SQLWays Documentation

Database and Application Migration Software

Version 6.0

Ispirer Systems Ltd.

Contents

Application Migration Concepts

User's Guide

SQLWays Troubleshooting Guide

Frequently Asked Questions

Ispirer Systems Resources and Contacts

Legal Notices

Conventions Used in This Manual

The following conventions are used for describing grammar elements in this manual:

TABLE 1. Conventions

Convention	Meaning
B	**Compulsory** element, which **must** appear in the statement. Compulsory elements are defined in the grammar without any brackets.
[B]	**Optional** element, which can **optionally** appear in the statement. Optional elements are delimited with square brackets.
B1[,BN]...	**List** element, containing **one or more** statements delimited by comma. List elements are defined in the grammar by specifying the first element followed by the second element delimited with square brackets and ellipsis.
{B1 | B2}	**Selective** element, specifying that **one** value from the list **must** appear in the statement. Selective elements are delimited with braces and are divided by '|' or keyword 'OR'.

Introduction

SQLWays is innovative database migration software converting database structure and data between IBM DB2, Oracle, Microsoft SQL Server, Sybase, MySQL, Pervasive.SQL and other database management systems. It supports migration of tables, constraints, indexes, views, stored procedures, functions and triggers.

The present documentation contains detailed description of SQLWays, its characteristics and functions.

To learn about the distinguishing features and benefits of SQLWays, read the chapter Key Benefits. To find the information about the supported databases and their versions, see Supported Databases. To learn what particular objects can be converted between each database pair, go to the chapter Database Object and Features Support. To find the information on the latest product updates, read What's New.

To find out how to perform migration tasks with the help of SQLWays Wizard and SQLWays command line, go to User's Guide. To become familiarized with the concepts of the migration process, see Database Migration Concepts.

If you are a new user, read Getting Started.

Key Benefits

Complete, Functionality-Rich and Adaptive

- Transfers data including LOB (images, binary data, formatted text) and Spatial (GIS) data and converts tables (default values, null and identity properties, check and unique constraints, primary and foreign keys, comments), indexes, views, triggers, stored procedures and functions
- Offers powerful capabilities for data type conversion that can be specified on the global level applicable for all tables and on the local level, for each table individually, taking into account different maximum length limits of identical data types in different databases.
- Resolves reserved word and identifier conflicts
- Makes possible renaming each database object and table column

Flexible, Open and Extensible Migration Process

- Uses the export/import process that provides the most flexible capabilities for managing migration
- Allows migrating between geographically divided databases
- Uses open formats facilitating customization and extension of migration tasks
- Exports database structures to SQL scripts and data to ASCII text files (CSV, SQL INSERT statements, TAB-delimited, fixed length and other formats).
- Provides you with a wizard-driven user interface that makes migration easy
- Allows you to automate migration process with the help of the SQLWays command line tool

Migration to Multiple Database Platforms

- Makes possible migration of databases to IBM DB2, Oracle, Microsoft SQL Server, Sybase and MySQL on different platforms
- Supports a variety of database versions, not being limited to the latest versions only

High Performance

- Uses high-performance native database tools to import data
- Data export and transformation modules written in C/C++

Supported Databases

This chapter contains the list of databases supported by SQLWays. Source databases are those from which the database structure and data are converted. Target databases are those to which the migrated database structure and data are imported.

Source databases supported by SQLWays are as follows:

TABLE 2. Supported Source databases

Database Product	Version
IBM DB2 for Linux, Unix and Windows	8.2, 8.1, 7.2, 7.1, 7.0, 6.1 and earlier
IBM DB2 for z/OS, OS/390 and MVS	8.1, 7.1, 6.1, 5.2, 5.1 and 4.1
IBM DB2 for iSeries and AS/400	V5R3, V5R2, V5R1, V4R5, V4R4 and earlier
Oracle	10g, 9i, 8i, 8.0.x and 7.x
Microsoft SQL Server	2005, 2000, 7.0 and 6.5
Sybase Adaptive Server Enterprise (ASE)	12.5.1, 12.5, 12.0, 11.x and earlier
Sybase Adaptive Server Anywhere (ASA), Sybase SQL Anywhere	9.0, 8.0.x, 7.0.x, 6.0.x, 5.5 and 5.0
Sybase IQ	12.x
Informix Dynamic Server (IDS)	10, 9.4, 9.3, 9.2, 9.1, 7.3, 7.2, 7.1 and earlier
Informix Standard Engine (SE)	7.x, 5.x, 4.x and earlier
MySQL	5.x, 4.x, 3.23 and earlier
PostgreSQL	8.x, 7.x and 6.x
Progress	10.x, 9.x and 8.x
SAP DB	7.4, 7.3 and earlier
Pervasive.SQL	v8, 2000 and 7
Microsoft Access	2003, 2000, 97, 95 and 2.0
Interbase, Firebird	7.1, 6.x, 5.x and 4.x
Lotus Notes	6, 5 and earlier
dBase, FoxPro, Excel, Paradox, Gupta SQLBase, Clipper and any other ODBC-compliant database	any version

Target databases supported by SQLWays are as follows:

TABLE 3. Supported Target Databases

Database Product	Version
IBM DB2 for Linux, Unix and Windows	8.2, 8.1, 7.2, 7.1, 7.0, 6.1 and earlier
IBM DB2 for z/OS and OS/390	8.1, 7.1, 6.1, 5.2, 5.1
IBM DB2 for iSeries and AS/400	V5R3, V5R2, V5R1, V4R5, V4R4
Oracle	10g, 9i, 8i and 8.0.x
Microsoft SQL Server	2005, 2000, 7.0 and 6.5
MySQL	5.x, 4.x and 3.23
PostgreSQL	8.x and 7.x

TABLE 3. Supported Target Databases

Database Product	Version
Sybase Adaptive Server Enterprise	12.5.1, 12.5, 12.0, 11.x
Sybase Adaptive Server Anywhere	9.0, 8.0.x, 7.0.x, 6.0.x, 5.5 and 5.0
Sybase IQ	12.x
Informix Dynamic Server (IDS)	10, 9.4, 9.3, 9.2, 9.1, 7.3, 7.2, 7.1 and earlier
Pervasive.SQL	v8

Database Object and Features Support

This chapter illustrates the capacity of SQLWays to support migration of a wide range of databases and particular database objects and features to the following database management systems:

- IBM DB2 for Linux, Unix and Windows
- IBM DB2 for z/OS and OS/390
- IBM DB2 for iSeries and AS/400
- Oracle
- Microsoft SQL Server
- MySQL
- PostgreSQL
- Sybase Adaptive Server Enterprise
- Pervasive.SQL

IBM DB2 for Linux, Unix and Windows Support

Table 4 contains databases and specific database objects and features that SQLWays migrates to IBM DB2 for Linux, Unix and Windows.

For more information about databases versions, see Supported Databases.

TABLE 4. Database Object and Features Support for IBM DB2 for Linux, Unix and Windows

Source	Target - IBM DB2 for Linux, Unix and Windows
IBM DB2 for Linux, Unix and Windows	**Tables** (Default values, Null and Identity properties, Primary and Foreign Keys, Check and Unique constraints, Comments), **Indexes, Views** and **Triggers**
IBM DB2 for z/OS, OS/390 and MVS	**Tables** (Default values, Null and Identity properties, Primary and Foreign Keys, Check and Unique constraints, Comments), **Indexes, Views, Sequences, Stored Procedures, Functions** and **Triggers**
IBM DB2 for iSeries and AS/400	**Tables** (Default values, Null and Identity properties, Check and Unique constraints, Primary and Foreign Keys, Comments)**, Indexes, Views, Sequences, Stored Procedures, Functions** and **Triggers**
Oracle	**Tables** (Default values, Null properties, Primary and Foreign Keys, Check and Unique constraints, Comments)**, Indexes, Views, Sequences, Stored Procedures, Functions** and **Triggers**
Microsoft SQL Server	**Tables** (Default values, Null and Identity properties, Primary and Foreign Keys, Check and Unique constraints)**, Indexes, Views, Stored Procedures, Functions** and **Triggers**
Sybase Adaptive Server Enterprise	**Tables** (Default values, Null and Identity properties , Primary and Foreign Keys, Check and Unique constraints), **Indexes, Views, Stored Procedures, Functions** and **Triggers**
Sybase Adaptive Server Anywhere, Sybase SQL Anywhere	**Tables** (Default values, Null and Identity properties, Primary and Foreign Keys , Check and Unique constraints), **Indexes, Views, Stored Procedures, Functions** and **Triggers**
Informix	**Tables** (Default values, Null and Serial properties , Primary and Foreign Keys, Check and Unique constraints)**, Indexes, Views, Stored Procedures, Functions, Triggers and 4GL Applications** (Logic, Forms and Reports)
MySQL	**Tables** (Default values, Null and Auto-increment properties , Primary and Foreign Keys, Unique constraints)**, Indexes, Views, Stored Procedures, Functions** and **Triggers**
PostgreSQL	**Tables** (Default values, Null properties, Primary and Foreign Keys, Check and Unique constraints), **Indexes**, **Views**, **Sequences**, **Stored Procedures, Functions** and **Triggers**
Progress	**Tables** (Default values, Null properties, Primary and Foreign Keys, Check and Unique constraints), **Indexes, Views, Synonyms, 4GL Code**
Pervasive.SQL	**Tables** (Default values, Null properties, Primary and Foreign Keys)**, Indexes** and **Views**

TABLE 4. **Database Object and Features Support for IBM DB2 for Linux, Unix and Windows**

Source	Target - IBM DB2 for Linux, Unix and Windows
Microsoft Access	**Tables** (Default values, Null and Identity properties , Primary and Foreign Keys) and **Indexes**
Interbase, Firebird	**Tables** (Default values, Null property , Primary and Foreign Keys, Check and Unique constraints), **Indexes,Generators, Views, Stored Procedures, Functions** and **Triggers**
dBase, FoxPro, Excel, Paradox, Gupta SQLBase, Clipper and any other ODBC-compliant database	**Tables** (Default values, Null properties, Primary Keys) and **Indexes**

IBM DB2 for z/OS and OS/390 Support

Table 5 contains databases and specific database objects and features that SQLWays migrates to IBM DB2 for z/OS and OS/390.

For more information about databases versions, see Supported Databases.

TABLE 5. Database Object and Features Support for IBM DB2 for z/OS and OS/390

Source	Target - IBM DB2 for z/OS and OS/390
IBM DB2 for Linux, Unix and Windows	**Tables** (Default values, Null and Identity properties, Primary and Foreign Keys, Check and Unique constraints, Comments)**, Indexes, Views, Sequences, Stored Procedures, Functions** and **Triggers**
IBM DB2 for z/OS, OS/390 and MVS	**Tables** (Default values, Null and Identity properties, Primary and Foreign Keys, Check and Unique constraints, Comments), **Indexes, Views, Sequences, Stored Procedures, Functions** and **Triggers**
IBM DB2 for iSeries and AS/400	**Tables** (Default values, Null and Identity properties, Check and Unique constraints, Primary and Foreign Keys, Comments)**, Indexes, Views, Sequences, Stored Procedures, Functions** and **Triggers**
Oracle	**Tables** (Default values, Null properties, Primary and Foreign Keys, Check and Unique constraints, Comments)**, Indexes, Views, Sequences, Stored Procedures, Functions** and **Triggers**
Microsoft SQL Server	**Tables** (Default values, Null and Identity properties, Primary and Foreign Keys, Check and Unique constraints), **Indexes, Views, Stored Procedures, Functions** and **Triggers**
Sybase Adaptive Server Enterprise	**Tables** (Default values, Null and Identity properties , Primary and Foreign Keys, Check and Unique constraints)**, Indexes, Views, Stored Procedures, Functions** and **Triggers**
Sybase Adaptive Server Anywhere, Sybase SQL Anywhere	**Tables** (Default values, Null and Identity properties, Primary and Foreign Keys , Check and Unique constraints), **Indexes, Views, Stored Procedures, Functions** and **Triggers**
Informix	**Tables** (Default values, Null and Serial properties , Primary and Foreign Keys, Check and Unique constraints)**, Indexes, Views, Stored Procedures, Functions, Triggers and 4GL Applications** (Logic, Forms and Reports)
MySQL	**Tables** (Default values, Null and Auto-increment properties , Primary and Foreign Keys, Unique constraints), **Indexes, Views, Stored Procedures, Functions** and **Triggers**
PostgreSQL	**Tables** (Default values, Null properties, Primary and Foreign Keys, Check and Unique constraints), **Indexes, Views, Sequences, Stored Procedures, Functions** and **Triggers**
Progress	**Tables** (Default values, Null properties, Primary and Foreign Keys, Check and Unique constraints), **Indexes, Views, Synonyms, 4GL Code**
Pervasive.SQL	**Tables** (Default values, Null properties, Primary and Foreign Keys)**, Indexes** and **Views**

TABLE 5. **Database Object and Features Support for IBM DB2 for z/OS and OS/390**

Source	Target - IBM DB2 for z/OS and OS/390
Microsoft Access	**Tables** (Default values, Null and Identity properties , Primary and Foreign Keys) and **Indexes**
Interbase, Firebird	**Tables** (Default values, Null property , Primary and Foreign Keys, Check and Unique constraints), **Indexes,Generators, Views, Stored Procedures, Functions** and **Triggers**
dBase, FoxPro, Excel, Paradox, Gupta SQLBase, Clipper and any other ODBC-compliant database	**Tables** (Default values, Null properties, Primary Keys) and **Indexes**

IBM DB2 for iSeries and AS/400 Support

Table 6 contains databases and specific database objects and features that SQLWays migrates to IBM DB2 for iSeries and AS/400.
For more information about databases versions, see Supported Databases.

TABLE 6. Database Object and Features Support for IBM DB2 for iSeries and AS/400

Source	Target - IBM DB2 for iSeries and AS/400
IBM DB2 for Linux, Unix and Windows	**Tables** (Default values, Null and Identity properties, Primary and Foreign Keys, Check and Unique constraints, Comments)**, Indexes, Views, Sequences, Stored Procedures, Functions** and **Triggers**
IBM DB2 for z/OS, OS/390 and MVS	**Tables** (Default values, Null and Identity properties, Primary and Foreign Keys, Check and Unique constraints, Comments), **Indexes, Views, Sequences, Stored Procedures, Functions** and **Triggers**
IBM DB2 for iSeries and AS/400	**Tables** (Default values, Null and Identity properties, Check and Unique constraints, Primary and Foreign Keys, Comments), **Indexes, Views, Sequences, Stored Procedures, Functions** and **Triggers**
Oracle	**Tables** (Default values, Null properties, Primary and Foreign Keys, Check and Unique constraints, Comments)**, Indexes, Views, Sequences, Stored Procedures, Functions** and **Triggers**
Microsoft SQL Server	**Tables** (Default values, Null and Identity properties, Primary and Foreign Keys, Check and Unique constraints), **Indexes, Views, Stored Procedures, Functions** and **Triggers**
Sybase Adaptive Server Enterprise	**Tables** (Default values, Null and Identity properties , Primary and Foreign Keys, Check and Unique constraints), **Indexes, Views, Stored Procedures, Functions** and **Triggers**
Sybase Adaptive Server Anywhere, Sybase SQL Anywhere	**Tables** (Default values, Null and Identity properties, Primary and Foreign Keys , Check and Unique constraints), **Indexes, Views, Stored Procedures, Functions** and **Triggers**
Informix	**Tables** (Default values, Null and Serial properties , Primary and Foreign Keys, Check and Unique constraints)**, Indexes, Views, Stored Procedures, Functions, Triggers and 4GL Applications** (Logic, Forms and Reports)
MySQL	**Tables** (Default values, Null and Auto-increment properties , Primary and Foreign Keys, Unique constraints), **Indexes, Views, Stored Procedures, Functions** and **Triggers**
PostgreSQL	**Tables** (Default values, Null properties, Primary and Foreign Keys, Check and Unique constraints), **Indexes, Views, Sequences, Stored Procedures, Functions** and **Triggers**
Progress	**Tables** (Default values, Null properties, Primary and Foreign Keys, Check and Unique constraints), **Indexes, Views, Synonyms, 4GL Code**
Pervasive.SQL	**Tables** (Default values, Null properties, Primary and Foreign Keys), **Indexes** and **Views**

TABLE 6. **Database Object and Features Support for IBM DB2 for iSeries and AS/400**

Source	Target - IBM DB2 for iSeries and AS/400
Microsoft Access	**Tables** (Default values, Null and Identity properties , Primary and Foreign Keys) and **Indexes**
Interbase, Firebird	**Tables** (Default values, Null property , Primary and Foreign Keys, Check and Unique constraints), **Indexes,Generators, Views, Stored Procedures, Functions** and **Triggers**
dBase, FoxPro, Excel, Paradox, Interbase, Gupta SQLBase, Clipper and any other ODBC-compliant database	**Tables** (Default values, Null properties, Primary Keys) and **Indexes**

Oracle Support

Table 7 contains databases and specific database objects and features that SQLWays migrates to Oracle. For more information about databases versions, see Supported Databases.

TABLE 7. Database Object and Features Support for Oracle

Source	Target - Oracle
IBM DB2 for Linux, Unix and Windows	**Tables** (Default values, Null and Identity properties, Primary and Foreign Keys, Check and Unique constraints, Comments)**, Indexes, Views, Sequences, Stored Procedures, Functions** and **Triggers**
IBM DB2 for z/OS, OS/390 and MVS	**Tables** (Default values, Null and Identity properties, Primary and Foreign Keys, Check and Unique constraints, Comments), **Indexes, Views, Sequences, Stored Procedures, Functions** and **Triggers**
IBM DB2 for iSeries and AS/400	**Tables** (Default values, Null and Identity properties, Check and Unique constraints, Primary and Foreign Keys, Comments), **Indexes, Views, Sequences, Stored Procedures, Functions** and **Triggers**
Oracle	**Tables** (Default values, Null properties, Primary and Foreign Keys, Check and Unique constraints, Comments), **Indexes, Views, Sequences, Users, Stored Procedures, Functions** and **Triggers**
Microsoft SQL Server	**Tables** (Default values, Null and Identity properties, Primary and Foreign Keys, Check and Unique constraints), **Indexes, Views, Stored Procedures, Functions** and **Triggers**
Sybase Adaptive Server Enterprise	**Tables** (Default values, Null and Identity properties , Primary and Foreign Keys, Check and Unique constraints), **Indexes, Views, Stored Procedures, Functions** and **Triggers**
Sybase Adaptive Server Anywhere, Sybase SQL Anywhere	**Tables (**Default values, Null and Identity properties, Primary and Foreign Keys , Check and Unique constraints), I**ndexes, Views, Stored Procedures, Functions** and **Triggers**
Informix	**Tables** (Default values, Null and Serial properties , Primary and Foreign Keys, Check and Unique constraints), **Indexes, Views, Stored Procedures, Functions, Triggers and 4GL Applications** (Logic, Forms and Reports)
MySQL	**Tables** (Default values, Null and Auto-increment properties , Primary and Foreign Keys, Unique constraints), **Indexes, Views, Stored Procedures, Functions** and **Triggers**
PostgreSQL	**Tables** (Default values, Null properties, Primary and Foreign Keys, Check and Unique constraints), **Indexes, Views, Sequences, Stored Procedures, Functions** and **Triggers**
Progress	**Tables** (Default values, Null properties, Primary and Foreign Keys, Check and Unique constraints), **Indexes, Views, Synonyms, 4GL Code**
Pervasive.SQL	**Tables** (Default values, Null properties, Primary and Foreign Keys), **Indexes** and **Views**

TABLE 7. Database Object and Features Support for Oracle

Microsoft Access	**Tables** (Default values, Null and Identity properties , Primary and Foreign Keys) and **Indexes**
Interbase, Firebird	**Tables** (Default values, Null property , Primary and Foreign Keys, Check and Unique constraints), **Indexes,Generators, Views, Stored Procedures, Functions** and **Triggers**
dBase, FoxPro, Excel, Paradox, Interbase, Gupta SQLBase, Clipper and any other ODBC-compliant database	**Tables** (Default values, Null properties, Primary Keys) and **Indexes**

Microsoft SQL Server Support

Table 8 contains databases and specific database objects and features that SQLWays migrates to Microsoft SQL Server.
For more information about databases versions, see Supported Databases.

TABLE 8. Database Object and Features Support for Microsoft SQL Server

Source	Target - Microsoft SQL Server
IBM DB2 for Linux, Unix and Windows	**Tables** (Default values, Null and Identity properties, Primary and Foreign Keys, Check and Unique constraints, Comments), **Indexes, Views, Sequences, Stored Procedures, Functions** and **Triggers**
IBM DB2 for z/OS, OS/390 and MVS	**Tables** (Default values, Null and Identity properties, Primary and Foreign Keys, Check and Unique constraints, Comments), **Indexes, Views, Sequences, Stored Procedures, Functions** and **Triggers**
IBM DB2 for iSeries and AS/400	**Tables** (Default values, Null and Identity properties, Check and Unique constraints, Primary and Foreign Keys, Comments)**, Indexes, Views, Sequences, Stored Procedures, Functions** and **Triggers**
Oracle	**Tables** (Default values, Null properties, Primary and Foreign Keys, Check and Unique constraints, Comments), **Indexes, Views, Sequences, Users, Stored Procedures, Functions** and **Triggers**
Microsoft SQL Server	**Tables** (Default values, Null and Identity properties, Primary and Foreign Keys, Check and Unique constraints), **Indexes, Views, Stored Procedures, Functions** and **Triggers**
Sybase Adaptive Server Enterprise	**Tables** (Default values, Null and Identity properties , Primary and Foreign Keys, Check and Unique constraints), **Indexes, Views, Stored Procedures, Functions** and **Triggers**
Sybase Adaptive Server Anywhere, Sybase SQL Anywhere	**Tables** (Default values, Null and Identity properties, Primary and Foreign Keys , Check and Unique constraints), **Indexes, Views, Stored Procedures, Functions** and **Triggers**
Informix	**Tables** (Default values, Null and Serial properties , Primary and Foreign Keys, Check and Unique constraints)**, Indexes, Views, Stored Procedures, Functions, Triggers** and **4GL Applications** (Logic, Forms and Reports)
MySQL	**Tables** (Default values, Null and Auto-increment properties , Primary and Foreign Keys, Unique constraints), **Indexes, Views, Stored Procedures, Functions** and **Triggers**
PostgreSQL	**Tables** (Default values, Null properties, Primary and Foreign Keys, Check and Unique constraints), **Indexes, Views, Sequences, Stored Procedures, Functions** and **Triggers**
Progress	**Tables** (Default values, Null properties, Primary and Foreign Keys, Check and Unique constraints), **Indexes, Views, Synonyms, 4GL Code**
Pervasive.SQL	**Tables** (Default values, Null properties, Primary and Foreign Keys)**, Indexes** and **Views**

TABLE 8. Database Object and Features Support for Microsoft SQL Server

Source	Target - Microsoft SQL Server
Microsoft Access	**Tables** (Default values, Null and Identity properties , Primary and Foreign Keys) and **Indexes**
Interbase, Firebird	**Tables** (Default values, Null property , Primary and Foreign Keys, Check and Unique constraints), **Indexes,Generators, Views, Stored Procedures, Functions** and **Triggers**
dBase, FoxPro, Excel, Paradox, Interbase, Gupta SQLBase, Clipper and any other ODBC-compliant database	**Tables** (Default values, Null properties, Primary Keys) and **Indexes**

MySQL Support

Table 9 contains databases and specific database objects and features that SQLWays migrates to MySQL. For more information about databases versions, see Supported Databases.

TABLE 9. Database Object and Features Support for MySQL

Source	Target - MySQL
IBM DB2 for Linux, Unix and Windows	**Tables** (Default values, Null and Identity properties, Primary and Foreign Keys, Check and Unique constraints, Comments), **Indexes, Views, Sequences, Stored Procedures, Functions** and **Triggers**
IBM DB2 for z/OS, OS/390 and MVS	**Tables** (Default values, Null and Identity properties, Primary and Foreign Keys, Check and Unique constraints, Comments), **Indexes, Views, Sequences, Stored Procedures, Functions** and **Triggers**
IBM DB2 for iSeries and AS/400	**Tables** (Default values, Null and Identity properties, Check and Unique constraints, Primary and Foreign Keys, Comments), **Indexes, Views, Sequences, Stored Procedures, Functions** and **Triggers**
Oracle	**Tables** (Default values, Null properties, Primary and Foreign Keys, Check and Unique constraints, Comments), I**ndexes, Views, Sequences, Users, Stored Procedures, Functions** and **Triggers**
Microsoft SQL Server	**Tables** (Default values, Null and Identity properties, Primary and Foreign Keys, Check and Unique constraints), **Indexes, Views, Stored Procedures, Functions** and **Triggers**
Sybase Adaptive Server Enterprise	**Tables** (Default values, Null and Identity properties , Primary and Foreign Keys, Check and Unique constraints), **Indexes, Views, Stored Procedures, Functions** and **Triggers**
Sybase Adaptive Server Anywhere, Sybase SQL Anywhere	**Tables** (Default values, Null and Identity properties, Primary and Foreign Keys , Check and Unique constraints), **Indexes, Views, Stored Procedures, Functions** and **Triggers**
Informix	**Tables** (Default values, Null and Serial properties , Primary and Foreign Keys, Check and Unique constraints), **Indexes, Views, Stored Procedures, Functions, Triggers** and **4GL Applications** (Logic, Forms and Reports)
MySQL	**Tables** (Default values, Null and Auto-increment properties , Primary and Foreign Keys, Unique constraints), **Indexes, Views, Stored Procedures, Functions** and **Triggers**
PostgreSQL	**Tables** (Default values, Null properties, Primary and Foreign Keys, Check and Unique constraints), **Indexes**, **Views**, **Sequences**, **Stored Procedures**, **Functions** and **Triggers**
Progress	**Tables** (Default values, Null properties, Primary and Foreign Keys, Check and Unique constraints), **Indexes**, **Views**, **Synonyms**, **4GL Code**
Pervasive.SQL	**Tables** (Default values, Null properties, Primary and Foreign Keys), **Indexes** and **Views**

TABLE 9. Database Object and Features Support for MySQL

Source	Target - MySQL
Microsoft Access	**Tables** (Default values, Null and Identity properties , Primary and Foreign Keys) and **Indexes**
Interbase, Firebird	**Tables** (Default values, Null property , Primary and Foreign Keys, Check and Unique constraints), **Indexes,Generators, Views, Stored Procedures, Functions** and **Triggers**
dBase, FoxPro, Excel, Paradox, Interbase, Gupta SQLBase, Clipper and any other ODBC-compliant database	**Tables** (Default values, Null properties, Primary Keys) and **Indexes**

PostgreSQL Support

Table 10 contains databases and specific database objects and features that SQLWays migrates to PostgreSQL.

For more information about databases versions, see Supported Databases.

TABLE 10. Database Object and Features Support for PostgreSQL

Source	Target – PostgreSQL
IBM DB2 for Linux, Unix and Windows	**Tables** (Default values, Null and Identity properties, Primary and Foreign Keys, Check and Unique constraints, Comments), **Indexes, Views, Sequences, Stored Procedures, Functions** and **Triggers**
IBM DB2 for z/OS, OS/390 and MVS	**Tables** (Default values, Null and Identity properties, Primary and Foreign Keys, Check and Unique constraints, Comments), **Indexes, Views, Sequences, Stored Procedures, Functions** and **Triggers**
IBM DB2 for iSeries and AS/400	**Tables** (Default values, Null and Identity properties, Check and Unique constraints, Primary and Foreign Keys, Comments), **Indexes, Views, Sequences, Stored Procedures, Functions** and **Triggers**
Oracle	**Tables** (Default values, Null properties, Primary and Foreign Keys, Check and Unique constraints, Comments), **Indexes, Views, Sequences, Users, Stored Procedures, Functions** and **Triggers**
Microsoft SQL Server	**Tables** (Default values, Null and Identity properties, Primary and Foreign Keys, Check and Unique constraints), **Indexes, Views, Stored Procedures, Functions** and **Triggers**
Sybase Adaptive Server Enterprise	**Tables** (Default values, Null and Identity properties , Primary and Foreign Keys, Check and Unique constraints), I**ndexes, Views, Stored Procedures, Functions** and **Triggers**
Sybase Adaptive Server Anywhere (ASA), Sybase SQL Anywhere	**Tables** (Default values, Null and Identity properties, Primary and Foreign Keys , Check and Unique constraints), **Indexes, Views, Stored Procedures, Functions** and **Triggers**
Informix	**Tables** (Default values, Null and Serial properties , Primary and Foreign Keys, Check and Unique constraints), **Indexes**, **Views**, **Stored Procedures, Functions, Triggers** and **4GL Applications** (Logic, Forms and Reports)
MySQL	**Tables** (Default values, Null and Auto-increment properties , Primary and Foreign Keys, Unique constraints), **Indexes**, **Views**, **Stored Procedures**, **Functions** and **Triggers**
PostgreSQL	**Tables** (Default values, Null properties, Primary and Foreign Keys, Check and Unique constraints), **Indexes**, **Views**, **Sequences**, **Stored Procedures**, **Functions** and **Triggers**
Progress	**Tables** (Default values, Null properties, Primary and Foreign Keys, Check and Unique constraints), **Indexes**, **Views**, **Synonyms**, **4GL Code**

TABLE 10. Database Object and Features Support for PostgreSQL

Source	Target – PostgreSQL
Pervasive.SQL	**Tables** (Default values, Null properties, Primary and Foreign Keys), **Indexes** and **Views**
Microsoft Access	**Tables** (Default values, Null and Identity properties , Primary and Foreign Keys) and **Indexes**
Interbase, Firebird	**Tables** (Default values, Null property , Primary and Foreign Keys, Check and Unique constraints), **Indexes,Generators, Views, Stored Procedures, Functions** and **Triggers**
dBase, FoxPro, Excel, Paradox, Interbase, Gupta SQLBase, Clipper and any other ODBC-compliant database	**Tables** (Default values, Null properties , Primary Keys) and **Indexes**

Sybase Adaptive Server Enterprise Support

Table 11 contains databases and specific database objects and features that SQLWays migrates to Sybase Adaptive Server Enterprise.
For more information about databases versions, see Supported Databases.

TABLE 11. Database Object and Features Support for Sybase Adaptive Server Enterprise

Source	Target - Sybase Adaptive Server Enterprise
IBM DB2 for Linux, Unix and Windows	**Tables** (Default values, Null and Identity properties, Check and Unique constraints, Primary and Foreign Keys, Comments), **Indexes**, **Views**, **Stored Procedures** and **Triggers**
IBM DB2 for z/OS, OS/390 and MVS	**Tables** (Default values, Null and Identity properties, Check and Unique constraints, Primary and Foreign Keys, Comments), **Indexes**, **Views**, **Stored Procedures** and **Triggers**
IBM DB2 for iSeries and AS/400	**Tables** (Default values, Null and Identity properties, Check and Unique constraints, Primary and Foreign Keys, Comments), **Indexes**, **Views**, **Stored Procedures** and **Triggers**
Oracle	**Tables** (Default values, Null properties, Check and Unique constraints, Primary and Foreign Keys, Comments), **Indexes**, **Views, Functions**, **Stored Procedures**, **Packages** and **Triggers**
Microsoft SQL Server	**Tables** (Default values, Null and Identity properties, Check and Unique constraints, Primary and Foreign Keys), **Indexes**, **Views, Functions**, **Stored Procedures** and **Triggers**
Sybase Adaptive Server Enterprise	**Tables** (Default values, Null and Identity properties, Check and Unique constraints, Primary and Foreign Keys), **Indexes**, **Views Functions**, **Stored Procedures** and **Triggers**
Sybase Adaptive Server Anywhere, Sybase SQL Anywhere	**Tables** (Default values, Null and Identity properties, Check and Unique constraints, Primary and Foreign Keys), **Indexes**, **Views, Functions**, **Stored Procedures** and **Triggers**
Informix	**Tables** (Default values, Null properties, Check and Unique constraints, Primary and Foreign Keys) and **Indexes**
MySQL	**Tables** (Default values, Null and Identity properties, Primary Keys) and **Indexes**
Pervasive.SQL	**Tables** (Default values, Null properties, Primary and Foreign Keys), **Indexes** and **Views**
Microsoft Access	**Tables** (Default values, Null and Identity properties, Primary Keys) and **Indexes**
Interbase, Firebird	**Tables** (Default values, Null property , Primary and Foreign Keys, Check and Unique constraints), **Indexes,Generators, Views, Stored Procedures, Functions** and **Triggers**
dBase, FoxPro, Excel, Paradox, Interbase, Gupta SQLBase, Clipper and any other ODBC-compliant database	**Tables** (Default values, Null properties, Primary Keys) and **Indexes**

Pervasive.SQL Support

Table 12 contains databases and specific database objects and features that SQLWays migrates to Pervasive.SQL. For more information about supported versions, see Supported Databases.

TABLE 12. Database Object and Features Support for Pervasive.SQL

Source	Target - Pervasive.SQL
IBM DB2 for Linux, Unix and Windows	**Tables** (Default values, Null and Identity properties, Check and Unique constraints, Primary and Foreign Keys, Comments), **Indexes, Views, Stored Procedures** and **Triggers**
IBM DB2 for z/OS, OS/390 and MVS	**Tables** (Default values, Null and Identity properties, Check and Unique constraints, Primary and Foreign Keys, Comments), **Indexes, Views, Stored Procedures** and **Triggers**
IBM DB2 for iSeries and AS/400	**Tables** (Default values, Null and Identity properties, Check and Unique constraints, Primary and Foreign Keys, Comments), **Indexes, Views, Stored Procedures** and **Triggers**
Oracle	**Tables** (Default values, Null properties, Check and Unique constraints, Primary and Foreign Keys, Comments), **Indexes, Views, Stored Procedures** and **Triggers**
Microsoft SQL Server	**Tables** (Default values, Null and Identity properties, Check and Unique constraints, Primary and Foreign Keys), I**ndexes, Views, Stored Procedures** and **Triggers**
Sybase Adaptive Server Enterprise	**Tables** (Default values, Null and Identity properties , Check and Unique constraints , Primary and Foreign Keys), **Indexes, Views, Stored Procedures** and **Triggers**
Sybase Adaptive Server Anywhere, Sybase SQL Anywhere	**Tables** (Default values, Null and Identity properties , Check and Unique constraints , Primary and Foreign Keys), **Indexes, Views, Stored Procedures** and **Triggers**
Informix	**Tables** (Default values, Null properties , Check and Unique constraints , Primary and Foreign Keys), **Indexes, Views, Stored Procedures** and **Triggers**
MySQL	**Tables** (Default values, Null and Identity properties , Primary Keys) and **Indexes**
PostgreSQL	**Tables** (Default values, Null and Identity properties , Primary Keys) and **Indexes**
Progress	**Tables** (Default values, Null and Identity properties , Primary Keys) and **Indexes**
Pervasive.SQL	**Tables** (Default values, Null properties , Primary and Foreign Keys)**, Indexes** and **Views**
Microsoft Access	**Tables** (Default values, Null and Identity properties , Primary Keys) and **Indexes**
Interbase, Firebird	**Tables** (Default values, Null property , Primary and Foreign Keys, Check and Unique constraints), **Indexes,Generators, Views, Stored Procedures, Functions** and **Triggers**
dBase, FoxPro, Excel, Paradox, Interbase, Gupta SQLBase, Clipper and any other ODBC-compliant database	**Tables** (Default values, Null properties, Primary Keys) and **Indexes**

What's New

SQLWays is permanently upgraded by its developers. The major features of the latest version SQLWays 6.0 include:

- **Conversion to HP NonStop SQL/MX**

SQLWays now supports conversion of data, schema and business logic from Oracle database to HP NonStop SQL/MX.

- **Added conversion Embedded SQL in PowerBuilder**

In the new version, SQLWays offers conversion of the embedded SQL in PowerBuilder application.

- **Java to C# migration**

SQLWays introduces basic support for converting classes, methods and functions form Java to C#.

- **Improvement conversion almost for all Databases**

New version of SQLWays offers significant improvements almost for all database conversion.

Getting Started

This title implies a detailed information about installing and starting to use SQLWays. The information is structured into stages covering preparatory steps, installation procedure and putting the product into operation.

- Preparing for Installation
- Installing SQLWays
- Putting SQLWays in Operation

Preparing for Installation

Where to Install

SQLWays is not to be necessarily installed on the same computer with either target or source database, but can be installed on any computer, which meets the software and hardware requirements.

In order to provide access to each database available on the computer that runs SQLWays, you should install client software for the applied databases.

Software Requirements

- Operating System: Windows 98, Windows NT 4.0, Windows ME, Windows 2000, Windows XP, Windows Server 2003, Windows Vista, Windows 7, Windows Server 2008 and Windows Server 2012.
- Database client software.
- ODBC-driver (for source databases only).

Hardware Requirements

TABLE 13. Hardware Requirements

Computer	Intel® or compatible Pentium 166 MHz or higher
Memory	At least 20 MB of free memory (more is recommended)
Hard disk space	At least 20 MB of free space in the installation location
Monitor	VGA or higher resolution 800x600 or higher resolution

Hard disk space

At least 20 MB of free space in the installation location for SQLWays (additional space required for export files, its volume defined by the volume of the information exported from the source database).

Ensure administrator-level rights

You must have administrator rights to install SQLWays on a Windows Operating System.

Verify Compatibility of Vendor Databases

Check the chapter Supported Databases to ensure that the version of SQLWays due to be installed supports the database you are going to migrate.

Installing SQLWays

This chapter contains the information about installing SQLWays. Before the installation, please read the chapter "Preparing for Installation" to ensure you have made all the required preparatory steps.

To install SQLWays on a Windows computer, you should do the following:

1. Unzip SQLWays Install Package to any folder. Launch setup.exe file from this folder.

2. The Welcome screen appears. Click **Next** to proceed with the installation.

3. Review the License Agreement. Click "I Agree" to accept the terms of the agreement.

4. If the same version of SQLWays as that one, which you want to install, is already installed on your computer, you will be offered two options: "Repair" and "Uninstall". "Repair" will enable you to repair the previous installation, e.g. by restoring erroneously deleted program files. "Uninstall" allows you to remove the software from your computer.

Select the operation you want to perform and click **Next** to continue.

5. The **Next** window allows you to choose the installation directory. By default, SQLWays is installed in Program Files.

Choose between the options "Install for all users" and "Just me". By choosing "Install for all users" you opt for creating the program's shortcuts in the **Programs** Menu and on the desktop for all users. After selecting "Just me" the program's shortcuts are created only for the current user installing the software.

6. Specify the **Programs** Menu, in which you would like to create the program's shortcuts. You can also enter a name to create a new folder.

If you select the **Do not create shortcuts** check box, the shortcuts will not be created. In this case the program can be launched directly from the installation directory.

7. Click "Install" to start the installation process.

Putting SQLWays in Operation

When installed, SQLWays can be used either with the help of SQLWays Wizard or the command line tool.

SQLWays Wizard can be launched right after the installation if you select the **Run SQLWays Wizard** check box or afterwards from Start/Program Menu.

In order to use the command line tool, first open the Windows Command Line window (to do that, launch cmd.exe if you work on Windows NT/2000/XP or command.exe if you use Windows 98/ME). Then go to the Installation Directory and launch SQLWays.exe.

For more information on how to use SQLWays Wizard and command line tool, read User's Guide.

Database Migration Concepts

This title implies the concepts of the migration process performed with the help of SQLWays.

