block in the third parameter gets executed.

```commit`

```commit`

```self database commitUnitOfWork```

The database connection commits all transactions to the database since the last commit point.

```getRecordFrom:`

getRecordFrom: id

```"return the customer record for the given id."
```

```| result |
```

```"prepares the parameters structure"
```

```self setParametersWith: id.
```

```"evaluates the statement."
```

```The result is a cursor that can be fetched".
```

```result := self database resultTableFromQuerySpec: (self staticSelectStatement)
```

```withValues: parameters
```

```ifError: [:error | ^CwMessagePrompter
```

```errorMessage: 'Error retrieving a row ', error errorText].```

```"fetches the first row and returns it"
```

```^result firstIfError: [:error | ^CwMessagePrompter
```

```errorMessage: 'Error retrieving a row ',
```

```error errorText].```

This method first prepares the parameters to be in the format necessary for the second message `resultTableFromQuerySpec:withValues:ifError:`. The database connection executes the query that corresponds with the first parameter with the values in the second parameter inserted into the query, and returns a table that is the result of the query. If an error occurs while executing the query, the block in the third parameter gets executed. Finally, the method returns the first record in the table.

```record`

```record```

```^self database emptyRowForQuerySpec: (self staticSelectStatement)
```

```ifError: [:error | ^CwMessagePrompter
```

```errorMessage: 'Error getting an empty row ', error errorText].```

This method sends the `emptyRowForQuerySpec:ifError:` message to the database connection. The database connection returns an empty row for the query specification in the first parameter.
rollback

rollback

self database rollbackUnitOfWork

The database connection rolls back all transactions since the last point of commit.

updateRecord:

updateRecord: anAbtRecord
  " adds record to the database. 
  Uses Smalltalk to issue SQL commands"

self database executeQuerySpec: (self staticUpdateStatement)
  withValues: anAbtRecord
  ifError: [:error | ^CwMessagePrompter
  errorMessage: 'Error updating a row', error errorText].

^#OK

This method sends the database instance variable the message executeQuerySpec:withValues:ifError:. The database connection executes the query that corresponds with the first parameter with the values in the second parameter. If an error occurs while executing the query, the block in the third parameter gets executed.

6.2.2.3 Query Specifications

In the implementation of the different protocol methods, you see that we use methods such as staticSelectStatement. If you look at their implementation in TpcODBCSqlTablePM, you see that they refer to an implementation in a subclass of this class.

staticSelectStatement
  " returns the select statement."

self subclassResponsibility

For every Manager-Managed Object relationship, we create a subclass that corresponds with the table in the database that holds the information for the Managed Objects. The only methods these subclasses implement are the methods that return the query specifications for the queries that access this table.

For example, the object Item is managed by the Catalog and for this manager we create a subclass TpcODBCSqlTableCatalogPM of TpcODBCSqlTablePM. This subclass implements a method that returns the correct query specification from the TpcODBCAccessSet for this specific query:

staticSelectStatement
  " returns the select statement."

selectStatement isNil ifTrue:
  [selectStatement := TpcODBCAccessSet runtimeQuerySpecNamed:
    #selectItemFromId].

^selectStatement

This method returns the query specification for the query that retrieves an Item record from the ITEM table:
SELECT * FROM ITEM WHERE (ITEM.IID = :id)

**Note**

The easiest way to create these queries is by using one of the visual parts from the database functions category. Once these queries are added to the TpcODBCAccessSet, you can delete the visual part.

In our design, we created three subclasses of the TpcODBCSqlTablePM:

- TpcODBCSqlTableCatalogPM
- TpcODBCSqlTableCustomerPortfolioPM
- TpcODBCSqlTableStockPM

These subclasses implement the following methods:

- staticSelectStatement
- staticInsertStatement
- staticUpdateStatement

When you want your design to use the ODBC approach to access the database, you must use these subclasses when composing your final application.

### 6.2.3 The DDM Persistency Manager

This publication focuses on the use of ODBC support in a VisualAge for Smalltalk environment with the Client Access/400 ODBC drivers. This is an environment where you can choose to use the VisualAge for Smalltalk AS/400 Connection feature. This feature offers you the same possibilities to access the AS/400 database as you have with ODBC. That is why we feel that this publication is incomplete without a comparison between the two approaches. See 6.5, “ODBC or DDM “That’s the Question”” on page 96 for more information about this comparison. This section covers a DDM implementation of the Abstract Persistency Manager.

This approach gives us the possibility to create a generic implementation of the Abstract Persistency Manager. The implementation is independent of the file you are accessing. Only at the moment of composition of the final application, you must specify which file you want to access with this Persistency Manager.

You find the DDM implementation in the VisualAge application TpcWsASDDMFilesApp. This application has two subapplications:

- **TpcASDDMFilesPM** This subapplication contains all of the Persistency Manager classes.
- **TpcASRecords** This subapplication holds the Record Description classes for the different files used.

The implementation of the Abstract Persistency Manager for the DDM approach is the class TpcASFilePM.
6.2.3.1 Record Based Access

The DDM implementation of the Persistency Manager is independent of the file used. But when you compose the final application, you have to specify which file and record descriptions the Persistency Manager part has to use. In VisualAge, record descriptions are classes that describe the file. You can compare these classes with classical DDS (Data Description Specification) on the AS/400 system. You do not have to create these classes yourself. VisualAge generates them for you.

Follow these steps to create, for example, the record description for the ITEM file.

1. Open the settings notebook of a Keyed file part from the AS/400 Data Access category. This opens the panel in Figure 61. Specify the file name ITEM, and library name CSDB in the respective entry fields. Choose Apply to apply these changes.

2. Choose Retrieve record description.... This opens the Record description - Settings panel in Figure 62 on page 90.
3. Change the name of the record in the **Record format name** field. This name is the name of the class for this record description.

4. Choose **OK** to close the panel and to save the record description. This opens the panel in Figure 63. Choose from the list the application where you want the record description class to be added.

**Note**

VisualAge does not allow you to add record description classes to a subapplication. If you want to add the record descriptions in a subapplication, you have to move them to this subapplication after you have created them in an application.
6.2.3.2 Protocol Implementation

In this section, we focus on the implementation of the protocol methods.

**addRecord:**

addRecord: aRecord

" adds aRecord to the database.
   Returns symbol OK if no errors are encountered "

[sel openFileReadWrite.
  fileRW write: aRecord.
  onExceptionExecute: [:signal | ^self handleExceptions: signal].
^#OK

This method opens the file for ReadWrite and writes the record that was passed in as a parameter to the file. When an exception occurs in this block, the exception handling block is executed. The methods openFileReadWrite and handleExceptions are internal methods.

**commit**

commit

" commits the transaction "

[fileRW commit.
  onExceptionExecute: [:signal | ^self handleExceptions: signal].

This method commits a unit of work. All transactions since the last commit point are applied to the database. When an exception occurs in this block, the exception handling block is executed.

