Novell®

SQL *Connector*

Overview

Printing Date: **October 1, 1999**

About This Manual

Purpose of this Manual

This manual presents an overview of SQL *Connector* on client (first tier), data broker (second tier) and data driver (third tier) operating systems. Examples are provided to illustrate how SQL *Connector* can be used in a variety of environments.

Intended Audience

This document is intended for end users, programmers and system administrators who will be using SQL *Connector*. A working knowledge of Java, ODBC, JDBC, database systems, Windows and Netware operating systems and networking is suggested for using SQL *Connector*.

Structure of this Manual

This manual consists of chapters that provide an overview of SQL *Connector*, a description of its architecture and a discussion of Windows environments for using SQL *Connector*.

Associated Documents

The document set contains these manuals:

- SQL *Connector Overview*
- SQL *Connector Installation Guide*
- SQL *Connector Administration Guide*
- SQL *Connector SQL Grammar Manual*
- SQL *Connector ODBC Programmer's Guide*
- SQL *Connector JDBC Programmer's Guide*

Operating System Conventions

When there are differences in commands, examples, or syntax between operating systems, the following abbreviations are used:

Abbreviation	**Meaning**
NetWare	the Novell NetWare operating system
Windows	the Microsoft Windows 95/98/NT operating systems

Table of Contents

1 Introduction

1.1 Overview

SQL *Connector* is a *Data Request Broker* that supports a multiple-tier, multiple-database enterprise environment for connecting ODBC and JDBC applications between client computer systems and databases on Netware or Windows NT. The Data Request Broker runs on the middle-tier (between the client systems and database servers) and supports a methodology known as Enterprise Data Access (EDA). EDA has the following features:

- All databases can be accessed using one middleware connection from the client system, as opposed to one connection per database.
- All data can be accessed using a standard SQL grammar and set of datatypes, as opposed to vendor-specific SQL and datatypes.

Applications that connect to SQL *Connector* are database independent. These applications connect to the Data Broker which connects to the physical database. Applications are thus isolated from database specific variations, such as alternate forms of SQL grammar and alternate datatypes. ANSI-92 standard SQL statements can be used (such as `SELECT...`, `UPDATE...`, etc.), without regard for the database source. ANSI standard datatypes (such as `CHAR`, `INTEGER`, `TIMESTAMP`) can be used, without regard for database specific datatypes.

From the client viewpoint, SQL *Connector* presents enterprise physical databases as uniform standard ANSI-92 SQL Data Sources. Only one connection is required from the client system to SQL *Connector*, which then connects to the physical databases.

Please see the *Installation Guide* for additional information about supported databases and platforms.

1.2 Components

SQL *Connector* includes *Client*, *Data Broker* and *Data Driver* components. These components can reside on the same or different computer systems within a network. The *Client components* support the connection of SQL *Connector* to end-user or web-based applications, or application development tools, that use OBDC or JDBC. The *Data Broker components* support client connectivity over a network, and query parsing, optimization and distribution. There is also a *Data Broker component* for definining and managing Data Sources. The *Data Driver components* support connectivity to physical databases. The components can be summarized in the following table:

Component Name	**Component Functionality**
Client Components	

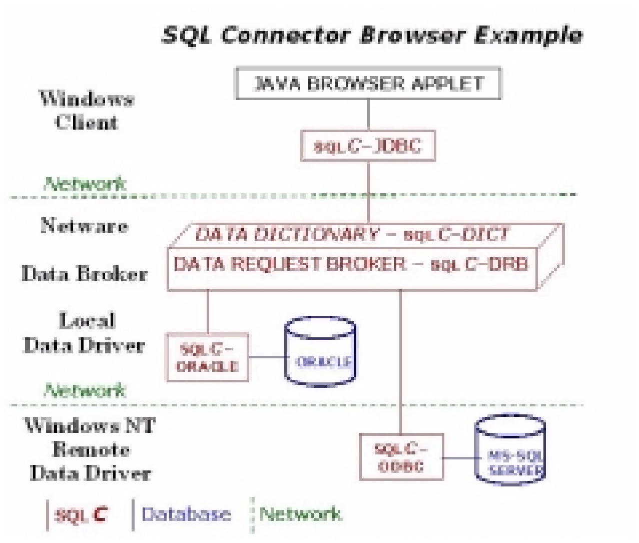
Component Name	**Component Functionality**
SQL *C*–ODBC	Microsoft® ODBC API
SQL *C*–JDBC	JavaSoft™ Java™ SQL classes
Data Broker Components	
SQL *C*–DSA	Data Source Administrator
SQL *C*–DRB	Data Request Broker
Data Driver Components (local or remote to the Data Broker)	
SQL *C*–ORA-DD	Oracle® Database Connection
SQL *C*–ODBC-DD	Microsoft ODBC™ Database Connection