- Migration Process in SQLWays
- Comparative Analysis of Databases
- Migrating to IBM DB2
- Migrating to Oracle
- Migrating to Microsoft SQL Server
- Migrating to Sybase
- Migrating to MySQL
- Migrating to Pervasive.SQL
- Tables
- Importing Tables

Migration Process in SQLWays

SQLWays uses an export and import migration process that implies the exportation of the whole source database or certain database objects and their subsequent importation into the target database.

Such migration process provides you with the most flexible way of managing database migration and has a number of advantages over the extraction and simultaneous transfer of information from the source database to the target database. These advantages include:

- Independent Export and Import

Each stage (export and import) can be executed any number of times without the necessity to perform the other stage (e.g. if you need to solely repeat import, you do not have to start again with the export). Thus, both stages and migration process in general can be adjusted and accomplished in the most effective way.

- Supporting Geographically Divided Databases

The migration process used in SQLWays allows you to migrate between geographically divided databases. SQLWays does not require simultaneous connection to the source and target databases, and export files can be transferred to a different location for further import.

- Flexible and Extensible Migration Process

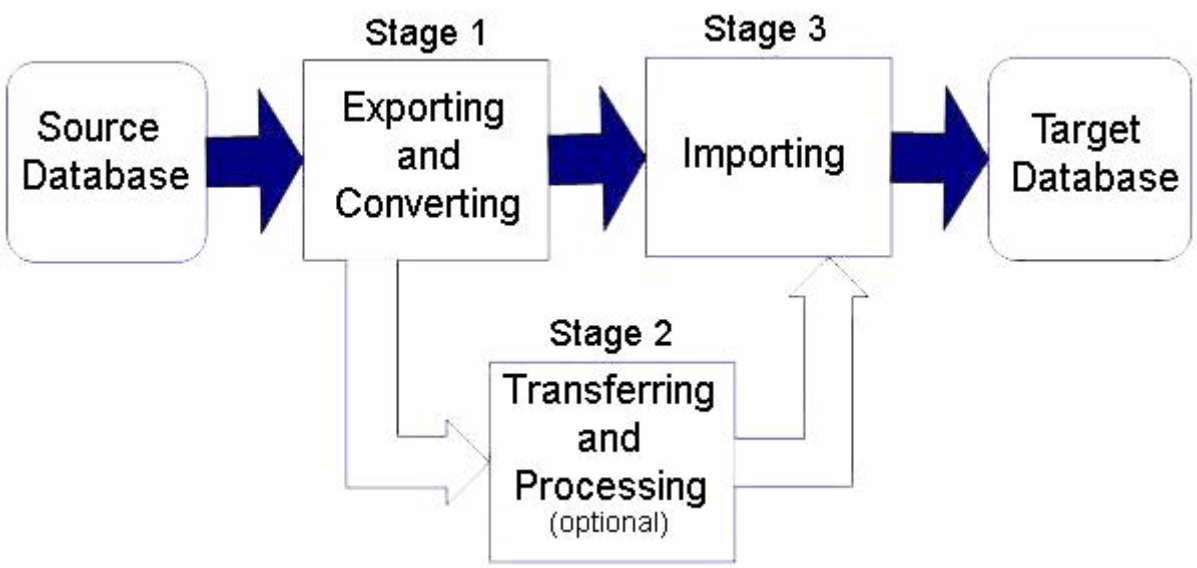
Migration process can be adapted to the needs of each particular project by processing export files with additional tools. SQLWays uses open standard formats for representing database structure (SQL files) and data (ASCII files) in export.

- High Performance

The import of data is executed by SQLWays with the help of high-performance data loading tools supplied with the databases (e.g. Oracle SQL Loader). It helps to dramatically reduce the time of migration, which is particularly important when dealing with large volumes of information.

Stages of Migration Process

Migration process consists of the following stages:



- Stage 1. Exporting and Converting
- Stage 2. Transferring and Processing (optional)
- Stage 3. Importing

Stage 1. Exporting and Converting

At this stage, SQLWays exports and converts the whole source database or certain database objects. In the latter case, you are to choose any of the objects belonging to the following types:

- Tables
- Views
- Stored procedures
- Functions
- Triggers

Having chosen the objects, you can change the name of each table, column, view, stored procedure, function or trigger. If required, you can also redefine the data type mapping between the source and target databases used by default. It can be done by changing the global data type mapping (for all tables) or local data type mapping (solely for selected tables). Besides, you can select the properties of the objects due to be converted.

During the export, SQLWays converts the source database and creates SQL files, which contain SQL statements used for creating database structure in the target database. Also, there are created ASCII files that contain data exported from tables and import scripts used for importing the database.

Export files

The created files depend on the target database.

- Files Created for Importing to IBM DB2
- Files Created for Importing to Oracle
- Files Created for Importing to Microsoft SQL Server
- Files Created for Importing to MySQL

How to perform export and conversion with SQLWays Wizard

Launch SQLWays Wizard. You will see the Welcome page. Click Next and indicate a source database. On the next page, indicate a target database. On the page “Specify Database Objects or Query” indicate the database objects (tables, views etc.) to be exported.

On the next three Wizard pages, you can specify conversion, export and import options. Then click Next to start export. The Wizard will export and convert the source database with the specified options and create import scripts for the specified target database.

For more information about using SQLWays Wizard, see SQLWays Wizard in User’s Guide.

How to perform export and conversion with SQLWays Command Line

Launch Windows Command Line environment (cmd.exe on Windows NT/2000/XP or command.exe on Windows 9x). Type sqlways.exe, set required options and run the command. SQLways.exe will export and convert the source database with the specified options and create import scripts for the specified target database.

For more information about SQLWays Command Line and command options, see Command Line in User’s Guide.

Files Created for Importing to IBM DB2

When the target database is IBM DB2, the following files are created by SQLWays while exporting and converting the source database:

- SQL files (.sql)

SQL files contain SQL statements used for creating database structure (tables, stored procedures etc.) in IBM DB2 using the CLP utility.

- Data files (.txt)

Data files are ASCII files that contain data from tables in the text format. Each table has its own txt file.

- Import scripts (.bat)

Import scripts are command files used for importing the source database into IBM DB2. Import scripts execute the CLP utility for creating database structure and the IBM DB2 IMPORT or LOAD commands for importing data.

Files Created for Importing to Oracle

When the target database is Oracle, the following files are created by SQLWays while exporting and converting the source database:

- SQL files (.sql)

SQL files contain SQL statements used for creating database structure (tables, stored procedures etc.) in Oracle using the Oracle SQL Plus utility.

- Data files (.txt)

Data files are ASCII files that contain data from tables in the text format. Each table has its own txt file.

- Control files (.ctl)

SQLWays uses the Oracle SQL Loader utility to import data from data files (.txt) to Oracle. Control files are used by Oracle SQL Loader to define data file structure (data format, field and row delimeters, import options etc.).

- Import scripts (.bat)

Import scripts are command files used for importing the source database into Oracle. Import scripts execute the SQL Plus utility for creating database structure and the Oracle SQL Loader untility for importing data.

Files Created for Importing to Microsoft SQL Server

When the target database is SQL Server, the following files are created by SQLWays while exporting and converting the source database:

- SQL files (.sql)

SQL files contain SQL statements used for creating database structure (tables, stored procedures etc.) in SQL Server using the SQL Server ISQL utility.

- Data files (.txt)

Data files are ASCII files that contain data from tables in the text format. Each table has its own txt file.

- Format files (.fmt)

SQLWays uses the SQL Server BCP utility to import data from data files (.txt) to SQL Server. Format files are used by BCP to define data file structure (field length, field terminals etc.).

- Import scripts (.bat)

Import scripts are command files used for importing the source database into SQL Server. Import scripts execute the ISQL utility for creating database structure and the BCP untility for importing data.

Files Created for Importing to MySQL

When the target database is MySQL, the following files are created by SQLWays while exporting and converting the source database:

- SQL files (.sql)

SQL files contain SQL statements used for creating database structure (tables, stored procedures etc.) in MySQL using the MySQL command line utility.

- Data files (.txt)

Data files are ASCII files that contain data from tables in the text format. Each table has its own txt file.

- Load command files (.ldi)

SQLWays uses the LOAD DATA INFILE command to import data from data files (.txt) to MySQL. Load command files contain LOAD DATA INFILE commands that are executed by the MySQL command line utility.

- Import scripts (.bat)

Import scripts are command files used for importing the source database into MySQL. Import scripts execute the MySQL command line utility for creating database structure and importing data.

Stage 2. Transferring and Processing (optional)

This stage is optional and is accomplished in specific migration projects requiring the transfer of export files. This may be required while performing migration between geographically divided databases or when it is impossible for some reason to establish simultaneous connection to the source and target databases at the same location.

Although SQLWays is capable of running solely on the Windows platform, it can also generate import scripts for Unix operating systems, such as Linux, Solaris, AIX, HP-UX etc., thus making it possible to perform import directly on Unix servers.

Provided SQLWays represents exported databases as SQL and ASCII files, it is also possible to extend the migration process with data processing, reporting and other 3rd party tools.

Stage 3. Importing

At this stage, the import scripts created at the previous (export) stage are executed and the source database is imported into the target database. Being executed, the import scripts invoke utilities, supplied with the databases for creating the structure of the target database and loading the data into the tables.

The database structure is created with the help of command line utilities intended for the execution of SQL statements, such as:

- SQL Plus for Oracle
- CLP (Command Line Processor) for IBM DB2
- ISQL for Microsoft SQL Server and Sybase
- MySQL command line for MySQL

The utilities used for the loading of data from ASCII files are as follows:

- SQL Loader for Oracle
- LOAD/IMPORT for IBM DB2
- BCP for SQL Server and Sybase
- LOAD DATA INFILE command for MySQL
- BUTIL for Pervasive.SQL

Import scripts are command files (.bat files for Windows or .sh shell scripts for Unix) that are automatically executed by SQLWays Wizard or must be executed manually or from another script while using SQLWays command line.

This is the final stage of the migration process. Upon its completion you will receive the converted database.

How to perform import with SQLWays Wizard

If you chose to start import automatically on the Wizard page “Specify Import Options”, the import will be started immediately after the completion of export. If you did not choose this option, you will have to click the “Start Import” button on the page “Migration Status” to start import. In the latter case, you will be able to examine the export results before the import.

For more information about using SQLWays Wizard, see SQLWays Wizard in User’s Guide.

How to perform import with SQLWays Command Line

Having launched Windows Command Line environment at the export stage, you will have to launch from the export directory “sqlways_all.bat” or “sqlways_imp.exe /bat=sqlways_all.bat” in order to start import. The first command outputs import log messages to the screen only. The second command outputs import log messages to the screen and sqlways_imp.log file.

For more information about SQLWays Command Line and command options, see Command Line in User’s Guide.

Comparative Analysis of Databases

This chapter describes the differences between databases. Databases use various SQL extensions, data types and extended features that are specific for this or that particular database and may not be supported or have different syntax in other databases.

The chapter covers these differences in detail, and shows how SQLWays performs conversion while migrating one database to another.

Identifiers

This section contains information about the identifiers and their use in various databases. The section helps you understand how SQLWays handles identifiers when migrating databases between various database management systems.

- Maximum Length of Identifiers
- Allowed Characters in Identifiers
- Delimited Identifiers

Maximum Length of Identifiers

Table 14 shows the information about the maximum length, in characters, of identifiers in databases.

TABLE 14. Maximum Length of Identifiers

Identifiers	Oracle	IBM DB2			Microsoft SQL Server	Sybase		MySQL	Pervasive.SQL
		Unix, Windows	OS/ 390	AS/400		ASE	ASA		
Table	30	128	18	128	128	30	128	64	20
Column	30	30	18	30	128	30	128	64	20
Primary	30	30	18	18	128	30	128	64	20
Foreign	30	30	18	18	128	30	128	64	20
Check	30	30	18	18	128	30	128	64	N/A
Unique	30	30	18	18	128	30	128	64	20
Index	30	18	18	128	128	30	128	N/A	20
View	30	128	18	128	128	30	128	N/A	20
Procedure	30	128	18	128	128	30	128	N/A	30
Function	30	18	18	128	128	30	128	N/A	N/A
Trigger	30	18	8	128	128	30	128	N/A	30

Allowed Characters in Identifiers

Table 15 shows the information about the characters allowed to use in identifier names in databases.

TABLE 15. Allowed Characters in Identifiers

Database		Identifiers Must Begin with	Subsequent Characters	Delimited Identifiers
Oracle		an alphabetical character	alphanumeric characters, _, $ and #	Supported
IBM DB2	Unix, Windows	an alphabetical character, $, # and @	alphanumeric characters, _, $, # and @	Supported
	OS/390	an alphabetical character	alphanumeric characters and _	Supported
	AS/400	an alphabetical character	alphanumeric characters and _	Supported
Microsoft SQL Server		an alphabetical character, _ , @ and #	alphanumeric characters, _, $, # and @	Supported
Sybase	ASE	an alphabetical character and _	alphanumeric characters, _, $, # and @	Supported
	ASA	an alphabetical character and _	alphanumeric characters, _, $, # and @	Supported
MySQL	prior to 3.23.6	alphanumeric characters, _ and $	alphanumeric characters, _ and $	Not supported
	3.23.6 or later	characters that are allowed in a file name, except / or .	characters that are allowed in a file name, except / or .	Supported
Pervasive.SQL		an alphabetical character	alphanumeric characters and _	Supported

Delimited Identifiers

Delimited identifiers are identifiers which do not need to follow the rules of regular identifiers. Such identifiers can include sequence of printable characters excluding those which are not allowed to use in delimited identifiers in the specified database. Usually delimited identifiers are used when the user need to use SQL reserved word as an identifier.

The rules for using delimited identifiers in various databases are shown in the following table.

TABLE 16. Delimited Identifiers

Database		Rules for Using Delimited Identifiers
Oracle		In Oracle, delimited identifier is a sequence of printable characters enclosed in double quotes. It cannot contain double quotation marks and exceed 30 characters in length not counting the double quotes. It is allowed to use SQL reserved words as delimited identifiers. For example: **SELECT book_id, "TYPE", author INTO...** In contrast to regular identifiers, delimited identifiers in Oracle are case sensitive. That is why you cannot name the column like "TYPE" (as it is shown in the example) and then refer to this column like "Type" or "type".
IBM DB2	Unix, Windows	In IBM DB2, delimited identifier is a sequence of characters enclosed in double quotes. If you need to represent double quotation mark within the delimited identifier you should use two consecutive double quotation marks without a space between them. For example, if there is a column in the table, which has a name **"Last "Name""** , then to refer to this column you must type something like: **"Last ""Name"""** It is allowed to use SQL reserved words as delimited identifiers. In contrast to regular identifiers, delimited identifiers in IBM DB2 are case sensitive.
	OS/390	In IBM DB2 for OS/390, delimited identifier is a sequence of characters enclosed in escape characters. The escape character is a double quotation mark (**"**), except for those cases when the string delimiter is set to double quotation mark. In this case the escape character is apostrophe (**'**). It is allowed to use SQL reserved words as delimited identifiers. If you need to represent escape character within the delimited identifier you should use two consecutive escape characters without a space between them. The necessary shift characters must be present if there are double byte characters in a delimited identifier.
	AS/400	In IBM DB2 for AS/400, delimited identifier is a sequence of characters enclosed in escape characters. The length of a delimited identifier includes the two escape characters only for column names. The escape character is a double quotation mark (**"**), except for those cases when the string delimiter is set to double quotation mark. In this case the escape character is apostrophe (**'**). To use SQL reserved words as delimited identifiers, you should specify them in uppercase. You cannot use the following characters within delimited identifiers: **X'00'** through **X'3F'** and **X'FF'**
Microsoft SQL Server		In Microsoft SQL Server delimited identifier is a sequence of characters enclosed in double quotation marks (**"**) as default. When **quoted_identifier** option is set **off**, only brackets (**[]**) are used to delimit identifiers and double quotation marks can be used to delimit character strings. It is allowed to use SQL reserved words as delimited identifiers. The length of the delimited identifier cannot exceed 128 characters not counting the delimiter characters. The body of the identifier can contain any combination of characters in the current code page except for the delimiting characters themselves.

TABLE 16. **Delimited Identifiers**

Database		Rules for Using Delimited Identifiers
Sybase	ASE	In Sybase Adaptive Server Enterprise, delimited identifier is a 28 characters long identifier enclosed in double quotes, which denotes table, view or column. You cannot use it to identify other objects in the database. Delimited identifier can be a reserved word, can begin with a number or with another non-alphabetical character and can include those characters, which are not allowed to use in regular identifiers. Before creating or referencing a delimited identifier you must execute: **set quoted_identifier on** Characters or data strings cannot be enclosed in double quotes, while the **quoted_identifier** option is turned **on** .
	ASA	The same as for the ASE. The only difference is that in Adaptive Server Anywhere the **quoted_identifier** option is set to **on** as a default and the maximum length for the delimited identifier is 126 characters.
MySQL	prior to 3.23.6	Not supported
	3.23.6 or later	In MySQL delimited identifier is an identifier enclosed in back ticks (`) by default. When MySQL is running in ANSI double quotes (**"**) will also work to quote identifiers. Delimited identifier cannot contain **ASCII(0)**, **ASCII(255)** or the quoting character. If the delimited identifier is a reserved word or contains special characters, you must always quote it with a ` (back tick). For example: **SELECT * FROM `types` WHERE `types`.id < 10;**
Pervasive.SQL		In Pervasive.SQL, delimited identifier is an identifier enclosed in double quotes, which can contain any combination of characters and can contain reserved words. If you need to represent double quotes within the delimited identifier, you should use a pair of double quotes without a space between them. The length is the same as for the regular identifier, counting double quotes. For example: **Column_name123456789** - (regular identifier, 20 characters) **"Column_name1234567"** - (delimited identifier, 20 characters)

Expressions and Statements

This section describes the conversion of SQL statements and expressions between different databases by SQLWays.

- Variable Declaration
- Assignment Statements
- Conditional Expressions
- SELECT Statement
- Executing Procedures and User-Defined Functions
- Executing Dynamic SQL Statements with Parameters
- Cursors
- Transaction Control

Variable Declaration

This subsection describes declaration of local variables in various databases and their conversion by SQLWays.

TABLE 17. Declaration of Local Variables

Database	Syntax	Description
Oracle	*var1 datatype* [{:= | **DEFAULT** } *exp1*]	The DECLARE statement is used for declare variables in the declarative part of any PL/SQL block or subprogram. Declarations allocate storage space for a value, specify its data type, and name the storage location so that maybe reference it. The keyword DEFAULT maybe used instead of the assignment operator to initialize variables.

	var1 table_name%ROWTYPE	The %ROWTYPE attribute provides a record type that represents a row in a database table. Fields in a record and corresponding columns in a row have the same names and data types.
		
		The %ROWTYPE attribute may be used in variable declarations as a data type specifier. Variables declared using %ROWTYPE are treated like those declared using a data type name.
		
		table_name – this identifies a database table (or view) that must be accessible when the declaration is elaborated.
		
		The %ROWTYPE attribute lets declare records structured like a row of data in a database table. To reference a field in the record, you use dot notation. For example,you might reference the deptno field as follows:
		IF emp_rec.deptno = 20 THEN ...
		The value of an expression may be assign to a specific field, as follows:
		emp_rec.sal := average * 1.15;
		
		Examples:
		In the example below, %ROWTYPE is used to store a row selected from the emp table:
		emp_rec emp%ROWTYPE;
		In the next example, you select a row from the emp table into a %ROWTYPE record:
		DECLARE
		emp_rec emp%ROWTYPE;
		...
		BEGIN
		SELECT * INTO emp_rec FROM emp WHERE empno = my_empno;
		IF (emp_rec.deptno = 20) AND (emp_rec.sal > 2000) THEN
		...
		END IF;
		END;

TABLE 17. Declaration of Local Variables

Database	Syntax	Description
	var1 variable_name | column_name %TYPE	The %TYPE attribute is used for defining data types of variables. Variables, declared with %TYPE attribute get type identical with variable or column, which located before %TYPE. *variable_name* – data type of this variable is used for the declared variable var1. *column_name* – data type of this column is used for the declared variable *var1*. column_name is a compound clause that must contain table or view name, where the column is defined. **Examples**: In the following example *var2* is declared through var1 with data type NUMBER: var1 number; var2 var1%TYPE; In the sample below var2 is declared through column col1 of the table tab1: var2 tab1.col1%TYPE;
MySQL	**DECLARE** *var1* [, *varN*]... *datatype* [**DEFAULT** *exp1*]	The DECLARE statement is used to declare local variables. DECLARE may only be used inside a BEGIN ... END compound statement and must be at its start, before any other statements. The scope of a variable is within the BEGIN ... END block.

TABLE 17. Declaration of Local Variables

Database	Syntax	Description
Microsoft SQL Server	**DECLARE** *@var1* [AS] *datatype* [, *@varN* [AS] *datatype*]	The DECLARE statement is used to declare variables anywhere within the procedure body, before their usage. After declaration all variables are initialized as NULL.
Informix	DEFINE var1 [, varN] datatype	The DEFINE statement is used to declare local variables. DEFINE may only be used inside a stored procedure and must be at the beginning of statements block, before any other statement. ***Datatype*** can be any datatype except for the SERIAL, SERIAL8, BYTE or TEXT.
	DEFINE var1 [, varN] LIKE {table|synonym|view}.column	The DEFINE statement with the LIKE clause is used to declare local variables as table columns declared. ***Column*** is any column existing in the table or view. **Remarks**: If ***column*** has SERIAL or SERIAL8 datatype, it is declared as INT or INT8 variable.

Conversion of Variable Declarations from Microsoft SQL Server to MySQL

The DECLARE statement is used to declare variables in Microsoft SQL Server and allows declaring variables with different data types. The MySQL DECLARE statement allows declaring a list of variables with one data type only, which is specified at the end of the variable list.

Microsoft SQL Server local variable name (***@var***) is changed to MySQL local variable name with the only difference that symbol "@" (which in Microsoft SQL Server designates local variable) is changed to MySQL "v_". In MySQL symbol "@" is used to designate user variable. User variable means that variable is working during the whole connection to the database. In our case we must used local variable, which works only within procedure body.

SQLWays used "v_" before local variable name for discern between local variable and column name. For example:

TABLE 18. Example, where column has the same name as variable (with "v_"):

Microsoft SQL Server	MySQL
select col1 from test where c=@c	select col1 from test where c=v_c

TABLE 19. Example, where column has the same name as variable (without "v_"):

Microsoft SQL Server	MySQL
select col1 from test where c=@c	select col1 from test where c=c

SQLWays converts the Microsoft SQL Server DECLARE statement with a list of variables to a standalone DECLARE statement for each variable in MySQL

TABLE 20. Example of variable declaration conversion

Microsoft SQL Server	MySQL

Create procedure mssql_declare_mysql
As
Begin
DECLARE @x INTEGER, @y CHAR
end
	Create procedure mssql_declare_mysql( )
	Begin
	DECLARE v_x INT;
	DECLARE v_y CHAR;
	end ;

Conversion of the Informix variable declarations to Oracle

The Informix DEFINE statement allows declaring a list of variables with one data type, which is specified at the end of the variable list. In Oracle you can declare only one variable with the appropriate datatype.

SQLWays converts the Informix DEFINE statement with a list of variables to a standalone declaration statement for each variable in Oracle.

SQLWays converts Informix LIKE to the Oracle %TYPE attribute.

TABLE 21. Example of variable declaration conversion

Informix	Oracle

create procedure with_multidef_stmt (var1 int)
define a, b, c int;
define d, e, f char(20);
let a = 10;
end procedure;
	create or replace procedure with_multidef_stmt (var1 int)
	as
	a int;
	b int;
	c int;
	d char(20);
	e char(20);
	f char(20);
	begin
	set a := 10;
	end;

create procedure with_LIKE (var1 int)
define i LIKE tab.c1;
define j,k LIKE tab.c2;
let i = 20;
end procedure;
	CREATE OR REPLACE PROCEDURE with_LIKE(var1 NUMBER)
	AS
	i tab.c1 %TYPE;
	j tab.c2 %TYPE;
	k tab.c2 %TYPE;
	BEGIN
	i := 20;
	end;

create procedure with_LIKE1 (var1 LIKE tab.c1)
define i LIKE tab.c1;
define j,k LIKE tab.c2;
let i = 20;
end procedure;
	CREATE OR REPLACE PROCEDURE with_LIKE(var1 tab.c1 %TYPE)
	AS
	i tab.c1 %TYPE;
	j tab.c2 %TYPE;
	k tab.c2 %TYPE;
	BEGIN
	i := 20;
	end;

Conversion of Oracle %ROWTYPE to MySQL

The Oracle %ROWTYPE attribute provides a record type that represents a row in a database table. If you use %ROWTYPE and any column's type changes or the number of columns in the table is changed, your application code remains correct and uses the latest column and type information. This provides data independence from applications, and allows reducing application maintenance code.

MySQL has no an equivalent of Oracle %ROWTYPE. In MySQL it is necessary to declare a variable for every column.

SQLWays changes Oracle variable declaration with the %ROWTYPE attribute to the MySQL variable declaration list with the same names and data types as column's names and data types in the database table.

If there is variable with reference to a field in the record in the procedure body, SQLWays changes it to appropriate variable. If there is an assignment statement with variable using %ROWTYPE in the procedure body, SQLWays changes the assignment statement to the list of assignment statements for each variable.

Examples of the conversion:

Given the table ***ora_rt*** contains two columns: ***ID*** of ***NUMBER***(10) and ***Name*** of ***VARCHAR2***(10).

TABLE 22. Using %ROWTYPE without references to the record fields

Oracle	MySQL

CREATE PROCEDURE ORA_SP_ROWTYPE (ROW1 OUT ora_rt%ROWTYPE) IS
BEGIN
SELECT ID, Name
INTO ROW1
FROM ora_rt
WHERE ID = 1;
END;
	CREATE PROCEDURE ORA_SP_ROWTYPE (OUT SWV_ROW1_ID INT, OUT SWV_ROW1_NAME VARCHAR(10))
	BEGIN
	SELECT ID,Name
	INTO SWV_ROW1_ID,SWV_ROW1_NAME
	FROM ora_rt WHERE ID = 1;
	END;/

Remarks: in the table ***ora_rt*** there are two columns: ***ID*** of ***NUMBER(10)*** and ***Name*** of ***VARCHAR2(10)***.

TABLE 23. Using %ROWTYPE with reference to a field in the record.

Oracle	MySQL

CREATE PROCEDURE ora_sp_rowtype3 IS
ROW1 ora_rt%ROWTYPE;
BEGIN
ROW1.ID := 5;
ROW1.Name := 'Tenth';
INSERT INTO ora_rt values (ROW1.ID, ROW1.Name);
END;
	CREATE PROCEDURE ORA_SP_ROWTYPE3()
	BEGIN
	declare SWV_ROW1_ID INT;
	declare SWV_ROW1_NAME VARCHAR(10);
	SET SWV_ROW1_ID = 5;
	SET SWV_ROW1_Name = 'Tenth';
	INSERT INTO ora_rt values(SWV_ROW1_ID,SWV_ROW1_Name);
	END;/

TABLE 24. Using %ROWTYPE with the record assignment.

Oracle	MySQL

CREATE PROCEDURE ora_sp_rowtype2 IS
ROW1 ora_rt%ROWTYPE;
ROW2 ora_rt %ROWTYPE;
BEGIN
SELECT ID, Name
INTO ROW1
FROM ora_rt
WHERE ID = 1;
ROW2 := ROW1;
END;
	REATE PROCEDURE ORA_SP_ROWTYPE2()
	BEGIN
	declare SWV_ROW1_ID INT;
	declare SWV_ROW1_NAME VARCHAR(10);
	declare SWV_ROW2_ID INT;
	declare SWV_ROW2_NAME VARCHAR(10);
	SELECT ID,Name
	INTO SWV_ROW1_ID,SWV_ROW1_NAME
	FROM ora_rt WHERE ID = 1;
	SET SWV_ROW2_ID = SWV_ROW1_ID;
	SET SWV_ROW2_NAME = SWV_ROW1_NAME;
	END;/

Conversion of Oracle %TYPE to Microsoft SQL Server

The Oracle %TYPE attribute adapts code as table definitions change. The %TYPE attribute provides the datatype of a variable or database column. If the column's type changes, your variable uses the correct type at run time. This provides data independence and reduces maintenance costs.

Microsoft SQL Server does not support and has no an equivalent of Oracle %TYPE.

SQLWays changes Oracle variable declaration with the %TYPE attribute to the Microsoft SQL Server variable declaration with the same data type as column's data type in the database table or as variable's data type.

Examples of the conversion:

TABLE 25. Using table_name.column_name%TYPE

Oracle	Microsoft SQL Server

CREATE PROCEDURE ORA_SP_TYPE IS
v_name ora.ora_rt.Name%TYPE;
BEGIN
SELECT Name
INTO v_name
FROM ora_rt
WHERE ID = 1;
END;
	CREATE PROCEDURE ORA.ORA_SP_TYPE AS
	BEGIN
	DECLARE @v_name VARCHAR(10)
	select @v_name = Name FROM ora_rt WHERE ID = 1
	END

Remarks: in the table ***ora_rt*** there is column: ***Name*** of ***VARCHAR2(10)***.

TABLE 26. Using variable%TYPE

Oracle	Microsoft SQL Server

CREATE procedure ora_sp_type1
IS
v_name1 varchar(10);
v_name v_name1%TYPE;
BEGIN
SELECT Name
INTO v_name
FROM ora_rt
WHERE ID = 1;
END;
	CREATE procedure ORA.ORA_SP_TYPE1
	AS
	BEGIN
	DECLARE @v_name1 VARCHAR(10)
	DECLARE @v_name VARCHAR(10)
	select @v_name = Name FROM ora_rt WHERE ID = 1
	END

Declaration of Local Variables with Composite Data Types

In this chapter describes declaration of local variable with composite (no scalar) data types. Scalar data type - is data type which hasn't internal components, such as numeric data types (NUMBER, DECIMAL, FLOAT,...), character data types (CHAR, VARCHAR2, RAW,...), datetime data types (DATE, TIMESTAMP,...) and boolean data type. Composite data type - is data type which has internal components, such as RECORD, TABLE and VARRAY.

Before declare variables with composite data type necessary define such data type.

TABLE 27. Declaration of Local Variables with Composite Data Types

Database	Statement	Description

Oracle	TYPE *type_name* IS RECORD (
	{
	field_name field_type
	[
	[NOT NULL]
	:= | DEFAULT expression
	]
	[,]
	}
	);
		Using this syntax, in Oracle defined RECORD type. Record is a group of related data items stored in fields, each with its own name and datatype. Field declaration is similar with local variable declaration.
		RECORD type can contain objects, collections, and other records.
		
		type_name – name of RECORD type.
		field_name – field name in record.
		field_type – field type in record.
		
		Example: Next sample defined RECORD type R1 with fields a (number) and b (varchar2(10)):
		
		TYPE R1 IS RECORD (
		a NUMBER, b VARCHAR2(10));

	var_name type_name;	If type is RECORD data type, then Oracle declared record variable.
Microsoft SQL Server	Not supported	

Conversion of Oracle RECORD Variable to Microsoft SQL Server

Since Microsoft SQL Server doesn't support records as type, SQLWays emulates such functionality through use set of distinct variables. For full substitution record variables SQLWays changes definition of record data type, variable declaration with record type, use record variables and use of internal element of record variables.

At first SQLWays converts declaration of variable with record type to set of distinct declaration with corresponding data types. Number and data types of declaring variables are equal with internal element of sources record variable.

Then SQLWays drops definition of record data type.

After that SQLWays looks for use of record variables and converts them as follow:

- If record variable uses without references to internal element then SQLWays converts it's to list of variables which generate by SQLWays for this record variable.
- If record variable uses with references to internal element then SQLWays converts its to corresponding variable which generates by SQLWays.

TABLE 28. Conversion of Oracle RECORD Variable to Microsoft SQL Server

Oracle	Microsoft SQL Server

CREATE OR REPLACE PROCEDURE
ora_sp_record_type as
begin
DECLARE
TYPE RecType IS RECORD(
a number,
b varchar2(15));
RecVar RecType;
a number;
b varchar2(15);
CURSOR cursoitem IS SELECT a, b
FROM tab1;
BEGIN
OPEN cursoitem;
LOOP
FETCH cursoitem INTO **RecVar**;
EXIT WHEN cursoitem%NOTFOUND;
BEGIN
SELECT a, b INTO a,b FROM tabl
WHERE c = **RecVar.a;**
END;
END LOOP;
CLOSE cursoitem;
COMMIT;
END;
end;
	CREATE PROCEDURE ORA_SP_RECORD_TYPE AS
	begin
	DECLARE @a FLOAT
	DECLARE @b VARCHAR(15)
	DECLARE cursoitem CURSOR FOR SELECT @a, @b
	FROM tab1
	DECLARE @SWV_RecVar_a
	FLOAT
	DECLARE @SWV_RecVar_b
	VARCHAR(15)
	OPEN cursoitem
	while 1 = 1
	begin
	FETCH cursoitem INTO **@SWV_RecVar_a, @SWV_RecVar_b**
	if @@FETCH_STATUS <> 0
	break
	BEGIN
	select @a = a, @b = b FROM tabl WHERE a = **@SWV_RecVar_a**
	
	END
	end
	CLOSE cursoitem
	DEALLOCATE cursoitem
	COMMIT
	end

Assignment Statements

Assignment statements are used to assign values to variables. Different databases support various variants of assignment statements.

TABLE 29. Various variants of assign statements

Databases	Syntax	Description
Microsoft SQL Server	**SELECT** *@local_variable* = *expression* { ,*@local_variable* = *expression* } **SET** *@local_variable* = *expression*	Sets the specified local variable, previously created with the DECLARE *@local_variable* statement, to the given value. Expression is any valid expression including a scalar subquery.
Oracle	*local_variable* := *expression*	Assignment operator assigns a value to a variable. Expression is any valid expression. You cannot specify a subquery in the assignment operator.

Conversion of Assignment Statement from Microsoft SQL Server to Oracle

SQL Server SELECT | SET @local_variable assignment statements allow specifying a subquery to assign a value to a variable, while the Oracle assignment operator (:=) does not allow specifying a subquery.

a) **Expression is any expression, except a scalar subquery.**

In this case SQLWays converts the SELECT and SET clause to the Oracle assignment operator (:=).

Examples:

TABLE 30. Expression is any expression, except a scalar subquery

Microsoft SQL Server	Oracle
SELECT @A=5+7	v_A:=5+7;
SET @B='String'	v_B:='String';

b) **Expression is a scalar subquery.**

If SQL Server expression is a scalar subquery, SQLWays converts the SELECT and SET clause to the Oracle SELECT INTO statement that allows assigning a SQL query result to a variable.

Examples:

TABLE 31. Expression is a scalar subquery

Microsoft SQL Server	Oracle
SELECT @D = (SELECT col1 FROM tab1)	**SELECT** col1 INTO v_D FROM tab1;
SELECT @C = (SELECT col2 FROM tab2)	**SELECT** col2 INTO v_C FROM tab2;

c) **SELECT | SET containing several assignment clauses.**

The SQL Server SELECT @local_variable assignment statement can contain several assignments while Oracle allows only one assignment. If SQL Server SELECT contains multiple assignments, SQLWays converts them to multiple assignment operators in Oracle.

Examples:

TABLE 32. SELECT | SET which contains several assign clauses

Microsoft SQL Server	Oracle
SELECT @E = (SELECT col3 FROM tab3), @G = 9.8	**SELECT** col3 INTO v_E FROM tab3; v_G:=9.8;
SELECT @F= (SELECT col4 FROM tab4), @H= (SELECT col5 FROM tab5)	**SELECT** col4 INTO v_F FROM tab4; **SELECT** col5 INTO v_H FROM tab5;
SELECT @K = 'Test string', @L = 10	v_K:= 'Test string'; v_L:=10;

Conditional Expressions

This subsection describes functions which compare an expression to each search value one by one and return the corresponding result, in various databases and their conversion by SQLWays.

TABLE 33. Conditional Expressions

Database	Syntax	Description
Oracle	**DECODE** (*comp_exp*, *search_exp1*, *result_exp1* [, *search_expN*, *result_expN*]… [, *default_exp*])	Compares expression (***comp_exp***) to each search value (***search_exp1***, …, ***search_expN***) one by one and if expression (***comp_exp***) is equal to search value then returns the corresponding result (***result_exp1***, …, ***result_expN***), if no match is found then returns default (***default_exp***) or if default is omitted then returns NULL. ***comp_exp***, **search_exp** and ***result_exp*** can be any of the datatypes CHAR, VARCHAR2, NCHAR, or NVARCHAR2. The string returned is of VARCHAR2 data type and is in the same character set as the first result parameter. The **search_exp**, ***result_exp*** and ***default_exp*** values can be derived from expressions. The DECODE function automatically converts ***comp_exp*** and each search value (**search_exp**) to the data type of the first search value before comparing and also automatically converts the return value to the same data type as the first result (***result_exp***). If the first result has the CHAR data type or if the first result is null, then the function converts the return value to the VARCHAR data type. The DECODE function considers two nulls to be equivalent. If ***comp_exp*** is null, then DECODE returns the result of the first search that is also null. The maximum number of components in the DECODE function, including ***comp_exp***, searches (***search_exp1***, …, ***search_expN***), results (***result_exp1***, …, ***result_expN***), and default (***default_exp***), is 255.
		Example
		DECODE (emp_type, 1, ‘clerk’, 2, ‘book-keeper’, ‘Unknown’)

TABLE 33. Conditional Expressions

Database	Syntax	Description
	CASE *comp_exp* WHEN *search_exp1* THEN *result_exp1* [WHEN *search_expN* THEN *result_expN*]… [ELSE *default_exp*] END	Compares expression (***comp_exp***) to each search value (***search_exp1***, …, ***search_expN***) one by one and if expression (***comp_exp***) is equal to search value then returns the corresponding result (***result_exp1***, …, ***result_expN***), if no match is found then returns default (***default_exp***) or if default is omitted then returns NULL. All of the expressions (***comp_exp***, ***search_exp*** and ***result_exp***) must be of CHAR, VARCHAR2, NCHAR, or NVARCHAR2 data type.
		Example:
		CASE emp_type WHEN 1 THEN ‘clerk’ WHEN 2 THEN ‘book-keeper’ ELSE ‘Unknown’ END
	CASE WHEN *condition_exp1* THEN *result_exp1* [WHEN *condition_expN* THEN *result_expN*]… [ELSE *default_exp*] END	Searches from left to right until it finds an occurrence of condition (***condition_exp***) that is true, and then returns ***result_exp***. If no condition is found to be true then returns default (***default_exp***) or if default is omitted then returns NULL.
		Example:
		CASE WHEN salary>500 THEN salary WHEN salary<200 THEN 200 ELSE 500 END
MySQL	**CASE** *comp_exp* WHEN *search_exp1* THEN *result_exp1* [WHEN *search_expN* THEN *result_expN*]… [ELSE *default_exp*] END	Compares expression (***comp_exp***) to each search value (***search_exp1***, …, ***search_expN***) one by one and if expression (***comp_exp***) is equal to search value then returns the corresponding result (***result_exp1***, …, ***result_expN***), if no match is found then returns default (***default_exp***) or if default is omitted then returns NULL. The type of the **return value** is the same as the type of the first returned value (the expression after the first THEN).
		Example:

TABLE 33. Conditional Expressions

Database	Syntax	Description
		CASE workgroup WHEN A THEN 'Administration' WHEN B THEN 'Book-keeping' WHEN C THEN 'Customer department' ELSE 'Others' END
	CASE WHEN *condition_exp1* THEN *result_exp1* [WHEN *condition_expN* THEN *result_expN*]… [ELSE *default_exp*] END	Searches from left to right until it finds an occurrence of condition (***condition_exp***) that is true, and then returns ***result_exp***. If no condition is found to be true then returns default (***default_exp***) or if default is omitted then returns NULL. The type of the **return value** is the same as the type of the first returned value (the expression after the first THEN).
		Example:
		CASE WHEN salary between 200 and 500 THEN 500 WHEN salary<200 THEN 200 ELSE salary END
Microsoft SQL Server	**CASE** *comp_exp* WHEN *search_exp1* THEN *result_exp1* [WHEN *search_expN* THEN *result_expN*]… [ELSE *default_exp*] END	Compares expression (***comp_exp***) to each search value (***search_exp1***, …, ***search_expN***) one by one and if expression (***comp_exp***) is equal to search value then returns the corresponding result (***result_exp1***, …, ***result_expN***), if no match is found then returns default (***default_exp***) or if default is omitted then returns NULL. **Return types**: the highest precedence type from the set of types in ***result_exp*** and the optional ***default_exp***
		Example:
		CASE level WHEN 1 THEN 'First level' WHEN 2 THEN 'Second level' WHEN 3 THEN 'Third level' ELSE 'Top level' END

TABLE 33. Conditional Expressions

Database	Syntax	Description
	CASE WHEN *condition_exp1* THEN *result_exp1* [WHEN *condition_expN* THEN *result_expN*]… [ELSE *default_exp*] END	Searches from left to right until it finds an occurrence of condition (***condition_exp***) that is true, and then returns ***result_exp***. If no condition is found to be true then returning default (***default_exp***) or if default is omitted then returning NULL. **Return types**: the highest precedence type from the set of types in ***result_exp*** and the optional ***default_exp***.
		Example:
		CASE WHEN price IS NULL THEN 'Not yet priced' WHEN price < 10 THEN 'Very Reasonable Title' WHEN price >= 10 and price < 20 THEN 'Coffee Table Title' ELSE 'Expensive book!' END
IBM DB2	**CASE** *comp_exp* WHEN *search_exp1* THEN {*result_exp1* | NULL} [WHEN *search_expN* THEN {*result_expN* | NULL}]… [ELSE *default_exp*] END	Compares expression (***comp_exp***) to each search value (***search_exp1***, …, ***search_expN***) one by one and if expression (***comp_exp***) is equal to search value then returns the corresponding result (***result_exp1***, …, ***result_expN***), if no match is found then returns default (***default_exp***) or if default is omitted then returns NULL.
		Example:
		CASE credit_limit WHEN 100 THEN 'Low' WHEN 5000 THEN 'High' ELSE 'Medium' END
	CASE WHEN *condition_exp1* THEN {*result_exp1* | NULL} [WHEN *condition_expN* THEN {*result_expN* | NULL}]… [ELSE *default_exp*] END	Searches from left to right until it finds an occurrence of condition (***condition_exp***) that is true, and then returns ***result_exp***. If no condition is found to be true then returns default (***default_exp***) or if default is omitted then returns NULL.
		Example:
		CASE WHEN1>0 THEN 'true' ELSE 'false' END

Conversion of Oracle DECODE to MySQL CASE

The Oracle DECODE function is used to compare the specified expression to each search value one by one and return the corresponding result.