**getRecordFromId:**

getRecordFrom: id

" returns the record read with id as key,
   or nil if there is no such record "
    | record |

[sel openFileReadOnly.
  record := fileRO readAt: id ifNotFound: [nil]]
  onExceptionExecute: [:signal | ^self handleExceptions: signal].
^record

This method opens the file for ReadOnly and reads the record at the ID specified in the parameter. When an exception occurs in this block, the exception handling block is executed.

**record**

record

" returns an empty record "

^((self subpartNamed: 'Keyed file Read Only') emptyRecord

This method performs the emptyRecord method on the keyed file part on the Composition Editor. This method returns an empty record.

**rollback**
rollback
  " rollbacks the transaction "

[fileRW rollback.]
  onExceptionExecute: [:signal | ^self handleExceptions: signal].

This method rolls back a unit of work. When an exception occurs in this block, the exception handling block is executed.

updateRecord:
updateRecord: aRecord
  " updates the aRecord
    Returns symbol OK if no errors are encountered "

[file openFileReadWrite.]
  fileRW update: aRecord.]
  onExceptionExecute: [:signal | ^self handleExceptions: signal].
  ^#OK

This method opens the file for ReadWrite and updates the record that was passed in as a parameter to the file. When an exception occurs in this block, the exception handling block is executed.

6.3 Implementation of the Domain Objects

In this section, we focus on the domain objects, and more exactly, on the way the objects retrieve the information from the database.

When we talk about the domain objects, we have to make a distinction between Manager objects and Managed Objects. The function of the Manager object is described in 6.1.1, “A Design Pattern” on page 77. In short, we can summarize this role as follows:

1. Creating, deleting, returning, accepting managed objects (in other words, managing the managed objects).
2. Enforcing environmental rules.
3. Accessing the database.

The role of the Managed Objects is very limited. It only stores information.

Much the same as we did in previous sections, we use the Catalog-Item relationship as an illustration for the Manager-Managed Object relationship.

6.3.1 Manager Objects

One of the roles of these objects is to access the database and retrieve information from it. In order to accomplish this task, the Manager uses a Persistency Manager. Every manager has an instance variable `persistencyManager` that points to the correct Persistency Manager.

Every access to the database is handled by the Persistency Manager. This Persistency Manager implements a certain `protocol` that we defined in 6.2.1, “The Abstract Persistency Manager” on page 82. This protocol consists of the following methods:

- addRecord:
When a Manager object needs to access the database, it has to use this protocol.

Catalog, for example, implements a method to build a new Item based on an item ID. This method looks similar to this:

```
itemFromId: id
    "returns the item corresponding to id"
    | aRecord anItem|
    (id isNil or: [id trimBlanks = ''])
        ifTrue: ['nil']
        ifFalse: [aRecord := persistencyManager getRecordFrom: id].
    (aRecord isNil)
        ifTrue: [self signalEvent: #itemNotFound with: id.
            nil].
    anItem := TpcItem newFromRecord: aRecord.
    self signalEvent: #itemFound with: anItem.
    ^anItem
```

This method uses the protocol method `getRecordFrom:` to read a record from the database. The Catalog does not have to know which database it is targeting. This is handled by the implementation of Persistency Manager.

The interface between the Manager and the Persistency Manager is on a record level, but the interface between the Manager and the other domain objects is on an object level. The `itemFromId:` method converts the item record that it received from the Persistency Manager in an Item object by calling a class method of `TpcItem` that creates a new Item and fills the object with the information in the record. We discuss this functionality in the next section.

### 6.3.2 Managed Objects

As was mentioned in the previous section, these classes implement a class method that creates a new instance of the class based on a record. The class is also responsible for putting the information contained in the object into a record structure whenever the Manager object has to write the object to the database.

All instances of these objects have an instance variable `record` that holds the record that was read from the database. When you use the ODBC Persistency Manager, this record is an instance of `AbtRecord`, and when you use the DDM implementation of the Persistency Manager, this is an instance of one of the subclasses of `RECORD` that you created.

For the `TpcItem` class, the methods to convert records to objects and objects into records are implemented as follows:

```
newFromRecord:
```
newFromRecord: record
    " creates a TpcItem instance filling it with the data of the record"
    | anItem |

    anItem := super new.
    anItem id: (record at: 'IID');
    name: (record at: 'INAME');
    price: (record at: 'IPRICE').
    anItem record: record.

    ^anItem

putDataIntoRecordFor:
    putDataIntoRecordFor: item
    " synchronizes the record instance variable
    with the data stored in the item"
    item record at: 'IID' put: item id;
    at: 'INAME' put: item name;
    at: 'IPRICE' put: item price.

This leaves us with one problem to solve. The record instance variable can be
either an instance of AbtRecord or an instance of one of the subclasses of
RECORD. AbtRecord understands the methods at: and at:put:, but RECORD does
not. This problem can be solved by extending RECORD with this functionality. A
possible implementation for these methods looks the same as the following:

    at:
    at: fieldName

    ^self at: fieldName asLowercase

    at:put:
    at: fieldName put: value

    self atField: (fieldName asLowercase) put: value

Note

If you are not familiar with extending existing classes, you can find more

6.4 Changing Persistency Managers in Your Application

In 6.1.3, “Composition of Objects” on page 80, we discussed the principle of
Composition of Objects. In this section, we show how you can actually change
the database access used by the application using this technique of composition.

In our sample application, every manager has an instance variable
persistencyManager. This variable is only assigned in the visual part
TpcNewOrderWindow in the subapplication TpcWsViewsApp in the TpcWsLabApp.
When you open a Composition Editor on this part, you see the Composition
Editor in Figure 64 on page 95.
There are four parts that have a Manager role:

- Customer portfolio
- Order portfolio
- Stock
- Catalog

These are the objects that have the responsibility to retrieve information from the database. The actual access to the database is performed by a Persistency Manager.

You can see that every Manager has a connection from its attribute `persistencyManager` to the `self` attribute of the corresponding ODBC Persistency Manager:

- Customer Portfolio ODBC PM
- Order Portfolio ODBC PM
- Stock ODBC PM
- Catalog ODBC PM

These are instances of the respective subclass of `TpcODBCSqlTablePM`. These parts execute the actual retrieval of information from the database using ODBC.

On the same Composition Editor, you see a DDM Persistency Manager for every Manager object. These objects are instances of `TpcASFilePM` pointing to the correct file and using the correct record descriptions.

- Customer Portfolio DDM PM
- Order Portfolio DDM PM
- Stock DDM PM
- Catalog DDM PM
These Persistency Managers are not connected to any other object. When we want to change the way the application accesses the database form ODBC to DDM, we only have to change the connection from the `persistencyManager` attribute of the Manager object to the `self` attribute of the corresponding DDM Persistency Manager. The result of this switch looks the same as the visual part in the Composition Editor in Figure 65.