1.3 Architecture

The SQL *Connector* architecture includes the Client, Data Broker and Data Driver components listed in the above table. The architecture is designed for maximum flexibility in meeting the needs of enterprise wide application development and deployment. The components have been modularized so that they can be installed on multiple tiers within an enterprise network. The following examples illustrate several possible configurations. The examples show how SQL queries from a client can be distributed by the Data Broker to multiple databases.

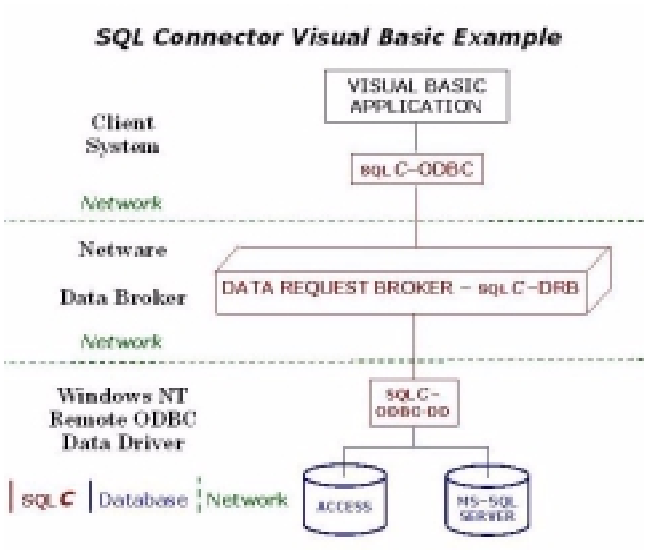
1.3.1 Three Tier Application (Intranet or Internet)

A three tier application example is a Java applet running on a Windows browser (client) that uses SQL *C*–JDBC to access a NetWare Data Broker. The Data Broker accesses an Oracle database running on the same system (with a local data driver) and uses the network to access an MS–SQL Server database on a Windows NT system (using Remote Data Driver on NT).