MySQL also supports the DECODE function, but it is used to decode the encrypted strings using the specified password. The MySQL CASE expression is an equivalent of the Oracle DECODE function.

SQLWays converts the Oracle DECODE function to the MySQL CASE expression.

Example:

TABLE 34. Conversion of Oracle DECODE to MySQL CASE

Oracle	MySQL

create procedure ora_sp_decode2 (job_level out varchar2) as
begin
select **DECODE**(job_lvl,1,'level 1',2,'level 2','Unknown level') into job_level from employee where job_id>0;
end;
	create procedure ora_sp_decode2 (out job_level TEXT)
	begin
	select **CASE** job_lvl
	WHEN 1 THEN 'level 1
	WHEN 2 THEN 'level 2'
	ELSE 'Unknown level' end
	into job_level from employee where job_id>0 ;
	end ;

SELECT Statement

This section describes SQL SELECT statement syntax in various databases and its conversion by SQLWays.

- Restricting Number of Rows in Result Set

Restricting Number of Rows in Result Set

This subsection describes ways for restricting result sets in various databases and their conversion by SQLWays.

TABLE 35. Restricting Number of Rows in Result Set

Database	Clause	Descriptions
Microsoft SQL Server	**TOP** *n* [**PERCENT**]	The TOP clause of the SELECT statement limits the number of rows returned in the result set. If PERCENT is not specified, n is the number of rows to return. If PERCENT is specified, n is the percentage of the result set rows to return. In this case n must be an integer between 0 and 100. If a SELECT statement that includes TOP also has an ORDER BY clause, the rows to be returned are selected from the ordered result set. **Example**: The following example retrieves only first 7 rows from the query: SELECT TOP 7 col1 FROM tab1
Oracle	ROWNUM	The ROWNUM pseudo column returns a number starting from 1 which indicates the order in which the row was selected in the result set. ROWNUM can be used to limit the number of rows in the result set. If a SELECT statement contains an ORDER BY clause, ROWNUM is assigned before the sort is done. **Example**: The following example retrieves only first 7 rows from the query: SELECT col1 FROM tab1 WHERE ROWNUM<=7;

Conversion of Microsoft SQL Server TOP clause to Oracle

The Microsoft SQL Server TOP clause limits the number of rows returned by a SELECT statement. TOP allows specifying the number or percentage of rows to return. If the SELECT statement that includes TOP also has an ORDER BY clause, the rows to be returned are selected from the ordered result set.

The Oracle ROWNUM pseudocolumn returns a number indicating the order in which the row was selected in the result set. ROWNUM can be used to limit rows in Oracle, but ROWNUM is assigned before ordering.

SQLWays converts the Microsoft SQL Server TOP clause to Oracle as follows:

a)**SELECT statement with TOP does not contain the ORDER BY clause.**

If Microsoft SQL Server does not contain the ORDER BY clause, SQLWays implements the TOP clause using the ROWNUM pseudo column in the WHERE clause of the SELECT statement in Oracle.

If TOP is specified with the PERCENT clause SQLWays calculates the total number of rows retuned by the query and the number of rows corresponding to the specified percentage.

TABLE 36. SELECT statement with TOP does not contain the ORDER BY clause

Microsoft SQL Server	Oracle

create procedure sql_sp_select_top
as
DECLARE @a number
select top 1 @a=col1 from tab1
	CREATE OR REPLACE PROCEDURE sql_sp_select_top
	AS
	v_a VARCHAR2(255);
	BEGIN
	select col1 INTO v_a from tab1 WHERE ROWNUM <=1;
	END;

create procedure sql_sp_select_top2
as
DECLARE @a number
select top 1 @a=col1 from tab1 WHERE col2>0
	CREATE OR REPLACE PROCEDURE sql_sp_select_top2
	AS
	v_a VARCHAR2(255);
	BEGIN
	select col1 INTO v_a from tab1 WHERE col2>0 and ROWNUM <=1;
	END;

create procedure sql_sp_select_top3
as
DECLARE @a number
select top 30 percent @a=col1 from tab1
	CREATE OR REPLACE PROCEDURE sql_sp_select_top3
	AS
	v_a VARCHAR2(255);
	BEGIN
	select col1 INTO v_a from tab1 WHERE ROWNUM <= 30 *(SELECT COUNT(*) from tab1) / 100;
	END;

b) **SELECT statement with TOP also contains the ORDER BY clause.**

Unlike Microsoft SQL Server, Oracle applies comparison with ROWNUM before ordering the result set. If the Microsoft SQL Server SELECT statement contains the ORDER BY clause, SQLWays converts the source query to the

query with the subquery. The subquery performs the ordering, while the query performs row restricting using ROWNUM.

TABLE 37. SELECT statement with TOP also contains the ORDER BY clause

Microsoft SQL Server	Oracle

create procedure sql_sp_select_top4
as
DECLARE @a number
select top 1 @a=col1 from tab1 **order by** col1
	create or replace procedure sql_sp_select_top4
	as
	a number;
	begin
	select * into a from (select col1 from tab1 order by col1)
	where rownum<=1;
	end;

create procedure sql_sp_select_top5
as
DECLARE @a number
select top 15 percent @a=col1 from tab1
order by col1
	create or replace procedure sql_sp_select_top5
	as
	a number;
	begin
	select * into a from (select col1 from tab1 order by col1)
	where rownum<=15*(select count(*) from tab1 order by col1)/100;
	end;

Executing Procedures and User-Defined Functions

This section describes an execution of procedures and user-defined functions from other procedures or functions in

various databases and their conversion by SQLWays.

TABLE 38. Executes Procedures and User-Defined Functions

Database	Syntax	Description
Microsoft SQL Server	**EXEC** [**UTE**] [*@return_status* =] *procedure_name* [[*@par1*=] {*value* | *@variable* [OUTPUT] | [DEFAULT] [,[*@parN*=] {*value* | *@variable* [OUTPUT] | [DEFAULT]]...]	The EXECUTE statement is used to execute procedures and user-defined functions in Microsoft SQL Server. @return_status is the value, returning by the function (procedure) procedure_name – the name of the procedure (function) @parN – the name of a procedure (function) parameter **Examples**: The sample below calls the procedure proc2: EXEC proc2 The following example calls f1 function with parameters 1 and @var1. The function returns a value in the @ret_val variable: EXECUTE @ret_value = f1 1, @var1

TABLE 38. Executes Procedures and User-Defined Functions

Database	Syntax	Description
Oracle	[*return_status* =] *procedure_name* ([{ *value* | *variable* }])	In order to execute a procedure or function in Oracle, you have to explicitly specify the name of the procedure (function) and its parameters in PL/SQL. **Examples**: The following example calls the *proc2* stored procedure with parameters 77 and 'test'. proc2 (77,'test'); In the example below, function func2 returns a value into the *ret_func2* variable. The function takes one parameter – 0. ret_func2:=func2(0) ;
Sybase Adaptive Server Anywhere	[@variable =] CALL procedure_name ([[@par1=] exp1 [,[@parN=] expN]…])	CALL invokes a procedure that has been previously created. The argument list can be specified by position or by using keyword format: •by position, the arguments will match up with the corresponding parameter in the parameter list for the procedure; •by keyword, the arguments are matched up with the named parameters (par1, …, parN). Inside a procedure, a CALL statement can be used in a DECLARE statement when the procedure returns result sets (for example: in the DECLARE CURSOR statement). Procedures can return an integer value (as a status indicator) using the RETURN statement.
	EXEC[UTE] [@return_status =] [creator.]procedure_name [[@par1=] {exp1 | @variable1 [OUTPUT]} [,[@parN=] {expN | @variableN [OUTPUT]}]…]	EXECUTE invoke a procedure, optionally supplying procedure parameters and retrieving output values and return status information. EXECUTE is Sybase Adaptive Server Enterprise-compatible alternative to the CALL statement. @return_status is a value, which contains return status information. @parN is the name of a procedure parameter. Examples: The sample below executes the procedure p2: EXECUTE p2 The following example executes the procedure and stores the return value in the @ret_val variable: EXECUTE @ret_value = p1 1

Conversion of Execution Procedures and User-Defined Functions from Microsoft SQL Server to Oracle

The Microsoft SQL Server EXEC statement is used for executing user-defined functions and stored procedures. In order to execute a user-defined functions or stored procedure in Oracle, you have to specify its name and parameters in the procedure body.

SQLWays converts the Microsoft SQL Server EXEC statement to the Oracle syntax for calling stored procedures and functions. .

TABLE 39. Conversion of Execution Procedures and User-Defined Functions from Microsoft SQL Server to Oracle

Microsoft SQL Server	Oracle

create procedure sql_sp_exec
as
DECLARE @a varchar(20)
EXEC @a=func1 1
	create or replace procedure sql_sp_exec
	as
	v_a varchar2(20);
	begin
	v_a:=func1(1);
	end;

create procedure sql_sp_exec2
as
DECLARE @a varchar(20)
EXECUTE proc1 @par1=@a,3
	create or replace procedure sql_sp_exec2
	as
	v_a varchar2(20);
	begin
	proc1 (v_a,3);
	end;

create procedure sql_sp_exec3
as
EXECUTE proc3
	create or replace procedure sql_sp_exec3
	as
	begin
	proc3();
	end;

Conversion of Sybase Adaptive Server Anyware CALL to Microsoft SQL Server

The Sybase Adaptive Server Anyware CALL statement is used to invoke a procedure that has been previously created. Sybase Adaptive Server Anyware CALL allow specifying expressions in the argument list.

The EXECUTE statement is used to invoke procedures in Microsoft SQL Server. Microsoft SQL Server EXECUTE allows specifying only constants or variables (not expressions) in the argument list.

SQLWays converts Sybase ASA CALL to the Microsoft SQL Server EXECUTE statement. For each expression in CALL, SQLWays creates a local variable, assigns the expression to the variables, and use this variable in the EXECUTE statement.

TABLE 40. Example of the conversion

Sybase Adaptive Server Anyware	Microsoft SQL Server

create procedure asa_sp_call (@a int)
begin
call sp_func('MTN' + ':' + 'FTN',@a+3);
end;
	create procedure asa_sp_call @a INT
	AS begin
	DECLARE @par01 CHAR(10)
	DECLARE @par02 INT
	set @par01 = 'MTN' + ':' + 'FTN'
	set @par02 = @a + 3
	EXECUTE sp_func @par01, @par02
	end

Executing Dynamic SQL Statements with Parameters

This section describes execution of parameterized dynamic SQL statements in various databases and their conversion by SQLWays.

TABLE 41. Executing Dynamic SQL Statements with Parameters

Database	Syntax	Description
Microsoft SQL Server	**EXEC** [**UTE**] sp_executesql *N'dynamic_compound_string'* | *@dynamic_statement* [*N'param_defined_string'* | *dynamic_param_definition* [*@param1*=]*value1* [,[*@paramN*=]*valueN*]…]	The EXECUTE statement with sp_executesql executes a SQL statement including dynamic compound statements. Dynamic compound statements can contain embedded parameters. **Parameters**: *N'dynamic_compound_string'* | *@dynamic_statement* – executable SQL statement that can be defined as a string or variable. *N'param_defined_string'* | *dynamic_param_definition* - string or variable which specifies input parameters for the executable SQL statement. Each parameter definition consists of a parameter name and its data type. The default value for the parameter is NULL *[@param1=] value1* is a value of the parameter. The value can be a constant or a variable. There must be a value specified for every parameter included in the dynamic statement. **Example**: The following statement executes a parameterized dynamic select statement with the input parameter: execute sp_executesql N'select * from tab1 where col1 = @param', N'@param int', @param = 35

TABLE 41. Executing Dynamic SQL Statements with Parameters

Database	Syntax	Description
Oracle	**EXECUTE IMMEDIATE** dynamic_string [INTO { *ret_value1* [, *retvalueN*]… | *record_name*}] [USING [{*IN* | *OUT* | *IN OUT*}] value1, [,valueN]…];	The EXECUTE IMMEDIATE statement executes dynamic SQL statements in Oracle **Parameters**: dynamic_string - executable SQL statement that can be defined as a string, variable or as expression. Input parameters in are marked as :N, where N is the number of the parameter in the USING clause. *ret_valueN*, *record_name* – the INTO clause specifies the variables or a record into which the column values are retrieved. The INTO clause can be used only for single-row queries. [*IN* | *OUT* | *IN OUT*] *valueN* – the USING clause specifies a list of input/output values of the parameters. By default, valueN is IN parameter. **Examples**: The sample below executes a dynamic compound insert statement with input parameters: sql_dString := 'INSERT INTO tab1 VALUES (:1, :2, :3)'; EXECUTE IMMEDIATE sql_dString USING 77, 21, variable1;

Conversion of Dynamic Statement Execution from Microsoft SQL Server to Oracle

The EXECUTE sp_executesql statement is used to execute dynamic SQL statements with parameters in Microsoft SQL Server. The second string parameter of EXECUTE sp_executesql describes the dynamic SQL statement parameters and their types.

The EXECUTE IMMEDIATE statement is used to execute dynamic SQL statements with parameters in Oracle. Input parameters are marked as :N, where N is the number of the parameter in the USING clause.

SQLWays converts the Microsoft SQL Server EXECUTE sp_executesql statement to Oracle EXECUTE IMMEDIATE. SQLWays changes input parameters in MSQL dynamic statements to the appropriate Oracle syntax. SQLWays replaces parameter names in Microsoft SQL Server dynamic string to the number of the parameter when converting to Oracle.

TABLE 42. Conversion of Dynamic Statement Execution from Microsoft SQL Server to Oracle

Microsoft SQL Server	Oracle

create procedure sql_sp_executesql as
declare @param int
execute sp_executesql N'select * from tab1 where col1 = @param',
N'@param int',
@param = 35
	CREATE OR REPLACE PROCEDURE sql_sp_executesql AS
	v_param NUMBER(10,0);
	BEGIN
	EXECUTE IMMEDIATE 'select * from tab1 where col1 = :1'
	USING 35;
	end;

create procedure sql_sp_executesql2 as
declare @InsOrderID int
declare @InsertString varchar(50)
SET @InsertString = N'INSERT INTO tab1' +
' VALUES (@InsOrderID)'
EXEC sp_executesql @InsertString,
N'@InsOrderID INT', @InsOrderID
	CREATE OR REPLACE PROCEDURE sql_sp_executesql2 AS
	v_InsOrderID NUMBER(10,0);
	v_InsertString VARCHAR2(50);
	BEGIN
	v_InsertString := 'INSERT INTO tab1' ||
	' VALUES (:1)';
	EXECUTE IMMEDIATE v_InsertString USING
	v_InsOrderID;
	END;

create procedure sql_sp_executesql3 as
declare @val int
declare @InsertString varchar(50)
SET @InsertString = N'DELETE FROM tab1 WHERE col1 = @par1 and col2=@par2'
EXEC sp_executesql @InsertString,
N'@par1 INT, @par2 INT', @par1=1,
@par2=@val
	CREATE OR REPLACE PROCEDURE sql_sp_executesql3 AS
	v_val NUMBER(10,0);
	v_InsertString VARCHAR2(50);
	BEGIN
	v_InsertString := 'DELETE FROM tab1 WHERE col1 = :1 and col2=:2' ;
	EXECUTE IMMEDIATE v_InsertString
	USING 1,v_val;
	end;

Cursors

The set of rows returned by a query is called the result set. The result set can consist of zero, one, or multiple rows, depending on how many rows meet your search criteria.

For queries that return more than one row, you must declare a cursor to process the rows. A cursor points to the current row in the result set of a multi-row query.

You can use the following commands to control a cursor: OPEN, FETCH, and CLOSE. First, you initialize the cursor with the OPEN statement, which identifies the result set. Then, you can execute FETCH repeatedly until all rows have been retrieved. When the last row has been processed, you release the cursor with the CLOSE statement. You can process several queries in parallel by declaring and opening multiple cursors.

- Cursor Declaration
- Conversion of Cursor with parameters from Oracle to MySQL

Cursor Declaration

This subsection describes cursor declaration statements in various databases and their conversion by SQLWays. You must declare a cursor before referencing it in other statements. When declaring a cursor you associate it with a specific query

TABLE 43. Cursor Declaration Statements in Various Databases

Database	Statement	Description
Microsoft SQL Server	**DECLARE** *cursor_name* **CURSOR** [**LOCAL** | **GLOBAL**] [**FORWARD_ONLY** | **SCROLL**] FOR *select_statement* [FOR **UPDATE** [OF *column1* [{ , *columnN*...}]]]	Declares a cursor and defines its attributes such as scrolling behavior and the query used to build the result set for the cursor. FORWARD_ONLY specifies that the cursor can only be scrolled from the first to the last row. FETCH NEXT is the only supported fetch option. SCROLL specifies access to fetch options (FIRST, LAST, PRIOR, NEXT, RELATIVE, ABSOLUTE) are available. *select_statement* specifies a SELECT statement that defines the result set of the cursor.
Oracle	**CURSOR** *cursor_name* [(*cur_param1* [IN] *data_type* [{:= | **DEFAULT**} *exp*] [, *cur_paramN* [IN] *data_type* [{:= | **DEFAULT**} *exp*]...)] IS *select_statement*;	Declares a cursor and associates it with a specific query. cur_paramN is a formal cursor parameter that can appear in a query wherever a constant can appear. The formal parameters of a cursor must be IN parameters. The query can also reference other PL/SQL variables within its scope. *Select_statement* is a query that returns a result set. If the cursor declaration declares parameters, each parameter must be used in the query.
MySQL	**DECLARE** *cursor_name* **CURSOR** FOR *select_statement*	Declares a cursor and associates it with a specific query. *select_statement* specifies a SELECT statement that defines the result set of the cursor.

a) Converting the FORWARD_ONLY cursor

The FORWARD_ONLY option in Microsoft SQL Server specifies that rows scrolled in forward direction from the first to the last row. This scrolling is default in Oracle, so the FORWARD_ONLY option is removed by SQLWays.

Examples:

TABLE 44. Cursor declaration statements in various databases

Microsoft SQL Server	Oracle
DECLARE cur1 CURSOR FORWARD_ONLY FOR SELECT col1 FROM tab1	CURSOR cur1 IS SELECT col1 FROM tab1;

b) Converting SCROLL cursor

The SCROLL option in Microsoft SQL Server specifies that rows scrolled in various directions (FIRST, LAST, PRIOR, NEXT, RELATIVE, ABSOLUTE). Whereas Oracle supports only forward direction (NEXT).

SQLWays removes the SCROLL option when converting to Oracle. This declaration conversion is correct if an application uses fetch with NEXT, otherwise the application must be revised.

Examples:

TABLE 45. Converting SCROLL cursor

Microsoft SQL Server	Oracle
DECLARE cur1 CURSOR SCROLL FOR SELECT col1 FROM tab1	CURSOR cur1 IS SELECT col1 FROM tab1;

Conversion of Cursor with Parameters from Oracle to MySQL

In Oracle cursor may have parameters that can appear in a query wherever a constant can appear. Parameters allow using one cursor with different values of input parameters in the query.

MySQL does not support parameters in cursors.

When converting an Oracle cursor with parameters to MySQL, SQLWays declares and uses local variables instead of the cursor parameters. Before opening the cursor, SQLWays assigns the values of the cursor parameters for these local variables.

Examples:

TABLE 46. Cursor with Parameters Conversion from Oracle to MySQL

Oracle	MySQL

create procedure ora_sp_cur_with_param_mysql
a
var1 number := 0;
CURSOR cur1 (val1 number) IS
SELECT col1 FROM tab1
WHERE col5 = val1;
BEGIN
open cur1 (0);
fetch cur1 into var1;
close cur1;
open cur1 (5);
fetch cur1 into var1;
close cur1;
END;
	Create procedure ora_sp_cur_with_param_mysql()
	BEGIN
	DECLARE var1 FLOAT DEFAULT 0;
	DECLARE val1 FLOAT;
	DECLARE cur1 CURSOR FOR
	SELECT col1 FROM tab1
	WHERE col5 = val1;
	SET val1 = 0;
	open cur1;
	fetch cur1 into var1;
	close cur1;
	SET val1 = 5;
	open cur1;
	fetch cur1 into var1;
	close cur1;
	end;

Transaction Control

This section contains information about transaction conversion when exporting data from one database to another by SQLWays.

- Starting Transaction
- BEGIN TRANSACTION Conversion from Microsoft SQL Server to Oracle
- COMMIT Statement
- COMMIT Conversion from Microsoft SQL Server to Oracle

Starting Transaction

Some databases allow explicit creation of user-defined transaction by special statements. When such statement database is defined a new user-defined transaction starts.

TABLE 47. Starting Transaction

Database	Syntax	Description
Microsoft SQL Server	**BEGIN** [**TRAN** [**SACTION**] [*transaction_name* | *@tran_name_variable*] [WITH MARK ['description']]]	BEGIN TRANSACTION starts an explicit or nested local transaction for the connection issuing the statement. Each transaction continues either until it completes without errors and COMMIT TRANSACTION ends transaction, or if errors are encountered and all modifications are erased with a ROLLBACK TRANSACTION statement. BEGIN TRANSACTION increments @@TRANCOUNT by 1. The WITH MARK option is the transaction name which places in the transaction log. When restoring a database to an earlier state, the marked transaction can be used in place of a date and time.
Oracle	Not supported.	

BEGIN TRANSACTION Conversion from Microsoft SQL Server to Oracle

Microsoft SQL Server allows starting explicit or nested transactions using BEGIN TRANSACTION. COMMIT statements for such transaction decrease @@TRANCOUNT by 1 without making updates permanent. If a name is specified, Microsoft SQL Server application can rollback the transaction to its beginning or to a defined BEGIN TRANSACTION statement.

Explicit transactions are not supported in Oracle but Oracle can rollback a part of work. For this purpose use a SAVEPOINT statement. The SAVEPOINT statement allows identify a point in a transaction to which you can roll back later. After the SAVEPOINT statement has been created, you can continue processing, commit your work, rollback the entire transaction, or rollback to the SAVEPOINT.

Therefore, SQLWays emulates the Microsoft SQL Server BEGIN TRANSACTION statement for ROLLBACK using Oracle SAVEPOINT statement.

SQLWays changes BEGIN TRANSACTION statement with name to SAVEPOINT statement. BEGIN TRANSACTION statements without name are dropped.

Examples:

TABLE 48. BEGIN TRANSACTION Conversion from Microsoft SQL Server to Oracle

Microsoft SQL Server	Oracle	Description
BEGIN	BEGIN	
BEGIN TRAN		
BEGIN TRANSACTION		
BEGIN TRANSACTION tran1	SAVEPOINT tran1	
BEGIN TRANSACTION @val_tran1	SAVEPOINT tran1	If @val_tran1 equal 'tran1'
BEGIN TRANSACTION @val_tran1 WITH MARK 'transaction 1	SAVEPOINT tran1	If @val_tran1 equal 'tran1'

COMMIT Statement

For explicitly end of transactions you can use a COMMIT statement. A COMMIT statement guarantees that all the transaction's modifications are made in the permanent part of the database. A COMMIT also frees resources, such as locks, used by the transaction.

TABLE 49. COMMIT Statement

Database	Syntax	Description
Microsoft SQL Server	**COMMIT** [**TRAN** [**SACTION**] [*transaction_name* | *@tran_name_variable*]]	Marks the end of a successful implicit or explicit transaction. If COMMIT is defined with transaction name or variable, then Microsoft SQL Server closes a user-defined transaction. If @@TRANCOUNT is 1, COMMIT TRANSACTION makes all data modifications performed since the start of the transaction a permanent part of the database and decrements @@TRANCOUNT to 0. If @@TRANCOUNT is greater than 1, COMMIT TRANSACTION decrements @@TRANCOUNT only by 1. When it is used in nested transactions, it commits the inner transactions, which do not free resources or make their modifications permanent. The data modifications are made permanent and resources freed only when the outer transaction is committed. Each COMMIT TRANSACTION is issued when @@TRANCOUNT is greater than 1 simply decrements @@TRANCOUNT by 1. When @@TRANCOUNT is finally decremented to 0, the entire outer transaction is committed. Because transaction_name is ignored by SQL Server, issuing a COMMIT TRANSACTION referencing the name of an outer transaction when there are outstanding inner transactions only decrements @@TRANCOUNT by 1.
Oracle	**COMMIT** [WORK] [COMMENT 'text']	The COMMIT statement explicitly makes permanent any changes made to the database during the current transaction. COMMENT keyword specifies a comment to be associated with the current transaction and is typically used with distributed transactions. The text must be a quoted literal no more than 50 characters long.

COMMIT conversion from Microsoft SQL Server to Oracle

Microsoft SQL Server COMMIT can end implicit and explicit (nested) transactions, while Oracle COMMIT ends the current transaction. Therefore SQLWays removes the TRANSACTION clause and transaction name (variable) when converting COMMIT from Microsoft SQL Server to Oracle.

Examples:

TABLE 50. COMMIT conversion from Microsoft SQL Server to Oracle

Microsoft SQL Server	Oracle
COMMIT	COMMIT;
COMMIT TRAN	COMMIT;
COMMIT TRANSACTION	COMMIT;
COMMIT TRANSACTION tran1	COMMIT;
COMMIT TRANSACTION @val_tran1	COMMIT;

Functions

This section contains information about the functions, their use in various databases and their conversion by SQLWays.

- Number Functions
- String Functions
- Returning Information about Database and Current Connection
- Replace NULL value functions

Number Functions

This subsection describes number functions used in various databases and their conversion by SQLWays.

- Converting String to Number

Converting String to Number

This subsection describes functions, which convert string to number in various databases and their conversion by SQLWays.

TABLE 51. Converting String to Number

<table>
<tr><th>Database</th><th>Syntax</th><th colspan="3">Description</th></tr>
<tr><td rowspan="8">Oracle</td><td rowspan="8">TO_NUMBER (exp1 [, exp2 [, exp3]])</td><td colspan="3">Converts exp1, a value of CHAR, VARCHAR2, NCHAR or NVARCHAR2 data type containing a number in format specified by the optional format model exp2, to a value of NUMBER data type.
Return type: NUMBER.

The exp3 specifies characters that are returned by number format elements:
•Decimal character
•Group separator
•Local currency symbol
•International currency symbol.

A number format model (exp2) is composed of one or more number format elements as listed in following:</td></tr>
<tr><td>Element</td><td>Example</td><td>Description</td></tr>
<tr><td>, (comma)</td><td>9,999</td><td>Returns a comma in the specified position. Multiple commas maybe specified.
Restrictions:
•A comma element cannot begin a number format model.
•A comma cannot appear to the right of a decimal character or period in a number format model.</td></tr>
<tr><td>. (period)</td><td>99.99</td><td>Returns a decimal point, which is a period (.) in the specified position.
Restriction: Only one period maybe specified.</td></tr>
<tr><td>$</td><td>$9999</td><td>Returns value with a leading dollar sign.</td></tr>
<tr><td rowspan="2">0</td><td>0999</td><td>Returns leading zeros.</td></tr>
<tr><td>9990</td><td>Returns trailing zeros.</td></tr>
</table>

TABLE 51. Converting String to Number

Database	Syntax	Description		
		9	9999	Returns value with the specified number of digits with a leading space if positive or with a leading minus if negative. Leading zeros are blank, except for a zero value, which returns a zero for the integer part of the fixed-point number.
		B	B9999	Returns blanks for the integer part of a fixed-point number when the integer part is zero (regardless of "0"s in the format model).
		C	C999	Returns in the specified position the ISO currency symbol (the current value of the NLS_ISO_CURRENCY parameter in ***exp3***).
		EEEE	9.9EEEE	Returns a value using in scientific notation.
		D	99D99	Returns in the specified position the decimal character, which is the current value of the NLS_NUMERIC_CHARACTER parameter in ***exp3***. The default is a period (.). **Restriction**: Only one decimal character maybe specify.
		FM	FM90.9	Returns a value with no leading or trailing blanks.
		G	9G999	Returns in the specified position the group separator (the current value of the NLS_NUMERIC_CHARACTER parameter in ***exp3***). Multiple group separators maybe specify in a number format model. **Restriction**: A group separator cannot appear to the right of a decimal character or period.
		L	L999	Returns in the specified position the local currency symbol (the current value of the NLS_CURRENCY parameter in ***exp3***).
		MI	9999MI	Returns negative value with a trailing minus sign (-). Returns positive value with a trailing blank. **Restriction**: The MI format element can appear only in the last position of a number format model.
		PR	9999PR	Returns a negative value in <angle brackets>. Returns a positive value with a leading and trailing blank. **Restriction**: The PR format element can appear only in the last position of a number format model.
		RN	RN	Returns a value as Roman numerals in uppercase. A value can be an integer between 1 and 3999.

TABLE 51. Converting String to Number

Database	Syntax	Description		
		rn	rn	Returns a value as Roman numerals in lowercase. Value can be an integer between 1 and 3999.
		S	S9999	Returns a negative value with a leading minus sign (-). Returns a positive value with a leading plus sign (+). **Restriction**: The S format element can appear only in the first or last position of a number format model.
			9999S	Returns a negative value with a trailing minus sign (-). Returns a positive value with a trailing plus sign (+).
		TM	TM	"Text minimum". Returns (in decimal output) the smallest number of possible characters. This element is case-insensitive. The default is TM9, which returns the number in fixed notation unless the output exceeds 64 characters. If output exceeds 64 characters, then Oracle automatically returns the number in scientific notation. **Restrictions**: •This element cannot precede with any other element. •This element can follow only with 9 or E (only one) or e (only one).

TABLE 51. Converting String to Number

Database	Syntax	Description		
		U	U9999	Returns in the specified position the "Euro" (or other) dual currency symbol (the current value of the NLS_DUAL_CURRENCY parameter in ***exp3***).
		V	999V99	Returns a value multiplied by 10n (and if necessary, round it up), where n is the number of 9's after the "V".
		X	XXXX	Returns the hexadecimal value of the specified number of digits. If the specified number is not an integer, then Oracle rounds it to an integer. **Restrictions**: •This element accepts only positive values or 0. Negative values return an error. •This element can precede only with 0 (which returns leading zeroes) or FM. Any other elements return an error. If you specify neither 0 nor FM with X, then the return always has 1 leading blank.
MySQL	**CAST** (*exp1* AS UNSIGNED [INTEGER]) **CAST** (*exp1* AS SIGNED [INTEGER])	Converts a value of ***exp1*** to a value of UNSIGNED or SIGNED INTEGER data type. If it uses numerical operations (like +) and one of the operands is unsigned as integer, the result will be unsigned. This can be overrided by using the SIGNED and UNSIGNED cast operators to cast the operation to a signed or unsigned 64-bit integer, respectively.		

String Functions

This subsection describes string functions used in various databases and their conversion by SQLWays.

- Concatenating Strings
- Converting Expression to String
- Converting ASCII Code to Character
- Converting Datetime Expression with Format String to String
- Conversion of Oracle TO_CHAR(datetime) with Format String to MySQL
- Returning Substring from String
- Returning String in Uppercase

Concatenating Strings

This subsection describes string concatenation functions used in various databases and their conversion by SQLWays.

TABLE 52. Concatenating Strings

Database	Syntax	Description
Oracle	*exp1* || *exp2* [|| *expN*]...	Returns the concatenation of two or more expressions of CHAR, VARCHAR2, CLOB data type. Return type: •If all expressions (for example ***exp1*** and ***exp2***) are of data type CHAR, the result has data type CHAR and is limited to 2000 characters. •If either expression is of data type VARCHAR2, the result has data type VARCHAR2 and is limited to 4000 characters. •If either expression is CLOB, the result is CLOB.
	CONCAT (*exp1*, *exp2*)	Returns ***exp1*** concatenated with ***exp2***. Both ***exp1*** and ***exp2*** can be any of the datatypes CHAR, VARCHAR2, NCHAR, NVARCHAR2, CLOB, or NCLOB. **Return type**: •The string returned is in the same character set as ***exp1***. Its data type depends on the datatypes of the arguments. •In concatenations of two different datatypes, CONCAT function returns the data type that results in a lossless conversion. •Therefore, if one of the arguments is a LOB, then the returned value is a LOB. If one of the arguments is a national data type, then the returned value is a national data type. CONCAT function is equivalent to the concatenation operator (||).

TABLE 52. Concatenating Strings

Database	Syntax	Description
MySQL	**CONCAT**(*exp1*, *exp2* [,*expN*]…)	Returns the string that results from concatenating the arguments (***expressions***). Returns NULL if any argument (***expression***) is NULL. A numeric argument is converted to its equivalent string form.
Microsoft SQL Server	*exp1* + *exp2* [+ *expN*]…	Returns the concatenation of two or more character or binary strings, columns, or a combination of strings and column names into one expression (a string operator). expression (***exp1***, ***exp2***, …, ***expN***) – is expression of any of the data types in the character and binary data type category, except the image, ntext, or text data types. Both expressions must be of the same data type, or one expression must be able to be implicitly converted to the data type of the other expression. An explicit conversion to character data must be used when concatenating binary strings and any characters between the binary strings. **Return type**: the data type of the argument with the highest precedence. The precedence order data types is following: •**sql_variant** (highest) •datetime •smalldatetime •float •real •decimal •money •smallmoney •bigint •int •smallint •tinyint •bit •ntext •text •image •timestamp •uniqueidentifier •nvarchar •nchar •varchar •char •varbinary •**binary** (lowest)
IBM DB2	**CONCAT** (*exp1*, *exp2*) *exp1* || *exp2* [|| *expN*]… *exp1* **CONCAT** exp2 [**CONCAT** *expN*]…	Returns the concatenation of two or more string arguments. Arguments must be compatible types. The result of the function and operator (||) is a string. Its length is the sum of the lengths of the two arguments. If either argument is null, the result is the null value. **Note**: a binary string cannot be concatenated with a character string, including character strings defined as FOR BIT DATA.

Converting Expression to String

This subsection describes functions that allow converting expression to a string in various databases and their conversion by SQLWays.

TABLE 53. Converting Expression to a String

Database	Syntax	Description
Oracle	**TO_CHAR**(*n* [, *fmt* [, '*nlsparam*']])	Converts *n* of NUMBER data type to a value of VARCHAR2 data type, using the optional number format *fmt*. If *fmt* is omit, then n is converted to a VARCHAR2 value exactly long enough to hold its significant digits. The '*nlsparam*' specifies these characters that are returned by number format elements: - Decimal character - Group separator - Local currency symbol • International currency symbol This argument can have this form: 'NLS_NUMERIC_CHARACTERS = ''dg'' NLS_CURRENCY = ''text'' NLS_ISO_CURRENCY = territory ' The characters *d* and *g* represent the decimal character and group separator, respectively. They must be different single-byte chara¬cters. Note that within the quoted string, you must use two single quotation marks around the parameter values. Ten characters are available for the currency symbol. If '*nlsparam*' or any of the parameters are omitted, this function uses the default parameter values for session.
	TO_CHAR (*nchar* | *clob* | *nclob*)	Converts NCHAR, NVARCHAR2, CLOB, or NCLOB data to the database character set.
Microsoft SQL Server	**STR** (*fl_exp* [, *length* [, *decimal*]])	Returns character data converted from numeric data. ***fl_exp*** – is an expression of approximate numeric (float) data type with a decimal point. ***length*** – is the total length, including decimal point, sign, digits, and spaces. The default is 10. ***decimal*** – is the number of places to the right of the decimal point. Return type – **char**. If supplied, the values for *length* and *decimal* parameters to STR should be positive. The number is rounded to an integer by default or if the decimal parameter is 0. The specified length should be greater than or equal to the part of the number before the decimal point plus the number's sign (if any). A short *fl_exp* is right-justified in the specified length, and a long *fl_exp* is truncated to the specified number of decimal places. For example, STR(12,10) fields the result of 12, which is right-justified in the result set. However, STR(1223, 2) truncates the result set to **.