![Composition Editor - Using DDM Persistency Managers](image)

Figure 65. Composition Editor - Using DDM Persistency Managers

This is just an illustration of how a good design can make your application easier to port from one DBMS to another.

### 6.5 ODBC or DDM “That’s the Question”

In this section, we look at the question you can ask yourself, should I go for ODBC or DDM, are there any other specific considerations?

ODBC is a good choice if you have SQL skills and your application needs to access multiple vendor databases.

DDM is a good choice if you want to avoid the more complex setup of the ODBC datasources, and you know your application will only be used to access an AS/400 system.

With VisualAge Version 3.0, both ODBC and DDM have every thing to build robust client/server applications (commitment control). With a good application implementation, such as in our example, it is very easy to test both of the access methods against the AS/400 database. This is very helpful when the choice between ODBC or DDM is not so obvious and easy to make.
6.5.1 Runtime Behavior of the Different Persistency Managers

There are two main issues when it comes to runtime responses:

• Size of the runtime image
• Response times of the application

We discuss our observations in the following paragraphs.

6.5.2 Size of Runtime Image

With the approach that we followed, we have to prepare the application before we can create a runtime:

1. Choose which technique you want to use to access the database, ODBC or DDM.

2. In the Composition Editor of the different visual parts, remove the Persistency Managers of all access techniques except those of the technique that you chose in Step 1.

3. Change the prerequisites of the main application. Remove all of the applications that contain implementations of Persistency Managers that you chose not to use. If you don’t do this, these applications and all of its prerequisites are packaged as well.

If you package your application now, only those applications are packaged that relate to the access technique that you chose. The image has a minimal size.

There are other preparations you might consider before packaging:

• Starting from a clean image (that is an image where you only loaded the different VisualAge features without applications created by yourself), load the applications that you want to package.

• Clean the image by executing the following code in your transcript or in a workspace:

  System abtScrubImage

After going through this procedure for both implementations of the Persistency Managers, packaging the applications gave us the runtime image sizes in Table 9.

<table>
<thead>
<tr>
<th></th>
<th>OS/2 Warp</th>
<th>Windows 3.1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ODBC</td>
<td>DDM</td>
</tr>
<tr>
<td>Size in Mbytes</td>
<td>2.22</td>
<td>2.53</td>
</tr>
</tbody>
</table>

6.5.3 Response Times

After creating the runtime images, we measured the response time of the application. We measured how long it takes to register one order with 10 order lines. This is one transaction.

Registering one order of x order lines performs the following operations to the database:

• 2X read operations
X+1 write operations
X update operations

The response times that we measured are displayed in Table 10. The table shows the response times for each transaction and the plus average response time. The response times were measured using OS/2 Warp and Windows 3.1 and this both for the DDM and ODBC implementation.

<table>
<thead>
<tr>
<th>Table 10. Response Times for One Order with 10 Order Lines (in Milliseconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td>Transaction #1</td>
</tr>
<tr>
<td>3,469</td>
</tr>
<tr>
<td>Transaction #2</td>
</tr>
<tr>
<td>Transaction #3</td>
</tr>
<tr>
<td>Transaction #4</td>
</tr>
<tr>
<td>Transaction #5</td>
</tr>
<tr>
<td>Transaction #6</td>
</tr>
<tr>
<td>Transaction #7</td>
</tr>
<tr>
<td>Transaction #8</td>
</tr>
<tr>
<td>Transaction #9</td>
</tr>
<tr>
<td>Transaction #10</td>
</tr>
<tr>
<td>Average</td>
</tr>
</tbody>
</table>

These response times were measured on an AS/400 system, Model D60, with OS/400 V3R1 installed.

Both implementations use commitment control. The DDM approach commits the transaction after saving the last row. The ODBC data source is configured to use uncommitted read. Updates and inserts are committed explicitly after completing the transaction.

Note
These figures cannot be taken as a benchmark between DDM and ODBC. The implementation of the different Persistency Managers is not optimized. They only give an idea about the response times of this specific implementation of the order entry application.

6.5.4 Conclusion
The differences in our scenario between ODBC and DDM are not very large. The runtime images are almost the same in size, and the response times do not vary a lot. With both implementations, you are able to create robust client/server applications.

If SQL skills or the knowledge that you will only access the AS/400 database does not force you into a certain implementation, and with an application
implementation such as our sample application, it could be worthwhile to run a little benchmark to try them both.
Chapter 7. Optimizing ODBC Access

In this chapter, we provide some tips and hints that we found during the development of the TPC-C application. These tips and hints can be divided in three major groups:

- Optimizing the queries and the way information is fetched from the database to your application.
- Fine-tuning the configuration of the ODBC data sources.
- Query behavior.

Most of these recommendations are not specific to this very specialized topic (VisualAge for Smalltalk and Client Access/400 ODBC support) but fit in a larger context. Fine-tuning the configuration of the ODBC data sources, for example, has to be done for every application, independent of the tool used to create this application. But we think that it is harmless to repeat them in this publication.

7.1 Optimizing the Queries

In this section, we focus on how to improve performance by manipulating the SQL query and the settings of the SQL parts. We use the visual database parts to illustrate the techniques, but evidently you can do the same when you want to perform the queries from a script.

The improvements all focus on the same central issue: try to limit the amount of data that you have to transfer from the AS/400 system to your workstation.

7.1.1 Subfiling

A first improvement can be made by using a technique that can be compared to subfiling. Subfiling is a very popular technique in native AS/400 programming. The traditional character-based applications can only show a limited amount of records from a file on one display. To see more information, you have to use the page up and down keys. You only see a sub-part of the file and this part is called a subfile.

The multi-row query part enables you to do a similar kind of subfiling. On page 1 of the Fetch Options section of the settings notebook of this part, you have the choice to enable packeting. Enabling packeting enables the container details part to request the result table rows as the application needs them. This speeds the display of result table data by loading into memory only those rows that can be displayed in the container details part (plus a number of buffered rows to make scrolling through the result table more smooth). This means that the multi-row query part executes the query completely but it only fetches a limited amount of the rows of the result table.

In the scenario of the TPC-C application, entering a new order starts with specifying the customer identification. Most probably, customers do not know their own customer ID. To make it easy for the operator to handle the order, you can provide an application that searches customers in the database based on their name or part of their name. To make the search seem faster, you can use packeting. This speeds up the display of the rows that match the search criteria. In the best case, the customer’s name is part of the first set of records shown.
In the worst case, you must scroll down until you reach the end of the result table.

The panel that helps the operator searching for the customer ID looks similar to the panel in the Composition Editor shown in Figure 66.

The operator can specify the name of the customer, or part of the name of the customer, and then select search. The first result rows are displayed in the container details part. Operators can scroll through this list and when they find the customer, they can select the row in the table and this information is then used in the order entry application. No further database access is necessary.