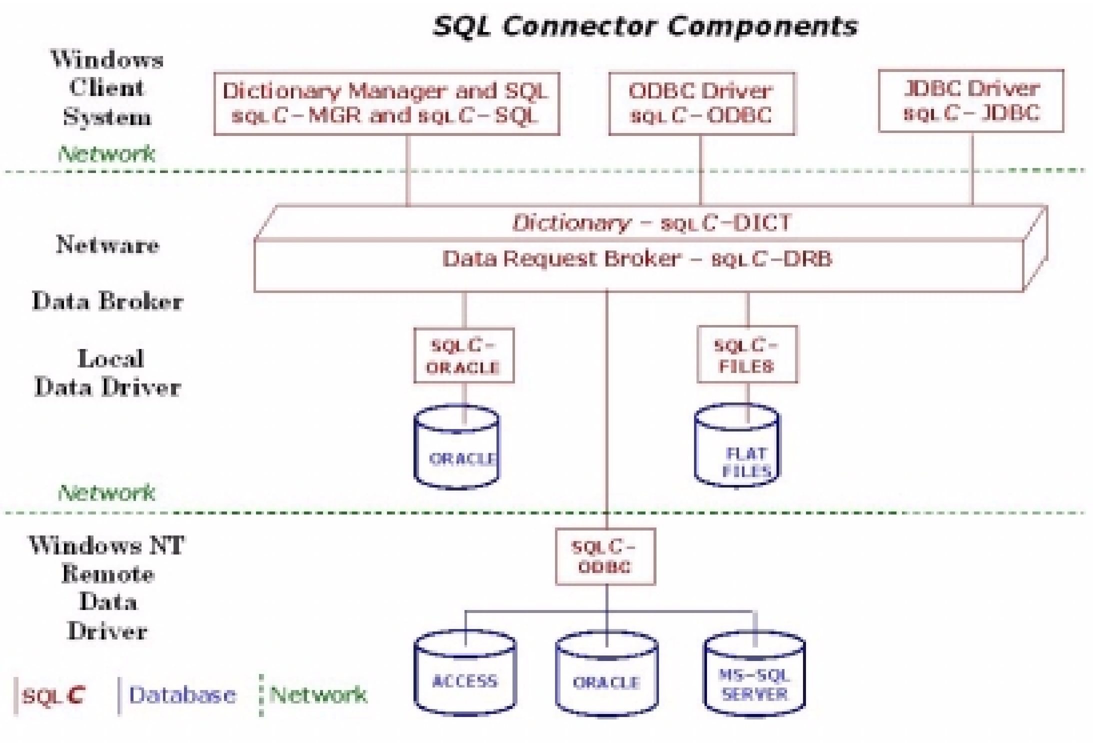
1.3.2 Two Tier Application (Two Data Sources)

A two tier application example is a Visual Basic application running on a Windows system that uses SQL *C*–ODBC to access a Netware Data Broker. The Netware Data Broker can be used to retrieve from both Access and MS–SQL Server (using Remote Data Drivers on NT).



1.3.3 Architecture Diagram

The complete SQL *Connector* component architecture is shown below:



The Client components (ODBC and JDBC drivers) can be used on both Windows (client) and Netware (broker) systems. Uses on Windows systems would include connecting to third party application development tools such as Microsoft Visual Basic (OBDC) and Symantec Visual Cafe (JDBC). Uses on Netware systems would include connecting to the Netscape Enterprise Server (ODBC) or IBM WebSphere Application Server (JDBC).

Also, the Data Drivers can be used on the same node as the Data Broker (local) or on a different node (remote), depending on the location of the physical database.

2 Architecture

2.1 Client Components

2.1.1 ODBC Driver

ODBC (Open Database Connectivity) is an API (Application Programming Interface) developed by Microsoft. ODBC is Microsoft's implementation of the X/Open SQL CLI (Call Level Interface) that defines how client/server interactions are implemented for database applications. ODBC also supports the SQL grammar and syntax specified in the ANSI SQL-92 standard.

ODBC was developed to provide vendor neutral access to data sources from a client application. With ODBC, a client application is not restricted to any one vendor's proprietary interface, and the client application can connect to any ODBC-compliant data source.

SQL *C*–ODBC is a Microsoft compliant implementation of the ODBC API (version 2.5). As such, any client application or application development environment can connect to SQL *C*–ODBC. SQL *C*–ODBC provides an enhanced ODBC SQL Grammar that can access database tables from multiple networked databases.

2.1.2 JDBC Driver

JDBC (Java Database Connectivity) is a vendor neutral API developed by JavaSoft. The interface is very similar in concept to the Microsoft ODBC API and provides Java programmers with a uniform database interface to a wide range of relational databases. JDBC is a standard part of Java and is included in JDK (Java Development Kit) 1.1. Like ODBC, a Java JDBC application is not restricted to any one vendor's proprietary interface, and the Java application can connect to any JDBC-compliant data source.

SQL *C*–JDBC is a 100% Java JDBC compliant implementation of the JDBC API (type IV, version 1.1). Any Java application, applet, or application development environment can connect to SQL *C*–JDBC. SQL *C*–JDBC provides a database independent SQL grammar that can access database tables from multiple databases.