TABLE 53. Converting Expression to a String

Database	Syntax	Description
IBM DB2	**CHAR** (*int_exp*)	Returns a fixed-length character string representation of an integer number. If argument can be null, the result can be null. If argument is null, the result is the null value. **int_exp** – is the expression that returns a value that is an integer data type (either SMALLINT, INTEGER or BIGINT). The result is the character string represent¬tation of the argument in the form of an SQL integer constant. The result consists of n characters that are the significant digits that represent the value of the argument with a preceding minus sign if the argument is negative. It is left justified. •If the first argument is a small integer: The length of the result is 6. If the number of characters in the result is less than 6, then the result is padded on the right with blanks to length 6. •If the first argument is a large integer: The length of the result is 11. If the number of characters in the result is less than 11, then the result is padded on the right with blanks to length 11. •If the first argument is a big integer: The length of the result is 20. If the number of characters in the result is less than 20, then the result is padded on the right with blanks to length 20.
	CHAR (*character_exp* [, *integer*])	Returns a fixed-length character string representation of a character string, if the first argument is any type of character string. If the first argument can be null, the result can be null. If the first argument is null, the result is the null value. ***character_exp*** – is an expression that returns a value that is CHAR, VARCHAR, LONG VARCHAR, or CLOB data type. ***integer*** – is the length attribute for the resulting fixed length character string. The value must be between 0 and 254. If the length of the character-expression is less than the length attribute of the result, the result is padded with blanks up to the length of the result. If the length of the character-expression is greater than the length attribute of the result, truncation is performed. A warning is returned unless the truncated characters were all blanks and the character-expression was not a long string (LONG VARCHAR or CLOB).

TABLE 53. Converting Expression to a String

Database	Syntax	Description
	CHAR (*decimal_exp* [, *decimal_ch*])	Returns a fixed-length character string representation of a decimal number, if the first argument is a decimal number. If the first argument can be null, the result can be null. If the first argument is null, the result is the null value. ***decimal-exp*** – is an expression that returns a value that is a decimal data type. If a different precision and scale is desired, the DECIMAL scalar function can be used first to make the change. ***decimal-ch*** – specifies the single-byte character constant that is used to delimit the decimal digits in the result character string. The character cannot be a digit, plus ('+'), minus ('-') or blank. The default is the period ('.') character. The result is the fixed-length character-string representation of the argument. The result includes a decimal character and *p* digits, where *p* is the precision of the decimal-exp with a preceding minus sign if the argument is negative. The length of the result is 2+p, where p is the precision of the *decimal-exp*. This means that a positive value will always include one trailing blank.
	CHAR (*fl_exp* [, *decimal_ch*])	Returns a fixed-length character string representation of a double-precision floating-point number, if the first argument is a DOUBLE or REAL. If the first argument can be null, the result can be null. If the first argument is null, the result is the null value. ***fl_exp*** – is an expression that returns a value that is a floating-point data type (DOUBLE or REAL). ***decimal_ch*** – specifies the single-byte character constant that is used to delimit the decimal digits in the result character string. The character cannot be a digit, plus (+), minus (-), or blank character. The default is the period (.) character. The result is the fixed-length character-string representation of the argument in the form of a floating-point constant. The length of the result is 24. If the argument is negative, the first character of the result is a minus sign. Otherwise, the first character is a digit. If the argument value is zero, the result is 0E0. Otherwise, the result includes the smallest number of characters that can represent the value of the argument so that the mantissa consist of a single digit other than zero followed by the decimal-character and a sequence of digits. If the number of characters in the result is less than 24, the result is padded on the right with blanks to length 24.

TABLE 53. Converting Expression to a String

Database	Syntax	Description
Sybase Adaptive Server Anywhere	**STR** (*fl_exp* [, *length* [, *decimal*]])	Returns the string equivalent of a number. ***fl_exp*** – is any approximate numeric (float, real, or double precision) expression. ***length*** – is the number of characters to be returned (including the decimal point, all digits to the right and left of the decimal point, and blanks). The default is 10. ***decimal*** – is the number of decimal digits to be returned. The default is 0. If the integer portion of the number cannot fit in the length specified, then the result is a string of the specified length containing all asterisks. For example, the following statement returns *** : SELECT STR(1234.56, 3)
Sybase Adaptive Server Enterprise	**STR** (*fl_exp* [, *length* [, *decimal*]])	Returns the character equivalent of the specified number. ***fl_exp*** – is any approximate numeric (float, real, or double precision) column name, variable, or constant expression. ***length*** – sets the number of characters to be returned (including the decimal point, all digits to the right and left of the decimal point, and blanks). The default is 10. ***decimal*** – sets the number of decimal digits to be returned. The default is 0. • STR, a string function, returns a character representation of the floating point number. • *length* and *decimal* are optional. If given, they must be non-negative. STR rounds the decimal portion of the number so that the results fit within the specified length. The length should be long enough to accom¬modate the decimal point and, if negative, the number's sign. The decimal portion of the result is rounded to fit within the specified length. If the integer portion of the number does not fit within the length, however, STR returns a row of asterisks of the specified length. For example: select str(123.456, 2, 4) A short *appr_num* is right justified in the specified length, and a long *appr_num* is truncated to the specified number of decimal places. • If *appr_num* is NULL, returns NULL.

Converting ASCII Code to Character

This subsection describes functions, which convert an int ASCII code to a character in various databases.

TABLE 54. Converting ASCII Code to Character

Database	Syntax	Description
Oracle	**CHR** (*exp1* [USING NCHAR_CS])	Returns the character having the binary equivalent to ***exp1*** in either the database character set or the national character set. If USING NCHAR_CS is not specified, then this function returns the character having the binary equivalent to ***exp1*** as a VARCHAR2 value in the database character set. If USING NCHAR_CS is specified, then this function returns the character having the binary equivalent to *n* as a NVARCHAR2 value in the national character set.
Microsoft SQL Server	**CHAR** (*exp1*)	A string function that converts an int ASCII code to a character. Return type – **char(1)**. ***exp1*** – is an integer from 0 through 255. NULL is returned if the integer expression is not in this range. CHAR can be used to insert control characters into character strings. The table shows some commonly used control characters.
MySQL	**CHAR**(*exp1* [,*expN*]…)	Interprets the arguments as integers and returns a string consisting of the characters given by the ASCII code values of those integers. NULL values are skipped.

TABLE 54. Converting ASCII Code to Character

Database	Syntax	Description
IBM DB2	**CHR** (*exp1*)	Returns the character that has the ASCII code value specified by the argument. The argument can be either INTEGER or SMALLINT. The value of the argument should be between 0 and 255; otherwise, the return value is null. The result of the function is CHAR(1). The result can be null; if the argument is null, the result is the null value.
Sybase Adaptive Server Anywhere	**CHAR** (*exp1*)	Returns the character with the ASCII value of a number. ***exp1*** – the number to be converted to an ASCII character. The number must be in the range 0 to 255, inclusive. CHAR returns NULL for integer expressions with values greater than 255 or less than zero.
Sybase Adaptive Server Enterprise	**CHAR** (*exp1*)	Returns the character equivalent of an integer. ***exp1*** – is any integer (tinyint, smallint, or int) column name, variable, or constant expression between 0 and 255. • CHAR, a string function, converts a single-byte integer value to a character value (char is usually used as the inverse of ascii.). • CHAR returns a char data type. If the resulting value is the first byte of a multibyte character, the character may be undefined. • If *char_expr* is NULL, returns NULL. Reformatting output with char • You can use concatenation and char values to add tabs or carriage returns to reformat output. char(10) converts to a return; char(9) converts to a tab.

Converting Datetime Expression with Format String to String

This subsection describes functions, which convert a datetime expression with a format string to a string in various databases and their conversion by SQLWays.

TABLE 55. Converting Datetime Expression with Format String to String

<table>
<tr><th>Database</th><th>Syntax</th><th colspan="2">Description</th></tr>
<tr><td rowspan="5">Oracle</td><td rowspan="5">TO_CHAR(date [, fmt [, 'nlsparam']])</td><td colspan="2">Converts date of DATE, TIMESTAMP, TIMESTAMP WITH TIME ZONE, or TIMESTAMP WITH LOCAL TIME ZONE data type to a value of VARCHAR2 data type in the format specified by the date format fmt.

If fmt is omit, then date is converted to a VARCHAR2 value as follows:
•DATE is converted to a value in the default date format.
•TIMESTAMP and TIMESTAMP WITH LOCAL TIME ZONE are converted to values in the default timestamp format.
•TIMESTAMP WITH TIME ZONE is converted to a value in the default timestamp with time zone format.

The 'nlsparams' specifies the language in which month and day names and abbreviations are returned. This argument can have this form:
'NLS_DATE_LANGUAGE = language'

A date format model (fmt) is composed of one or more datetime format elements as listed in following:</td></tr>
<tr><td>Element</td><td>Description</td></tr>
<tr><td>–
/
,
.
:
;
“text”</td><td>Punctuation and quoted text which is reproduced in the result.</td></tr>
<tr><td>AD
A.D.</td><td>AD indicator with or without periods.</td></tr>
<tr><td>AM
A.M.</td><td>Meridian indicator with or without periods.</td></tr>
</table>

TABLE 55. Converting Datetime Expression with Format String to String

Database	Syntax	Description	
		BC B.C.	BC indicator with or without periods.
		CC SCC	One greater than the first two digits of a four-digit year; “S” prefixes BC dates with “-”. For example, ’20’ from ’1900’.
		D	Day of week (1 – 7).
		DAY	Name of day, padded with blanks to length of 9 characters.
		DD	Day of month (1 – 31).
		DDD	Day of year (0 – 366).
		DY	Abbreviated name of day.
		FF [1..9]	Fractional seconds; no radix character is printed (use the X format element to add the radix character). Use the numbers 1 to 9 after FF to specify the number of digits in the fractional second portion of the datetime value returned.
		HH	Hour of day (1 – 12).
		HH24	Hour of day (0 – 23).
		IW	Week of year (1-52 or 1-53) based on the ISO standard.
		IYY IY I	Last 3, 2, or 1 digit(s) of ISO year.
		IYYY	4-digit year based on the ISO standard.
		J	Julian day; the number of days since January 1, 4712 BC.
		MI	Minute (0 – 59).
		MM	Month (01 – 12; Jan – 01).
		MON	Abbreviated name of month.

TABLE 55. Converting Datetime Expression with Format String to String

Database	Syntax	Description	
		MONTH	Name of month, padded with blanks to length if 9 characters.
		Q	Quarter of year (1, 2, 3, 4; JAN-MAR = 1).
		RM	Roman numeral month (I – XII; Jan – I).
		RR	Given a year with 2 digits: •If the year is <50 and the last 2 digits of the current year are >=50, then the first 2 digits of the returned year are 1 greater than the first 2 digits of the current year. •If the year is >=50 and the last 2 digits of the current year are <50, then the first 2 digits of the returned year are 1 less than the first 2 digits of the current year.
		RRRR	Round year. Accepts either 4-digit or 2-digit input. If 2-digit, provides the same return as RR. If you don't want this functionality, then simply enter the 4-digit year.
		SS	Second (0 – 59).
		SSSSS	Seconds past midnight (0 – 86399).
		TZD	Daylight savings information. The TZD value is an abbreviated time zone string with daylight savings information. It must correspond with the region specified in TZR.
		TZH	Time zone hour.
		TZM	Time zone minute.
		TZR	Time zone region information.
		WW	Week of year (1-53) where week 1 starts on the first day of the year and continues to the seventh day of the year.
		W	Week of month (1-5) where week 1 starts on the first day of the month and ends on the seventh.
		X	Local radix character.
		Y YYY	Year with comma in the position.
		YYYY SYYYY	4-digit year; "S' prefixes BC dates with "_".
		YEAR SYEAR	Year, spelled out; "S" prefixes BC dates with "-".
		YYY YY Y	Last 3, 2, or 1 digit(s) of year.
		These characters appear in the return value in the same location as they appear in the format model. The total length of a date format model cannot exceed 22 characters. Character literals, enclosed in double quotation marks.	

TABLE 55. Converting Datetime Expression with Format String to String

Database		Syntax	Description
IBM DB2	8	**VARCHAR_FORMAT** (*timestamp_exp*, *fmt_str*)	Converts ***timestamp_exp*** to a string in the format specified by the ***fmt-string***. ***timestamp-exp*** – is an expression that results in a timestamp. The argument must be a timestamp or a string representation of a timestamp that is neither a CLOB nor a LONG VARCHAR. The string expression returns a CHAR or a VARCHAR value whose maximum length is not greater than 254. ***fmt-string*** – is a character constant that contains a template for how the result is to be formatted. The length of the format string must not be greater than 254. The content of format-string can be specified only in full case. Format string must be following: 'YYYY-MM-DD HH24:MI:SS' where YYYY represents a 4-digit year value; MM represents a 2-digit month value (01-12; January=01); DD represents a 2-digit day of the month value (01-31); HH24 represents a 2-digit hour of the day value (00-24; If the hour is 24, the minutes and seconds values are zero.); MI represents a 2-digit minute value (00-59); and SS represents a 2-digit seconds value (00-59). The result of the function is a varying-length character string containing a formatted timestamp expression. In order to choose from a ***timestamp-exp*** some part (such as SECOND, MINUTE and etc.) is used accordingly SECOND, MINUTE and etc. functions. For example: if it is necessary to choose the second part from ***timestamp-exp*** then SECOND functions must be used.
		TO_CHAR (*timestamp_exp*, *fmt_str*)	Returns a character representation of a timestamp that has been formatted using a character template. TO_CHAR is a synonym for VARCHAR_FORMAT.
	<8	**CHAR** (*datetime_exp* [, ISO | USA | EUR | JIS | LOCAL])	Converts ***datetime_exp*** of DATE, TIME and TIMESTAMP data type to a string. If the ***datetime_exp*** is null, the result is the null value. If ***datetime-exp*** is an expression of: •DATE data type then the length of the result is 10. •TIME data type then the length of the result is 8. •TIMESTAMP data type then the length of the result is 26. A ***datetime_exp*** format model can be implemented with expressions of MINUTE, SECOND and etc. functions. For concatenate result of MINUTE function with result of SECOND function it is necessary to convert result of these functions to string, as MINUTE and SECOND functions return integer. "||" in IBM DB2 allows concatenate only two strings. Example of this concatenate is following: **CHAR(MINUTE(CURRENT TIMESTAMP))** || **CHAR(SECOND(CURRENT TIMESTAMP))**.

TABLE 55. Converting Datetime Expression with Format String to String

<table>
<tr><th>Database</th><th>Syntax</th><th colspan="2">Description</th></tr>
<tr><td rowspan="22">MySQL</td><td rowspan="22">DATE_FORMAT
(date, fmt_str)</td><td colspan="2">Converts date to a string in the format specified by the date format fmt_str.

Formats the date value according to the format string (fmt_str). The following specifiers may be used in the format string:</td></tr>
<tr><td>Specifier</td><td>Description</td></tr>
<tr><td>%a</td><td>Abbreviated weekday name (Sun..Sat)</td></tr>
<tr><td>%b</td><td>Abbreviated month name (Jan..Dec)</td></tr>
<tr><td>%D</td><td>Day of the month with English suffix (0th, 1st, 2nd, 3rd, etc.</td></tr>
<tr><td>%d
%e</td><td>Day of the month, numeric (00..31)</td></tr>
<tr><td>%j</td><td>Day of year (001..366)</td></tr>
<tr><td>%m
%c</td><td>Month, numeric (00..12)</td></tr>
<tr><td>%M</td><td>Month name (January..December)</td></tr>
<tr><td>%f</td><td>Microseconds (000000..999999)</td></tr>
<tr><td>%i</td><td>Minutes, numeric (00..59)</td></tr>
<tr><td>%h
%I
%l</td><td>Hour (01..12)</td></tr>
<tr><td>%H
%k</td><td>Hour (00..23)</td></tr>
<tr><td>%p</td><td>AM or PM</td></tr>
<tr><td>%r</td><td>Time, 12-hour (hh:mm:ss followed by AM or PM)</td></tr>
<tr><td>%S
%s</td><td>Seconds (00..59)</td></tr>
<tr><td>%T</td><td>Time, 24-hour (hh:mm:ss)</td></tr>
<tr><td>%U</td><td>Week (00..53), where Sunday is the first day of week</td></tr>
<tr><td>%u</td><td>Week (00..53), where Monday is the first day of week</td></tr>
<tr><td></td><td></td></tr>
<tr><td></td><td></td></tr>
</table>

TABLE 55. Converting Datetime Expression with Format String to String

Database	Syntax	Description	
		%V	Week (01..53), where Sunday is the first day of week, used with %X
		%v	Week (01..53), where Monday is the first day of week, used with %x
		%W	Weekday name (Sunday..Saturday)
		%w	Day of the week (0=Sunday .. 6=Saturday)
		%X	Year for the week, where Sunday is the first day of the week, numeric 4 digits; used with %V
		%x	Year for the week, where Monday is the first day of the week, numeric 4 digits; used with %v
		%Y	Year, numeric, 4 digits
		%y	Year, numeric, 2 digits
		%%	A literal '%'

Conversion of Oracle TO_CHAR(datetime) with format string to MySQL

The Oracle TO_CHAR(***datetime***, ***fmt***) function converts datetime values to a string in the format specified by the fmt option.

MySQL has the DATE_FORMAT function that allows datetime values converting to a string in the specified format.

SQLWays converts the Oracle TO_CHAR function to the MySQL DATE_FORMAT function and converts elements of format string from Oracle to corresponding specifier in MySQL as specified in the following table

TABLE 56. Conversion of Oracle TO_CHAR(datetime) with format string to MySQL

Mapping of datetime format specifiers between MySQL and Oracle		
MySQL	**Oracle (independently from register)**	**Description**
%a	DY	Abbreviated weekday name (Sun..Sat)
%b	MON	Abbreviated month name (Jan..Dec)
%D	–	Day of the month with English suffix (0th, 1st, 2nd, 3rd, etc.)
%d %e	DD	Day of the month, numeric ((00..31) and (0..31))
%j	DDD	Day of year (001..366)
%m %c	MM	Month, numeric ((00..12) and (0..12))
%M	MONTH	Month name (January..December)
%f	–	Microseconds (000000..999999)
%i	MI	Minutes, numeric (00..59)
%h %I %l	HH HH12	Hour ((01..12) and (1..12))
%H %k	HH24	Hour ((00..23) and (0..23))
%p	AM PM	AM or PM
%r	–	Time, 12-hour (hh:mm:ss followed by AM or PM)
%S %s	SS	Seconds ((00..59) and (0..59))
%T	–	Time, 24-hour (hh:mm:ss)
%u	WW IW	Week (00..53), where Monday is the first day of week
%U	–	Week (00..53), where Sunday is the first day of week
%V	–	Week (01..53), where Sunday is the first day of week, used with %X
%v	WW IW	Week (01..53), where Monday is the first day of week, used with %x
%W	DAY	Weekday name (Sunday..Saturday)
%w	–	Day of the week (0=Sunday .. 6=Saturday)

TABLE 56. **Conversion of Oracle TO_CHAR(datetime) with format string to MySQL**

Mapping of datetime format specifiers between MySQL and Oracle		
MySQL	**Oracle (independently from register)**	**Description**
%X	–	Year for the week, where Sunday is the first day of the week, numeric 4 digits; used with %V
%x	–	Year for the week, where Monday is the first day of the week, numeric 4 digits; used with %v
%Y	YYYY SYYYY IYYY	Year, numeric, 4 digits
%y	YY IYY	Year, numeric, 2 digits
–	J	Julian day; the number of days since January 1, 4712 BC.
–	Q	Quarter of year (1, 2, 3, 4; JAN-MAR = 1).
–	RR	Given a year with 2 digits: •If the year is <50 and the last 2 digits of the current year are >=50, then the first 2 digits of the returned year are 1 greater than the first 2 digits of the current year. •If the year is >=50 and the last 2 digits of the current year are <50, then the first 2 digits of the returned year are 1 less than the first 2 digits of the current year.
–	RRRR	Round year. Accepts either 4-digit or 2-digit input. If 2-digit, provides the same return as RR. If you don't want this functionality, then simply enter the 4-digit year.
–	W	Week of month (1-5) where week 1 starts on the first day of the month and ends on the seventh.
–	SSSSS	Seconds past midnight (0 – 86399).
–	X	Local radix character.
–	Y,YYY	Year with comma in the position.
–	YEAR SYEAR	Year, spelled out; "S" prefixes BC dates with "-".
–	YYY	3 digits of year.
–	Y	1 digit of year.
–	IY	2 digits of ISO year.
–	I	1 digit of ISO year.
–	AD A.D.	AD indicator with or without periods.
–	BC B.C.	BC indicator with or without periods.
–	CC SCC	One greater than the first two digits of a four-digit year; "S" prefixes BC dates with "-". For example, '20' from '1900'.
–	D	Day of week (1 – 7).
–	A.M. P.M.	Meridian indicator with periods.
–	TZH	Time zone hour.

TABLE 56. Conversion of Oracle TO_CHAR(datetime) with format string to MySQL

Mapping of datetime format specifiers between MySQL and Oracle		
MySQL	**Oracle (independently from register)**	**Description**
–	TZM	Time zone minute.
–	TZR	Time zone region information.
–	RM	

TABLE 57. Example of Conversion

Oracle	MySQL

create procedure sp_to_char_date_format
as
begin
-- GET ACTUAL TIME AND DATE
select to_char(sysdate,'DD-MON-YYYY:HH24:MI')
from dual;
end;
	create procedure sp_to_char_date_format()
	begin
	-- GET ACTUAL TIME AND DATE
	select DATE_FORMAT(CURRENT_TIMESTAMP, '%e-%M-
	%Y:%H:%i') from dual;
	end;

Returning Substring from String

This subsection describes functions returning a substring of a string, in various databases and their conversion by SQLWays..

TABLE 58. Returning a Substring from String

Database	Syntax	Description
Oracle	**SUBSTR** (*exp1*, *exp2* [,*exp3*])	Returns a substring of a string (***exp1***), beginning from ***exp2*** with length of ***exp3***. •If ***exp2*** is 0, then it is treated as 1. •If ***exp2*** is positive, then Oracle counts from the beginning of string to find the first character. •If ***exp2*** is negative, then Oracle counts backward from the end of string. •If ***exp3*** is omitted, then Oracle returns all characters to the end of string. •If ***exp3*** is less than 1, then a null is returned. **Return type**: the same as ***exp1***. ***exp1*** can be any of the datatypes CHAR, VARCHAR2, NCHAR, NVARCHAR2, CLOB, or NCLOB. Floating-point numbers passed as arguments to SUBSTR are automatically converted to integers.
IBM DB2	**SUBSTR** (*exp1*, *exp2* [,*exp3*])	Returns a substring of a string (***exp1***), beginning from ***exp2*** with length of ***exp3***. If any argument is null, the result is the null value. **Return type:** the same as ***exp1***. ***exp2*** must be an integer between 1 and the length or maximum length of ***exp1***, depending on whether ***exp1*** is fixed-length or varying-length. ***exp3*** is an expression that specifies the length of the result. If specified, ***exp3*** must be a binary integer in the range 0 to n, where n equals (the length attribute of ***exp1***) – ***exp2*** + 1. The default for ***exp3*** is the number of bytes from the byte specified by the ***exp2*** to the last byte of ***exp1*** in the case of character string or binary string or the number of double-byte characters from the character specified by the ***exp2*** to the last character of ***exp1*** in the case of a graphic string. However, if ***exp1*** is a varying-length string with a length less than ***exp2***, the default is zero and the result is the empty string.
MySQL	**SUBSTRING** (*exp1*, *exp2* [,*exp3*]) **SUBSTRING** (*exp1* FROM *exp2* [FOR *exp3*])	Returns a substring of a string (***exp1***), beginning from ***exp2*** with length of ***exp3***. Return type: the same as ***exp1***. •If ***exp2*** is positive, then SUBSTRING function counts from the beginning of string to find the first character. •If ***exp2*** is negative, then SUBSTRING function counts backward from the end of string. (Undocumented). •If ***exp3*** is omitted, then SUBSTRING function returns a substring from string ***exp1*** starting at position ***exp2***.

TABLE 58. Returning a Substring from String

<table>
<tr><th>Database</th><th>Syntax</th><th>Description</th></tr>
<tr><td rowspan="6">Microsoft SQL Server</td><td>SUBSTRING (exp1, exp2, exp3)</td><td colspan="2">Returns a substring of a string (exp1), beginning from exp2 with length of exp3.

exp1 – is a character string, binary string, text, image, a column, or an expression that includes a column.

exp2 – is an integer that specifies where the substring begins.

exp3 – is an integer that specifies the length of the substring (the number of characters or bytes to return).

Return type:
•If expression (exp1) is one of the character data types then function returns character data.
•If expression (exp1) is one of the binary data types then function returns binary data.
•The returned string is the same type as the given expression (exp1) with the exceptions shown in the following table:</td></tr>
<tr><td>given expression (exp1)</td><td>return type</td></tr>
<tr><td>text</td><td>varchar</td></tr>
<tr><td>image</td><td>varbinary</td></tr>
<tr><td>ntext</td><td>nvarchar</td></tr>
<tr><td>RIGHT (char_exp, int_exp)</td><td colspan="2">Returns a substring of a string (char_exp). Returning substring contains int_exp last symbols.

Return type: varchar.

int_exp is an expression that specifies the length of the result.</td></tr>
<tr><td>LEFT (char_exp, int_exp)</td><td colspan="2">Returns a substring of a string (char_exp). Returning substring contains int_exp first symbols.
Return type: varchar.
int_exp is an expression that specifies the length of the result.</td></tr>
</table>

a) Converting the RIGHT function.

SQLWays converts the Microsoft SQL Server RIGHT function to the Oracle SUBSTR function with negative length parameter.

TABLE 59. Converting RIGHT function

Microsoft SQL Server	Oracle
RIGHT (‘ABCDEF’,5)	SUBSTR (‘ABCDEF’,-5);

b) Converting the LEFT function.

SQLWays converts the Microsoft SQL Server LEFT function to the Oracle SUBSTR function with the start position 1 and the same length as defined in the LEFT function.

TABLE 60. Converting LEFT function

Microsoft SQL Server	Oracle
LEFT (‘ABCDEF’,5)	SUBSTR (‘ABCDEF’,1,5);

Returning String in Uppercase

This subsection describes functions which return string in uppercase in various databases and thier conversion by SQLWays.

TABLE 61. Returning string in uppercase

Database	Syntax	Description
Oracle	**UPPER** (*exp*)	Returns expression (***exp***) with all letters uppercase. ***exp*** - can be constant or variable containing constant of the CHAR, VARCHAR2, NCHAR, NVARCHAR2, CLOB, or NCLOB data type. **Return type**: the same as the data type of **exp**.
MySQL	**UPPER** (*exp*) **UCASE** (*exp*)	Returns expression (***exp***) with all letters uppercase. ***exp*** – can be constant or variable containing constant. **Return type**: the same as the data type of **exp**.

Returning Position of Substring in String

This subsection describes functions that return the position of a substring in the specified string in various databases

and their conversion by SQLWays.

TABLE 62. Returning Position of Substring in String

Database	Syntax	Description
Microsoft SQL Server	CHARINDEX (*substring*, *string* [,*start_location*])	The CHARINDEX function returns the starting position of a substring in the specified character string. ***Substring*** is an expression containing the sequence of characters to be found. ***String*** is a character string or expression searched for the specified sequence. ***Start_location*** is the character position to start searching for the substring. If ***start_location*** is not given, negative or zero, the search starts at the beginning of the string. If either of expressions is NULL, CHARINDEX returns NULL when the database compatibility level is 7.0 or later. If the database compatibility level is 6.5 or earlier, CHARINDEX returns NULL only when both expressions are NULL. If substring is not found within the string, CHARINDEX returns 0. **Return type**: int **Example**: Returns the position of point in the *title* variable CHARINDEX('.', @title)

TABLE 62. Returning Position of Substring in String

Database	Syntax	Description
Oracle	INSTR (*string*, *substring* [,*start_location* [,*occurrence*]])	The INSTR function returns the starting position of a substring in the specified character string. INSTR calculates strings using characters as defined by the input character set. ***String*** is an expression containing character string, searched for the specified sequence. ***Substring*** is an expression containing the sequence of characters to be found. ***start_location*** is a nonzero integer indicating the character of string to start searching for *substring*. If position is negative, then INSTR counts and searches backward from the end of *string*. The default value is 1. ***occurrence*** is an integer indicating which occurrence of *string* INSTR should search for. The value of occurrence must be positive. The default value is 1. If substring does not appear *occurrence* times after the *position* character of *string*, then the return value is 0. Both *string* and *substring* can be any of the datatypes CHAR, VARCHAR2, NCHAR, NVARCHAR2, CLOB, or NCLOB. If either of expressions is NULL, INSTR returns NULL **Return type:** NUMBER. **Example**: The following sample returns 7 - position of first comma in the constant string. INSTR('Austin,Boston,Cardiff', ',')

TABLE 62. Returning Position of Substring in String

Database	Syntax	Description
Sybase Adaptive Server Anyware	LOCATE (*string*, *substring* [,*start_location*])	Returns the starting position of a substring in the specified character string. ***string*** is the string to be searched. ***substring*** is the string to be searched for. The maximum length is the 255 bytes. ***start_location*** is the character position to start searching for the substring. The first character is position 1. If ***start_location*** is negative, the LOCATE function returns the last matching string offset rather than the first. A negative offset indicates how much of the end of the string is to be excluded from the search. •If ***start_location*** is specified, the search starts at that offset into the string. •If ***string*** is given as substring, the function returns a NULL value. • If ***string*** is not found, 0 is returned. Searching for a zero-length string will return 1. •If any of the arguments are NULL, the result is NULL.
MySQL	INSTR(*string*, *substring*)	Returns the starting position of a substring (***substring***) in the specified character string (***string***). INSTR is an equivalent of the LOCATE function with two arguments (when ***start_location*** is ommitted).
	LOCATE(*substring*, *string* [,*start_location*])	Returns the starting position of a substring (***substring***) in the specified character string (***string***). If ***start_location*** is specified then LOCATE returns the position of a substring (substring) in string (***string***), starting at position ***start_location***. •If ***substring*** is not in string then LOCATE returns 0.

Conversions of Microsoft SQL Server CHARINDEX to Oracle

The CHARINDEX function is used in Microsoft SQL Server to retrieve the position of a substring in the specified string. Oracle uses with this purpose the INSTR function. These functions have a similar syntax and differ in the parameter order.

SQLWays converts the Microsoft SQL Server function CHARINDEX to the Oracle function INSTR.

TABLE 63. Examples of Conversions of Microsoft SQL Server CHARINDEX to Oracle

Microsoft SQL Server	Oracle

CREATE PROCEDURE sql_sp_charindex
as
begin
DECLARE @a VARCHAR(10)
DECLARE @b VARCHAR(10)
charindex (@a,@b)
end;
	CREATE OR REPLACE PROCEDURE sql_sp_charindex
	AS
	v_a VARCHAR2(10);
	v_b VARCHAR2(10);
	BEGIN
	INSTR(v_b,v_a);
	end;

CREATE PROCEDURE sql_sp_charindex2
as
begin
DECLARE @a VARCHAR(10)
DECLARE @b VARCHAR(10)
charindex (@a,@b,1)
end;
	CREATE OR REPLACE PROCEDURE sql_sp_charindex2
	AS
	v_a VARCHAR2(10);
	v_b VARCHAR2(10);
	BEGIN
	INSTR(v_b,v_a,1);
	end;

Removing (or Trimming) Characters from a String

This section describes functions which remove (trim) characters from the specified string in various databases and their conversion by SQLWays.

TABLE 64. Removing (or Trimming) Characters from a String

Database	Syntax	Description
Sybase Adaptive Server Anyware	TRIM (*exp*)	TRIM removes leading and trailing blanks from ***exp*** string.
	RTRIM (*exp*)	RTRIM removes trailing blanks from ***exp*** string.
	LTRIM (*exp*)	LTRIM removes leading blanks from ***exp*** string.
Microsoft SQL Server	RTRIM (*exp*)	RTRIM removes trailing blanks from ***exp*** string. **Return type**: VARCHAR ***exp*** is an expression of character or binary data. exp can be a constant, variable, or column. ***exp*** must be of a data type that can be implicitly converted to VARCHAR.
	LTRIM (*exp*)	LTRIM removes leading blanks from ***exp*** string. **Return type**: VARCHAR ***exp*** is an expression of character or binary data. exp can be a constant, variable, or column. ***exp*** must be of a data type that can be implicitly converted to VARCHAR.

Conversion of Sybase Adaptive Server Anyware TRIM to Microsoft SQL Server

The Sybase Adaptive Server Anyware TRIM function removes leading and trailing blanks from the specified string. Microsoft SQL Server does not support the TRIM function, but RTRIM and LTRIM functions can be used to remove both leading and trailing blank characters.

SQLWays converts Sybase Adaptive Server Anyware TRIM to Microsoft SQL Server RTRIM with LTRIM, as shown in the example below.

TABLE 65. Example of the Conversion of Sybase Adaptive Server Anyware TRIM to Microsoft SQL Server

Sybase Adaptive Server Anyware	Microsoft SQL Server

create procedure asa_sp_trim ( )
begin
select trim (' hello ');
end;
	create procedure asa_sp_trim AS
	begin
	select rtrim(ltrim(' hello '))
	end

Returning Information about Database and Current Connection

- Returning Information about Current User

Returning Information about Current User

This subsection describes built-in functions that return the name of the current user connected to the database.

TABLE 66. Returning Information about Current User

Database	Syntax	Description
Oracle	USER	Returns the name of the user currently connected to the database. **Returned type**: varchar2 Remarks: Cannot be used as a condition for the CHECK constraints.
Informix	USER	The USER operator returns the name of the user currently connected to the database. **Returned type:** VARCHAR.

Replace NULL value functions

- Returning the first non-NULL expression
- Returning one of expressions depending on whether check expression is NULL or NOT NULL

Returning the first non-NULL expression

This chapter describes functions that return the first non-null expression (or replace NULL value) in various databases and their conversion by SQLWays.

Note. These functions differ from functions like IFNULL e.g. that check the first expression for NULL and return either second or third expression.

TABLE 67. Returning the first non-NULL expression

Database	Syntax	Description
Oracle	NVL (exp1, exp2)	Replaces NULL with the specified replacement value. Returns the same type as ***exp1***. If ***exp1*** is NULL, then NVL returns ***exp2***. If ***exp1*** is NOT NULL, then NVL returns exp1. The arguments ***exp1*** and ***exp2*** can have any data type. If expressions' data types are different, then Oracle converts ***exp2*** to the data type of ***exp1*** before comparing them.
	COALESCE (exp1, exp2 [,expN]…)	Returns the first non-null expression in the expression list. ***exp1***…***expN*** – are expressions of any data type If all arguments are NULL, COALESCE returns NULL.
Microsoft SQL Server	ISNULL (exp1, exp2)	Replaces NULL with the specified replacement value. Returns the same type as ***exp1***. If ***exp1*** is NULL, then ISNULL returns ***exp2***. If ***exp1*** is NOT NULL, then ISNULL returns ***exp1***. The arguments ***exp1*** and ***exp2*** can have any data type, but ***exp2*** must have the same type as ***exp1***.
	COALESCE (exp1, exp2 [,expN]…)	Returns the first non-null expression in the expression list. ***exp1***…***expN*** – are expressions of any data type If all arguments are NULL, COALESCE returns NULL. All expressions must be of the same type or must be implicitly convertible to the same type.

TABLE 67. Returning the first non-NULL expression

Database	Syntax	Description
MySQL	IFNULL (exp1, exp2)	Replaces NULL with the specified replacement value. Returns the same type as ***exp1***. If ***exp1*** is NULL, then IFNULL returns ***exp2***. If ***exp1*** is NOT NULL, then IFNULL returns ***exp1***.
	COALESCE (exp1, exp2 [,expN]…)	Returns the first non-null expression in the expression list. ***exp1***…***expN*** – are expressions of any data type If all arguments are NULL, COALESCE returns NULL.
IBM DB2	COALESCE (exp1, exp2 [,expN]…)	Returns the first non-null expression in the expression list. ***exp1***…***expN*** – are expressions of any data type If all arguments are NULL, COALESCE returns NULL. The selected argument is converted, if necessary, to the attributes of the result.
	VALUE (exp1, exp2 [,expN]…)	Returns the first non-null ***exp*** in the expression list. VALUE is a synonym for COALESCE.

TABLE 67. Returning the first non-NULL expression

Database	Syntax	Description
Sybase Adaptive Server Anywhere	COALESCE (exp1, exp2 [,expN]…)	Returns the first non-null expression in the expression list. ***exp1***…***expN*** – are expressions of any data type If all arguments are NULL, COALESCE returns NULL.
	ISNULL (exp1, exp2 [,expN]…)	Returns the first non-null ***exp*** in the expression list. ISNULL is a synonym for COALESCE.
Sybase Adaptive Server Enterprise	ISNULL (exp1, exp2)	Returns the first non-null expression in the expression list. The arguments ***exp1*** and ***exp2*** can have any data type. The data types of the expressions must convert implicitly, or must use the convert function.
	COALESCE (exp1, exp2 [,expN]…)	Returns the first non-null expression in the expression list. ***exp1***…***expN*** – are expression of any data type If all arguments are NULL, COALESCE returns NULL.

Returning one of expressions depending on whether check expression is NULL or NOT NULL

This chapter describes functions used in various databases to return one of two expressions depending on whether a specified check expression is NULL or NOT NULL, and their conversion by SQLWays.

Note. These functions differ from functions like ISNULL, COALESCE etc. that return the first non-null value.

TABLE 68. Returning one of expressions depending on whether check expression is NULL or NOT NULL

Database	Syntax	Description
Oracle	NVL2 (exp1, exp2, exp3)	NVL2 lets you determine the returned value depending on whether a specified expression is NULL or NOT NULL. If exp1 is NULL value, then the value of exp2 is returned. If exp1 is not NULL, the value of exp3 is returned. The argument exp1 can have any data type. If the data types of exp2 and exp3 are different, then Oracle converts exp3 to the data type of exp2 before comparing them unless exp3 is a NULL constant The data type of the return value is always the same as the data type of exp2, unless exp2 is character data, in this case the returned value's data type is VARCHAR2.
Sybase Adaptive Server Anywhere	IFNULL (exp1, exp2 [,exp3])	IFNULL lets you determine the returned value depending on whether a specified expression is NULL or NOT NULL. If exp1 is NULL value, then the value of exp2 is returned. If exp1 is not NULL, the value of exp3 is returned. If exp1 is not NULL and there is no exp3, NULL is returned.
Microsoft SQL Server	-	Not supported.

Conversion of Sybase ASA IFNULL to Microsoft SQL Server

The Sybase Adaptive Server Anyware IFNULL function returns one of two expressions, depending on whether a specified check expression is NULL or NOT NULL.

Microsoft SQL Server does not support IFNULL. The Microsoft SQL Server searched CASE expression can be used to implement the Sybase Adaptive Server Anyware IFNULL function.

SQLWays changes the Sybase Adaptive Server Anyware IFNULL function to the Microsoft SQL Server searched CASE expression that tests the first expression for NULL.

TABLE 69. Example of the conversion IFNULL with two expressions

Sybase Adaptive Server Anywhere	Microsoft SQL Server

create procedure asa_sp_ifnull (par int,
par2 int)
begin
declare res int;
SET res = IFNULL (par, par2);
end;
	create procedure asa_sp_ifnull @par INT,@par2 INT
	AS begin
	declare @res INT
	SET @res = CASE WHEN @par IS NULL THEN @par2 END
	end

TABLE 70. Example of the conversion IFNULL with three expressions

Sybase Adaptive Server Anywhere	Microsoft SQL Server

create procedure asa_sp_ifnull (par int,
par2 int, par3 int)
begin
declare res int;
SET res = IFNULL (par, par2, par3);
end;
	create procedure asa_sp_ifnull @par INT,@par2 INT, @par3 INT
	AS begin
	declare @res INT
	SET @res = CASE WHEN @par IS NULL THEN @par2 ELSE @par3 END
	end

Techniques

This section describes various techniques for writing applications in various databases and their conversion by SQLWays.