![Figure 66. Using Packeting to Speed Up the Display of Result Rows](image)

The query used in the multi-row query part, is the following:

```sql
SELECT *
FROM CSTMR
WHERE (CSTMR.CFIRST LIKE :name)
```

**Note**

*Like searches* on the AS/400 system that compare the value in a column of type CHARACTER with a certain value require that the compare value has the full length of the column in the database. If this is not the case, you have to add % characters to this value to the full length of the column.

Like searches on the AS/400 system are case sensitive.
7.1.2 Skinny Objects

In the previous section, we improved the performance of the application by only fetching those rows that we can actually display. By scrolling through the list, we fetch a new set of rows each time until we find the row that we are looking for.

This approach works fine but is not optimal. The rows contain much more information than we need. We are looking for a certain customer. This customer can be identified by a name and address. Instead of reading the complete row (or object), it is enough to read those columns that enable the operator to identify the customer uniquely, plus those columns that form the key of the table. This reduced row is called a skinny object. This approach can limit the transferred information dramatically and increases the response times as dramatically.

In the example in the previous section (7.1.1, “Subfiling” on page 101), we transfer the full customer record from the data source to the workstation. If we look at the layout of the table (Table 3 on page 11), you see that a row contains one column CDATA of length 500 with general information about the customer. This information is irrelevant when it comes to identifying the customer from a list.

To change the application in order to use skinny objects, you only have to define another query and quick-form the result table again. You might consider using the following query:

```
SELECT CSTMR.CDID, CSTMR.CFIRST, CSTMR.CID, CSTMR.CWID
FROM CSTMR
WHERE (CSTMR.CFIRST LIKE :name)
```

The Composition Editor on the visual part would look the same as the Composition Editor in Figure 67 on page 104. The difference with the other implementation is that the container details part only has four columns.
### 7.2 Fine-Tuning the ODBC Configuration

This section focuses on the configuration of your ODBC data sources. You learn how you can improve the performance of ODBC by manipulating the different configuration parameters. In this section, we refer explicitly only to the Client Access/400 for Windows 3.1 client. **The Client Access/400 Optimized for OS/2 client does not give you the freedom to manipulate these parameters.**

You can see this section as an extension to section 4.1.2, "Configuration of the Client Access/400 for Windows 3.1 ODBC Driver" on page 27. In this section, we tried to set the focus on creating and configuring data sources where this section concentrates on improving the performance of the data source.

#### 7.2.1 DefaultLibraries

DefaultLibraries selects the default libraries you want to see.

For example, to see libraries QGPL and QPLS, enter QGPL, QPLS for this parameter.

The more default libraries you select, the longer it takes ODBC to read all of them.

**Note:** Many ODBC applications use the ODBC’s catalog functions to retrieve lists of available owners/libraries, tables/files, and columns/fields. The libraries you enter for the DefaultLibraries setting make up the catalog view (that is, the catalog functions work only against these libraries).
This text box specifies the AS/400 library or libraries that the Client Access/400 ODBC driver searches. You can enter multiple libraries separated by commas or spaces in this text box.

To add to the existing user library list, you need to add a special entry, *USRLIBL, to the list of libraries. All of the libraries listed before *USRLIBL are added to the front of the library list and all of the libraries listed after *USRLIBL are added to the end of the list.

The first library in the list is used as the default library. If you do not want a default library, start the list with a comma.

If tables are specified in SQL statements without a library name, the default library is used. If the application does not specify a library name with the DBQ parameter in the SQLBrowseConnect or SQLDriverConnect API, the ODBC driver uses the first library in this text box. Specifying every library that you own or may ever want to access is not the best idea as far as performance is concerned. Each library you add to the list increases the number of search paths and the amount of data the host creates and transfers to your PC every time an application requests catalog information. If you must access a large number of libraries, consider creating multiple data sources, each with a limited number of libraries.

### 7.2.2 RecordBlocking

ODBC can fetch (retrieve) rows of data from an AS/400 file one at a time or in blocks, which reduces communications and other overhead. The RecordBlocking setting lets you control how the ODBC driver fetches rows of data. There are three possible values:

- 0 indicates that the driver fetches only one row at a time.
- 1 indicates the driver fetches a block of rows if the SQL SELECT statement ends with a FOR FETCH ONLY clause; otherwise, the driver fetches one row.
- 2, the default, indicates that the driver fetches a block of rows unless the SQL SELECT statement contains a FOR UPDATE OF clause.

The default value usually gives the best performance. However, if an application assumes it has a lock on any row it has fetched and does not use the FOR UPDATE OF clause in the SQL SELECT statement, the application cannot update the row if you are using the default setting. One result of reading data in blocks is that the SQL SELECT statement is opened for read only. If an application assumes that any SELECT statement is used for updating, then you must change this setting to 0 or 1 for the application to work properly.

When updating data and using the default setting, you must use the For Update Of clause in your SQL Select statement and set on the SQL_CONCURRENCY option using the SQLSetStmtOption interface.

### 7.2.3 BlocksizeKB

This parameter lets you select how much data (in kilobytes) is retrieved in one block from the AS/400 system.

Values are from 1 to 512 kilobytes with the DEFAULT=32.

Shorter block sizes may make certain applications look faster.
Larger block sizes should transfer the most data in the shortest time.

**Note:** The BlocksizeKB setting lets you specify the maximum number of kilobytes of AS/400 data the ODBC driver can retrieve in one block. Valid values for this setting are from 1 to 512 with a default value of 32. If you need to retrieve large amounts of data from the AS/400 system, you can reduce overhead by using a larger block size.

However, in some cases, a smaller block size can make an ODBC application's performance appear faster to a user even though the data retrieval is actually slower than it is if ODBC were using a larger block size. Some applications (Microsoft Access, for example) start displaying or operating on the first row or rows of data as soon as they are received. If you specify a very large block size, you delay the arrival of the first row while the data block is assembled on the AS/400 system and sent to the PC. Another case in which you might want to use a smaller block size is an ODBC application that selects large amounts of data but generally uses only the first few rows.

### 7.2.4 LazyClose

LazyClose improves performance by eliminating communication flows between the PC and the AS/400 system.

LazyClose should almost always be used as DEFAULT.

When using LazyClose, review how your programs uses locks.

**Note:** Another way to improve performance is to reduce the number of “flows” of information between the PC and the AS/400 system. Each instance of the communications flow changing direction costs time and impedes performance. LazyClose improves performance in most situations by reducing the time spent in communications handshaking. With LazyClose, when the ODBC application closes a statement, the driver sends data to indicate that close to the AS/400 system, but the communications buffer is not flushed. The close does not actually happen until the next operation that forces the communications to be flushed.

The LazyClose setting has two possible values: 0 disables LazyClose, and 1, the default, enables it.

If LazyClose is enabled, a SQLFreeStmt with the SQLClose option is not sent to the AS/400 system until the next request is sent. If disabled, a SQLFreeStmt with the SQLClose option causes an explicit flow to the AS/400 system to close the statement. Why would you want to disable LazyClose? Consider an application that waits for user input, locks and potentially updates a row of data, and then waits for more input. If the application counts on the close to unlock the locked row, LazyClose can cause rows in the database to remain locked for extended periods of time (until the application receives more input from the user).