2.2 Data Broker Components

2.2.1 Data Sources

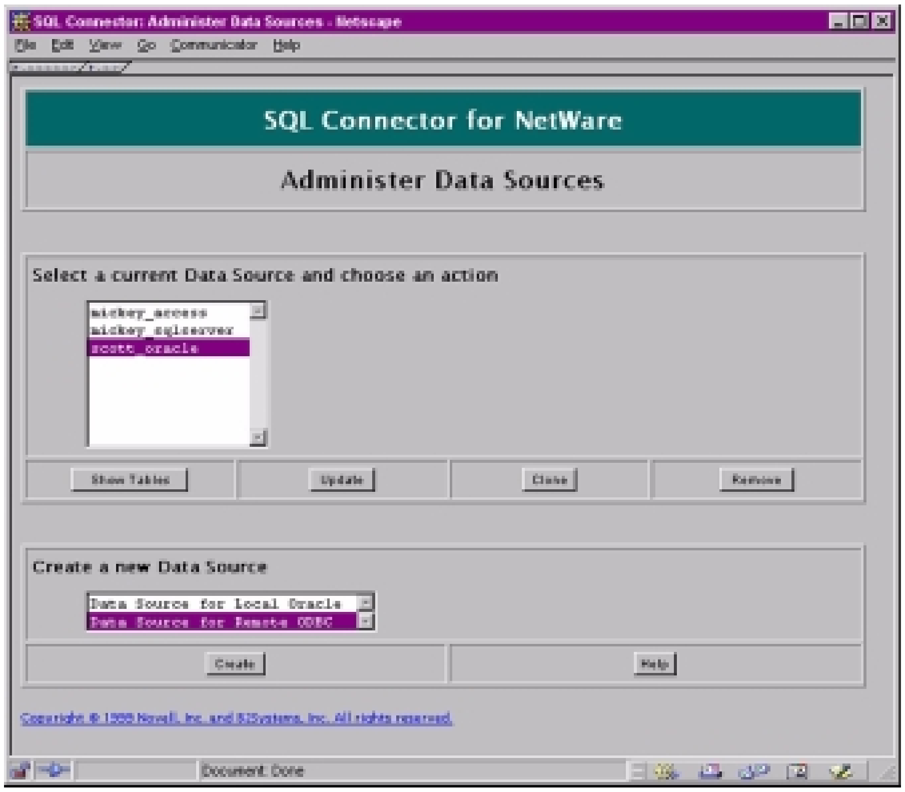
SQL *Connector* supports access to multiple physical databases by using *Data Sources*, which are defined on the same node as the SQL *Connector* Data Broker. The Data Sources have table entries that *point* to physical database tables or views on the same node or other nodes elsewhere in the network. Database views are very useful when the application only needs a subset of fields in a table, or the application requires frequent joins of multiple database tables.

A client application built using SQL *Connector* is only aware of the database tables defined by a Data Source and not the physical source of the data. The physical database source can be changed, for example, from MS Access to Oracle, or from Oracle to MS–SQL, with no change required in the application.

SQL *Connector* Data Sources support client applications which are database independent. Even though ODBC and JDBC provide vendor neutral programming interfaces, these interfaces themselves are not database independent. For example, some database vendors do not support ANSI SQL datatypes for decimal numbers. Consequently, applications built with vendor supplied interfaces must be aware of which datatypes are supported. In contrast, SQL *Connector* is database independent. If an ANSI SQL datatype is not supported by the physical database, the datatype is emulated by the Data Source and mapped to an appropriate physical database type.

2.2.2 Data Source Administrator

The SQL *Connector* Data Source Administrator (SQL *C*–DSA) is a web browser application for creating and maintaining SQL *Connector* Data Sources and for importing physical database table information. The SQL *Connector* Data Source Administrator primary web page is shown below:



SQL Connector: Administer Data Sources - Netscape

SQL Connector for NetWare

Administer Data Sources

Select a current Data Source and choose an action

mickey_access
mickey_sqlserver
scott_oracle

Create a new Data Source

Data Source for Local Oracle
Data Source for Remote ODBC

Copyright © 1998 Novell, Inc. and B2Systems, Inc. All rights reserved.