- Returning Non-Table Data as Result Set (Dummy Tables)
- Returning Result Sets from Procedure

Returning Non-Table Data as Result Set (Dummy Tables)

This subsection describes techniques to return non-table data as a result set in various databases and their conversion by SQLWays.

Sometimes it is required to return information that not contained in the database tables, as a result set. For example, returning the current user, current date, constants or arithmetic expressions without table columns.

TABLE 71. Returning Non_Table Data and Result Set (Dummy Tables)

Database	Syntax	Description
Sybase Adaptive Server Anywhere	SELECT select_list [FROM [SYS.]DUMMY]	The DUMMY table can be used in Sybase Adaptive Server Anywhere to return non-table result set. DUMMY is a read only system table that has one row and one column (dummy_col integer NOT NULL). Use of FROM DUMMY in the FROM clause is optional. If no table is specified in the FROM clause, the table is assumed to be SYS.DUMMY.
Microsoft SQL Server	SELECT select_list	The SELECT statement without the FROM clause and a variable assignment (@var1=col1) returns non-table result set in Microsoft SQL Server.

Non-Table Result Set Conversion from Sybase Adaptive Server Anywhere to Microsoft SQL Server

The DUMMY system table can be used in order to return non-table data as a result set in Sybase Adaptive Server Anywhere.

The SELECT statement without the FROM clause and a variable assignments returns non-table result set in Microsoft SQL Server.

SQLWays removes the FROM [SYS.]DUMMY clause when converting Sybase Adaptive Server Anywhere non-table result sets to Microsoft SQL Server.

Examples:

TABLE 72. Sybase Adaptive Server Anywhere - Microsoft SQL Server Conversion

Sybase Adaptive Server Anywhere	Microsoft SQL Server
SELECT 1 FROM SYS.DUMMY	SELECT 1
SELECT 1 FROM DUMMY	SELECT 1

Returning Result Sets from Procedure

This subsectin describes how to return a result set to the client application from a procedure in various databases.

- Returning Result Set to Client
- Result Set Conversion from Microsoft SQL Server to IBM DB2

Returning Result Set to Client

The set of rows returned by a query is called the result set. The result set can consist of zero, one, or multiple rows, depending on how many rows meet your search criteria.

Returning result sets means that the client application will be able to retrieve and process rows returned by the procedure.

TABLE 73. Returning Result Sets from Procedure

Database	Syntax	Description
Microsoft SQL Server	create procedure proc_name as begin select_statement end	Standalone SELECT statement in a Microsoft SQL Server procedure returns the result set to the client application. For example, “SELECT * FROM Products” generates a default result set of all the products in the Products table. SQL Server sends this result set to the client.
IBM DB2	create procedure proc_name () LANGUAGE SQL [DYNAMIC RESULT SETS N begin declare cursor_name CURSOR WITH RETURN [TO CALLER | TO CLIENT] FOR select_statement; open cur1; end	Cursor, which is declared with the WITH RETURN attribute and which isn’t closed in the procedure body, returns the result set to the client. WITH RETURN indicates that the cursor is intended for use as a result set from a stored procedure. DYNAMIC RESULT SETS N – Indicates the estimated upper bound of returned result sets for the stored procedure. Within an SQL procedure, cursors, declared using the WITH RETURN clause, are still open when the SQL procedure ends, define the result sets from the SQL procedure. All other open cursors in an SQL procedure are closed when the SQL procedure ends. The default for all cursors is WITH RETURN TO CALLER. TO CALLER – Specifies that the cursor can return a result set to the caller. For example, if the caller is another stored procedure, the result set is returned to that stored procedure. If the caller is a client application, the result set is returned to the client application. TO CLIENT – Specifies that the cursor can return a result set to the client application. This cursor is invisible to any intermediate nested procedures.

Result Set Conversion from Microsoft SQL Server to IBM DB2

A standalone SELECT statement returns the result set to the client application from a Microsoft SQL Server procedure.

In order to return a result set from a DB2 procedure, you have to declare a cursor with the WITH RETURN option, and open the cursor.

SQLWays converts Microsoft SQL Server standalone SELECT statements to DECLARE WITH RETURN cursors and OPEN statements in IBM DB2.

TABLE 74. Example of Result Set Conversion from Microsoft SQL Server to IBM DB2

Microsoft SQL Server	IBM DB2
create procedure SalesOrder_Test as begin select * from Customers end	create procedure SalesOrder_Test ( ) LANGUAGE SQL DYNAMIC RESULT SETS 1 begin DECLARE cur1 CURSOR WITH RETURN FOR select * from Customer s; open cur1; end

Migrating to IBM DB2

This chapter is devoted to migrating databases to IBM DB2 server, an object-relational database management system.

IBM DB2 Data Types

This section describes the data types supported by IBM DB2.

- CHAR, VARCHAR and LONG VARCHAR
- GRAPHIC, VARGRAPHIC and LONG VARGRAPHIC
- BIGINT, INTEGER and SMALLINT
- FLOAT, REAL and DOUBLE
- DECIMAL or NUMERIC
- DATE, TIME and TIMESTAMP
- BLOB, CLOB and DBCLOB
- DATALINK

CHAR, VARCHAR and LONG VARCHAR

CHAR, VARCHAR and LONG VARCHAR data types are used to store character data. A character data is a sequence of bytes, the length of the character value is the number of bytes in this sequence. If the length is zero, the value is called the *empty string*. This value should not be confused with the NULL value.

CHAR[(*n*)] [**FOR BIT DATA**]
CHARACTER[(*n*)] [**FOR BIT DATA**]

A fixed-length character data. The range of *n* is 1 to 254 characters. When *n* is not specified the dafault value is 1.

When CHAR values are stored, they are right-padded with spaces to the specified length. When CHAR values are retrieved, trailing spaces are removed.

FOR BIT DATA option specifies that the contents of the column are to be treated as bit (binary) data. During data exchange with other systems, code page conversions are not performed.

VARCHAR(*n*) [**FOR BIT DATA**]
CHARACTER VARYING(*n*) [**FOR BIT DATA**]
CHAR VARYING(*n*) [**FOR BIT DATA**]

A variable-length character string. The range of *n* is 1 to 32 672 characters.

Value is stored in database using only as many characters as it contains, plus a few bytes to record the length. When the value is stored trailing spaces are removed.

FOR BIT DATA option specifies that the contents of the column are to be treated as bit (binary) data. During data exchange with other systems, code page conversions are not performed.

LONG VARCHAR [**FOR BIT DATA**]

A variable-length character string. The maximum length is 32 700 characters.

FOR BIT DATA option specifies that the contents of the column are to be treated as bit (binary) data. During data exchange with other systems, code page conversions are not performed.

GRAPHIC, VARGRAPHIC and LONG VARGRAPHIC

GRAPHIC, VARGRAPHIC and LONG VARGRAPHIC data types are used to store graphic data. A graphic data is a sequence of bytes that represents double-byte character data, the length of the graphic value is the number of double-byte characters in this sequence. If the length is zero, the value is called the *empty string*. This value should not be confused with the NULL value.

For a Unicode database, the GRAPHIC data types are considered to be equivalent to CHAR data types.

GRAPHIC[(*n*)]

A fixed-length graphic string. The range of *n* is 1 to 127 double-byte characters. When *n* is not cpecified the default value is 1.

VARGRAPHIC(*n*)

A variable-length graphic string. The range of *n* is 1 to 16 336 double-byte characters.

LONG VARGRAPHIC

A variable-length graphic string. The maximum length is 16 350 double-byte characters.

BIGINT, INTEGER and SMALLINT

BIGINT

An 8-byte integer with a precision of 19 digits. The range of big integers is -9 223 372 036 854 775 808 to +9 223 372 036 854 775 807.

INTEGER
INT

A 4-byte integer with a precision of 10 digits. The range of large integers is -2 147 483 648 to +2 147 483 647.

SMALLINT

A 2-byte integer with a precision of 5 digits. The range is -32 768 to 32 767.

DECIMAL or NUMERIC

DECIMAL[(p[,s])]
DEC[(p[,s])]
NUMERIC[(p[,s])]
NUM[(p[,s])]

A packed decimal number with an implicit decimal point. The position of the decimal point is determined by the precision (*p*) and the scale (*s*) of the number.

The *p* specifies the total number of the digits. The range of the precision is 1 to 31 digits.

The *s* is the number of digits in the fractional part of the number. It cannot be negative or greater than the precision. The range of scale is 0 to the precision of the number.

If *p* is omitted, the default is 5.

If *s* is omitted the default is 0, that is, values will have *no* decimal point or fractional part.

The range of a decimal variable or the numbers in a decimal column is -*n* to +*n*, where the absolute value of *n* is the largest number that can be represented with the applicable precision and scale. The maximum range is -10^31+1 to 10^31-1.

FLOAT, REAL and DOUBLE

FLOAT[(*n*)]

FLOAT data type is used to store single-precision and double-precision floating-point numbers.

A single-precision floating-point number is a 32-bit approximation of a real number. The number can be zero or can range from -3.402E+38 to -1.175E-37, or from 1.175E-37 to 3.402E+38. The range of *n* is 1 to 24. IBM DB2 internally represents the single-precision FLOAT data type as the REAL data type.

A double-precision floating-point number is a 64-bit approximation of a real number. The number can be zero or can range from -1.79769E+308 to -2.225E-307, or from 2.225E-307 to 1.79769E+308. The range of *n* is 25 to 53. IBM DB2 internally represents thedouble-precision FLOAT data type as the DOUBLE [PRECISION] data type.

If *n* is not specified the default value is 53.

REAL

A single-precision floating-point number.

DOUBLE [**PRECISION**]

A double-precision floating-point number.

DATE, TIME and TIMESTAMP

DATE

The DATE data type stores the date information (year, month, day). The supported range is January 1, 0001 to December 31, 9999.

TIME

The TIME data type stores the time information (hours, minutes, seconds). The supported range is 00 hours 00 minutes 00 seconds to 24 hours 00 minutes 00 seconds.

If the hour is 24, the minute and second specifications are zero.

TIMESTAMP

The TIMESTAMP data type stores the date and time information (year, month, day, hour, minute, second, and microsecond). The supported range is January 1, 0001 00 hours 00 minutes 00 seconds 000000 microseconds to December 31, 9999 24 hours 00 minutes 00 seconds 000000 microseconds.

BLOB, CLOB and DBCLOB

BLOB(*n*[K|M|G])

A BLOB (binary large object) is a variable-length binary string that can be up to 2 gigabytes (2 147 483 647 bytes) long.

If only *n* by itself is specified, the value of *n* is the maximum length. If K, M or G is specified the maximum length is 1 024, 1 048 576 or 1 073 741 824 times *n*, and the maximum value for *n* is 2 097 152, 2 048 or 2, respectively.

Binary large objects are used to hold data, such as pictures, video and audio. Like FOR BIT DATA character strings, BLOB strings are not associated with a code page.

CLOB(*n*[K|M|G])

A CLOB (character large object) is a variable-length character string that can be up to 2 gigabytes (2 147 483 647 bytes) long. If the length is zero, the value is called the *empty string*. This value should not be confused with the null value.

If only *n* by itself is specified, the value of *n* is the maximum length. If K, M or G is specified the maximum length is 1 024, 1 048 576 or 1 073 741 824 times *n*, and the maximum value for *n* is 2 097 152, 2 048 or 2, respectively.

Character large objects are used to store large SBCS's (single-byte character sets) character-based data such as formatted text, HTML and XML documents.

DBCLOB(*n*[K|M|G])

A DBCLOB (double-byte character large object) is a variable-length graphic string of double-byte characters that can be up to 1 073 741 823 characters long.

If only *n* by itself is specified, the value of *n* is the maximum length. If K, M or G is specified the maximum length is 1 024, 1 048 576 or 1 073 741 824 times *n*, and the maximum value for *n* is 1 048 576, 1 024 or 1, respectively.

Double-byte character large objects are used to store large DBCS's (double-byte character sets) character-based data.

DATALINK

DATALINK

A DATALINK value is a character string, which references an object stored external to a database. The DATALINK values encode the file and the filename in terms of a Uniform Resource Locator (URL).

A URL is a text string of the general format:

http://www.ispirer.com/datalinks/datalinks.txt

IBM DB2 Functions and Expressions

This section describes functions and expressions that can be used in IBM DB2 SQL statements.

- IBM DB2 Expressions
  - Simple CASE Expression
- IBM DB2 Functions
  - COALESCE
  - LEFT
  - LENGTH
  - RIGHT
- IBM DB2 Special Registers
  - CURRENT TIMESTAMP

IBM DB2 Expressions

An expression is a combination of one or more values, operators, and SQL functions that evaluate to a value. An expression generally assumes the data type of its components.

- Simple CASE Expression

Simple CASE Expression

Syntax

CASE expr
WHEN when_expr **THEN** result_expr
{**WHEN** when_expr **THEN** result_expr}
[**ELSE** else_expr]
END

Simple CASE evaluates *input_expr*, and then, in the order specified, evaluates *input_expr = when_expr* for each WHEN clause. Simple CASE returns the *result_expr* of the first WHEN clause that evaluates to TRUE. If *no* match is found, CASE returns the *else_expr* if an ELSE clause is specified, or a NULL value if *no* ELSE clause is specified.

Example

This example uses a CASE expression to list the full name of the division to which each employee belongs.

```
SELECT EMPNO, LASTNAME,
CASE WORKDEPT
WHEN 'A' THEN 'Administration'
WHEN 'B' THEN 'Human Resources'
WHEN 'C' THEN 'Accounting'
WHEN 'D' THEN 'Design'
WHEN 'E' THEN 'Operations'
END
FROM EMPLOYEE;
```

Equivalents in other databases

TABLE 75. Equivalents in other databases

Oracle	DECODE expression
Microsoft SQL Server	Simple CASE expression

IBM DB2 Functions

A function is a relationship between a set of input data values and a set of result data values. Functions can either be built-in or user-defined.

Built-in functions are provided with the database manager, while user-defined functions are used to extend the function of the database system by users.

- COALESCE
- LEFT
- LENGTH
- RIGHT

COALESCE

Syntax

COALESCE(expr, expr2 {, expr })

COALESCE returns the first argument that is not NULL. The arguments are evaluated in the order in which they are specified. This function is usually used to replace NULL values with a not NULL value.

Example

This example selects title and price for all books. If the price for a given title is NULL, the price shown is 0.00.

```
SELECT title, COALESCE(price, 0.00) AS price
FROM titles;
```

TABLE 76. Equivalents in other databases

Oracle	NVL function
Microsoft SQL Server	COALESCE and ISNULL functions

LEFT

Syntax

LEFT(char_expr, int_expr)

LEFT returns int_expr leftmost characters of char_expr.

Example

This example returns the five leftmost characters of each book title.
SELECT LEFT(title, 5)
FROM TITLES;
Here is the result set:

_____
The B
Cooki
You C
The G
The P

Equivalents in other databases

TABLE 77. Equivalents in other databases

Oracle	SUBSTR function
Microsoft SQL Server	LEFT function

LENGTH

Syntax

LENGTH(*expression*)

The LENGTH function returns the length of a value. The argument can be an expression that returns a value of any built-in data type.

If the argument is a string, the function returns the number of characters in the string. The length of fixed-length strings includes blanks. The length of a varying-length string is the actual length, not the maximum length.

Example

This example returns the company name and the number of characters in the name for each company.

```
SELECT CompanyName, LENGTH(CompanyName)
FROM CUSTOMERS
```

Here is the result set:

```
CompanyName
_________________
Consolidated Holdings      21
Eastern Connection         18
France restauration        19
Great Lakes Food Market    23
Island Trading             14
Let's Stop N Shop          17
```

Equivalents in other databases

TABLE 78. Equivalents in other databases

Oracle	LENGTH function
Microsoft SQL Server	LEN function

RIGHT

Syntax

RIGHT(*char_expr*, *int_expr*)

RIGHT returns *int_expr* rightmost characters of *char_expr*.

Example

This example returns the five rightmost characters of each author's first name.
SELECT RIGHT(au_fname, 5)
FROM AUTHORS
Here is the result set:

```
_____________________
lbert
Ann
Anne
chael
ichel
```

Equivalents in other databases

TABLE 79. Equivalents in other databases

Oracle	SUBSTR function
Microsoft SQL Server	RIGHT function

IBM DB2 Special Registers

A special register is a variable that defined for an application process by the database manager and can be referenced in SQL statements.

- CURRENT TIMESTAMP

CURRENT TIMESTAMP

Syntax

CURRENT TIMESTAMP

The CURRENT TIMESTAMP special register specifies a timestamp that is based on a reading of time-of-day clock when the SQL statement is executed. If this special register is used more than once within a single SQL statement, all values are based on a single clock reading.

Example

This example inserts a row into the CUSTOMERS table. The value of the second column is a timestamp that indicates when the row was inserted.

```
INSERT INTO CUSTOMERS VALUES ('ABC Corp', CURRENT TIMESTAMP);
```

Equivalents in other databases

TABLE 80. Equivalents in other databases

Oracle	SYSDATE function
Microsoft SQL Server	GETDATE function

IBM DB2 LOAD Command

This section describes the IBM DB2 LOAD utility, which moves data from files, named pipes, or devices into a DB2 table.

The load utility is capable of efficiently moving large quantities of data into tables. The utility can handle all data types, including large objects (LOBs) and user-defined types (UDTs). The load utility is faster than the import utility, because it writes formatted pages directly into the database, while the import utility performs SQL INSERTs. The load utility does not fire triggers, and does not perform referential or table constraints checking (other than validating the uniqueness of the indexes).

LOAD Command Options

Syntax

LOAD [CLIENT] FROM *filename* | *pipename* | *device* **OF** *filetype*
LOBS FROM *lobpath* **MODIFIED BY** *filetype_mod* **METHOD** *load_method*
[**SAVECOUNT** *n*] [**ROWCOUNT** *n*] [**WARNINGCOUNT** *n*] [**MESSAGES** *message_file*]
INSERT | REPLACE | RESTART | TERMINATE INTO *table_name* [*(column {, column})*]

Options

- **CLIENT** - Specifies that the data to be loaded resides on a remotely connected client. If this option is not specified the data must reside on the server.

The CLIENT option is available in IBM DB2 7.1 or later.

Example

This example uses the LOAD command to load an ASCII text file to a IBM DB2 table.
LOAD FROM customers.txt OF del MESSAGES customers.log INSERT INTO customers

IBM DB2 Version Differences

This section describes differences between IBM DB2 8.x, 7.x, 6.x and 5.x versions.

- LOAD Command

LOAD Command

The IBM DB2 LOAD utility moves data from files, named pipes, or devices into a IBM DB2 table. The load utility is faster than the import utility, because it writes formatted pages directly into the database, while the import utility performs SQL INSERTs.

Options

- **CLIENT**

Requirements: IBM DB2 7.1 or later

Specifies that the data to be loaded resides on a remotely connected client. If this option is not specified the data must reside on the server.

For example, the following command allows you to load an ASCII text file located on the client to a remote IBM DB2 database.

```
LOAD CLIENT FROM customers.txt OF del MESSAGES customers.log INSERT INTO customers
```


Migrating to Oracle

This chapter is devoted to migrating databases to Oracle server, an object-relational database management system.

- Oracle Data Types
- Oracle Functions and Expressions
- Oracle Reserved Words
- Oracle Version Differences

Oracle Data Types

This section describes the data types supported by Oracle.

- CHAR, NCHAR, VARCHAR2 and NVARCHAR2
- NUMBER and FLOAT
- DATE and TIMESTAMP
- INTERVAL YEAR TO MONTH and INTERVAL DAY TO SECOND
- LONG, RAW and LONG RAW
- BLOB, CLOB, NCLOB and BFILE
- ROWID and UROWID

CHAR, NCHAR, VARCHAR2 and NVARCHAR2

Character data types store character data in the database character or national character set.

CHAR[(*n* [**BYTE** | **CHAR**])]

Fixed-length character data of length *n* bytes or characters. The maximum length is 2000 bytes, the default value is 1.

BYTE indicates that the length semantics for the column is byte. CHAR indicates that the length semantics for the column is character. The default is BYTE. CHAR and BYTE qualifiers are not available prior to Oracle Version 9.

NCHAR[(*n*)]

Fixed-length character data of length *n* characters or bytes, depending on the national character set. The maximum length is determined by the number of bytes required to store each character, with an upper limit of 2000 bytes. The default value is 1.

VARCHAR2(*n* [**BYTE** | **CHAR**])

Variable-length character data with length of *n* characters. The maximum length is 4000 bytes. You must specify the maximum length for VARCHAR2 columns.

BYTE indicates that the length semantics for the column is byte. CHAR indicates that the length semantics for the column is character. The default is BYTE. CHAR and BYTE qualifiers are not available prior to Oracle Version 9.

NVARCHAR2(*n*)

Variable-length character data having maximum length *n* characters or bytes, depending on the national character set. The maximum length is determined by the number of bytes required to store each character, with an upper limit of 4000 bytes. You must specify the maximum length for NVARCHAR2 columns.

NUMBER and FLOAT

NUMBER(p,s)

The NUMBER data type stores zero, positive, and negative fixed and floating point numbers with 38 digits of precision.

p is the precision, or the total number of digits. The precision can range from 1 to 38.

s is the scale, or the number of digits to the right of the decimal point. The scale can range from -84 to 127.

To specify integers use the following form: NUMBER(p) or NUMBER(p,0).

FLOAT[(p)]

Oracle supports the ANSI data type FLOAT. Therefore, specifying a floating-point number you can use FLOAT data type instead of NUMBER data type.

p is the binary precision that can range from 1 to 126. If *p* is not specified the default value is binary 126. To convert from binary to decimal precision, multiply *p* by 0.30103. To convert from decimal to binary precision, multiply the decimal precision by 3.32193. 126 digits of binary precision is roughly equivalent to 38 digits of decimal precision.

DATE and TIMESTAMP

DATE

The DATE data type stores date and time information. For each DATE value, Oracle stores the following information: year, month, day, hour, minute, and second.

The date value can be specified as an ANSI date literal, an Oracle date literal or can be converted from a character or numeric value with the TO_DATE function. The ANSI date literal contains *no* time portion, and must be specified in the format 'YYYY-MM-DD'. The default date format for an Oracle date literal can be changed by the initialization parameter NLS_DATE_FORMAT.

Date data can range from January 1, 4712 BC to December 31, 9999.

If a date value is specified without a time component, then the default time is 12:00:00 AM. If a date value is specified without a date, then the default date is the first day of the current month.

TIMESTAMP[(fractional_seconds_precision)]

The TIMESTAMP data type is an extension of the DATE data type. For each TIMESTAMP value, Oracle stores the following information: year, month, day, hour, minute, second and fraction of second.

fractional_seconds_precision optionally specifies the number of digits in the fractional part of second and can be a number in the range 0 to 9. The default is 6.

The TIMESTAMP data type is available in Oracle 9i Release 1 (9.0.1) or later.

INTERVAL YEAR TO MONTH and INTERVAL DAY TO SECOND

INTERVAL YEAR[(p)] **TO MONTH**

The INTERVAL YEAR TO MONTH data type stores a period of time represented as years and months.

p is the number of digits in the YEAR datetime field. The range of p is 0 to 9. The default value is 2.

This data type is available in Oracle 9i Release 1 (9.0.1) or later.

INTERVAL DAY[(p1)] **TO SECOND**[(p2)]

The INTERVAL DAY TO SECOND data type stores a period of time represented as days, hours, minutes and seconds with a fractional part.

p1 is the number of digits in the DAY datetime field. The range of p1 is 0 to 9. The default value is 2.

p2 is the number of digits in the fractional part of the SECOND datetime field. The range of p2 is 0 to 9. The default value is 6.

This data type is available in Oracle 9i Release 1 (9.0.1) or later.

LONG, RAW and LONG RAW

LONG

LONG columns store variable-length character data with a maximum length of 2^31 - 1 (2,147,483,647) bytes. A table cannot contain more than one LONG column.

RAW(*n*)

RAW is variable-length data type. Oracle does not perform character conversion when transmitting RAW data. The maximum length is 2000 bytes. You must specify the size for RAW columns.

LONG RAW

LONG RAW columns store variable-length data with a maximum length of 2^31 - 1 (2,147,483,647) bytes. Oracle does not perform character conversion when transmitting LONG RAW data.

BLOB, CLOB, NCLOB and BFILE

The built-in LOB data types BLOB, CLOB and NCLOB (stored internally), and BFILE (stored externally), can store large and unstructured data such as text, images and spatial data up to 4 gigabytes in size.

BLOB

The BLOB data type stores binary large objects. BLOB can store up to 4 gigabytes of binary data.

CLOB

The CBLOB data type stores character large objects. CLOB can store up to 4 gigabytes of character data.

NCLOB

The NCBLOB data type stores character large objects in multibyte national character set. NCLOB can store up to 4 gigabytes of character data.

BFILE

The BFILE data type enables access to binary file LOBs that are stored in file systems outside the Oracle database. A BFILE column stores a locator, which serves as a pointer to a binary file on the server's file system. The maximum file size supported is 4 gigabytes.

ROWID and UROWID

ROWID

The ROWID data type is used to store physical address of each row in the database. This data type is primarily for values returned by the ROWID pseudocolumn.

UROWID[(*n*)]

The UROWID data type is used to store the logical addresses of index-organized and foreign tables.

n is the size of a UROWID column. The range of *n* is 1 to 4000. The default value is 4000.

This data type is available in Oracle 9i Release 1 (9.0.1) or later.

Oracle Functions and Expressions

This section describes functions and expressions that can be used in Oracle SQL statements and PL/SQL programs.

Oracle Expressions

An expression is a combination of one or more values, operators, and SQL functions that evaluate to a value. An expression generally assumes the data type of its components.

- DECODE Expression

DECODE Expression

Syntax

DECODE(*expr, search, result {, search, result} [, default]*)

Oracle compares expr to each search value one by one. If expr is equal to a search, Oracle returns the corresponding result. If *no* match is found, Oracle returns default, or, if default is omitted, returns null.

The search, result, and default values can be derived from expressions.

Example

This query decodes the value DEPTNO. If DEPTNO is 10, the query returns 'ACCOUNTING'; if DEPTNO is 20, it returns 'RESEARCH'; etc. If DEPTNO is not 10, 20, 30, or 40, the query returns 'NONE'.

```
SELECT DECODE (deptno,10, 'ACCOUNTING',
20, 'RESEARCH',
30, 'SALES',
40, 'OPERATION',
'NONE') name, loc
FROM dept;
```

Equivalents in other databases

TABLE 81. Equivalents in other databases

IBM DB2	Simple CASE expression
Microsoft SQL Server	Simple CASE expression

Oracle Functions

A function is a relationship between a set of input data values and a set of result data values. Functions can either be built-in or user-defined.

Built-in functions are provided with the database manager, while user-defined functions are used to extend the function of the database system by users.

- LENGTH
- NVL
- SUBSTR
- SYSDATE

LENGTH

Syntax

LENGTH(string_expr)

LENGTH returns the number of characters of the giving string expression.

Example

This example returns the company name and the number of characters in the name for each company.

```
SELECT CompanyName, LENGTH(CompanyName)
FROM CUSTOMERS
```

Here is the result set:

```
CompanyName
_____________________________
Consolidated Holdings        21
Eastern Connection           18
France restauration          19
Great Lakes Food Market      23
Island Trading               14
Let's Stop N Shop            17
```

Equivalents in other databases

TABLE 82. Equivalents in other databases

IBM DB2	LENGTH function
Microsoft SQL Server	LEN function

NVL

Syntax

NVL(expr1, expr2)

If expr1 is NULL, the function returns expr2. If expr1 is not NULL, the function returns expr1. The NVL function is usually used to replace NULL values with specified value.

Example

This example selects title and price for all books. If the price for a given title is NULL, the price shown is 0.00.

```
SELECT title, NVL(price, 0.00) AS price
FROM titles;
```

Equivalents in other databases

TABLE 83. Equivalents in other databases

IBM DB2	COALESCE function
Microsoft SQL Server	ISNULL and COALESCE functions

SUBSTR

Syntax

SUBSTR(string, start [, length])

SUBSTR returns part of a character string beginning at start position, length characters long.

if start is positive, Oracle counts from the beginning of the string.

if start is negative, Oracle counts backwards from the end of the string.

if length is omitted, Oracle returns all characters to the end of the string.

Example

This example returns the full first name in one column, and only the first initial in the second column.

```
SELECT au_fname, SUBSTR(au_fname, 1, 1)
FROM AUTHORS;
```

Here is the result set:

```
au_fname
_____________________________
Albert                    A
Ann                       A
Anne                      A
Michael                   M
Michel                    M
```

Equivalents in other databases

TABLE 84. Equivalents in other databases

IBM DB2	SUBSTR, LEFT and RIGHT functions
Microsoft SQL Server	SUBSTRING , LEFT and RIGHT functions

SYSDATE

Syntax

SYSDATE

The SYSDATE function returns the current date and time.

Example

This example selects the current system date and time.

```
SELECT SYSDATE
FROM dual;
```

Equivalents in other databases

TABLE 85. Equivalents in other databases

IBM DB2	CURRENT TIMESTAMP special register
Microsoft SQL Server	GETDATE function

Oracle Reserved Words

There are reserved words in Oracle which cannot be used as identifiers (table or column names etc.) without being delimited with double quotation marks ("). The only exception is that you cannot use the uppercase reserved word ROWID as an identifier, even in double quotation marks.

If an identifier name is a reserved word delimited with double quotation marks then you must use double quotation marks whenever you refer to this object. As the delimited identifiers are case sensitive, you should refer to this object using the same case of the letters as you were using when naming the object.

For example, if you create the table with the name “ELSE”, then you should refer to this table like “ELSE” and not like “Else” or “eLSE”.

Many Oracle reserved words are valid object or column names in other databases. So tables migrated from other databases to Oracle may contain Oracle reserved words.

For example, LEVEL is a reserved word in Oracle, but it is not a reserved word in IBM DB2 and Microsoft SQL Server. A column can be named LEVEL in IBM DB2 and Microsoft SQL Server, but *no* column is allowed to have the name LEVEL in Oracle.

When a database is migrated to Oracle, SQLWays automatically resolves Oracle reserved words conflicts. SQLWays checks the table and column names for Oracle reserved words, and encloses them in double quotation marks. The reserved word ROWID is replaced with the delimited identifier “Rowid”.

SQLWays supports the following Oracle reserved words:

TABLE 86. Oracle reserved words

ACCESS	ADD	ALL
ALTER	AND	ANY
AS	ASC	AUDIT
BETWEEN	BY	CHAR
CHECK	CLUSTER	COLUMN
COMMENT	COMPRESS	CONNECT
CREATE	CURRENT	DATE
DECIMAL	DEFAULT	DELETE
DESC	DISTINCT	DROP
ELSE	EXCLUSIVE	EXISTS
FILE	FLOAT	FOR
FROM	GRANT	GROUP
HAVING	IDENTIFIED	IMMEDIAE
IN	INCREMENT	INDEX
INITIAL	INSERT	INTEGER
INTERSECT	INTO	IS
LEVEL	LIKE	LOCK
LONG	MAXEXTENTS	MINUS
MISLABEL	MODE	MODIFY
NOAUDIT	NOCOMPRESS	NOT
NOWAIT	NULL	NUMBER

TABLE 86. Oracle reserved words

OF	OFFLINE	ON
ONLINE	OPTION	OR
ORDER	PCTFREE	PRIOR
PRIVILEGES	PUBLIC	RAW
RENAME	RESOURCE	REVOKE
ROW	ROWID	ROWNUM
ROWS	SELECT	SESSION
SET	SHARE	SIZE
SMALLINT	START	SUCCESSFUL
SYNONYM	SYSDATE	TABLE
THEN	TO	TRIGGER
UID	UNION	UNIQUE
UPDATE	USER	VALIDATE
VALUES	VARCHAR	VARCHAR2
VIEW	WHENEVER	WHERE
WITH		

Oracle Version Differences

Oracle versions are compatible in most ways. This section describes differences between Oracle 9i, 8i, 8.0.x and 7.3.

- TIMESTAMP data type

TIMESTAMP data type

Requirements: Oracle 9i Release 1 (9.0.1) or later.

The TIMESTAMP data type is an extension of the DATE data type. For each TIMESTAMP value, Oracle stores the following information: year, month, day, hour, minute, second and fraction of second.

fractional_seconds_precision optionally specifies the number of digits in the fractional part of second and can be a number in the range 0 to 9. The default is 6.

For more information, see Oracle Data Types

Migrating to Microsoft SQL Server

This chapter is devoted to migrating databases to Microsoft SQL Server, a relational database management system.

Migration from Oracle to Microsoft SQL Server

This section provides detailed information about migrating a database from Oracle 9i and 8i to Microsoft SQL Server 2000, 7.0 and 6.5. This manual describes differences between Microsoft SQL Server and Oracle and outlines how those differences are handled by SQLWays during the conversion process.

Microsoft SQL Server and Oracle Functions and Expressions

This chapter provides detailed descriptions of the differences in functions and expressions used by Microsoft SQL Server and Oracle databases.

- Oracle DECODE Expressions

In Oracle, DECODE expressions are used to easily manipulate the data representation of a table. SQLWays converts Oracle DECODE expressions to SQL Server simple CASE expressions.

For example, this query decodes the value DEPTNO. If DEPTNO is 10, the query returns 'ACCOUNTING'; if DEPTNO is 20, it returns 'RESEARCH'; etc. If DEPTNO is not 10, 20, 30, or 40, the query returns 'NONE'.

TABLE 87. DECODE Expression

Oracle	Microsoft SQL Server
SELECT DECODE (deptno,10, 'ACCOUNTING', 20, 'RESEARCH', 30, 'SALES', 40, 'OPERATION', 'NONE') name, loc FROM dept;	SELECT CASE deptno WHEN 10 THEN 'ACCOUNTING' WHEN 20 THEN 'RESEARCH' WHEN 30 THEN 'SALES' WHEN 40 THEN 'OPERATION' ELSE 'NONE' END name, loc FROM dept;

For more information, see Oracle DECODE Expressions and Microsoft SQL Server Simple CASE Expressions

Migration from Sybase to Microsoft SQL Server

Microsoft SQL Server and Sybase relational database systems were developed together until their respective 4.2 versions. These databases are similar in many respects.

This section outlines the differences between Microsoft SQL Server and Sybase.

- Data Types

Microsoft SQL Server and Sybase Data Types Differences

This chapter provides detailed descriptions of the differences in data types used by Microsoft SQL Server and Sybase Adaptive Server databases.

For more information about the data types, see Microsoft SQL Server Data Types and Sybase Data Types

TABLE 88. Data Types Differences

SQL Server Data Type	Sybase Data Type	Description	Conversion
char(*n*) 1<=*n*<=8000	char(*n*) 1<=*n*<=255	Fixed-length character data	if *n*>255, SQL Server nchar(*n*) should be converted to Sybase text
nchar(*n*) 1<=*n*<=4000	nchar(*n*) 1<=*n*<=255	Fixed-length character data (Unicode, multibyte sets)	if *n*>255, SQL Server nchar(*n*) should be converted to Sybase text
varchar(*n*) 1<=*n*<=8000	varchar(*n*) 1<=*n*<=255	Variable-length character data	if *n*>255, SQL Server varchar(*n*) should be converted to Sybase text
nvarchar(*n*) 1<=*n*<=4000	nvarchar(*n*) 1<=*n*<=255	Variable-length character data (Unicode, multibyte sets)	if *n*>255, SQL Server nvarchar(*n*) should be converted to Sybase text
bigint	-	Integer, 64-bit	SQL Server bigint should be converted to Sybase decimal(19,0)
int	int	Integer, 32-bit	Not required
smallint	smallint	Integer, 16-bit	Not required
tinyint	tinyint	Integer, 8-bit	Not required
datetime	datetime	Date and time	Not required
smalldatetime	smalldatetime	Date and time	Not required
money	money	Monetary data	Not required
smallmoney	smallmoney	Monetary data	Not required
decimal(p,s) 1<=p<=38; 0<=s<=p;	decimal(p,s) 1<=p<= 38; 0<=s<=p;	Fixed-point numeric data	Not required
numeric(p,s) 1<=p<=38; 0<=s<=p;	numeric(p,s) 1<=p<=38; 0<=s<=p;	Fixed-point numeric data	Not required
float(*n*) 1<=*n*<=53	float(*n*) 1<=*n*<=machine dependent	Floating point numeric data	Not required
-	double precision	Floating point numeric data, 64-bit	Sybase double precision should be converted to SQL server float(53)

TABLE 88. Data Types Differences

SQL Server Data Type	Sybase Data Type	Description	Conversion
real	real	Floating point numeric data, 32-bit	Not required
text	text	Variable-length character data, up to 2 Gb	Not required
ntext	-	Variable-length Unicode character data, up to 1 Gb	SQL Server ntext should be converted to Sybase text
binary(*n*) 1<=*n*<=8000	binary(*n*) 1<=*n*<=255	Fixed-length binary data	if *n*>255, SQL Server binary(*n*) should be converted to Sybase image
varbinary(*n*) 1<=*n*<=8000	varbinary(*n*) 1<=*n*<=255	Variable-length binary data	if *n*>255, SQL Server varbinary(*n*) should be converted to Sybase image
image	image	Variable-length binary data, up to 2 Gb	Not required
bit	bit	Integer value 0 or 1	SQL Server bit can be NULL, while Sybase bit cannot be NULL Nullable SQL Server bit should be converted to Sybase tinyint or char(1)
uniqueidentifier	-	Globally unique identifier (GUID)	SQL Server uniqueidentifier should be converted to Sybase char(36)
timestamp	timestamp	Unique number that gets updated every time a row gets updated	Not required

Microsoft SQL Server Data Types

This section describes the data types supported by Microsoft SQL Server.

- char, nchar, varchar and nvarchar
- bigint, int, smallint and tinyint
- datetime and smalldatetime
- money and smallmoney
- decimal and numeric
- float and real
- text and ntext
- binary, varbinary and image
- bit, uniqueidentifier and timestamp

char, nchar, varchar and nvarchar

char[(*n*)]

Fixed-length non-Unicode character data with length of *n* characters. *n* must be a value from 1 through 8,000. Storage size is *n* bytes.

nchar[(*n*)]

Fixed-length Unicode character data with length of *n* characters. *n* must be a value from 1 through 4,000. Storage size is two times *n* bytes.

varchar[(*n*)]

Variable-length non-Unicode character data with length of *n* characters. *n* must be a value from 1 through 8,000. Storage size is the actual length of the data entered, not *n* bytes. The data entered can be 0 characters in length.

nvarchar[(*n*)]

Variable-length Unicode character data with length of *n* characters. *n* must be a value from 1 through 4,000. Storage size, in bytes, is two times the number of characters entered. The data entered can be 0 characters in length.

Remarks

When *n* is not specified in a data definition, the default length is 1.

bigint, int, smallint and tinyint

bigint

Integer (whole number) data from -2^63 (-9223372036854775808) through 2^63-1 (9223372036854775807). Storage size is 8 bytes.