### 7.2.5 ExtendedDynamic

ExtendedDynamic support improves performance by:

- Preparing the SQL statement.
- Optimizing the SQL statement.
- Saving the statement for reuse.
Use of ExtendedDynamic support is the DEFAULT.

Always use ExtendedDynamic support except:

- During development.
- If you rarely use the same SQL statement more than once.

**Note:** Extended Dynamic SQL support, also known as package support, offers perhaps the biggest potential performance boost to users of both off-the-shelf and custom applications. Package support lets you store prepared SQL statements in a package file on the AS/400 system so you can use the statements in the future without the overhead of preparing and optimizing the statement again.

With extended dynamic support, the first time the statement is run, information about an SQL statement is saved in an SQL package object on the AS/400 server. The next time the statement is run, the SQL statement utilizes the information saved on the AS/400 server, thus saving significant processing time. Statements that are cached include SELECT, positioned UPDATE and DELETE, INSERT with subselect, DECLARE PROCEDURE, and all other statements that include parameter markers. A system without package support for SQL statements is the same as a system that stores programs only in source form and then has to compile a program every time a user needs to run it. The ExtendedDynamic setting has two possible values: 0 disables extended dynamic support; 1, the default, enables package support. The setting works in conjunction with the PackageAPPLICATIONNAME setting.

Package support as implemented by the CA/400 applications driver is transparent to ODBC applications (and ODBC application programmers). The ODBC driver creates the SQL packages, and no special programming is required from the ODBC application to add statements to and use them from the package. You can use package support with an application right out of the box or with your own custom applications. You can also control package usage on a per application basis, as you see in the PackageAPPLICATIONNAME section.

Most applications benefit from using SQL package support. However, an application that rarely or never uses the same SQL statement more than once does not benefit much from using a package. In fact, storing information in a package that is never used may cause slight performance degradation. You might want to disable package support with a 0 setting for such an application. You might also want to disable package support for an ODBC application in development. The many changes a developing application goes through can fill a package with statements that are never reused. Once the application is complete and ready to be shipped, the package should be primed with the SQL statements that are used.

### 7.2.6 PackageAPPLICATIONNAME

Use this parameter to customize your ExtendedDynamic support. You can customize the following:

- In which AS/400 library the package information is stored.
- The package name.
- How a package is created and used:
  - 0-Never created.
1. Used if already created.
2. Created and used-DEFAULT.

- What to do if the package is full:
  0. Be read only-DEFAULT.
  1. Be cleared.

- What to do if the package is unusable:
  0. Give an error message.
  1. Give a warning message-DEFAULT.
  2. Give no indication.

Note: When you run an ODBC application with extended dynamic support enabled, the CA/400 ODBC driver checks the ODBC.INI file for an entry for that application. If the driver finds no entry (if this is the first time you have run the application, the driver does not find an entry), it creates one using default values. If the driver finds an entry, (usually because the application has been run before), it uses the information in the entry to control extended dynamic support for the application. Thus, you can use extended dynamic support on an application-by-application basis. For example, the Windows program name for the Database Access application that comes with CA/400 is SQLIBM. So, the ODBC driver looks for an entry in the data source’s section of the ODBC.INI file with the label PackageSQLIBM= in the appropriate data source section of the ODBC.INI file. The entry looks similar to the following:

- PackageSQLIBM=QGPL/SQLIBM(FBA),2,0,1

The entries for application names are in the form:

- PackageAPPLICATIONNAME=LIBL/PACKAGE(SFX),USAGE,FULL,ERROR

where LIB is the AS/400 library where the package resides or is created.
PACKAGE is the package name (if the name is longer than seven characters, the first seven are used) and SFX is a three-character suffix that completes the package name. As was mentioned previously, the ODBC driver creates the application entry using default values for all of the entry’s parameters the first time you run an ODBC application with extended dynamic support enabled. You can subsequently change the library and package names; however, you cannot change the suffix. The ability to change the package and library names offers flexibility in using package support. For an ODBC application with fixed SQL statements accessing one library, it may make sense for all users to use the same package. For an interactive query application that accesses a variety of databases in many libraries, it may be valuable to create a separate package or even multiple packages for each user.

When the ODBC driver creates an SQL package, several settings for the job creating the package are stored in or associated with the package. These include the current naming convention, commitment control mode, date and time formats and separators, decimal separator, and the default collection. With the exception of the default collection (the first library in the DefaultLibraries setting), the ODBC driver uses these settings to determine the three-character suffix that is part of the package name. This is why you are not allowed to change the package name suffix. You may change the first seven characters of the package name and the library name where it is created. Because of the way SQL packages work, the current default collection (the first library in the
Default Libraries setting) must be the same as when the package was created or the package is not usable.

The first numeric value after the suffix specifies USAGE. The USAGE parameter has three possible values.

- 0 indicates that the ODBC driver does not use a package for this application.
- 1 indicates the driver uses an existing package in read-only mode (that is, the driver does not add new statements to the package).
- 2, the default, indicates the driver creates a new package if one does not exist and adds new statements to the package.

The 1 parameter deserves a little more explanation. It is used when there are a limited number of SQL statements that are always the same for all users. The developer for such an application can “prime” the package with the statements by running the application once to add the statements to the package. The developer can then set the USAGE parameter to 1, read-only mode, keeping the package contents constant.

A package holds a finite number of SQL statements. FULL lets you decide whether the package is read-only when it is full (0, the default) or the package is cleared when it is full (1). Full = 0 lets you continue to use any statements you already have in the package and does not let you add any new ones. Full = 1 clears all statements out of your package and lets you start over again.

Packages can store up to 512 statements.

ERROR lets you decide whether the ODBC driver returns an error when a package cannot be used. Situations where the package cannot be used include:

- A package is unusable if the default library name in the ODBC.INI file is different than the one the package was created with. For example, if you use the ODBC administrator and change the default library name (the default library name is the first library in the ODBC’s Administrator’s Default libraries list), then run the application again; the old package is unusable (and a new package is not created).
- The job CCSID does not match the job CCSID under which the package was created.

Error is set as follows:

- 0 indicates that the driver returns an error if the package is unusable.
- 1, the default, indicates the driver returns a warning message if the package is unusable.
- 2 indicates the driver does not return an error if the package is unusable.

Package usability is determined on the first call of the SQLPrepare or SQLExecuteDirect API that tries to use the package.

Some ODBC applications may not let you proceed even if only a warning is returned. In this case, you probably want to use a 2 parameter for the ERROR value. This way your application can continue.
7.2.7 Summary - ODBC.INI

Default settings are usually the best performing.

There are several ways to control and tune the performance of the AS/400 system’s Client Access for Windows ODBC driver. The controlling and tuning are mostly done through the ODBC.INI file.

The most important parameter for improved ODBC performance is most often the use of ExtendedDynamic support.