2.2.3 Data Request Broker

SQL *Connector* contains a Data Request Broker (DRB) which includes a sophisticated SQL processor, optimizer and distributor. The SQL processor converts ANSI standard

SQL into database vendor specific SQL, regardless of the database. *The SQL processor supports full read/write capability for all database connections.*

Structured Query Language (SQL) is a well-defined ANSI (American National Standards Institute) standard language for database access. Database vendors strive to support the grammar and syntax of the SQL language, but inevitably change the implementation to support database specific features. The SQL processor is based on the ANSI SQL-92 standard and is database independent. Database independent SQL eliminates the need to learn each vendor's dialect of SQL, not all of which are SQL-92 compatible. The SQL processor also provides enhanced datatype support that may be missing from a vendor-specific database. See the *SQL Grammar Manual* for additional information.

SQL processing begins by decomposing an SQL statement (which may have embedded subqueries) into its constituent parts:

- data query statements (read data)
  - `select`
- data manipulation statements (write data)
  - `update`
  - `insert`
  - `delete`

Data queries are processed first, since the results may be needed for a data manipulation statement. For example, a statement like:

```
update emp set bday = (select bday from personnel where empno = 1234)
```

would require the `select` statement to be processed first, since the result is needed for the `update` statement.

Data Query Processing

SQL *C*–DRB can decompose, rewrite and distribute data queries as close to the data source as possible. SQL *C*–DRB also performs query pushdown and query optimization.

Query Decomposition and Rewrite

An incoming query from a client application that is using ANSI-standard SQL (independent of the database) must be rewritten using vendor-specific SQL before being sent from the broker to the database. The rewrite process also analyzes the syntax for built-in SQL functions (such as `SUM`, `AVG`, `MIN`, and `MAX`) that may or may not be supported by the database vendor's SQL. If the function is not supported by the vendor, than a vendor-supported function is substituted if one is available. If there is no vendor-equivalent function, then a rewritten query (without the function) is sent to the database, and the function will be performed by the broker when the rows are returned.

As example of using vendor specific functions, consider the conversion of a date string, such as a statement like `WHERE EMP_DATE > "07/08/99"`. If the database is Oracle, then the broker will use the Oracle `TO_DATE` function when the query is rewritten. If the database is ODBC, then the broker will use the ODBC `CONVERT` function.

Query Pushdown

Decomposition of a query also leads to decisions regarding how much of the query can be *pushed down* into the physical database. Examples of pushdown operations are `aggregates`, `join` clauses, `where` clauses and `order by` clauses. SQL *C*–DRB will determine how much of the query can be sent directly to the database and how much must be processed by SQL *C*–DRB. SQL *C*–DRB attempts to push as much processing down to the underlying database as possible to maximize the overall performance.

Pushdown processing reduces network traffic by reducing data flow across the network. For example, the SUM function is pushed down into the database if the function is supported. Only the result is returned across the network. If the summation was not pushed down, then all of the field values would be returned across the interface, which would result in increased network traffic.

Query Optimization

SQL *C*–DRB has the capability to rewrite a database query based on keys and indexes in the physical database. SQL *C*–DRB bases its decisions on the cardinality and selectivity of the tables involved in the query. This information determines the order and sequence of interactions with the tables in the query. The primary goal of the query optimization is to reduce the bandwidth requirements between SQL *Connector* and the underlying physical databases.

2.3 Data Driver Components

The SQL *Connector* Data Broker connects to physical databases (local Oracle or Remote ODBC) using the SQL *Connector* Data Driver components. There is a Data Driver component for each supported database. The Database Drivers generally connect to the physical databases in the following ways:

- SQL *Connector* direct connection to a vendor supplied connectivity library
- SQL *Connector* network connection to a vendor supplied connectivity library

2.3.1 Direct Connection to a Database Vendor Library

The most direct form of connection is to use the database vendor supplied connectivity library running on the same Netware system as the SQL *Connector* Data Broker and Data Driver components. The vendor library will then connect to the vendor database, which is executing on a database server that is the same Netware system or another computer system.