The bigint data type is available in SQL Server 2000 or later.

int

Integer (whole number) data from -2^31 (-2,147,483,648) through 2^31 - 1 (2,147,483,647). Storage size is 4 bytes.

smallint

Integer data from -2^15 (-32,768) through 2^15 - 1 (32,767). Storage size is 2 bytes.

tinyint

Integer data from 0 through 255. Storage size is 1 byte.

datetime and smalldatetime

Date and time data types for representing date and time of day.

datetime

Date and time data from January 1, 1753, through December 31, 9999, with an accuracy of three-hundredths of a second, or 3.33 milliseconds.

smalldatetime

Date and time data from January 1, 1900, through June 6, 2079, with an accuracy of one minute.

money and smallmoney

Monetary data types for representing monetary or currency values.

money

Monetary data values from -2^63 (-922,337,203,685,477.5808) through 2^63 - 1 (+922,337,203,685,477.5807), with accuracy to a ten-thousandth of a monetary unit. Storage size is 8 bytes.

smallmoney

Monetary data values from - 214,748.3648 through +214,748.3647, with accuracy to a ten-thousandth of a monetary unit. Storage size is 4 bytes.

decimal and numeric

Numeric data types with fixed precision and scale.

decimal[(p[,s])]

Fixed precision and scale numbers. When maximum precision is used, valid values are from - 10^38 +1 through 10^38 - 1.

p (precision) specifies the maximum total number of decimal digits that can be stored, both to the left and to the right of the decimal point. The precision must be a value from 1 through the maximum precision. The maximum precision is 38.

s (scale) specifies the maximum number of decimal digits that can be stored to the right of the decimal point. Scale must be a value from 0 through p. The default scale is 0; therefore, 0 <= s <= p. Maximum storage sizes vary, based on the precision.

numeric[(p[,s])]

A synonym for decimal.

float and real

Approximate number data types for use with floating point numeric data. Floating point data is approximate; not all values in the data type range can be precisely represented.

float[(*n*)]

Floating point number data from - 1.79E + 308 through 1.79E + 308. *n* is the number of bits used to store the mantissa of the float number in scientific notation and thus dictates the precision and storage size. *n* must be a value from 1 through 53.

real

Floating point number data from -3.40E + 38 through 3.40E + 38. Storage size is 4 bytes. In SQL Server, the synonym for real is float(24).

text and ntext

Variable-length character strings.

text

Variable-length non-Unicode data in the code page of the server and with a maximum length of 2^31 - 1 (2,147,483,647) characters. Storage size is the actual length in bytes of the data entered.

ntext

Variable-length Unicode data with a maximum length of 2^30 - 1 (1,073,741,823) characters. Storage size, in bytes, is two times the number of characters entered.

binary, varbinary and image

Binary data types.

binary[(*n*)]

Fixed-length binary data of *n* bytes. *n* must be a value from 1 through 8,000. Storage size is *n*+4 bytes. When *n* is not specified in a data definition, the default length is 1.

varbinary[(*n*)]

Variable-length binary data of *n* bytes. *n* must be a value from 1 through 8,000. Storage size is the actual length of the data entered + 4 bytes, not *n* bytes. The data entered can be 0 bytes in length. When *n* is not specified in a data definition, the default length is 1.

image

Variable-length binary data from 0 through 2^31-1 (2,147,483,647) bytes.

bit, uniqueidentifier and timestamp

bit

Integer data type that can be 1, 0, or NULL.

uniqueidentifier

A globally unique identifier (GUID). A column of uniqueidentifier data type can be initialized using the NEWID function or converting from a string constant in the following form: xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx, in which each x is a hexadecimal digit in the range 0-9 or A-F.

timestamp

A database-wide unique number that gets updated every time a row gets updated. The value of a timestamp column is unique within a database. The storage size is 8 bytes.

Microsoft SQL Server Functions and Expressions

This chapter describes functions and expressions that can be used in Microsoft SQL Server SQL statements and Transact SQL programs.

SQL Server Expressions

An expression is a combination of one or more values, operators, and SQL functions that evaluate to a value. An expression generally assumes the data type of its components.

- Simple CASE Expression

Simple CASE Expression

Syntax
CASE expr
WHEN when_expr **THEN** result_expr
{**WHEN** when_expr **THEN** result_expr}
[**ELSE** else_expr]
END

Simple CASE evaluates expr, and then, in the order specified, evaluates expr = when_expr for each WHEN clause. Simple CASE returns the result_expr of the first WHEN clause that evaluates to TRUE. If *no* match is found, CASE returns the else_expr if an ELSE clause is specified, or a NULL value if *no* ELSE clause is specified.

Example

This example uses a CASE expression to alter the display of book categories to make them more understandable.

```
SELECT Category =
CASE type
WHEN 'popular_comp' THEN 'Popular Computing'
WHEN 'mod_cook' THEN 'Modern Cooking'
WHEN 'business' THEN 'Business'
WHEN 'psychology' THEN 'Psychology'
WHEN 'trad_cook' THEN 'Traditional Cooking'
ELSE 'Not yet categorized'
END,
price AS Price
FROM titles
```

Equivalents in other databases

TABLE 89. Equivalents in other databases

IBM DB2	Simple CASE expression
Oracle	DECODE expression

SQL Server Functions

A function is a relationship between a set of input data values and a set of result data values. Functions can either be built-in or user-defined.

Built-in functions are provided with the database manager, while user-defined functions are used to extend the function of the database system by users.

- COALESCE
- GETDATE
- ISNULL
- LEFT
- LEN
- RIGHT
- SUBSTRING

COALESCE

Syntax

COALESCE(*expr1 {, exprN }*)

COALESCE returns the first expression that is not NULL. The expressions are evaluated in the order in which they are specified. This function is usually used to replace NULL values with a not NULL value.

Example

This example selects title and price for all books. If the price for a given title is NULL, the price shown is 0.00.

```
SELECT title, COALESCE(price, 0.00) AS price
FROM titles;
```

Equivalents in other databases

TABLE 90. Equivalents in other databases

IBM DB2	COALESCE function
Oracle	NVL function

GETDATE

Syntax

GETDATE()

The GETDATE function returns the current system date and time.

Example

This example selects the current system date and time.
SELECT GETDATE()
Here is the result set:

```
_____________________
July 31 1977   7:00    AM
```

This example creates the Employees table and uses GETDATE for a default value for the employee hire date.

```
CREATE TABLE Employees
(
id INT NOT NULL,
name VARCHAR(100) NOT NULL,
hire_date DATETIME DEFAULT GETDATE()
)
```

Equivalents in other databases

TABLE 91. Equivalents in other databases

IBM DB2	CURRENT TIMESTAMP special register
Oracle	SYSDATE function

ISNULL

Syntax

ISNULL(*check_expr, replacement_value*)

ISNULL replaces NULL with the specified replacement value. The function returns the value of *check_expr* if it is not NULL, otherwise, *replacement_value* is returned.

Example

This example selects title and price for all books. If the price for a given title is NULL, the price shown is 0.00.

```
SELECT title, ISNULL(price, 0.00) AS price
FROM titles
```

Equivalents in other databases

TABLE 92. Equivalents in other databases

IBM DB2	COALESCE function
Oracle	NVL function

LEFT

Syntax

LEFT(*char_expr*, *int_expr*)

LEFT returns *int_expr* leftmost characters of *char_expr*.

Example

This example returns the five leftmost characters of each book title.
SELECT LEFT(title, 5)
FROM titles
Here is the result set:

```
_____________________
The B
Cooki
You C
The G
The P
```

Equivalents in other databases

TABLE 93. Equivalents in other databases

IBM DB2	LEFT function
Oracle	SUBSTR function

LEN

Syntax

LEN(*string_expr*)

LEN returns the number of characters of the giving string expression, excluding trailing blanks.

Example

This example returns the company name and the number of characters in the name for each company.

```
SELECT CompanyName, LEN(CompanyName)
FROM Customers
```

Here is the result set:

```
CompanyName
_____________________________
Consolidated Holdings       21
Eastern Connection          18
France restauration         19
Great Lakes Food Market     23
Island Trading              14
Let's Stop N Shop           17
```

Equivalents in other databases

TABLE 94. Equivalents in other databases

IBM DB2	LENGTH function
Oracle	LENGTH function

RIGHT

Syntax

RIGHT(*char_expr*, *int_expr*)

RIGHT returns *int_expr* rightmost characters of *char_expr*.

Example

This example returns the five rightmost characters of each author's first name.
SELECT RIGHT(au_fname, 5)
FROM authors
Here is the result set:

_____

lbert
Ann
Anne
chael
ichel

Equivalents in other databases

TABLE 95. Equivalents in other databases

IBM DB2	RIGHT function
Oracle	SUBSTR function

SUBSTRING

Syntax

SUBSTRING(*expr, start, length*)

SUBSTRING returns part of a character, binary or text expression beginning at *start* position, *length* characters (bytes) long.

Example

This example returns the full first name in one column, and only the first initial in the second column.
SELECT au_fname, SUBSTRING(au_fname, 1, 1)
FROM authors
Here is the result set:
au_fname
_____
Albert A
Ann A
Anne A
Michael M
Michel M

Equivalents in other databases

TABLE 96. Equivalents in other databases

IBM DB2	SUBSTR function
Oracle	SUBSTR function

Microsoft SQL Server Version Differences

Microsoft SQL Server versions are compatible in most ways. This chapter describes differences between Microsoft SQL Server 2000, 7.0 and 6.5 versions.

- bigint data type
- INFORMATION_SCHEMA.ROUTINES view

bigint data type

Requirements: Microsoft SQL Server 2000 or later

The **bigint** data type allows you to store integer (whole number) data from -2^63 (-9223372036854775808) through 2^63-1 (9223372036854775807). Storage size is 8 bytes.

The **bigint** data type is intended for special cases where the integer values may exceed the range supported by the **int** data type. The **int** data type remains the primary integer data type in SQL Server.

INFORMATION_SCHEMA.ROUTINES view

Requirements: Microsoft SQL Server 2000 or later

You can use the INFORMATION_SCHEMA.ROUTINES view to retrieve information about stored procedures. This view contains one row for each stored procedure accessible to the current user in the current database.

For example, the following query returns owner, name and definition text of stored procedures in the current database:

```
select ROUTINE_SCHEMA, ROUTINE_NAME, ROUTINE_DEFINITION
from INFORMATION_SCHEMA.ROUTINES
where ROUTINE_TYPE='PROCEDURE'
```

The INFORMATION_SCHEMA.ROUTINES view was introduced in SQL Server 2000. This view is based on the sysobjects, syscomments and other system tables.

To retrieve the same information about stored procedures from a Microsoft SQL Server 7.0 database, you can use the following query:

```
select su.name, so.name, sc.text
from sysobjects so, syscomments sc, sysusers su
where xtype='P' and so.id=sc.id and so.uid=su.uid
order by su.name, so.name, sc.colid
```

The query above may return several rows per each stored procedure if the definition text is longer than 4000 characters.

Migrating to Sybase

This chapter is devoted to migrating databases to Sybase.

Sybase develops and promotes two basic database products: Sybase Adaptive Server Enterprise (ASE) and Sybase Adaptive Server Anywhere (ASA).

ASE is the company's enterprise-class relational database management system (RDBMS) with a long market history and well-established market reputation.

ASA is a relational database management system for mobile, desktop and workgroup use.

Sybase Adaptive Server Enterprise (ASE)

This section is devoted to migrating databases to Sybase Adaptive Server Enterprise, a relational database management system.

- Sybase ASE Data Types
- Sybase ASE History
- Sybase ASE Versions Evolution

Sybase ASE Data Types

This subsection describes the data types supported by Sybase.

- char, nchar, varchar and nvarchar
- int, smallint and tinyint
- datetime and smalldatetime
- money and smallmoney
- decimal and numeric
- float, double precision and real
- text
- binary, varbinary and image
- bit and timestamp

char, nchar, varchar and nvarchar

char[(*n*)]

Fixed-length character data in single-byte character sets with length of *n* characters. *n* must be a value from 1 through 255. Storage size is *n* bytes.

nchar[(*n*)]

Fixed-length character data in multibyte character sets with length of *n* characters. *n* must be a value from 1 through 255. Storage size is *n**@@ncharsize bytes.

varchar[(*n*)]

Variable-length character data in single-byte character sets with length of *n* characters. *n* must be a value from 1 through 255. Storage size is the actual length of the data entered, not *n* bytes.

nvarchar[(*n*)]

Variable-length character data in multibyte character sets with length of *n* characters. *n* must be a value from 1 through 255. Storage size, in bytes, is the actual number of characters entered * @@ncharsize.

Remarks

When *n* is not specified in a data definition, the default length is 1.

int, smallint and tinyint

int

Integer (whole number) data from -2^31 (-2,147,483,648) through 2^31 - 1 (2,147,483,647). Storage size is 4 bytes.

smallint

Integer (whole number) data from -2^15 (-32,768) through 2^15 - 1 (32,767). Storage size is 2 bytes.

tinyint

Integer (whole number) data from 0 through 255. Storage size is 1 byte.

datetime and smalldatetime

Date and time data types for representing date and time of day.

datetime

Date and time data from January 1, 1753, through December 31, 9999, with an accuracy of 1/300 of a second, or 3.33 milliseconds. Storage size is 8 bytes.

smalldatetime

Date and time data from January 1, 1900, through June 6, 2079, with an accuracy of one minute. Storage size is 4 bytes.

money and smallmoney

Monetary data types for representing monetary or currency values.

money

Monetary data values from -2^63 (-922,337,203,685,477.5808) through 2^63 - 1 (+922,337,203,685,477.5807), with accuracy to a ten-thousandth of a monetary unit. Storage size is 8 bytes.

smallmoney

Monetary data values from - 214,748.3648 through +214,748.3647, with accuracy to a ten-thousandth of a monetary unit. Storage size is 4 bytes.

decimal and numeric

Numeric data types with fixed precision and scale.

decimal[(p[,s])]

Fixed precision and scale numbers. When maximum precision is used, valid values are from - 10^38 +1 through 10^38 - 1.

p (precision) specifies the maximum total number of decimal digits that can be stored, both to the left and to the right of the decimal point. The precision must be a value from 1 through the maximum precision. The maximum precision is 38. The default precision is 18 digits.

s (scale) specifies the maximum number of decimal digits that can be stored to the right of the decimal point. Scale must be a value from 0 through p. The default scale is 0; therefore, 0 <= s <= p. Maximum storage sizes vary, based on the precision.

numeric[(p[,s])]

The **numeric** and **decimal** datatypes are identical in all respects but one: only numeric datatypes with a scale of 0 can be used for the IDENTITY column.

float, double precision and real

Approximate number data types for use with floating point numeric data. Floating point data is approximate; not all values in the data type range can be precisely represented.

float[(precision)]

Floating point number. Storage size is 4 bytes if precision < 16, and 8 bytes if precision >= 16. The range and storage precision is machine dependent.

float columns with a precision of 1-15 are stored as real; those with higher precision are stored as double precision.

double precision

Floating point number. Storage size is 8 bytes. The range and storage precision is machine dependent.

real

Floating point number. Storage size is 4 bytes. The range and storage precision is machine dependent.

text

Variable-length character strings.

text

Variable-length character data with a maximum length of 2^31 - 1 (2,147,483,647) characters.

binary, varbinary and image

Binary data types.

binary[(*n*)]

Fixed-length binary data of *n* bytes. *n* must be a value from 1 through 255. When *n* is not specified, the default length is 1.

varbinary[(*n*)]

Variable-length binary data of *n* bytes. *n* must be a value from 1 through 255. When *n* is not specified, the default length is 1.

image

Variable-length binary data from 0 through 2^31-1 (2,147,483,647) bytes.

bit and timestamp

bit

Integer data type that can be 1 or 0. Integer values other than 0 or 1 are accepted, but are always interpreted as 1. Columns with a data type of bit cannot be NULL.

timestamp

Data type that is defined as varbinary(8). It requires 8 bytes of storage.

Sybase updates the **timestamp** column each time its row is modified. A table can have only one column of **timestamp** data type.

Sybase ASE History

Sybase Adaptive Server Enterprise started its life as Sybase SQL Server - the first relational database management system (RDBMS) manufactured and sold by Sybase.

It was originally created for UNIX platforms in 1987. In 1988, SQL Server for OS/2 was co-developed for PC by Sybase, Microsoft, and Ashton-Tate. Then Ashton-Tate lost its interest in the project and Microsoft became the lead partner after porting SQL Server to Microsoft Windows NT. For several years, Microsoft was a Sybase distributor, reselling the Sybase product for OS/2 and Windows NT under the name Microsoft SQL Server (Microsoft SQL Server).

Since releasing version 4.21, Microsoft and Sybase sold and supported the product together. In 1993, the co-development licensing agreement between Microsoft and Sybase expired and the cooperation between the companies ended. After Microsoft purchased a copy of the source code of Sybase SQL Server, the both companies continued to develop the products independently, as competitors.

Microsoft put emphasis on the ease-of-use and "windowising" the product while Sybase focused on maximizing the product's performance and reliability and running it on high-end hardware.

In 1995, Sybase released SQL Server 11.0 and in 1997 - version 11.5. Following the release of the latter version, Sybase renamed its product as Adaptive Server Enterprise (ASE) in order to better distinguish itself from Microsoft SQL Server. However, due to their common background, the present-day versions of ASE and Microsoft SQL Server still have many similarities. For example, both ASE and Microsoft SQL Server have a very similar (though not identical) SQL implementation called "Transact-SQL".

Sybase SQL Server was the first actual client-server RDBMS also capable of handling real-world workloads. Besides, Sybase SQL Server was the first commercially successful RDBMS supporting stored procedures and triggers, and a cost-based query optimizer.

The most recent ASE release is ASE 12.5.1. It was brought into life in Oct. 2003. It runs on the main flavors of Unix, Linux, Windows NT/2000, and Mac OS X.

Sybase ASE Versions Evolution

Version 12.5.1

1. Dynamically altering the data cache without having to restart the server

2. Automatic expansion of databases and devices. Databases can be configured to expand automatically when they run out of space.

3. Supports the UTF-8 sort order. In earlier versions, when using UTF-8, the two sort order choices were binary and no-case (ASCII-only). Version 12.5.1 provides the ability to perform non-binary sort orders in UTF-8. As such, all sort orders available for the unichar and univarchar data types can be used for char or varchar data when the default character set is set to UTF-8.

4. Allows using unichar and univarchar datatypes with any server's default character set without having first to configure your default character set to UTF-8.

5. SQL derived tables. A SQL derived table is created with a nested select statement, as in the following example: *select * from (select * from t) dt*.

6. Date and time have been added as separate data types. In earlier versions, only datetime and smalldatetime were available.

7. Added XML mapping. The *for xml* clause in select statements and the *forxmlj* function map SQL result sets to SQL-XML document, using the SQLX-XML format defined by the ANSI SQLX standard.

8. Uses the native XML processor supporting SQL extensions that perform XML query functions on XML documents. Integration of the native XML processor in Adaptive Server provides remarkable performance improvements over the Java-based XQL processor of earlier versions of Adaptive Server. The native XML processor supports standard XML documents and standard XPath queries, which are a subset of the new XQuery language.

9. Adaptive Server extends its LDAP support to include storage of user information. With LDAP services enabled:

- Adaptive Server authenticates clients with data from an LDAP server. Users authenticate with passwords stored on an LDAP server rather than in the syslogins catalog. The LDAP server provides a centralized location for login accounts-both names and passwords.
- Adaptive Server servers share user login data stored on the LDAP server. Information formerly stored in syslogins is now managed and stored on an LDAP server. It is cached locally to preserve referential integrity and for other, database-specific uses.
- With LDAP enabled, users have a single login and password throughout the enterprise.

10. Adaptive Server Enterprise Web Services enable Adaptive Server Enterprise to both provide and use Web services. Client applications can access SQL and stored procedures in Adaptive Server using SOAP.

11. Migrate more easily between Adaptive Server and Microsoft-SQL Server. Adaptive Server 12.5.1 provides enhanced compatibility with Microsoft SQL extensions:

- default in the insert statement;
- set for variable and value assignments;
- square brackets surrounding identifier names;
- new built-in function cast() and those added in version 12.5.0.3: len(), left(), day(), month(), year(), str_replace(), newid(), square().

- derived tables, which can be used wherever a view can be used, and which can be used in the FROM LIST of SELECT, SELECT INTO, CREATE VIEW, and INSERT statements.

Version 12.5.0.3

1. Asynchronous log service (ALS) and optimistic index locking help resolve drastically increased contention on such key resources as the log, the spinlocks that guard the log, and the address locks, increasing Adaptive Server scalability in systems using four or more online engines.

2. Allows creating and managing multiple temporary databases in addition to the system tempdb, which was the only temporary database in the server in earlier versions of Adaptive Server.

3. Improved performance for select into.

4. New functions:

- year;
- month;
- day;
- str_replace;
- square;
- left;
- len;
- next_identity;
- identity_burn_max;

Version 12.5.0.1

1. Adaptive Server is available in the Enterprise Edition, the Small Business Edition, and the Developer's Edition. The Enterprise Edition of Adaptive Server is a full-featured server that can run all optional features. The Small Business Edition of Adaptive Server includes the features required by most small businesses, but excludes some of the more advanced features. The Developer's Edition is designed for you to design and build applications for Adaptive Server in a development environment. Earlier versions of Adaptive Server were sold either at the Enterprise Level or the Workplace Level.

2. ASE Replicator extends Adaptive Server Enterprise capabilities and provides basic replication from a primary database to one or more replicate databases. ASE Replicator:

- Replicates both tables and stored procedures;
- Maintains database integrity and transactional consistency at multiple sites;
- Provides guaranteed delivery of replicated data;
- Manages database objects with a publish-and-subscribe model;

3. The Extensible Query Language (XQL) result set feature allows accessing query results as objects, rather than as strings, and extracting SQL base types easily from an XML document, without parsing results or writing code. The XQL engine is written in Java, so creating a Java object is easy. The engine allows:

- Iterating through XQL result sets to extract XML fragments;

- Converting XML into SQL data, for insertion into database tables;
- Getting the results as a DOM (Document Object Model) object;

Version 12.5

1. Dynamic reconfiguration.

- Dynamic memory allocation. Allows allocating total physical memory dynamically. Many of the configuration parameters that affect memory are now dynamic, and the server does not have to be restarted for them to take effect.
- For Adaptive Server 12.5, both the data cache and the procedure cache are specified as absolute values. The sizes of the caches do not change until DBA reconfigure them. In earlier versions of Adaptive Server, the size of the procedure cache was based on a percentage of the available memory. After DBA configured the data cache, whatever was left over was allocated to the procedure cache.

2. Extends Java capabilities. Developers can now wrap Java static methods in SQL names and create SQLJ stored procedures or functions that developers can use as would Transact-SQL stored procedures or built-in functions. This new functionality:

- Allows Java methods to return output parameters and result sets to the calling environment.
- Allows developers to take advantage of traditional SQL syntax, metadata, and permission capabilities.
- Complies with Part 1 of the ANSI SQLJ standard specification.
- Allows developers to use existing Java methods as SQLJ procedures and functions on the server, on the client, and on any SQLJ-compliant, thirdparty database.

3. XML in the database. XML allows defining own application-oriented markup tags. This feature, written entirely in Java, also includes methods for storing XML documents and generating them from SQL data.

User can:

- Select raw data from Adaptive Server using XQL and display it as an XML document. User can also store XML documents in Adaptive Server.
- Use the Java XML parser provided by Sybase, which allows you to install his query engine as either a standalone program or inside Adaptive Server.

4. Supporting union operators in select statements that define views.

5. Using Internet directory services (LDAP).

6. Implementing Secure Sockets Layer (SSL) protocol.

7. Provides the Enterprise Java Beans (EJB Server), a component transaction server. EJB Server provides the framework for creating, deploying and managing middle-tier business logic in the form of EJBs in a multitier environment.

8. External file system support. Enables SQL access to file system directories and files. The supported syntax is: *create [existing] table fname external file at "directory_pathname" column delimiter "delimiter"*.

9. Row-level access control. Row-level access control enables the Database owner or table owner to control the rows in a table that user can access, based on their identification or profile and the privileges the user has from the application level.

10. Java.net support. With java.net, developers can create client-side Java networking applications within the server. Developers can create a network Java client application that connects to any server, which enables Adaptive Server to function as a client to external servers. Developers can:

- Download documents from any URL address on the Internet.
- Send e-mail messages from inside the server.
- Access documents using XML.

11. unichar support. Adds two new datatypes using UTF-16 encoding of the Unicode char. The unichar and univarchar datatypes are independent of the existing char and varchar datatypes, but have the same functionality. Unichar is a fixed-width, non-nullable data type (like char) and univarchar is a variable-width, nullable data type (like varchar). The set of the built-in string functions that operate on char and varchar, also operate on unichar and univarchar. Unlike the existing char and varchar, the new unichar and univarchar only store UTF-16 characters and have no relation to the default character set or default sort order of Adaptive Server. To use these news datatypes, the default character set for the server must be set to UTF-8.

Version 12.0

Availability and manageability:

1. Sybase Failover for a high availability system. Sybase Failover enables Adaptive Server to work in a high availability cluster in an active-active configuration. Both nodes in the cluster include Adaptive Servers managing independent workloads and capable of taking over each other's workload in the event of a failure. The Adaptive Server that takes over the workload is called a secondary companion, and the Adaptive Server that fails is called the primary companion. Together they are companion servers. This movement from one node to another is called failover. After the primary companion is ready to resume its workload, it is moved back to its original node. This movement is called failback. Clients connected to the failed Adaptive Server automatically reestablish their connection via the second machine.

2. Supporting UNIX files system. Introduces a new dsync setting for database device files, which controls whether or not data written to those files are buffered. When the dsynch setting is on, Adaptive Server opens a database device file using the UNIX dsync flag. The dsync flag ensures that data is written to the device file directly on the physical storage media, and Adaptive Server can recover data on the device in the event of a system failure.

3. Modifying an existing table's schema with alter table. Includes new syntax for alter table that allows users to add, drop, or modify a table's columns. Alter table allows users to modify a column's data type, length, locking scheme, or default value. Alter table also allows users to add, drop, or modify an IDENTITY column and to add a null or non-null column.

4. Suspending database updates with quiesce database command. User may want to suspend database updates to use external database backup utility, or to separate a database from its mirror for reporting purposes.

Performance and productivity:

1. Provides a runtime environment for Java for executing Java code in the server. Adaptive Server's Java capabilities provide users with powerful ways of managing and storing both data and logic, using a language that is both portable and widely available.

- Java programming language can be used as an integral part of Transact-SQL.
- Java code can be reused in different layers of an application - client, middle-tier, server - and can be consequently used wherever it makes most sense.
- Java provides a more powerful language than stored procedures for building logic in the database.
- Java classes become rich, user-defined datatypes.
- Methods of Java classes provide new functions accessible from SQL.

2. Merge joins as a query execution method for equijoins. Previous versions of Adaptive Server performed all join queries as nested-loop joins. Merge joins can produce dramatic improvements in query execution time, especially for joins involving several very large tables.

3. Query costing improvements. The following changes help improve the accuracy of optimizer costing and provide additional paths for query execution:

- Transitive closure can be applied to joins.
- Predicate factoring and transformation can improve costing for queries using or.
- Special like optimization for leading wildcards in strings.

4. ANSI syntax for joining tables or views. Previous versions of Adaptive Server provided syntax only for a transact-SQL join, which include *= and =* symbols for specifying a right or a left join, respectively.

5. Dynamic execution of Transact-SQL. Adaptive Server version 12.0 provides an extension to the execute command that enables defining transact-SQL commands dynamically at execution time. Applications and procedures can use the new execute() syntax in cases where table and column names are not known until the application or procedure executes.

6. Text and image data type enhancements. Adaptive Server version 12.0 improves the storage format for text and image datatypes. Changes to the storage format are transparent to end-user applications, but they enable Adaptive Server to perform random access when querying data. This improves query performance over early versions, which had to access text and image data sequentially. The new storage format for text and image datatypes also enables Adaptive Server to asynchronously prefetch the data into an existing buffer pool.

7. Adaptive Server can capture query text and save an abstract plan for a query in a new system table called sysqueryplans. Using a rapid hashing method, incoming SQL queries can be compared to stored query text, and if a match is found, the saved abstract plan is used to execute the query.

8. Adaptive Server version 12.0 introduces *disable trigger* option of the *alter table* command to disable any triggers in a database before load the database, reducing the time required to load the database.

9. Adaptive Server version 12.0 allows dividing each cache into partitions, each with its own spinlock. In the previous versions, each task that needs to access the data cache holds a spinlock on the cache while accessing it. With a large number of engines and a high transaction rate, contention for buffer cache spinlocks can slow performance.

Distributed processing:

1. Adaptive Server version 12.0 introduces several Distributed Transaction Management features, which:

- Bring Adaptive Server into full compliance with the X/Open XA protocol when acting as a resource manager, without requiring additional services as XA-Server.
- Provided support for distributed transactions coordinated by Microsoft Distributed Transaction Coordinator (MSDTC).
- Ensure consistent commit or rollback for all transactions that update Adaptive Server data via remote procedure calls (RPCs) and Component Integration Services.
- Provide the framework to support additional distributed transaction management protocols in the future.

Version 11.9.2

1. Direct Updates Through Joins. In version 11.9.2, many restrictions on when direct updates can be performed have been removed.

2. Character Set Changes:

- Adaptive Server includes support for the European currency symbol, or "Euro".
- Adaptive Server can now perform character set conversions that cause a change in the data length;

Version 11.9

1. New Locking Schemes. The Adaptive Server version 11.9 provide two new locking schemes to improve the concurrence and performance of Adaptive Server:

- Datapages locking, which locks only the data pages;
- Datarows locking, which locks only the data rows, also known as row-level locking;

The datapages and datarows locking schemes share many characteristics, including the fact that they do not acquire locks on index pages, so together they are often referred to as the data-only locking schemes.

The pre-11.9 locking scheme continues to be support; it is called allpages locking. Allpages locking locks the data pages and index pages affected by query.

When a query updates a value in a row in an allpages-locked table, the data page is locked with an exclusive lock. Any index pages affected by the update are also locked with exclusive locks. These locks are transactional, meaning they are held until the end of the transaction.

In datapages locking, entire data pages are still locked, but index pages are not locked. When a row needs to be changed on a data page, that page is locked, and the lock is held until the end of the transaction.

In datarows locking, row-level locks are acquired on individual rows on data pages. Index rows and pages are not locked. When a row needs to be changed on a data page, a nontransactional latch is acquired on the page. The latch is held while the physical change is made to the data page, and then the latch is released. The lock on the data row is held until the end of the transaction. The index rows are updated using latches on the page, but are not locked. Index entries are implicitly locked by acquiring a lock on a data row.

2. The create index command allows you to specify ascending or descending order for each column in the index. In earlier versions, all indexes were created in ascending order for all columns. Scans that needed to read the data in reverse order could scan the index backward, but if the required order was a mix of ascending and descending order on the keys, the query needed to perform a sort step.

3. Transact-SQL syntax provides new commands and options that affect locking:

- Select, delete, update, readtext, and writetext commands now include the readpast option. This option allows the command to skip all pages or rows that are not immediately accessible due to incompatible locks.
- A new command, lock table, explicitly locks entire tables in either shared or exclusive mode.
- Server-level and session level options that specify how long tasks wait for locks.
- Support for repeatable reads transaction isolation level (level 2) for data-only-locked tables.

Version 11.5

1. Asynchronous prefetch improves performance by anticipating the pages that will be required for certain well-defined classes of database activities whose access patterns are predictable. The I/O requests for these pages are issued before the query needs them so that most pages are in cache by the time query processing needs to access the page.

2. Simplifies standard SQL expressions by allowing a case expression instead of an if...else construct.

3. OmniConnect has been replaced by Component Integration Services (CIS). The Component Integration Services (CIS) feature allows connecting to remote Sybase and non-Sybase databases. It presents a uniform view of enterprise data to client applications and provides location transparency to the data sources that clients access.

4. Create index enhancements. Create clustered index…with sorted_data does not sort or copy data except when certain optional clauses are specified to create index. If these clauses are specified for partitioned tables, a parallel sort must be performed. In earlier releases of SQL Server, use of the create clustered index…with sorted_data allowed to skip the sort step on data that was already in sorted order, however, the data pages always copied to a new location on the data devices.

5. Descending index scan optimization. Descending index scan optimization is a performance enhancement that can improve the performance of queries that use the desc keyword in order by queries to return result sets in descending order. In earlier releases of SQL Server, descending result sets required a worktable and a sort. In Adaptive Server 11.5, the optimizer can choose to use the index and avoid the sort step, if the strategy reduces the query cost.

Descending index scan can speed the use of both clustered and non-clustered index access, reduce the tempdb space required for temporary tables, save the CPU time required for sorts, and shorten the time that locks are held, if descending scans use holdlock or transaction isolation level 3.However, there can be increased chance of deadlocking in some applications.

This feature does not change the syntax for the order by clause, it only changes the way that order by clauses with the desc keyword can be optimized.

6. Directory services. Adaptive Server 11.5 provides an alternative to using the traditional interfaces file by supporting the ability to connect to network service information throw third-party directory service provider.

7. Extended Stored Procedures (ESPs). This release support both user-defined and system-defined extended stored procedures.

ESPs provide a method for calling procedural language function from within the Adaptive Server. The procedural language in which the functions are written must be capable of calling C language functions and manipulating C data types.

ESPs allow Adaptive Server to perform a task outside Adaptive Server in response to an event occurring within Adaptive Server. For example, an email notification could be sent in response to an event occurring within the RDBMS.

The interface to ESPs is similar to the interface to system procedures and user-defined stored procedures. The difference is that an ESP executes procedural language code, rather than Transact-SQL statements.

Extended stored procedures are implemented by an Open Server application called XP Server, which runs on the same machine as Adaptive Server. Running ESPs in a separate process protects Adaptive Server from failing because of faulty ESP code. Adaptive Server and XP Server communicate through the remote procedure calls (RPCs).

The function that implements the ESP is compiled and linked into a dynamic linked library (DLL) or shared library. Adaptive server looks in the system tables for the function that has the same name as requested ESP and passes the function name and the DLL name to XP server.

8. Adaptive Server allows managing individual metadata caches for:

- User indexes.
- Objects such as procedures, triggers, views, rules, and defaults.
- Databases.

Managing individual metadata caches for these objects is a beneficial for a server that contains a large number of user indexes and objects and where there is high concurrency among users.

9. Parallel bulk copy. Parallel bulk copy allows coping data in parallel to Adaptive Server from files. Parallel bulk copy substantially increases performance during bcp sessions because large bulk copy jobs can be split in multiple sessions and run concurrently.

10. Parallel queries and enhanced partitioning. When Adaptive server is configured for parallel query processing, the optimizer evaluates each query to determinate whether it is eligible for parallel execution. If it is, the query is divided into components that process simultaneously. The results are combined and delivered to the client in a shorter period of time than it would take to process the query serially as a single component.

Fully enabled parallel query processing requires multiple processes, engines, and partitions, resulting in increased overhead for Adaptive Server, additional CPU requirements, and increased disk I/O.

Pont-in-time recovery. Point-in-time recovery allows recovering a database by rolling it forward to a particular time in its transaction log.

11. Recovery fault isolation. With recovery fault isolation, a System Administrator has the option of isolating the corrupt pages marked suspect by recovery while the rest of database is brought online and available to users.

Prior to this release, when recovery detected corruption in a database, it would make entire database inaccessible. The database remained unavailable to users until the suspect pages has been repaired or removed from database.

12. SQL Server Manager has been replaced by Sybase Central Sybase's common management interface and the Adaptive Server plug-in for Sybase Central.

The Adaptive Server plug-in for Sybase central allows managing adaptive Server installations using the Sybase central graphical management tool.

Version 11.0

1. User-defined caches. Release 11.0 allows System Administrators to split the SQL Server data cache into separate named data caches and to bind databases or database objects to those caches. Also, System Administrators can create pools within caches that perform large I/O to disk, improving performance for many queries.

2. Data storage changes.

- System Administrators can now configure maximum numbers of rows stored on a data page or index leaf page. This reduces lock contention and improves concurrency for frequently accessed tables.

- In Release 11.0, SQL Server provides the ability to partition heap tables. A partition is simple another term for a page chain. Partitioning a table creates multiple page chains (partitions) for the table and, therefore, multiple last pages for insert operations.

When a transaction inserts data into a portioned table, SQL Server randomly assigns the transaction to one of the table's partitions. Concurrent inserts are less likely to block, since multiple last pages are available for inserts.

3. Transaction log changes. There is one user log cache for each configured user connection. SQL Server uses this user log caches to buffer the user transaction records, which reduces the contention at the end of the transaction log. When a user log cache becomes full or another event occurs (such as transaction completes), SQL server "flushes" all log records from the user log cache to the database transaction log.

4. Isolation level 0. SQL Server allows specify isolation level 0 for the queries in transactions along with isolation levels 1 and 3 supported in release 10.0. Isolation level 0 prevents other transactions from changing data that has already been modified by uncommitted transaction. There other transactions are blocked from modifying data until the transaction commits. However, other transactions can still read the uncommitted data (known as dirty reads).

Version 10.0

1. SQL Server release 10.0 provides full support for cursors by implementing the new SQL commands: declare cursor, fetch, open, close, deallocate; keywords in the delete and update statements that allow updating and deleting at cursor positions;

2. Data Definitions Enhancements.

- a) Integrity constraints. In addition to the rules, triggers, defaults, and unique indexes already provided by SQL Server by allowing specify integrity constraints in the create table statement. These include:

  •unique and primary key constraints to insure unique values in a column;

  •Referential integrity constraints to guarantee that a primary key exists in another foreign key table;

  •Check constraints to limit the values that can be inserted into a table;

  •Defaults to specify default values for a column;

- b) Changes to Views.

  •The distinct keyword can now be used in view definition, allowing creating views that do not contain duplicate rows.

  •The with check option clause has been added to the create view statement. When a view is created with check option, all rows inserted or updated through the view must meet the view criteria.

  •Create view statements can been included in batches with other transact-SQL statements.

- c) The new create schema command allows creating several objects and grant permissions on those objects in a single statement, which can be committed or rolled back as a unit.

- d) Release 10.0 of SQL Server provides these new datatypes:

  •Numeric or decimal (dec) specify exact numeric values, with specified precision and scale to indicate the number of digits and number of digits to the right of the decimal point.

  •Double precision specifies an approximate numeric type.

  •The following changes have been made to existing datatypes:

  •Float now accepts an optional precision specification.

  •The default length for char, varchar, nchar, and nvarchar columns is 1 in create table statements, variable declarations, and parameter specifications.

- e) Automatic sequential values using the IDENTITY property for a column. Each table in a database can have a single IDENTITY column. This column is used to store numbers that are automatically generated by SQL Server. The value of the IDENTITY column uniquely identifies each row in a table.

3. ANSI compatibility. SQL Server release 10.0 meets SQL89/FIPS 127-1 and SQL92 / FIPS 127-2 standards. In addition to major features such as declarative integrity constraints and cursors, which described earlier, the following changes are included in release 10.0.

- FIPS Flagger. For customers writing application that must conform to the ANSI standard, SQL Server provides a set fipsflagger option to flag incompatible syntax.

- SQLSTATE messages and codes. By default, SQL Server returns SQLSTATE values to embedded SQL applications, as required by entry level SQL92.

- The escape clause in the like predicate. SQL Server now supports the ANSI escape clause, which allows specifying an escape character in the like predicate to search for wildcard characters. This is an addition to Transact-SQL's use of square brackets for the same purpose.

- ANSI-style comments. In Transact-SQL, comments are delimited by /* */ pairs, and can be nested. Transact-SQL now also supports ANSI-style comments, which consists of any string beginning with two unspaced minus signs, a comment, and a terminating newline. This syntax conflicts with subtraction of a negative number.

- SQL Server now provides ANSI-compliant "chained" transaction behaviors as option. In chained mode, all data retrieval and modification commands (delete, insert, open, fetch, select, and update) implicitly begin a transaction.

- Delimited identifiers. SQL Server now supports delimited identifiers for table, view, and column names. Delimited identifiers are object names enclosed within double quotation marks. Using them allow avoiding certain restrictions on object names.