Other control and tuning parameters include:
- DefaultLibraries
- RecordBlocking
- BlocksizeKB
- LazyClose
- PackageAPPLICATIONNAME

Note: As you can see from this tour of the CA/400 ODBC driver .INI settings, the default settings provide good performance to most users. But some applications may benefit from analysis and tuning for optimal performance. The biggest performance gains are probably made by adjusting the SQL package support setting to suit your situation.

The text for the section on “Performance Tuning IBM’s ODBC Driver” was supplied by Bob Nelson and reprinted with permission from his article in the July 1995 issue of NEWS 3X/400 magazine.

7.3 Query Behavior

In Chapter 5, “Using the VisualAge Database Functions” on page 45, we found out that there are two ways to perform INSERT, UPDATE, and DELETE queries to the database.

You can use the SQL Statement part to perform these queries. When you follow this approach, you have to create explicitly a part and a query, and you perform this query by issuing the executeQuery action on the part.

The other possibility is to perform actions directly on a result table or on the query parts. You can, for example, edit the information in a result table and this is translated for you in an UPDATE statement.

The way these two approaches are handled by the different parts is completely different and this can have a serious impact on the performance of your application.

7.3.1 Explicit Query

Explicit queries are queries that are created by the developer of the application, and that are stored in one of the access sets of the application.

Let’s take a look at the following query:
UPDATE DSTRCT
    SET DNAME=:DNAME
    WHERE ((DID = :DID) AND (DWID = :DWID))

This statement is translated into the following ODBC calls by the SQL Statement part:

SQLAllocStmt()
SQLPrepare("UPDATE DSTRCT SET DNAME=? WHERE DID=? AND DWID=?")
SQLSetParam( new_name )
SQLSetParam( district_id )
SQLSetParam( warehouse_id )
SQLExecute()
SQLExecute()
...
SQLExecute()
SQLFreeStmt( SQL_CLOSE )

The statement gets prepared explicitly and ODBC re-uses this statement every time the executeQuery action is issued on the SQL Statement part. This approach is very useful when you have to perform multiple updates on a table.

Another advantage of this approach is that when you enabled extendedDynamic support for your data source, every explicit query creates an SQL statement in the SQL package associated with your application. This improves performance even more.

### 7.3.2 Implicit Query

An implicit query is a query that is executed by a database function part without the explicit cooperation of the developer of the application.

For example, the UPDATE query in the previous section can also be performed by directly editing a result table that displays the different districts. Changing the name column for one of the rows results in the following query:

UPDATE DSTRCT
    SET DNAME='D00101jer'
    WHERE DID=1 AND DWID='0001'

This statement is translated into the following ODBC calls by the SQL Statement part:

SQLAllocStmt()
SQLExecDirect("UPDATE DSTRCT SET DNAME='D00101jer' WHERE DID=1 AND DWID='0001'")
SQLFreeStmt( SQL_DROP )

ODBC creates these calls for every row that you edit in the result table. This approach is very useful when you only have to change small amounts of data. In case you have to change serious amounts of data, for example, with an order entry application, the explicit queries give much better performance.

In case you enabled extendedDynamic support, you do not see any SQL statements for these implicit queries in the SQL package associated with the application.
Appendix A. Installation of the Sample Application Diskette

This appendix describes the contents of the diskette that is included with this publication.

Not only the VisualAge sample applications will be installed, but we also included a TpcWsSetupApp application. This application automatically creates a library called CSDB on an AS/400 system of your choice, creates physical files in this library, and copies the contents from the files of the diskette to this library. In this way, you are able to use the sample application on a small subset of the CSDB database.

A.1 Installation of the Tpc Applications

1. File installation:
   a. Copy the following files in the directory where the VisualAge Client is installed:
      • SYSTEM.NAM
      • DSTRCT
      • CSTMR
      • ORDERS
      • ORDLIN
      • ITEM
      • STOCK
      • LABSAPP.DAT
   b. Open the SYSTEM.NAM file with any editor and substitute the name XXXXXXXXXX with the name of the AS/400 system you are going to work with.

2. Create the ODBC Datasource:
   a. If you are going to use ODBC, define a datasource CSDB. This datasource must refer to a library called CSDB in the system you are going to work with.

   Note: The library CSDB is going to be built during the installation procedure.

3. Start VisualAge for Smalltalk:
   a. Go to the VisualAge Organizer window, choose Application in the menu bar, and from there, click on Import/Export (see Figure 68 on page 114).
   b. From the following menu, choose Import Application. Select the file LABSAPP.DAT as the file to open (see Figure 69 on page 114). That file should be in your VisualAge client directory.
Figure 68. Importing Applications

Figure 69. Select Library
c. A three-pane window is shown (see Figure 70 on page 115). On the left side, there is the list of applications contained in the library. When you choose one of them, the central pane shows you a list of all the versions of the application. From this list, you are asked to choose the versions you want to load. Do this by selecting it on the central pane and then add it to the list of the selected versions (shown in the right pane) using the >> button.

Select all the versions of all the applications (see Figure 71 on page 116). Click the OK button to start the loading.

At the end of this operation, you have loaded the selected versions in the VisualAge library.

Figure 70. Select Versions to Load in the Library
d. When the loading is finished, go back to the VisualAge Organizer, choose Application in the menu bar, and from there, click on Load (see Figure 72 on page 117). From the submenu, choose Available Applications.

Another three-pane window appears (see Figure 73 on page 117). This time, you have to choose the applications that you want to load into your image.

You need to load the following applications:
- TpcLabApp
- TpcWsLabApp
- TpcPerformanceMonitorApp
- TpcWsSetupApp
- ExxASToolsApp
Figure 72. Load Applications in the Image

Figure 73. Select Applications to Load in the Image
4. Loading a different version:
   a. If you want to change the version of TpcLabApp that you work with, go to the VisualAge Organizer Window and select the application TpcLabApp.
   b. Open the contest menu and choose Load. From the submenu, choose Another Edition (see Figure 74).
   c. A new window appears (see Figure 75 on page 119). From there, you can choose the version you want to load.

![VisualAge Organizer Window](image)

*Figure 74. Select Another Version to Load in the Image*
Appendix A. Installation of the Sample Application Diskette 119

Figure 75. Select the Version

Figure 76. Starting the Setup Procedure
5. Setup the environment:
   You are now ready to set up the environment.
   a. Go to the System Transcript. Type the following Smalltalk code:
      ```smalltalk
      AS400System reconfigureAllSystems
      ```
      Select this line of code with your mouse, open the contest menu, and choose execute.
      This method ensures that VisualAge for Smalltalk knows the names of the AS/400 systems you have currently defined in the configuration of your PC.
   b. Go back to the VisualAge Organizer window. Choose the TpcWsSetupApp. On the right pane, select TpcSetupView (see Figure 76 on page 119).
   c. Click on the Test button of the tools bar (see Figure 76 on page 119). That button is the one showing the circle arrow.
   d. The Setup window starts (see Figure 77). Enter the system of the name you want to work with (this system must be at least at OS/400 V3R1). Leave CSDB as the name of the library. Click the GO button to start the setup procedure.