The following table shows the SQL *Connector* Data Drivers and the respective vendor libraries:

Database Component	Component Name	Vendor Library
SQL *C*–ORA-DD	Data Driver for Oracle	OCI

A vendor library connects to the physical database if it runs on the same computer system as the SQL *C*–DRB.

2.3.2 Network Connection to a Database Vendor Library

The Data Broker can connect across the network to a SQL *Connector* ODBC Data Driver running on a remote Windows NT system. The remote ODBC Data Driver then connects to the vendor supplied connectivity library running on the same remote system (Microsoft Access, Microsoft SQL Server, or Oracle).

3 Usage

3.1 Overview

The steps required to use SQL *Connector* will vary from site to site and within systems at a given site. These steps include the following:

- DBA (Database Administrator) activities
  - Installation
  - Configuration
  - Testing
- Application Development Activities
  - Programming
  - Testing

3.2 Database Administration

3.2.1 Installation

SQL *Connector* installation should be planned around which computer systems are client systems, which systems are application or web servers and which systems are database servers. Use of SQL *Connector* may require third party software such as ODBC driver managers, Java development kits, database vendor connectivity software and database server software.

3.2.2 Configuration

SQL *Connector* is configured using the client component SQL *C*–DSA. The Data Source Administrator runs on a web browser and accesses Data Sources on a Data Broker system. The Data Sources then access data from physical databases.

SQL *C*–DSA is used to create the Data Sources and to import metadata information from physical databases on database servers.

SQL statements executed by the Netscape Server "dbadmin" facility may be used to query the Data Source tables (and hence the physical database tables) using ANSI standard SQL statements. Thus the Data Source can be tested before a client application is available.

3.2.3 Testing

There are sample ODBC and JDBC programs and a database that are supplied for testing purposes. These programs can test a database connection and retrieve joined data (between departments and employees). The Data Source and sample programs are supplied with source code (SQL, C and Java) to provide a starting point for programmers who will be developing SQL *Connector* applications.

3.3 Application Development

3.3.1 Programming

Once a Data Source and data access has been tested, application development can begin. Application development typically uses one of the following environments:

- Application development tools that support ODBC, for example:
  - Microsoft Access
  - Microsoft Visual Basic
  - Powersoft PowerBuilder
  - Borland Delphi
  - Netware Data Objects
- Application development tools that support JDBC, for example:
  - Borland JBuilder
  - Powersoft PowerJ
  - Symantec Visual Cafe
  - Netscape JavaScript
  - IBM WebSphere
- Programming languages that call the ODBC API or Java SQL classes.
  - Microsoft Visual C++
  - Watcom C++
  - Microsoft Visual J++

Regardless of the choice of application development environments, the use of SQL *Connector* is transparent because of the ODBC and JDBC standards.

3.3.2 Testing

The application tools and programming languages have test capabilities that can be utilized during the development process. SQL statements can be sent to the Data Request Broker and analyzed in isolation from an application. These SQL statements can be tested for syntax, performance and efficiency. Database timing and database traces can be used to look for performance bottlenecks such as joins using non-unique indexes, sequential searches through large tables, and failure to push down SQL statements from the application to the physical database.

3.4 Summary

SQL *Connector* is a *data access Data Request Broker environment* that provides a multiple-tier, multiple-database enterprise environment for connecting ODBC and JDBC compliant applications to multiple database sources.

- SQL *Connector* includes client components for connecting to ODBC and JDBC environments and for maintenance and testing of SQL *Connector* Data Sources.
- SQL *Connector* includes Data Broker components for query parsing, optimizing and distribution, even among multiple databases.
- SQL *Connector* includes Data Driver components for connecting to vendor databases in the absence of a vendor connectivity library and provides additional connectivity not supplied by database vendors.