- Treatment of nulls. A new set option, ansinull, determines whether or not evaluation of NULL-valued operands in SQL equality (=) or inequality (!=) comparisons and in aggregate functions is ANSI-compliant.

Version 4.9.1

1. New system procedures.

- sp_syntax displays the syntax of Transact-SQL statements, system procedures, utilities, and DB-Library routines.

- New system procedures allowing users easy access to information about SQL Server. These stored procedures are compatible with the catalog interface for the Open database connectivity (ODBC) API. The procedures are: sp_column_privileges, sp_columns, sp_databases, sp_datatype_info, sp_fkeys, sp_pkeys, sp_server_info, sp_proc_columns, sp_statistics, sp_stored_procedures, sp_table_privileges, sp_tables.

Version 4.9

1. Multibyte character set feature. SQL Server now supports multibyte character sets, including EUC-JIS, SHIFT-JIS, and DEC-Kanji, for use in Asian installations. The following changes and features where introduced to provide flexible language support.

- SQL Server support conversion among a variety character sets.

- The System Administrator can now change the default, installed sort order, or character set used by SQL Server.

- New datatypes. The national character datatypes nchar and nvarchar allow a user to define a column length as containing a specific number of multibyte characters.

Version 4.8

1. Transact-SQL now provides the union operator. It allows combining the results of two or more queries into a single result set.

2. New "Not equals" operator. The new negative comparison operator <> (not equal to) is the same as the != operator.

3. New datatypes. Smallmoney, smalldatetime, and real are smaller versions of money, datetime, and float, respectively. They require 4 bytes of storage, rather than the 8 bytes required by their larger counterparts.

4. Quoted strings in column headings. Except in create table, create view, and select into statements, column headings now can include any characters, including blanks and SQL Server keywords, if the column heading is enclosed in quotes.

5. Symmetric multiprocessors features. Release 4.8 of the SQL Server explicitly designed to take full advantage of the capabilities of symmetric multiprocessor (SMP) systems.

Sybase Adaptive Server Anywhere (ASA)

This section is devoted to migrating databases to Sybase Adaptive Server Anywhere, a relational database management system.

- Sybase ASA Data Types
- Sybase ASA History
- Sybase ASA Versions Evolution

Sybase ASA Data Types

This subsection describes the data types supported by Sybase Adaptive Server Anywhere.

- char, varchar and long varchar
- bigint, int or integer, smallint and tinyint
- date, datetime, smalldatetime, time and timestamp
- money and smallmoney
- decimal and numeric
- float, double and real
- text
- binary, long binary, varbinary and image
- bit

char, varchar and long varchar

Character data types are used to store character data.

char[(n)]
character[(n)]

Fixed-length character data with length of n characters. n must be a value from 1 through 32767. For strings up to 254 bytes, the storage size is n +1 bytes. There are more additional bytes used for longer strings.

You can store multi-byte characters as the **char** data type.

varchar[(n)]
character varying[(n)]

Variable-length character data with length of n characters. n must be a value from 1 through 32767. For strings up to 254 bytes, the storage size is n +1 bytes. There are more additional bytes used for longer strings.

You can store multi-byte characters as the **varchar** data type.

long varchar

Variable-length character data. The size is limited by the maximum size of the database file (currently 2 Gb). There are some subsidiary bytes used for storage in addition to the length of the string.

Remarks

When n is not specified in a data definition, the default length is 1. Data type lengths of less than one are not allowed. After the 255th byte characters are stored separately form the row containing the long string value.

bigint, int or integer, smallint and tinyint

[**unsigned**] **bigint**

Integer (whole number) data. The range for signed **bigint** values is from -2^63 (-9223372036854775808) through 2^63 - 1 (9223372036854775807). The range for **unsigned bigint** values is from 0 through 2^64 - 1 (18446744073709551615).

Storage size is 8 bytes. The data type is signed as the default.

[**unsigned**] **int**
[**unsigned**] **integer**

Integer (whole number) data. The range for signed **integer** is from -2^31 (-2147483648) through 2^31 - 1 (2147483647). The range for **unsigned integer** is from 0 through 2^32 - 1 (4294967295).

Storage size is 4 bytes.

[**unsigned**] **smallint**

Integer (whole number) data. The range for signed **smallint** is from -2^15 (-32768) through 2^15 - 1 (32767). The range for **unsigned smallint** is from 0 through 2^16 - 1 (65535).

Storage size is 2 bytes.

[**unsigned**] **tinyint**

Unsigned integer (whole number) data from 0 through 2^8 - 1 (255). Storage size is 1 byte.

date, datetime, smalldatetime, time and timestamp

Date and time data types for representing date and time of day.

date

date **data** type is used to store year, month and day.

The range is from January 1, 0001, through December 31, 9999. Storage size is 4 bytes.

A **date** column can also contain an hour and minute if the TRUNCATE_DATE_VALUES option is set to OFF.

datetime
smalldatetime

These data types are the aliases for built-in data type **timestamp**. These data types are used to store year, month, day, hour, minute, second and fraction of second. The fraction is stored to 6 decimal places.

The range is from January 1, 0001 00:00:00.000000, through December 31, 9999 23:59:59.999999. Storage size is 8 bytes.

These data types are provided primarily for compatibility with Adaptive Server Enterprise.

time

This data type is used to store hour, minute, second and fraction of a second. The fraction is stored to 6 decimal places. Storage size is 8 bytes.

timestamp

This data type is used to store year, month, day, hour, minute, second and fraction of second. The fraction is stored to 6 decimal places.

The range of possible dates is the same as for the **date** type, but the usefull range lies between 1600-02-28 23:59:59 and 7911-01-01 00:00:00. If the value is not in this range, the time portion may be incomplete. It requires 8 bytes of storage.

money and smallmoney

Monetary data types for representing monetary values. These data types provide compatibility with the Adaptive Server Enterprise data types.

money

Monetary data values from -999999999999999.9999 through 999999999999999.9999, with accuracy to a ten-thousandth of a monetary unit. Storage size is 8 bytes.

smallmoney

Monetary data values from -999999.9999 through 999999.9999, with accuracy to a ten-thousandth of a monetary unit. Storage size is 4 bytes.

decimal and numeric

Numeric data types with fixed precision and scale.

decimal[(p[,s])]
dec[(p[,s])]

Fixed precision and scale numbers. When maximum precision is used, valid values are from -10^128 * 9 through 10^128 * 9.

p (precision) specifies the maximum total number of decimal digits that can be stored, both to the left and to the right of the decimal point. The precision must be a value from 1 through the maximum precision. The maximum precision is 128. The default precision is 30 digits.

s (scale) specifies the maximum number of decimal digits that can be stored to the right of the decimal point. Scale must be a value from 0 through p. The default scale is 6; therefore, 0 <= s <= p. Maximum storage sizes vary, based on the precision.

numeric[(p[,s])]

The **numeric** and **decimal** datatypes are identical in all respects but one: only the **numeric** data types with a scale of 0 can be used for the IDENTITY column.

float, double and real

Approximate number data types for use with floating point numeric data. Floating point data is approximate; not all values in the data type range can be precisely represented.

float[(precision)]

Floating point number. Storage size is 4 bytes if precision < 25, and 8 bytes if precision >= 25. The range and storage precision is machine dependent.

float columns with a precision of 1-24 are stored as float; those with higher precision are stored as double.

double

Floating point number. The value range is from 2.22507385850721e-308 through 1.79769313486231e+308. Storage size is 8 bytes. The range and storage precision is machine dependent.

real

Floating point number. The value range is from 1.175495e-38 through 3.402823e+38. Storage size is 4 bytes. The range and storage precision is machine dependent.

text

text

Variable-length character data with a size limited by the maximum size of the database file (currently 2 Gb). There are some subsidiary bytes used for storage in addition to the length of the string.

binary, long binary, varbinary and image

Binary data types, which are used to store images and other information that is not interpreted by the database.

binary[(n)]

Fixed-length binary data of n bytes. n must be a value from 1 through 32767. When n is not specified, the default length is 1.

long binary

Variable-length binary data. The maximum size is equal to the maximum size of the database file (currently 2Gb).

varbinary[(n)]

Variable-length binary data of n bytes. n must be a value from 1 through 32767. When n is not specified, the default length is 1.

image

Variable-length binary data. It is implemented in Adaptive Server Anywhere as **long binary** allowing NULL.

This data type is provided primarily for compatibility with Adaptive Server Enterprise.

bit

bit

Integer data type that can be 1 or 0. By default, columns with a data type of bit do not allow NULL, but you can explicitly allow NULL.

Sybase ASA History

Sybase Adaptive Server Anywhere (ASA) is a relational database management system developed by Sybase and used, either as standalone or as a network server, in a multi-user client-server environment.

It is specifically designed to have a small footprint (use fewer memory and disk resources than the average database management system) and extensive scalability for desktop, workgroup or mobile environments.

Sybase Adaptive Server Anywhere started its life as Watcom SQL - an SQL database server product manufactured by Watcom International Corporation. It was originally created for Microsoft-DOS and QNX in 1992.

Watcom International Corporation was founded in 1981, following the research of the computer Systems Group at the University of Waterloo, in Waterloo, Ontario, Canada. Watcom produced a variety of tools, including the well-known Watcom C compiler, introduced in 1988.

In 1994 Watcom was acquired by Powersoft. Then in 1995, Powersoft merged with Sybase and the latest version of Watcom SQL 5.0 was renamed as Sybase SQL Anywhere.

In 1999, Sybase produced SQL Anywhere Studio version 6.0, the complete relation database management system for mobile, desktop and workgroup use. It included the Adaptive Server Anywhere (renamed SQL Anywhere) relation database management system, supplemented with a set of tools for designing and managing databases, creating database reports and forms, using databases on the Web and replicating databases to mobile users.

In May 2000, Sybase transformed their mobile and embedded computing division into its own company, iAnywhere Solutions, set up on the basis of Watcom International Corporation. Since then, this division was engaged in the continuing development of SQL Anywhere Studio.

At the present moment, ASA runs on just about any piece of computing hardware including DOS, PalmOS, Windows, UNIX and Novell NetWare. Its light-weight variant is used in such equipment as mobile phones.

ASA supports two standards - Transact-SQL and Watcom SQL (switching ANSI standard SQL-92). Transact-SQL is compatible with the SQL language of the industrial RDBMS Sybase Adaptive Server Enterprise, having greater scalability and productivity potential. It allows migrating to a high level platform without modifying the existing software.

Sybase ASA Versions Evolution

Version 9.0.0

1. XML support. Adaptive Server Anywhere 9.0 supports XML, including storing XML document, exporting relational data as XML, importing XML, and returning XML from queries on relational data.

- SQL/XML support. SQL/XML is a standard that describes using SQL in conjunction with XML. As part of its SQL/XML support, Adaptive Server Anywhere includes an XML data type that can be used to store XML documents in the database.

- FOR XML clause. The SELECT statement support FOR XML clause that allows obtaining query results as an XML document.

2. HTTP server in the database. Adaptive Server Anywhere database servers can presently act as web servers, making possible the running of web-based applications using only an Adaptive Server Anywhere database and a web browser.

3. 64-bit version available. A full 64-bit version of the software is available for Windows Server 2003 on Itanium II. A deployment release is available on 64-bit Linux and HP-UX operating systems.

4. The WITH clause allows specifying common table expressions. Common table expressions are temporary view definitions existing only within the scope of a SELECT statement. They can be recursive, or non-recursive. They also permit to perform multiple level of aggregation within a single query.

5. Recursive union can now be performed using a common table expression of a particular form. Recursive common table expressions make possible the writing of recursive queries. These are particularly useful when querying tables representing hierarchical structures or directed graphs. Each recursive common table expression contains an initial sub-query, which is executed first, and a recursive sub-query. The reference to the view, which must appear within the FROM clause of the recursive sub-query, references the rows added to the view during the previous iteration.

6. INTERSECT and EXCEPT operations supported. These operations compute intersection and difference between two or more result sets. They complement the UNION operation.

7. SELECT statements can operate on stored procedure result sets. In SELECT statements, a stored procedure call can now appear at any place where a base table or view is allowed.

8. Online analytical processing features added.

- ROLLUP operation. For queries with a GROUP BY clause, the ROLLUP operation adds subtotal rows into the result set. Each subtotal row provides an aggregate over a set of rows in the GROUP BY clause.

- The LIST function can include ordered lists.

- Addition aggregate functions. Functions have been added to compute sample-based and population-based deviations and variances.

9. The CREATE INDEX statement permits an index to be created on a built-in function. This feature is a convenience method adding a new computed column to the table, and creating an index on that column.

10. ORDER BY clause allowed in all contexts. In the earlier releases, it was not allowed for many SELECT statements in view definitions, in sub-queries, or in UNION operations to use ORDER BY clause. This restriction has presently been removed.

11. SELECT statements can now include START AT as part of the TOP clause. START AT provides additional flexibility in queries that impose explicit limitations on the result set.

12. Constraints can now be named. Names can be now assigned to check constraints, unique constraints, and referential integrity constraints.

13. Lateral derived tables permit outer references in the FROM clause. Outer references can now be made from derived tables and from stored procedures in the FROM clause. The LATERAL keyword is used to indicate that an outer reference is being made.

14. CREATE FUNCTION and ALTER FUNCTION now permit Transact-SQL syntax. User defined functions can now be created in the Transact-SQL dialect that return a scalar value to the calling environment.

15. The new function EXPRTYPE returns the data type of an expression.

Version 8.0.2

1. Clustered index support. Creating a clustered index on a table causes the rows in that table to be stored approximately in the same order as they appear in the index.

2. Unique identifier support. Adaptive Server Anywhere supports unique identifiers (UUIDs and GUIDs). UUIDs (universally unique identifiers) and GUIDs (global unique identifiers) that allows uniquely identifying rows, even across distinct databases.

3. Update existing rows with the ON EXISTING clause. The ON EXISTING clause of the INSERT statement can be used to update existing rows with new values, as long as the table has a primary key.

4. New joins added. New joins added to this release include the nested loop semijoin, the nested loop anti-semijoin, the hash semijoin and the hash anti-semijoin.

5. Hide procedure text to keep logic confidential. The logic contained in stored procedures, functions, triggers, and views can be hidden by using the SET HIDDEN option. This allows applications and databases to be distributed without revealing the logic in stored procedures, functions, triggers, and views.

6. SET statement can be used to assign variable values.

7. INSERT statement presently supports WITH AUTO NAME. If WITH AUTO NAME is specified in an INSERT statement, the names of the items in the SELECT list determine the associations of values to destination columns. This is useful when the number of columns in the destination table is too large.

8. Triggers can discriminate among the actions that caused a trigger to fire. Different actions now carried out depending on whether the trigger was fired by an INSERT, UPDATE, or DELETE operation. This feature enables sharing logic among the different events within a single trigger, and carrying out some actions in an action-depend manner.

Version 8.0.1

1. Determine ANSI equivalency of non-ANSI statements. The REWRITE functions accept a new parameter, ANSI, which cause the function to return the ANSI equivalent of any SELECT, DELETE, or UPDATE statement.

2. Variable assignment allowed in the UPDATE statement. The SET clause of the UPDATE statement can now be used for assigning a value to a variable, in addition to updating the table. This feature is compatible with Adaptive Server Enterprise.

3. Square brackets can delimit identifiers.

4. Specify isolation level in the FROM clause. The WITH table-hint argument can be used for specifying a locking method for a particular table or view for particular SELECT, UPDATE, or DELETE statement.

Version 8.0.0

1. Full outer joins are now supported.

2. Two forms of ANSI standard CASE statements are now supported.

Version 7.0.3

1. Database properties for padding and case sensitivity. Two new properties to determine if database uses blank padding when comparing strings (BlankPadding) or if database is case sensitive (CaseSensitive).

Version 7.0.2

1. Oracle, IBM DB2, Microsoft SQL Server, Sybase Adaptive Server Enterprise, Sybase Adaptive Server Anywhere, and Microsoft Access databases can be migrated (imported) remotely into Adaptive Server Anywhere using the new sa_migrate set of stored procedures.

2. Truncate timestamp option for compatibility with non-Adaptive Server Anywhere databases, timestamp values can be truncated.

Version 7.0.1

1. Windows CE 3.0 support.

Version 7.0.0

1. Unloading result sets. The new UNLOAD SQL statement makes possible the unloading of query result set into a comma-delimited text file.

2. Distributed transactions and three-tired computing. Distributed transactions include operations on more than one server in a single transaction. A transaction server controls the commit and rollback behavior of distributed transactions.

In this release, Adaptive Server Anywhere can participate in distributed transactions coordinated by the Microsoft Distributed Transaction Coordinator (DTC). Products such as Sybase Application Server and Microsoft Transaction Server can use DTC for the coordination of transaction, so DTC support enables Adaptive Server Anywhere to participate in three-tired computing with these products.

Version 6.0.3

1. Read-only databases. Database can be designated as read-only when a database server is started. This feature makes deployment of databases on read-only media, such as CD-ROMs, more straightforward.

Version 6.0.2

1. The ODBC driver has been updated to ODBC 3.51. This version of ODBC includes support for Unicode Applications.

Version 6.0.1

1. Adaptive Server Anywhere is available for Windows CE.

Version 6.0

1. Java in the database. Adaptive Server Anywhere provides a runtime environment for Java in the database, and allows using Java classes as SQL data types.

2. Multi-processor support and improved multitasking.

3. SQL Anywhere Version 5 and other previous releases of the software have been able to handle multiple connections concurrently, but query processing has been carried out on only a single operation system thread. Consequently, even on multi-processor machines, only a single processor was used for query processing.

4. The data processing engine in Adaptive Server Anywhere can now run using multiple operation-system threads, and hence can take advantage of multiple processors. By default, Adaptive Server Anywhere uses the same number of operating system threads as the machine has CPU's.

5. Multi-processor support in Adaptive Server Anywhere enables transactions from separate connections to run simultaneously on separate CPU's. A single connection uses a single thread, and no single requests are split among CPU's.

6. Query optimization enhancements.

- Rewrite optimization to transform a query into a query that has a greater potential for cost-based optimization to find the most efficient access plan.
- Carries out the following kinds of transformation for rewriting optimization:
  - •Subquery rewriting. In many cases subquery rewritten as joins.
  - •Elimination of unnecessary DISTICT clauses.
- Subquery caching. There are subqueries that cannot be rewritten as joins - for example, subqueries involving aggregate functions. In these cases, there is a trade off between evaluating the subquery each time it is referenced (CPU time) or caching (storing) the subquery result for repeated use (memory use, and possible disk access).
- Adaptive Server Anywhere Version 6 carries out some caching of subqueries.
- Fractional user-supplied estimates. In those cases where the cost-based optimizer does not generate the best access plan, and where query performance is important, users can supply estimates of various parts of the query in order to prompt the optimizer.

7. Programming interface enhancements.

- New ODBC features:
  - •Native ODBC. Presently the ODBC driver sends commands directly to the database server, without using "native" library. An ODBC driver in previous versions translated ODBC calls to a set of "native" calls, which were passed to the native interface DLL.
  - •Adaptive Server Anywhere supports ODBC 3.0.

8. New SQL features.

- UNSIGNED data types. Unsigned versions of the INT and SMALLINT data types have been added. These versions can hold only non-negative numbers, but have a large maximum value than signed version.
- New data types. A 64-bit integer data type (BIGINT) has been added, in both signed and unsigned versions. Also, a VARBINARY data type is supported.
- SELECT list enhancements. The first few rows of a query can be retrieved using the following syntax: SELECT [FIRST|TOP n] select_list.
- Derived tables. Now, queries can be used in FROM clause. This makes possible writing complex queries without creating a view.
- Computed columns. The value of a computed column is an expression, usually based on other columns.

Version 5.5.1

1. A new special constant, LAST USER, has been added for identifying which user was the last to change a row.

2. Index names must now be unique for a given table, instead for a given user ID.

3. EXECUTE (string) is an alternative syntax for the EXECUTE IMMEDIATE statement.

4. A new database option, FIRE_TRIGGERS, allows trigger firing to be turned off. This is relevant for SQL Server to SQL Anywhere replication.

Version 5.5

With release 5.5, Sybase SQL Anywhere can be obtained in a Standard and a Professional Edition. The Professional Edition includes additional components. In particular, the following components are included in the Professional Edition:

- NetImpact Dynamo Internet features;
- The Powersoft Infomaker reporting tool;

1. Several database options have been added to improve compatibility with SQL Server and ISO/SQL92, while maintaining the ability to be compatible with the existing applications.

2. The CONVERT function presently converts strings to dates and times, as well as converse.

3. Live backup. A new option for the backup utility allows a continuous backup of the transaction log. In case of a server shutdown, this log file can be used for a rapid restart of the system.

Version 5.0.3

1. A new clause has been added to the DESCRIBE statement to retrieve column names with more than thirty characters.

2. ALTER VIEW and ALTER TRIGGER statements make possible the replacement of an existing view or trigger definition by a new definition.

3. Transact-SQL accepted by parser. To enhance the SQL Server compatibility, performance parameters that can be used in SELECT, UPDATE, and DELETE statements in System 11 are now accepted and discarded by SQL Anywhere. Also, the PREFETCH System 11 option is allowed, but has no effect.

Version 5.0.2

1. CREATE SCHEMA statement creates a collection of tables, views, and associated permissions for a user.

2. FLOAT (p) data type stored as REAL or DOUBLE, depending on the value of p.

3. More than one foreign keys per table can now be created.

4. Column default extension. Constant expressions are now allowed as column defaults, as long as they do not reference database objects.

5. Transact-SQL support.

CREATE TABLE and CREATE INDEX statements now support ON segment-name clause. CREATE INDEX permits CLUSTERED and UNCLUSTERED keywords, although no clustering is performed.

The GOTO control of flow statement is now supported.

The system objects owned by the DBO user ID are now not unloaded by either Unload or Extraction utilities.

6. Multiple-event triggers. A single trigger can presently be defined to handle a combination of INSERT, UPDATE, and DELETE operations on a table.

Version 5.0.1

1. Numbering scheme. Sybase SQL Anywhere uses numbers for indicating patch or maintenance-level release. Release 5.0.01 is the first maintenance-level release for Sybase SQL Anywhere 5.0. The numbering scheme replaces alphabetic scheme used for Watcom SQL patch-level releases.

2. Release 5.0.01 is released to address some problems in the initial 5.0 release, particularly a backwards-compatibility problem.

Version 5.0

Previous releases of this product were released under the name Watcom SQL.

With the Sybase\Powersoft merger of February 1995, Watcom became a part of Sybase, Inc. Watcom SQL is presently part of the Sybase System 11 product line, and this change is reflected in the new name of the product, SQL Anywhere. The principal dialect of SQL supported by SQL Anywhere is named Watcom-SQL.

1. SQL Central database management tool. SQL Central is a graphical database administration tool that runs on the Windows 95 and Windows NT 3.51 operation systems.

2. Transact-SQL compatibility. SQL Anywhere 5.0 includes a set of extensions to Watcom-SQL from the Sybase Transact-SQL dialect. This makes the development of compatible applications for SQL Anywhere and SQL Server database servers much more straightforward, and also brings new features to all SQL Anywhere users.

3. ODBC 2.5 support. SQL Anywhere 5.0 now supports ODBC 2.5 at level 2 for the Windows 95 and Windows NT operating systems, and ODBC 2.1 for the Windows 3.x operating systems.

4. Open Server Gateway. The Open Server Gateway allows client applications to work with both with SQL Server and SQL Anywhere database servers.

5. Transaction log mirroring. SQL Anywhere can optionally maintain two identical transaction logs. Transaction log mirroring provides additional protection against loss of data due to disk failure.

6. Calls to external DLLs from procedures. You can now create procedures that call external DLLs.

7. Stored procedure extensions. Stored procedures have more flexible parameter declarations (including default values and optional parameters in CALL statement), and can return status information.

8. User-defined functions. The new CREATE FUNCTION statement of the SQL Anywhere stored procedure language can be used to define new functions, which can be used just like other functions.

9. Batches. This release introduces support for batches of SQL statements. Control statements (IF, LOOP, and so on) are presently available in command file as well as in procedures and triggers.

10. Statement level triggers. Triggers can be made to fire once after each statement, as alternative to the row-level triggers supported in the earlier versions.

11. New Watcom-SQL elements. These include global variables, and new operators.

12. New built-in functions. New numeric, string, date and time, and data type conversion functions have been added, as well as new system functions. New functions include:

- System functions describing database properties;
- PATINDEX for pattern matching;

13. User-defined data types. User can define own data types from the existing base data types supported by SQL Anywhere.

14. Extensions to data manipulation statements. The DELETE and UPDATE statements have been extended to support criteria based on joins.

15. System stored procedures. SQL Server provides stored procedures for carrying out database management functions. The system tables of SQL Server and SQL Anywhere are different, but several of system procedures are provided to carry out analogous actions on each.

16. More flexible column constraints. Prior to this release, all constraints associated with a table, whether column constraints or table constraints, were held as a single table constraint. Column constraints are now held individually, which allows them to be individually deleted or replaced.

17. Other new statements have been added: LOAD TABLE, UNLOAD TABLE, TRUNCATE TABLE, MESSAGE, EXECUTE IMMEDIATE, RETURN, and others.

Migrating to MySQL

This chapter is devoted to migrating databases to MySQL server, a relational database management system.

- MySQL Data Types
- MySQL Reserved Words
- Importing Data into MySQL

MySQL Data Types

This section describes the data types supported by MySQL.

- CHAR, NCHAR and VARCHAR
- BIGINT,INT, INTEGER, MEDIUMINT, SMALLINT and TINYINT
- DECIMAL, DEC and NUMERIC
- FLOAT
- DOUBLE, DOUBLE PRECISION and REAL
- DATE, TIME, DATETIME, TIMESTAMP and YEAR
- TINYBLOB, BLOB, MEDIUMBLOB and LONGBLOB
- TINYTEXT, TEXT, MEDIUMTEXT and LONGTEXT
- BIT and BOOL
- ENUM and SET

CHAR, NCHAR and VARCHAR

[**NATIONAL**]**CHAR**[(*n*)] [**BINARY**]

A fixed-length character data. The range of *n* is 0 to 255 characters (1 to 255 prior to MySQL Version 3.23). When *n* is not specified the default value is 1.

When CHAR values are stored, they are right-padded with spaces to the specified length. When CHAR values are retrieved, trailing spaces are removed. BINARY keyword is given to sort and compare CHAR values in case-sensitive manner. NATIONAL CHAR (or NCHAR) is used to define that a CHAR column should use the default CHARACTER set. This is the default in MySQL.

If the column is defined as CHAR(0) and nullable then the column will occupy only one bit and can take two values: NULL or "" (empty string).

[**NATIONAL**] **VARCHAR**(*n*) [**BINARY**]

A variable-length string. The range of *n* is 0 to 255 characters (1 to 255 prior to MySQL Version 4.0.2). Value is stored in database using only as many characters as it contains, plus one byte to record the length.

When the value is stored trailing spaces are removed.

BINARY keyword is given to sort and compare VARCHAR values in case-sensitive manner.

BIGINT, INT, INTEGER, MEDIUMINT, SMALLINT and TINYINT

BIGINT [**UNSIGNED**]

Integer data. The signed range is -9223372036854775808 to 9223372036854775807. The unsigned range is 0 to 18446744073709551615. Storage size is 8 bytes.

INT [**UNSIGNED**]

Integer data. The signed range is -2147483648 to 2147483647. The unsigned range is 0 to 4294967295. Storage size is 4 bytes.

INTEGER [**UNSIGNED**]

This is a synonym for INT.

MEDIUMINT [**UNSIGNED**]

Integer data. The signed range is -8388608 to 8388607. The unsigned range is 0 to 16777215. Storage size is 3 bytes.

SMALLINT [**UNSIGNED**]

Integer data. The signed range is -32768 to 32767. The unsigned range is 0 to 65535. Storage size is 2 bytes.

TINYINT [**UNSIGNED**]

Integer data. The signed range is -128 to 127. The unsigned range is 0 to 255. Storage size is 1 byte.

DECIMAL, DEC and NUMERIC

DECIMAL[(p[,s])] [**UNSIGNED**]

An unpacked floating-point number. DECIMAL is stored as a string, using one character for each digit of the value.

The p specifies the total number of the digits not including the decimal point and, the `-' sign. Prior to MySQL Version 3.23, the p argument must include the space needed for the sign and the decimal point.

The s specifies the number of digits following the decimal point.

The maximum range of DECIMAL values is the same as for DOUBLE, but the actual range for a given DECIMAL column may be constrained by the choice of p and s.

If p is omitted, the default is 10.

If s is omitted, the default is 0, that is, values will have *no* decimal point or fractional part.

If UNSIGNED is specified, negative values are not allowed.

DEC[(p[,s])] [**UNSIGNED**]

NUMERIC[(p[,s])] [**UNSIGNED**]

These are synonyms for DECIMAL. DEC data type is available since MySQL version 4.00.

FLOAT

FLOAT[(p[,s])] [**UNSIGNED**]

A floating-point number. Precision p can be <=24 for a single-precision floating-point number (allowable values are -3.402823466E+38 to -1.175494351E-38, 0, and 1.175494351E-38 to 3.402823466E+38) and between 25 and 53 for a double-precision floating-point number.

The p is the precision, or the total number of digits. The s is the scale, or the number of digits to the right of the decimal point.

FLOAT without arguments stands for a single-precision floating-point number. In MySQL Version 3.23, this is a true floating-point value. In earlier MySQL versions, FLOAT(p) always has 2 decimals. All calculations in MySQL are done with double precision.

If UNSIGNED is specified, negative values are not allowed.

DOUBLE, DOUBLE PRECISION and REAL

DOUBLE[(p,s)] [**UNSIGNED**]

A normal-size (double-precision) floating-point number. Allowable values are -1.7976931348623157E+308 to -2.2250738585072014E-308, 0, and 2.2250738585072014E- 308 to 1.7976931348623157E+308.

The p is the precision, or the total number of digits. The s is the scale, or the number of digits to the right of the decimal point.

DOUBLE without arguments stands for a double-precision floating-point number.

If UNSIGNED is specified, negative values are not allowed.

DOUBLE PRECISION[(p,s)] [**UNSIGNED**]

REAL[(p,s)] [**UNSIGNED**]

These are synonyms for DOUBLE.

DATE, TIME, DATETIME, TIMESTAMP and YEAR

DATE

The DATE data type stores the date information (year, month, day). The supported range is January 1, 1000 to December 31, 9999.

TIME

The TIME data type stores the time information (hours, minutes, seconds). The supported range is -838 hours 59 minutes 59 seconds to 838 hours 59 minutes 59 seconds.

DATETIME

The DATETIME data type stores both date and time information (year, month, day, hours, minutes, seconds). The supported range is January 1, 1000 00 hours 00 minutes 00 seconds to December 31, 9999 23 hours 59 minutes 59 seconds.

TIMESTAMP[(p)]

The TIMESTAMP data type stores the date and time information (year, month, day, hours, minutes, seconds). The difference between TIMESTAMP and DATATYPE is that TIMESTAMP column automatically changes its value to the current date and time for the row, which is updated or inserted into the table.

The range is January 1, 1970 00 hours 00 minutes 00 seconds to January 1, 2037 00 hours 00 minutes 00 seconds.

The allowed values of p is 14 (or missing), 12, 8, 6, 4 or 2 (4 or 2 are not allowed in MySQL versions prior to v4.0). The p argument (4 bytes long) affects how a TIMESTAMP column is displayed. The TIMESTAMP(p) columns where p is 8 or 14 are reported to be numbers. The other TIMESTAMP(p) columns are reported to be strings.

YEAR[(2|4)]

The YEAR type is a 1-byte type used for representing years.

A year in 2- or 4-digit format (default is 4-digit). The allowable values are 1901 to 2155, 0000 in the 4-digit year format, and 1970 to 2069 if you use the 2-digit format (70-69). (The YEAR type is not available prior to MySQL Version 3.22.)

TINYBLOB, BLOB, MEDIUMBLOB and LONGBLOB

A binary large object that can hold a variable amount of binary data. The sorting and comparison of the values for these objects is performed in case-sensitive manner.

TINYBLOB

A binary large object column with a maximum length of 255 (2^8 - 1) characters.

BLOB

A binary large object column with a maximum length of 65535 (2^16 - 1) characters.

MEDIUMBLOB

A binary large object column with a maximum length of 16777215 (2^24 - 1) characters.

LONGBLOB

A binary large object column with a maximum length of 4294967295 (2^32 - 1) characters.

TINYTEXT, TEXT, MEDIUMTEXT and LONGTEXT

A character large object that can hold a variable amount of case-insensitive text data.

TINYTEXT

A character large object column with a maximum length of 255 (2^8 - 1) characters.

TEXT

A character large object column with a maximum length of 65535 (2^16 - 1) characters.

MEDIUMTEXT

A character large object column with a maximum length of 16777215 (2^24 - 1) characters.

LONGTEXT

A character large object column with a maximum length of 4294967295 (2^32 - 1) characters.

BIT and BOOL

BIT

BOOL

These are synonyms for TINYINT.

ENUM and SET

ENUM('value1','value2',...)

A character data that can have only one value, chosen from the list of values 'value1', 'value2', ..., NULL or the special "" error value specified when the table is created. An ENUM can have a maximum of 65535 distinct values.

SET('value1','value2',...)

A character data that can have zero or more values, which must be chosen from the list of values 'value1', 'value2', ... specified when the table is created. A SET can have a maximum of 64 members.

MySQL Reserved Words

There are reserved words in MySQL which cannot be used as identifiers (table or column names etc.) without being quoted with backticks (`). Quoting of identifiers was introduced in MySQL Version 3.23.6. You can also enclose identifiers with double quotation marks (") if you run MySQL in ANSI mode.

When migrating databases to MySQL, SQLWays automatically resolves conflicts with MySQL reserved words. SQLWays delimits identifiers which are reserved words in MySQL with single quotation marks.

The following words are reserved by MySQL:

TABLE 97. MySQL reserved words

ADD	ALL	ALTER
ANALYZE	AND	AS
ASC	AUTO_INCREMENT	BDB
BERKELEYDB	BETWEEN	BIGINT
BINARY	BLOB	BOTH
BTREE	BY	CASCADE
CASE	CHANGE	CHAR
CHARACTER	CHECK	COLLATE
COLUMN	COLUMNS	CONSTRAINT
CREATE	CROSS	CURRENT_DATE
CURRENT_TIME	CURRENT_TIMESTAMP	DATABASE
DATABASES	DAY_HOUR	DAY_MINUTE
DAY_SECOND	DEC	DECIMAL
DEFAULT	DELAYED	DELETE
DESC	DESCRIBE	DISTINCT
DISTINCTROW	DIV	DOUBLE
DROP	ELSE	ENCLOSED
ERRORS	ESCAPED	EXISTS
EXPLAIN	FALSE	FIELDS
FLOAT	FOR	FORCE
FOREIGN	FROM	FULLTEXT
FUNCTION	GEOMETRY	GRANT
GROUP	HASH	HAVING
HELP	HIGH_PRIORITY	HOUR_MINUTE
HOUR_SECOND	IF	IGNORE
IN	INDEX	INFILE
INNER	INNODB	INSERT
INT	INTEGER	INTERVAL
INTO	IS	JOIN
KEY	KEYS	KILL
LEADING	LEFT	LIKE
LIMIT	LINES	LOAD

TABLE 97. MySQL reserved words

LOCALTIME	LOCALTIMESTAMP	LOCK
LONG	LONGBLOB	LONGTEXT
LOW_PRIORITY	MASTER_SERVER_ID	MATCH
MEDIUMBLOB	MEDIUMINT	MEDIUMTEXT
MIDDLEINT	MINUTE_SECOND	MOD
MRG_MYISAM	NATURAL	NOT
NULL	NUMERIC	ON
OPTIMIZE	OPTION	OPTIONALLY
OR	ORDER	OUTER
OUTFILE	PRECISION	PRIMARY
PRIVILEGES	PROCEDURE	PURGE
READ	REAL	REFERENCES
REGEXP	RENAME	REPLACE
REQUIRE	RESTRICT	RETURNS
REVOKE	RIGHT	RLIKE
RTREE	SELECT	SET
SHOW	SMALLINT	SOME
SONAME	SPATIAL	SQL_BIG_RESULT
SQL_CALC_FOUND_ROWS	SQL_SMALL_RESULT	SSL
STARTING	STRAIGHT_JOIN	STRIPED
TABLE	TABLES	TERMINATED
THEN	TINYBLOB	TINYINT
TINYTEXT	TO	TRAILING
TRUE	TYPES	UNION
UNIQUE	UNLOCK	UNSIGNED
UPDATE	USAGE	USE
USER_RESOURCES	USING	VALUES
VARBINARY	VARCHAR	VARCHARACTER
VARYING	WARNINGS	WHEN
WHERE	WITH	WRITE
XOR	YEAR_MONTH	ZEROFILL

These are the words which are disallowed by ANSI SQL but allowed by MySQL as column or table names:

- ACTION
- BIT
- DATE
- ENUM
- *NO*

- TEXT
- TIME
- TIMESTAMP

Importing Data into MySQL

LOAD DATA INFILE Command

During the import stage SQLWays uses the LOAD DATA INFILE command to load data from text files into MySQL tables.

The command has the following syntax:

LOAD DATA [LOW_PRIORITY | CONCURRENT] [LOCAL] INFILE 'file_name.txt'

```
[REPLACE | IGNORE]
INTO TABLE tbl_name
[FIELDS
   [TERMINATED BY '\t']
   [[OPTIONALLY] ENCLOSED BY '']
   [ESCAPED BY '\\' ]
] [LINES
   [STARTING BY '']
   [TERMINATED BY '\n']
] [IGNORE number LINES]
[(col_name,...)]
```

There are several options used with the LOAD DATA INFILE command:

REPLACE

If the table already exists in MySQL, you can use this option when you want existing rows to be updated basing on primary or unique key information or added to the table if primary key is not matched.

IGNORE

If the table already exists in MySQL, you can use this option when you do not want to update existing rows, when primary or unique key values are equivalent to the existing in the table. The rows from the text file which do not match primary or unique key in the table are inserted.

Remarks:

IGNORE and REPLACE options works properly only if tables have primary or unique keys.

IGNORE number LINES

This option specifies the number of rows from the beginning of the text file to be skipped during the insertion into the table.

Migrating to Pervasive.SQL

This chapter is devoted to migrating databases to the Pervasive.SQL server, a relational database management system.

- Data Type

Pervasive.SQL Data Types

This section describes the data types supported by Pervasive.SQL.

- CHAR and VARCHAR
- BIGINT, UBIGINT, INTEGER, UINTEGER, SMALLINT, USMALLINT, TINYINT and UTINYINT
- DATE, TIME and TIMESTAMP
- DECIMAL, NUMERIC, NUMERICSA and NUMERICSTS
- MONEY and CURRENCY
- FLOAT, REAL, DOUBLE, BFLOAT4 and BFLOAT8
- LONGVARCHAR
- BINARY and LONGVARBINARY
- BIT, IDENTITY and SMALLIDENTITY

CHAR and VARCHAR

CHAR(*n*)

Fixed-length character data with length of *n* characters. *n* must be a value from 1 through 255. CHAR columns are padded with blanks. You must specify the size for CHAR columns.

VARCHAR(*n*)

Variable-length character data with length of *n* characters. *n* must be a value from 1 through 254. VARCHAR columns are not padded with blanks. You must specify the size for VARCHAR columns.

BIGINT, UBIGINT, INTEGER, UINTEGER, SMALLINT, USMALLINT, TINYINT and UTINYINT

BIGINT

Integer (whole number) data from -2^63 (-9223372036854775808) through 2^63-1 (9223372036854775807). Storage size is 8 bytes.