![Figure 77. Setup Installation Window](image)

You are now ready to use the sample application.
Appendix B. ODBC Error Messages

ODBC error messages can have three different sources:

- The ODBC driver:

  An error reported on an ODBC driver has the following format:

  \[[vendor] [ODBC_component] message\]

  ODBC_component is the component in which the error occurred.

  For example, an error message from INTERSOLV’s SQL Server driver looks similar to this:

  \[[INTERSOLV] [ODBC SQL Server driver] Login incorrect.\]

  If you get this type of error, check the last ODBC call your application made for possible problems or contact your IBM support representative.

- The database system:

  An error that occurs in the data source includes the data source name in the following format:

  \[[vendor] [ODBC_component] [data_source] message\]

  With this type of message, ODBC_component is the component that received the error from the data source indicated. For example, you may get the following message from an Oracle data source:

  \[[INTERSOLV] [ODBC Oracle driver] [Oracle] ORA-0919: specified length too long for CHAR column\]

  If you get this type of error, you did something incorrectly with the database system. Check your database system documentation for more information or consult your database administrator.

  In this example, check your Oracle documentation.

- The driver manager:

  The driver manager is a DLL that establishes connections with drivers, submits requests to drivers, and returns results to applications. An error that occurs in the driver manager has the following format:

  \[[vendor] [ODBC DLL] message\]

  The vendor can be Microsoft or INTERSOLV.

  For example, an error from the Microsoft driver manager might look similar to this:

  \[[Microsoft] [ODBC DLL] Driver does not support this function\]

  If you get this type of error, consult the Programmer’s Reference for the Microsoft ODBC Software Development Kit available from Microsoft.
Appendix C. Special Notices

This publication is intended to help application developers, IBM Business Partners, and IBM technical representatives to create client/server applications based on VisualAge for Smalltalk using the ODBC interface and DB2 for OS/400. The information in this publication is not intended as the specification of any programming interfaces that are provided by VisualAge for Smalltalk. See the PUBLICATIONS section of the IBM Programming Announcement for VisualAge for Smalltalk for more information about what publications are considered to be product documentation.

References in this publication to IBM products, programs or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM’s product, program, or service may be used. Any functionally equivalent program that does not infringe any of IBM’s intellectual property rights may be used instead of the IBM product, program or service.

Information in this book was developed in conjunction with use of the equipment specified, and is limited in application to those specific hardware and software products and levels.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Licensing, IBM Corporation, 500 Columbus Avenue, Thornwood, NY 10594 USA.

The information contained in this document has not been submitted to any formal IBM test and is distributed AS IS. The information about non-IBM (“vendor”) products in this manual has been supplied by the vendor and IBM assumes no responsibility for its accuracy or completeness. The use of this information or the implementation of any of these techniques is a customer responsibility and depends on the customer’s ability to evaluate and integrate them into the customer’s operational environment. While each item may have been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere. Customers attempting to adapt these techniques to their own environments do so at their own risk.

Any performance data contained in this document was determined in a controlled environment, and therefore, the results that may be obtained in other operating environments may vary significantly. Users of this document should verify the applicable data for their specific environment.

Reference to PTF numbers that have not been released through the normal distribution process does not imply general availability. The purpose of including these reference numbers is to alert IBM customers to specific information relative to the implementation of the PTF when it becomes available to each customer according to the normal IBM PTF distribution process.

The following terms are trademarks of the International Business Machines Corporation in the United States and/or other countries:

Advanced Peer-to-Peer Networking
APPN

Application System/400
AS/400
The following terms are trademarks of other companies:

Windows is a trademark of Microsoft Corporation.

PC Direct is a trademark of Ziff Communications Company and is used by IBM Corporation under license.

UNIX is a registered trademark in the United States and other countries licensed exclusively through X/Open Company Limited.

Other trademarks are trademarks of their respective companies.
Appendix D. Related Publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

D.1 International Technical Support Organization Publications

For information on ordering these ITSO publications see “How To Get ITSO Redbooks” on page 127.

A complete list of International Technical Support Organization publications, known as redbooks, with a brief description of each, may be found in:

International Technical Support Organization Bibliography of Redbooks, GG24-3070.

D.2 Other Publications

These publications are also relevant as further information sources:

• Getting Started with VisualAge, SC34-4535-00
• VisualAge User's Guide, SC34-4518-00
• VisualAge User's Reference, SC34-4519-00
• AS/400 Connection User's Guide and Reference, in online format with AS/400 Connection
• VisualAge Features Class Guide and Reference, in online format with VisualAge for Smalltalk and IBM Smalltalk products
• VisualAge ODBC Support Drivers Reference, in online format with VisualAge for Smalltalk products
• Programmer’s Guide to Building Parts for Fun and Profit, SC34-4496-01
• VisualAge for Smalltalk Reports Guide and Reference, in online format with the VisualAge for Smalltalk Reports feature
• Introduction to Object-Oriented Programming with IBM Smalltalk, SC34-4491-02
• IBM Smalltalk Programmer’s Reference, SC34-4493-02
• IBM Smalltalk User’s Guide, SC34-4536-00
• IBM Smalltalk and VisualAge Resource Catalog, G325-0813-02
• VisualAge: Building GUIs for Existing Applications, GG24-4244-00
• VisualAge: Concepts and Features, GG24-3946-00
• Client Access Windows 3.1 Client for OS/400 ODBC User’s Guide, SC41-3533-00
How To Get ITSO Redbooks

This section explains how both customers and IBM employees can find out about ITSO redbooks, CD-ROMs, workshops, and residencies. A form for ordering books and CD-ROMs is also provided.

This information was current at the time of publication, but is continually subject to change. The latest information may be found at URL http://www.redbooks.ibm.com/redbooks.

How IBM Employees Can Get ITSO Redbooks

Employees may request ITSO deliverables (redbooks, BookManager BOOKs, and CD-ROMs) and information about redbooks, workshops, and residencies in the following ways:

- **PUBORDER** — to order hardcopies in United States
- **GOPHER link to the Internet** - type GOPHER.WTSCPOK.ITSO.IBM.COM
- **Tools disks**
  To get LIST3820s of redbooks, type one of the following commands:
  ```
  TOOLS SENDTO EHONE4 TOOLS2 REDPRINT GET SG24xxxx PACKAGE
  TOOLS SENDTO CANVM2 TOOLS REDPRINT GET SG24xxxx PACKAGE (Canadian users only)
  ```
  To get lists of redbooks:
  ```
  TOOLS SENDTO WTSCPOK TOOLS REDBOOKS GET REDBOOKS CATALOG
  TOOLS SENDTO USDIST MKTTOOLS MKTTOOLS GET ITSOCAT TXT
  TOOLS SENDTO USDIST MKTTOOLS MKTTOOLS GET LISTSERV PACKAGE
  ```
  To register for information on workshops, residencies, and redbooks:
  ```
  TOOLS SENDTO WTSCPOK TOOLS ZDISK GET ITSOREGI 1996
  ```
  For a list of product area specialists in the ITSO:
  ```
  TOOLS SENDTO WTSCPOK TOOLS ZDISK GET ORGCARD PACKAGE
  ```
- **Redbooks Home Page on the World Wide Web**
- **IBM Direct Publications Catalog on the World Wide Web**
  IBM employees may obtain LIST3820s of redbooks from this page.
- **ITSO4USA category on INEWS**
- **Online** — send orders to: USIB6FPL at IBMMAIL or DKIBMBSH at IBMMAIL
- **Internet Listserver**
  With an Internet E-mail address, anyone can subscribe to an IBM Announcement Listserver. To initiate the service, send an E-mail note to announce@webster.ibmlink.ibm.com with the keyword subscribe in the body of the note (leave the subject line blank). A category form and detailed instructions will be sent to you.
How Customers Can Get ITSO Redbooks

Customers may request ITSO deliverables (redbooks, BookManager BOOKs, and CD-ROMs) and information about redbooks, workshops, and residencies in the following ways:

- **Online Orders** (Do not send credit card information over the Internet) — send orders to:

<table>
<thead>
<tr>
<th>In United States:</th>
<th>IBMMAIL</th>
<th>Internet</th>
</tr>
</thead>
<tbody>
<tr>
<td>usib6fpl@ibmmail</td>
<td></td>
<td><a href="mailto:usib6fpl@ibmmail.com">usib6fpl@ibmmail.com</a></td>
</tr>
<tr>
<td>In Canada:</td>
<td>caibmbkz@ibmmail</td>
<td><a href="mailto:lmannix@vnet.ibm.com">lmannix@vnet.ibm.com</a></td>
</tr>
<tr>
<td>Outside North America:</td>
<td>bookshop@dkibmbsh@ibmmail</td>
<td><a href="mailto:bookshop@dk.ibm.com">bookshop@dk.ibm.com</a></td>
</tr>
</tbody>
</table>

- **Telephone orders**

  United States (toll free) 1-800-879-2755
  Canada (toll free) 1-800-IBM-4YOU
  Outside North America (long distance charges apply)
  (+45) 4810-1320 - Danish
  (+45) 4810-1420 - Dutch
  (+45) 4810-1540 - English
  (+45) 4810-1670 - Finnish
  (+45) 4810-1220 - French
  (+45) 4810-1020 - German
  (+45) 4810-1620 - Italian
  (+45) 4810-1270 - Norwegian
  (+45) 4810-1120 - Spanish
  (+45) 4810-1170 - Swedish

- **Mail Orders** — send orders to:

  IBM Publications 144-4th Avenue, S.W. Sortemosevej 21
  Publications Customer Support Calgary, Alberta T2P 3N5 DK-3450 Allerød
  P.O. Box 29554 Raleigh, NC 27626-0570 Canada Denmark
  USA

- **Fax** — send orders to:

  United States (toll free) 1-800-445-9269
  Canada (toll free) 1-800-267-4455
  Outside North America (+45) 48 14 2207 (long distance charge)

- **1-800-IBM-4FAX (United States) or (+1) 415 855 43 29 (Outside USA)** — ask for:

  Index # 4421 Abstracts of new redbooks
  Index # 4422 IBM redbooks
  Index # 4420 Redbooks for last six months

- **Direct Services** - send note to softwareshop@vnet.ibm.com

- **On the World Wide Web**

  Redbooks Home Page http://www.redbooks.ibm.com/redbooks

- **Internet Listserver**

  With an Internet E-mail address, anyone can subscribe to an IBM Announcement Listserver. To initiate the service, send an E-mail note to announce@webster.ibm.link.ibm.com with the keyword subscribe in the body of the note (leave the subject line blank).
IBM Redbook Order Form

Please send me the following:

<table>
<thead>
<tr>
<th>Title</th>
<th>Order Number</th>
<th>Quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

• Please put me on the mailing list for updated versions of the IBM Redbook Catalog.

First name                      Last name

Company

Address

City                      Postal code   Country

Telephone number          Telefax number   VAT number

• Invoice to customer number

• Credit card number

Credit card expiration date Card issued to Signature

We accept American Express, Diners, Eurocard, Master Card, and Visa. Payment by credit card not available in all countries. Signature mandatory for credit card payment.

DO NOT SEND CREDIT CARD INFORMATION OVER THE INTERNET.
## List of Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>APA</td>
<td>all points addressable</td>
</tr>
<tr>
<td>IBM</td>
<td>International Business Machines Corporation</td>
</tr>
<tr>
<td>ITSO</td>
<td>International Technical Support Organization</td>
</tr>
<tr>
<td>PROFS</td>
<td>Professional Office System</td>
</tr>
<tr>
<td>APPC</td>
<td>Advanced Program-to-program Communication</td>
</tr>
<tr>
<td>DLL</td>
<td>Dynamic Link Library</td>
</tr>
<tr>
<td>ODBC</td>
<td>Open Database Connection</td>
</tr>
<tr>
<td>DDM</td>
<td>Distributed Database Manager</td>
</tr>
<tr>
<td>SQL</td>
<td>Structured Query Language</td>
</tr>
</tbody>
</table>
naming convention 25, 32

object design 77
object model 80
object model design 79
ODBC
  component 2
  definition 1
  driver, types of 3
  interface 1
ODBC error message 121
odbc or ddm 96
ODBC support, configuration of Client Access/400 21
ODBC.INI 110
ODBC.INI file 35
optimizing ODBC access 101
optimizing query 101
order line table layout (ORDLIN) 12
orders table layout (ORDERS) 12

packageAPPLICATIONNAME 107
packeting 50
persistency manager
  abstract persistency manager 82
  create 80
  DDM 88
  implementation of 82
  ODBC 84
  phase 1 object analysis 9
  PTF
    on Client Access/400 17
    SF28583 17
    SF28840 17

query behavior 110

real-life OO application design 77
record based access 89
RecordBlocking 105
response time 97
results of multi-row query 53
results of single-row query 65
running query 56
running query with host variable 59
runtime behavior 97
runtime image
  size of 97

sample ODBC trace log 26, 33
setting up database connection 37
single-row query part 63
single-tier driver 3
skinny objects 103
SQL 4
SQL statement part 68
stock table layout (STOCK) 13
subfiling 101

table
  conformance level 4
  customer 11
  database terminology 13
  district 10
  order 12, 13
  tearing-off result 53
  TPC-C benchmark
    application requirement specification 9
    database layout 10
    trace log
      sample ODBC 26, 33
      tracing ODBC calls 25, 32

UPDATE statement 68
update using a multi-row query part 60
updating row in table 67
using host variable 58
using quick form 54
using the single-row query for delete 66
using the single-row query for insert 66
using the single-row query for update 66

V3R1 ODBC support 7
VisualAge database access 35
VisualAge database functions 45

working with container details part 56