UBIGINT

Unsigned integer (whole number) data from 0 through 18446744073709551615. Storage size is 8 bytes.

INTEGER

Integer (whole number) data from -2^31 (-2,147,483,648) through 2^31 - 1 (2,147,483,647). Storage size is 4 bytes.

UINTEGER

Unsigned integer (whole number) data from 0 through 2^32 (4294967295) Storage size is 4 bytes.

SMALLINT

Integer data from -2^15 (-32,768) through 2^15 - 1 (32,767). Storage size is 2 bytes.

USMALLINT

Unsigned integer data from 0 through 2^16 (65535). Storage size is 2 bytes.

TINYINT

Integer data from -2^7 (-128) through 2^7 - 1 (127). Storage size is 1 byte.

UTINYINT

Unsigned integer data from 0 through 255. Storage size is 1 byte.

DATE, TIME and TIMESTAMP

Date and time data types for representing date and time of day.

DATE

The DATE data type stores date information (year, month, day) from 01-01-0001 through December 31, 9999. The DATE data type is stored internally as a 4-byte value.

TIME

The TIME data type stores time information (hour, minute, second and hundreds of a seconds) from 00:00:00 through 23:59:59.

The TIME data type is stored internally as a 4-byte value.

TIMESTAMP

The TIMESTAMP data type stores day and time information (year, month, day, hour, minute, second and fractional seconds) from 0001-01-01 00:00:00.0000000 through 9999-12-31 23:59:59.9999999 UTC.

The TIMESTAMP data type is stored internally as a 8-byte unsigned value representing septaseconds (10^-7 second) since January 1, 0001 in a Gregorian calendar, Coordinated Universal Time (UTC).

DECIMAL, NUMERIC, NUMERICSA and NUMERICSTS

Numeric data types with fixed precision and scale.

DECIMAL(p,s)

p (precision) specifies the maximum total number of decimal digits that can be stored, both to the left and to the right of the decimal point. The precision must be a value from 1 through 64.

s (scale) specifies the maximum number of decimal digits that can be stored to the right of the decimal point. Scale must be a value from 0 through p.

Precision and scale must be specified.

The DECIMAL data type is stored internally as a packed decimal number with two decimal digits per byte.

NUMERIC(p,s)

The NUMERIC data type is similar to DECIMAL. The maximum precision is 15.

NUMERIC values are stored as ASCII strings, right justified with leading zeros. Each digit occupies one byte internally. The embedded sign has an EBCDIC value.

NUMERICSA(p,s)

The NUMERICSA data type (sometimes called NUMERIC SIGNED ASCII) is a COBOL data type that is the same as the NUMERIC data type, except that the embedded sign has an ASCII value instead of an EBCDIC value.

The maximum precision is 15.

NUMERICSTS(p,s)

The NUMERICSTS data type (sometimes called SIGN TRAILING SEPARATE) is a COBOL data type that is the same as the NUMERIC data type, except that the sign is saved in a separate byte.

The maximum precision is 14.

MONEY and CURRENCY

MONEY

The MONEY data type has the same internal representation as the DECIMAL type, with an implied two decimal places.

MONEY is a legacy data type and Pervasive Software recommends to use the DECIMAL data type instead of MONEY.

CURRENCY

The CURRENCY data type has an implied four digit scale of decimal places, which represents the fractional component of the currency data value.

FLOAT, REAL, DOUBLE, BFLOAT4 and BFLOAT8

FLOAT

The FLOAT data type is consistent with the IEEE standard for single and double-precision real numbers.

REAL

The REAL data type is defined as a 4-byte FLOAT.

DOUBLE

The DOUBLE data type is defined as a 8-byte FLOAT.

BFLOAT4

The BFLOAT4 data type is a single precision real number.

The BFLOAT4 type is commonly used in legacy BASIC applications. Microsoft refers to this data type as MBF (Microsoft Binary Format), and *no* longer supports this type in the Visual Basic environment. Pervasive Software recommends to use the REAL data type instead of BFLOAT4.

BFLOAT8

The BFLOAT8 data type is a double precision real number.

The BFLOAT8 type is commonly used in legacy BASIC applications. Microsoft refers to this data type as MBF (Microsoft Binary Format), and *no* longer supports this type in the Visual Basic environment. Pervasive Software recommends to use the DOUBLE data type instead of BFLOAT4.

LONGVARCHAR

LONGVARCHAR

Variable-length character data with a maximum length of 2^31 - 1 (2,147,483,647) characters. Storage size is the actual length in bytes of the data entered + 8-byte header.

BINARY and LONGVARBINARY

Binary data types.

BINARY(*n*)

Fixed-length binary data of *n* bytes. *n* must be a value from 1 through 255. BINARY columns are padded with binary zeros. You must specify the size for BINARY columns.

LONGVARBINARY

Variable-length binary data from 0 through 2^31-1 (2,147,483,647) bytes. Storage size is the actual length in bytes of the data entered + 8-byte header.

BIT, IDENTITY and SMALLIDENTITY

BIT

Integer data type that can be 1 or 0. Integer values other than 0 or 1 are accepted, but are always interpreted as 1. Columns with a data type of BIT cannot be NULL.

If some BIT columns follow each other, they are saved in one byte, otherwise each BIT is stored in one byte.

IDENTITY

The IDENTITY data type is stored as a signed 4-byte INTEGER. IDENTITY is used to automatically assign the next highest value when a record is inserted into a table.

When an INSERT operation occurs for a record with ASCII zero in the IDENTITY field, the database engine assigns the next highest value.

SMALLIDENTITY

The SMALLIDENTITY data type is the same as the IDENTITY data type, except that a signed 2-byte integer (SMALLINT) is used to store SMALLIDENTITY values.

Tables

This chapter describes the various aspects of working with tables and implementation differences in databases.

- Column Default Values
- Dropping Tables
- Importing Tables

Column Default Values

This section describes usage of the DEFAULT option in various databases and their conversion by SQLWays.

DEFAULT is used to assign the default value to the column if a subsequent INSERT statement omits the value for the column.

TABLE 98. Column Default Values

Database	Syntax	Description
MySQL	DEFAULT expression	DEFAULT allows you to specify the default value to the column in MySQL. Expression can include a constant.
Microsoft SQL Server	DEFAULT expression	DEFAULT allows you to specify the default value to the column in Microsoft SQL Server. Expression can include a constant, built-in function or a mathematical expression.
Oracle	DEFAULT expression	DEFAULT allows you to specify the default value to the column in Oracle. Expression can include a constant, any SQL function except for those, which return a literal argument, a column reference or a nested function invocation. Restrictions: expression cannot contain references to PL/SQL functions or to other columns, date constants that are not fully specified and pseudo columns LEVEL, CURRVAL, NEXTVAL, and ROWNUM.
IBM DB2	DEFAULT expression	DEFAULT allows you to specify the default value to the column in DB2. Expression can include a constant, a special register like CURRENT DATE, CURRENT TIME or CURRENT TIMESTAMP, CURRENT SCHEMA, USER and a cast function. Restrictions: expression cannot contain the value of the DATALINK type, object identifier of a typed table, the data of the structured type.

Conversion of DEFAULT syntax to MySQL

MySQL allows you to specify only constants for the DEFAULT values whereas other databases (Oracle, DB2 etc.) allow you to use expressions like CURRENT DATE as the default. If a DEFAULT value is not constant string or number, SQLWays removes the default option when converting databases to MySQL.

The only exception is if a column of the source database has a datetime data type (date with time part) with the default expression which returns the current system date and time. In this case SQLWays removes the default and converts the data type to the MySQL TIMESTAMP data type. This is because the MySQL TIMESTAMP data type as the default takes the current system date and time:

TABLE 99. Conversion of DEFAULT syntax to MySQL

SQL Server	MySQL
CREATE TABLE table_name1 (col1 datetime default GETDATE())	CREATE TABLE table_name1 (col1 TIMESTAMP);

Conversion of DEFAULT values from IBM DB2 to Oracle

Oracle interprets DEFAULT '' as NULL whereas such syntax is interpreted as an empty string in IBM DB2. In order to insert an empty string as the default in Oracle, the syntax must be like DEFAULT ' ' (with the space between single quotes).

SQLWays converts DEFAULT '' from IBM DB2 to Oracle as DEFAULT ' ':

TABLE 100. Conversion of DEFAULT values from IBM DB2 to Oracle

IBM DB2	Oracle
CREATE TABLE table_name1 (col1 varchar(10) default '')	CREATE TABLE table_name1 (col1 varchar2(10) default ‘);

Dropping Tables

The DROP TABLE statement is used to drop tables. The following statement drops the BOOKS table:
DROP TABLE books

The table you want to drop may be referenced by a FOREIGN KEY constraint.

- IBM DB2 Universal Database drops foreign keys referencing the table.
- Oracle does not allow you to drop tables referenced by foreign keys of other tables without specifying the CASCADE CONSTRAINTS option in the DROP TABLE statement.

The CASCADE CONSTRAINTS option drops the FOREIGN KEY constraints of the child tables. For example,
DROP TABLE books CASCADE CONSTRAINTS

- Microsoft SQL Server and Sybase Adaptive Server do not allow you to drop referenced tables. The referencing FOREIGN KEY constraint or the referencing table must first be dropped.

To generate the CASCADE CONSTRAINTS option for Oracle

- SQLWays Wizard - DDL Options Page
- SQLWays command line - [DDL] SubSection

Importing Tables

This chapter describes importing tables to the target database using SQLWays.

Importing tables that already exist in the database

Tables that you import may already exist in the target database. For example, tables might be created by another tool and you use SQLWays to import data only. Or you perform the migration process several times, and after first migration all tables exist in the target database.

In this case, you may want to drop tables before importing them again.

SQLWays has the option to generate the DROP TABLE statement before each CREATE TABLE statement. This option allows you to drop the table before re-creating it.

To generate the DROP TABLE statement

- SQLWays Wizard - DDL Options Page
- SQLWays command line - [DDL] SubSection

Executing a DROP TABLE statement in some databases you may face a problem that the table cannot be dropped. For more information see, Dropping Tables

Application Migration Concepts

This title implies the concepts of the migration process performed with the help of SQLWays.

- Progress 4GL to C# .NET Migration
- PowerBuilder Migration
- Oracle PL/SQL to Java Rename

Progress 4GL to C# .NET Migration

Conversion Features

- **4GL and ABL Application Files**
  - Converts .p and .w files to .cs C#.NET files
  - Converts .i include files to C# code
- **4GL and ABL Triggers**
  - Converts 4GL database trigger procedures to database triggers (T/SQL triggers for SQL Server, and PL/SQL triggers for Oracle)

PowerBuilder Migration

SQLWays can help you automatically convert SQL statements, table and column names in PowerBuilder applications to conform to the syntax and requirements of the new database platform.

- **Challenges**

  The necessity of PowerBuilder applications conversion is often underestimated in database migration projects. But even if you continue using PowerBuilder (not migrating to Java, .NET i.e.), the application migration can still take 30-50% of the project cost.

  In the most cases, SQL statements in PowerBuilder applications are coded in the database native syntax that is different for Sybase, Oracle, Informix, SQL Server, PostgreSQL, MySQL etc.

  Even if you use the PowerBuilder graphical features to define the datasources (PBSELECT), you often need to modify table and column names throughout the entire application code.

  Some database migration projects involve significant database schema changes (renaming, removing, changing columns, tables) that also need to be reflected in PowerBuilder applications.

- **Conversion Features**

  - Supports major databases - Oracle, SQL Server, DB2, Sybase, Informix, MySQL, PostgreSQL etc.
  - Offers assessment tools
  - Processes PowerBuilder Source Files (extracted from .PBL libraries), finds and converts SQL statements and database access code.
  - Converts SQL SELECT and PowerBuilder PBSELECT statements in DataWindow objects (.SRD files) and Query (.SRQ files)
  - Converts SQL SELECT, INSERT, UPDATE and DELETE statements in Window objects (.SRW files), User objects (.SRU files) and Function Objects (.SRF files)
  - Supports Procedure calls
  - Converts SQL built-in functions
  - Supports Result Sets processing
  - Converts DECLARE PROCEDURE FOR and ALIAS FOR statements
  - Supports Dynamic SQL conversion
  - Table and column mapping

Oracle PL/SQL to Java Migration

SQLWays is capable of converting Oracle PL/SQL to Java classes.

- **Conversion Features**

  - Convert PL/SQL packages to Java class with saving 99% of business logic
  - Convert cursor to Java ResultSet
  - Converts Out Parameters to Java inner class
  - Converts DML into Java Embedded SQL.

User's Guide

This title implies how to use SQLWays. To learn about the concepts behind the migration process, read the title Database Migration Concepts.

You can use SQLWays either with the help of Wizard or command line tool. Wizard provides you with a user-friendly and intuitively clear interface that allows you to easily perform migration represented in the step-by-step form. The command line version of SQLWays can be used for automating migration process.

- SQLWays Wizard
- SQLWays Command Line
- Setting Up ODBC Data Sources

SQLWays Wizard

This chapter contains information on how the SQLWays wizard guides you through the migration process. In an easy step-by-step manner, SQLWays Wizard interface enables you to enter settings or configure migration just in several dialog boxes.

Scroll through the pages representing separate steps of the migration process via these links:

- "Welcome Page"
  This page contains an introduction into SQLWays Wizard and enables you to order SQLWays online or enter the acquired license key.

- Step 1 - "Choose a Source Database"
  Choosing a source database and specifying your user name and password for connecting to the source database.

- Step 2 - "Choose a Target Database"
  Choosing a target database and specifying your user name and password for connecting to the target database.

- Step 3 - "Specify Database Objects or Query"
  Selecting database objects (tables, views etc.) due to be migrated from the source database to the target database.

- Step 4 - "Set DDL and Data Options"
  Specifying options for SQL scripts and data files generated at the export stage.

- Step 5 - "Specify Export File Options"
  Specifying location and other options for export files.

- Step 6 - "Specify Import Options"
  Specifying options for the import stage.

Having successfully performed the previous 6 steps, you can review the selections and start the migration process.

Welcome Page

The welcome page contains an introduction into SQLWays Wizard and enables you to order SQLWays online or enter a license key.

- Until you register you can try the product **for evaluation**. In this case there are some restrictions on using SQLWays. For more information, see SQLWays License Agreement.
- In order to lift these restrictions, register SQLWays.

  •Apply for a license by clicking **Order Online**, or directly via http://www.ispirer.com/order/.

  •Click **Enter License** and enter in the **License Information** dialog box your **Registration Name** and the acquired **Registration Number**.

  After registration you can select the **Do not show this Welcome page again** check box at the bottom of this page to skip the welcome page after restarting SQLWays Wizard.

Step 1- Choose a Source Database

On this page you choose a source database and specify your user name and password for connecting to the source database.

- If **User/System DSN** is clicked, select the name of the source database in the available ODBC data sources box. To create a new ODBC data source, click **New** and select its name, type and ODBC driver for which you want to set up the data source.
- If **File DSN** is clicked, enter the path of the file which contains information on ODBC connection for the source database. You can click (...) to browse for the *.dsn file.
- In the **User name** and **Password** boxes, enter login information for the source database. You can optionally select the **Save password** check box in order not to enter the password in future.

- Setting Up ODBC Data Sources

Step 2- Choose a Target Database

On this page you choose a target database and specify your user name and password for connecting to the target database.

- In the **Target** box, select the target database from the list of supported target databases.
  In the **Version** box, enter its version.

- Depending on the selected target database, enter its connection options and/or click **Advanced** to adjust additional settings.

TABLE 101. Target Database Options

Target Database	Connection Options	Advanced Options
IBM DB2	• database	IBM DB2 Advanced Options
Oracle	• service	Oracle Advanced Options
Microsoft SQL Server	• server • database	Microsoft SQL Server Advanced Options
Sybase	• server • database	Sybase Adaptive Server Enterprise Advanced Options
MySQL	• host • database	MySQL Advanced Options
Pervasive.SQL	• ODBC alias	Pervasive.SQL Advanced Options

- In the **User name** and **Password** boxes, enter login information for the target database. You can optionally select the **Save password** check box in order not to enter the password in future.

  If the target database is Microsoft SQL Server you can select the **Use Windows authentication** check box in order to use your operating system login information for connecting to Microsoft SQL Server. In this case you do not need to enter login information in the **User name** and **Password** boxes.

- If the **Generate import scripts** check box is selected, SQLWays creates scripts for importing data and database structure to the target database.

IBM DB2 Advanced Options

This dialog box specifies advanced options for the IBM DB2 (for Linux, Unix and Windows; for z/OS and OS/390; for iSeries and AS/400) target database. The following options are available:

- In the **BIN directory** box, enter the path to the folder where the IBM DB2 utilities like CLP, IMPORT and LOAD are located. You can click (...) to browse for this folder.
- By default **Use the IMPORT command** is clicked in the **Utilities options** box, so the data is imported to a IBM DB2 database.

  •In the **Mode** box, select the mode in which the IMPORT utility is executed. Possible modes are *INSERT* (Adds the imported data to the table without changing the existing table data), *INSERT_UPDATE* (Adds rows of imported data to the target table, or updates existing rows of the target table with matching primary keys) and *REPLACE* (Deletes all existing data from the table by truncating the data object, and inserts the imported data. The table definition and the index definitions are not changed). The default mode is *INSERT*.

- If **Use the LOAD command** is clicked in the **Utilities options** box, the data is loaded to a IBM DB2 database.

  •In the **Mode** box, select the mode in which the LOAD utility is executed. Possible modes are *INSERT* (Adds the loaded data to the table without changing the existing table data) and *REPLACE* (Deletes all existing data from the table, and inserts the loaded data. The table definition and index definitions are not changed). The default mode is *INSERT*.

- If the **Load from a remote client** check box is selected, the data resides on a remote client. By default the **Load from a remote client** check box is clear, so the data resides on the IBM DB2 server. The CLIENT option in the LOAD command is available in IBM DB2 7.1 or later.
- If the **Nonrecoverable load** check box is selected, the load transaction is marked as non-recoverable, and it is not possible to recover it by a subsequent roll forward action. With this option, when the database is configured to support online backups, table spaces are not put in backup pending state following the load operation. By default the **Nonrecoverable load** check box is clear.
- In the **Modified by** box, enter additional options that are added to the modified by option of the IBM DB2 IMPORT and LOAD utilities. Some options like character, column delimiters and others are added to the IBM DB2 IMPORT/LOAD scripts by SQLWays. You can use this option if you need other options to be added to the modified by option. For example, set *modifiedby_options=usedefaults delprioritychar* and these options will be added to the modified by option in each script for the IBM DB2 IMPORT/LOAD utilities.
- In the Tablespaces box:

  •In the **Tables** box, enter the table space in which the tables will be created. The table space must exist, and be a REGULAR table space. If no other table space is specified, all table parts will be stored in this table space. The default table space is *USERSPACE1*.

  •In the **Indexes** box, enter the table space in which any indexes on the tables will be created. The specified table space must exist, be a REGULAR DMS table space.

  •In the Large objects box, enter the table space in which the values of any long columns (LONG VARCHAR, LONG VARGRAPHIC, LOB data types) will be stored. The table space must exist, be a LONG DMS table space.

- In the **Timestamp format** box, select the format of the timestamp columns in the text file when the target database is IBM DB2. Possible formats are *IBM DB2*, *ISO*. The default format is IBM DB2 that means using the IBM DB2 native format of timestamps. The IBM DB2 native format is YYYY-MM-DD-HH.MI.SS.FFFFFF. The IBM DB2 IMPORT command requires this format to import text files containing timestamps not enclosed by double quotes (TAB delimited output format e.g.). When ISO is specified, the ISO format of timestamps will be used. The ISO format is YYYY-MM-DD HH:MI:SS.FFFFFF. The IBM DB2 IMPORT command can import timestamp values in the ISO format, when they are enclosed by double quotes (CSV output format e.g.). **Note**. This option is ignored when any value is specified in the option DATETIME_FORMAT (see earlier in the [DATA] subsection). For example, when DATETIME_DATATYPE=DATE is specified, the datetime datatypes will be converted to DATE in IBM DB2.

To specify IBM DB2 options in the command line

- SQLWays command line - [IBM DB2] subsection

Oracle Advanced Options

This dialog box specifies advanced options for the Oracle target database. The following options are available:

- In the **BIN directory** box, enter the path to the folder where the Oracle utilities like SQL Plus and SQL Loader are located. You can click (...) to indicate this folder.

- In the SQL Loader box:

  •In the **Executable** box, enter the name of the executable module of the Oracle SQL Loader. The default name is *sqlldr*.exe.

  •In the **Data option** box, select the loading option in the generated SQL Loader control file. Possible options are *Insert*, *Append*, *Replace*, Truncate. The default option is *Insert*.

  •If **Use direct path load** check box is selected, it specifies a direct path load. By default the **Use direct path load** check box is clear, which specifies a conventional path load.

  •If **Use DECODE for empty CHARs** check box is selected, SQLWays generates DECODE function in the SQL Loader control file for CHAR NOT NULL columns for fixed-length text file to convert NULLs to blanks. If you what to load fields that contain only blanks into CHAR NOT NULL columns and the text file has the fixed length format, you have to use DECODE function, because SQL Loader treats blanks as NULLs.

- In the SQL Plus box:

  •In the **Executable box**, enter the name of the executable module of the Oracle SQL Plus. The default name is *sqlplus*.exe.

- The **SQL Loader column formats** box specifies the data type that represents the datetime datatypes in Oracle. Datetime data consist of valid date and time combinations. Depending on the database this is the DATE data type in Oracle, TIMESTAMP data type in IBM DB2 and DATETIME data type in Microsoft SQL Server and Sybase. SQLWays will convert the datetime datatypes to the specified data type. By default the datetime data type are converted to DATE. For example, when TIMESTAMP(6) is specified, the datetime datatypes will be converted to TIMESTAMP(6) in Oracle. The TIMESTAMP data type is available in Oracle 9i Release 1 (9.0.1) or later.

  •In the **DATE** box, enter the format string in SQL Loader control files for loading dates (year, month, day) into DATE columns from the text files. The default format is "YYYY-MM-DD". If any value is specified in the DATE_FORMAT option in the [Data] subsection, it overrides the DATE option.

  •In the **TIME** box, enter the format string in SQL Loader control files for loading times (hours, minutes, seconds) into DATE columns from the text files. The default format is "HH24:MI:SS". If any value is specified in the TIME_FORMAT option in the [Data] subsection, it overrides the TIME option.

  •In the **TIMESTAMP** box, enter the format string in SQL Loader control files for loading timestamps (year, month, day, hours, minutes, seconds) into DATE columns from the text files. The default format is "YYYY-MM-DD HH24:MI:SS". If any value is specified in the DATETIME_FORMAT option in the [Data] subsection, it overrides the TIMESTAMP option.

  •In the **TIMESTAMP9i** box, enter the format string in SQL Loader control files for loading Oracle 9i TIMESTAMP data type (year, month, day, hours, minutes, seconds, fractional seconds precision) from the text files. The default format is "YYYY-MM-DD HH24:MI:SS.FF".

- By default the **Generate the EXIT command** check box is selected, so the SQL Plus EXIT command is generated at the end of DDL scripts. This allows executing DDL scripts from multiple files in batch mode.

- By default the **Use the TO_DATE function** check box is selected, so the TO_DATE function with the corresponding date format is generated for DATE columns in SQL INSERT statements for SQL Plus. This allows inserting rows that contain DATE columns when the date format differs from the default date format for Oracle database. If the **Use the TO_DATE function** check box is clear, the TO_DATE function is not generated in SQL INSERT statements.

To specify Oracle options in the command line

- SQLWays command line - [Oracle] subsection

Microsoft SQL Server Advanced Options

This dialog box specifies advanced options for the Microsoft SQL Server target database. The following options are available:

- In the **BIN directory** box, enter the path to the folder where the Microsoft SQL Server utilities like BCP and ISQL are located. You can click (...) to indicate this folder .
- In the **Max errors** box, enter the maximum number of errors that can occur before the BCP utility is canceled. Each row that cannot be copied by BCP is ignored and counted as one error (Parameter -m of BCP). Default number is 10.
- In the **Codepage** box, select the code page of the data in the data file for the BCP utility (Parameter -C of BCP). The following code pages are:

  •ACP - ANSI/Microsoft Windows (ISO 1252).
  •OEM - Default code page used by the client. This is the default code page used by bcp if -C is not specified.
  •RAW - No conversion from one code page to another is taking place.
  •<value> - Specific code page number, for example, 437.

To specify Microsoft SQL Server options in the command line

- SQLWays command line - [Microsoft SQL Server] subsection

Sybase Adaptive Server Enterprise Advanced Options

This dialog box specifies advanced options for the Sybase Adaptive Server Enterprise target database. The following options are available:

- In the **BIN directory** box, enter the path to the folder the Sybase utilities like BCP and ISQL are located. You can click (...) to indicate this folder.

To specify Sybase Adaptive Server Enterprise options in the command line

- SQLWays command line - [Sybase] subsection

MySQL Advanced Options

This dialog box specifies advanced options for the MySQL target database. The following options are available:

- In the **BIN directory** box, enter the path to the folder where the MySQL utilities are located. You can click (...) to indicate this folder.
- In the **Table type** box, select a table type that is used when creating tables in the MySQL database. MySQL supports two different kinds of tables: transaction-safe tables (InnoDB and BDB) and not transaction-safe tables (HEAP, ISAM, MERGE, and MyISAM). For example, when InnoDB is selected, SQLWays generates the TYPE=InnoDB clause in the CREATE TABLE statements for MySQL.
- In the **Data load option** box, select an option to be added to the LOAD command. When *Replace* is selected existing rows are updated in the existing table basing on primary or unique key information or added to the table if primary key is not matched. When *Ignore* is selected the existing rows in the table are not updated, when primary or unique key values are equivalent to the existing in the table. The rows from the text file which do not match primary or unique key in the table are inserted.
- By default the **Import from client** check box is selected, so the LOCAL keyword is generated for the MySQL LOAD DATA INFILE command that is used for importing data to MySQL. When LOCAL is specified, data files can be located on the client host. If the **Import from client** check box is clear, the LOCAL keyword is not generated and data files must be located on the server before importing.

To specify MySQL options in the command line

- SQLWays command line - [MySQL] subsection

Pervasive.SQL Advanced Options

This dialog box specifies advanced options for the Pervasive.SQL target database. The following options are available:

- In the **BIN directory** box, enter the path to the folder the Pervasive.SQL utilities like BUTIL are located. You can click (...) to indicate this folder.
- In the **Data directory** box, enter the path to the folder the database data files (.mkd files) are located. You can click (...) to indicate this folder. This option is used for the BUTIL utility.

To specify Pervasive.SQL options in the command line

- SQLWays command line - [Pervasive] subsection

Step 3- Specify Database Objects or Query

On this page you select database objects (tables, views etc.) due to be migrated from the source database to the target database.

- If **Specify Objects** is clicked in the drop-down list, database objects are determined particularly.

  •Move the necessary objects from the left list to the right list.
  Hint: To add an object, you can either drag-and-drop it to the right list, or click the right arrow button. Reverse shift of moved objects to the starting position is available via clicking the left arrow button.

- If **Use Query** is clicked in the drop-down list:

  •If the **Specify a SQL query statement** is clicked, enter SQL query statement manually.
  •If the **Specify a file containing a SQL query statement** is clicked, enter the path or click (...) to open the file with this statement.

Step 4 - Set DDL and Data Options

On this page you specify options for SQL scripts and data files generated at the export stage.

Those source database objects (tables, views, stored procedures, functions, triggers and packages) which have been selected for migration to the target database at the previous step, appear on the left part of this page. Right part of the page is full of various settings for each of the selected object.

SQLWays creates SQL scripts when it exports the definitions of the source database objects. These scripts contain DDL statements that are used to create database objects in the target database.

- DDL Options
- Data Options

DDL Options

The following options are the following:

- **Tables** settings:

  •If the Change column names case to check box is selected, it is possible to select the case of column names in the SQL statements: *Upper*, *Lower*. If the Change column names case to check box is clear, the case of the column names is not changed and column names are used as they are provided by source database.

  •If the Use constraint names of the source database in DDL statement check box is selected, constraint names of the source database will be used in generated DDL scripts. Otherwise constraint names will be skipped. By default the Use constraint names of the source database in DDL statement check box is clear.

  •If the **Convert identity columns** check box is selected, SQLWays converts the identity properties of columns. If the **Convert identity columns** check box is clear, SQLWays does not extract the identity properties. Identity columns are commonly used in conjunction with primary key constraints to serve as the unique row identifier for the table.

  •By default the **Force NOT NULL constraints for columns that make up a primary key** check box is selected, so all primary key columns are created with NOT NULL constraints. Some databases (Oracle, Sybase, Access e.g.) allow not specifying NOT NULL constraints for primary key columns explicitly when creating the table, and they change columns to NOT NULL when adding a primary key. Other databases (IBM DB2, Microsoft SQL Server, MySQL e.g.) require primary key columns to be created with NOT NULL constraints before adding a primary key.

  •By default the **Remove NOT NULL constraints from all columns except primary key columns** check box is clear, so the NOT NULL constraints for all columns except primary key columns in the target database are not removed to avoid their conversion between databases by SQLWays.

- **SQL scripts** settings are:

  •Omit schema names in SQL scripts: This option specifies that schema (owner) names are omitted in SQL scripts for DDL statements. This allows you to use the default schema (owner) name for the user.

  •Change schema names to: This option allows you to change the schema (owner) name of the exported database objects. If a value is specified, SQLWays changes the source schema (owner) name to the specified value. If no value is specified, the schema (owner) name is not changed and the source value is used.

  •Use statement termination string: This option specifies a string that is used to terminate SQL statements in SQL scripts. The default value is a semicolon (;) unless the target database is Sybase or Microsoft SQL Server. If the target database is Sybase or Microsoft SQL Server the default value is GO.

If the **Generate DDL** check box is selected:

•If the **Generate the DROP TABLE statement for each table** check box is selected, the DROP TABLE statement is generated before the CREATE TABLE statement.

- If the **Generate the DROP INDEX statement for each index** check box is selected, the DROP INDEX statement is generated before each CREATE INDEX statement. By default the **Generate the DROP INDEX statement for each index** check box is clear. This option can be helpful when the scripts for indexes are re-executed without recreating the table.
- If the **Cascade referential integrity constarints** check box is selected, the CASCADE CONSTRAINTS option is generated in the DROP TABLE statement. Currently this option is supported by Oracle only. For more information, see Dropping Tables. This option is only available if the target database is Oracle and the **Generate the DROP TABLE statement** check box is selected.

- **Global Data Type Mapping:** These settings show how global data types in the source database are mapped to the target one.

- **Column Name Mapping:** These settings show how global data types in the source database are mapped to the target one.

To specify DDL options in the command line

- SQLWays command line - [DDL] subsection

Data Options

SQLWays exports data to text files.

- General

  •If the Export Data check box is selected:

  •In the Start row box, enter the number of the start row to convert. The default number is 1.

  •In the Number of rows box, enter the number of the exported rows. The default value is all rows matching the query condition.

  •In the Prefetch count box, enter the number of rows that are read before being written to the text file. The default number is 1000 rows.

- **Formats:** On this page you can specify text file formats and various properties.

  •If the **Use datetime format** check box is selected, you can specify a format of the datetime datatypes in the text files. Datetime data consist of valid date and time combinations. Depending on the database this is the DATE data type in Oracle, TIMESTAMP data type in IBM DB2 and DATETIME data type in Microsoft SQL Server and Sybase.

A datetime format can be composed of one or more datetime format elements as listed below.

YYYY - 4-digit year
YY - Last 2 digits of year
MM - 2-digit numeric abbreviation of month (01-12)
MON - 3-symbol abbreviated name of month (JAN, FEB etc)
DD - 2-digit day of month (01-31)
HH - 2-digit hour of day (01-12)
HH12 - 2-digit hour of day (01-12)
HH24 - 2-digit hour of day (00-23)
MI - 2-digit minute (00-59)
SS - 2-digit second (00-59)

F - Fraction of second. The number of F symbols represents the precision. For example, FFF for accuracy to the milliseconds and FFFFFF for accuracy to the microseconds. For example, to get datetime values like 21-DEC-2002 21-21-00 set the datetime format DD-MON-YYYY HH24-MI-SS. If the **Use datetime format** check box is clear, the default datetime format is used. By default, ODBC drivers convert datetime values using the ISO format YYYY-MM-DD HH24:MI:SS.FFF (the part FFF depends on database).

•Datetime data consists of valid date and time combinations. Time can contain a fraction part. For example, IBM DB2 TIMESTAMP data type can contain microseconds, Microsoft SQL Server and Sybase DATETIME can keep milliseconds. In the **Save datetime fraction** box, select how datetime fraction is saved.
If Yes is selected, the fractional seconds are saved in the text files.
If No is selected, the fractional seconds are not saved in the text files.
If Default is selected, the fractional seconds are saved unless the target database is Oracle. If Default is selected, the target database is Oracle and the datetime datatypes are converted to the Oracle DATE data type, the fractional seconds are not saved, since the Oracle DATE data type doesn't support fractional seconds. If Default is selected, the target database is Oracle and the datetime datatypes are converted to the Oracle TIMESTAMP

data type, the fractional seconds are saved, since the Oracle TIMESTAMP data type supports fractional seconds. Note. This option is ignored if any value is specified in the option Datetime format (see earlier).

- If the **Use date format** check box is selected, you can specify a format of the date datatypes in the text files. A date is a three-part value (year, month and day). Not all databases have a data type that allows you to keep date values without a time part. For example, such data type exists in IBM DB2 and MySQL (the DATE data type). A date format can be composed of one or more date format elements as listed below.

YYYY - 4-digit year
YY - Last 2 digits of year
MM - 2-digit numeric abbreviation of month (01-12)
MON - 3-symbol abbreviated name of month (JAN, FEB etc)
DD - 2-digit day of month (01-31)

For example, to get date values like 21-DEC-2002, set the datetime format DD-MON-YYYY. If the **Use date format** check box is clear, the default date format is used. By default, ODBC drivers convert date values using the ISO format YYYY-MM-DD.

- If the **Use time format** check box is selected, you can specify a format of the time datatypes in the text files. A time is a three-part value (hour, minute and second). Not all databases have a data type that allows you to keep time values without a date part. For example, such data type exists in IBM DB2 and MySQL (the TIME data type). A time format can be composed of one or more time format elements as listed below.

HH - 2-digit hour of day (01-12)
HH12 - 2-digit hour of day (01-12)
HH24 - 2-digit hour of day (00-23)
MI - 2-digit minute (00-59)
SS - 2-digit second (00-59)

For example, to get time values like 21-21-00 set the time format HH24-MI-SS. If the **Use time format** check box is clear, the default time format is used. By default, ODBC drivers convert time values using the ISO format HH24:MI:SS.

- By default the Use ODBC float conversion check box is selected, so numeric data (REAL, FLOAT, DOUBLE datatypes) are converted to the text representation by ODBC driver. Otherwise, SQLWays performs the conversion.
- **Decimal point:** This option specifies a single character, which is used instead of a decimal point character for numeric values. The default value depends on the Regional Settings.

- Files

- If the Default for the target check box is selected, it is possible to specify the output format for the text file. Available values are:
Column-delimited (CSV) output format
Fixed length output format
TAB-delimited output format
INSERT statements
XML output format
Btrieve output format

- In the **Column delimeter** box, enter a column delimiter used in the text files. This option can be used for the CSV format only. The default value is a comma. It is possible to specify one or multiple characters as a column delimiter.
- In the **Blanks count** box, enter number of blanks that are placed between columns for FIX (fixed length) output format. The default number is 1.
- In the **Line delimeter** box, enter a line delimiter used in the text files. The default value is carriage return and new line characters (0x0D0A or \r\n) unless import system is Unix. If import system is Unix, the default. value is a new line character (0x0A or \n).
- In the **Replace new line** box, enter a character string that replaces newline characters in the data. Newline characters are 0x0D0A or \r\n for Windows and 0x0A or \n for Unix.
- If the **Remove new line** check box is selected, newline characters are removed from the data. Newline characters are 0x0D0A or \r\n for Windows and 0x0A or \n for Unix.

To specify Data Export options in the command prompt

- SQLWays command prompt - [Data] subsection

Step 5 - Specify Export File Options

On this page you specify location and other options for export files such as:

- In the **Export directory** box, enter the path to the folder where data files, SQL, import scripts and log files should be exported. You can click (...) to indicate this folder. The current directory is a default folder. If the directory does not exist, SQLWays creates *\Ispirer\SQLWays 3.7\Project* directory.

- In the **Report file name** box, enter the name of the HTML report file for viewing the conversion results stored in the Export directory. Click the enabled after complete export **View Report...** button at the last step of the migration to view the report.

- Log file stored in the Export directory allows to understand every aspect of the migration process. In the **Log file name** box, enter its name and define whether to continue processing when an error occurs. By default the **Continue processing when an error occurs** check box is selected, so SQLWays does not stop processing and tries to perform the next task. If the **Run in trace mode** check box is selected, detailed information is written to this log file. It is typically requested by the technical support staff and can help to solve database export and conversion problems.

- By default the **Divide data file into parts of** check box is clear, so the text file with exported table is not divided into parts. In this case the log record contains (sqlways.log):

Output data file: C:\Program Files\Ispirer\SQLWays 3.7\Project\gsd_locker_assignment.txt

If the **Divide data file into parts of** check box is selected it is possible to define the largest possible file size for text files in the edit box (2G is specified by default). G, M, K postfix can be specified to define gigabytes, megabytes and kilobytes. When this limit is exceeded SQLWays divides the text file into parts. Each file contains a whole number of rows. SQLWays creates the files: table_name.txt, table_name2.txt, … table_nameN.txt.
Note: The minimal part size must exceed the maximum row size (without LOB data) multiplied by the prefetch row count.

If splitting is specified and file is split, log record contains (sqlways.log):

Data file was divided into 2 files of 1G:
Output data file: C:\Program Files\Ispirer\SQLWays 3.7\Project\gsd_locker_assignment.txt
Output data file: C:\Program Files\Ispirer\SQLWays 3.7\Project\gsd_locker_assignment2.txt

- In the **LOB objects options** box:

  •In the **LOB directory** box, enter the path or click (...) to indicate in which folder the data of LOB objects (large objects such as CLOB and BLOB data types in Oracle, IBM DB2; TEXT and IMAGE data types in Microsoft SQL Server, Sybase; MEMO data types in Access and FoxPro) are exported. It is used only for migration to IBM DB2 or Oracle. For each LOB value, SQLWays creates one file in the LOB directory, specifying the file name in the ASCII file containing the table rows. The default value is the subdirectory of the export directory.

  •Select in the **Write the LOB data inside the text file** is the data of LOB columns (CLOB, BLOB, TEXT, IMAGE and MEMO) be written inside the text file. If target database is Microsoft SQL Server, Sybase, MySQL or Pervasive.SQL, the LOB data are written inside the text file even if this option is not specified. This allows the LOB data to be transferred into Microsoft SQL Server and Sybase using the BCP utility, into MySQL using LOAD DATA INFILE command and into Pervasive.SQL using the BUTIL utility. By default, if target database is Oracle, IBM DB2 or not specified, each LOB value is written into a separate

file. This allows the LOB data to be transferred into Oracle using SQL Loader and IBM DB2 IMPORT/LOAD utilities.

•By default the **Create a LOB subdirectory for each table** check box is selected, so SQLWays creates a subdirectory in the LOB directory for each table in order to organize LOB files in it and improve performance. The name of the subdirectory is a table name. The name of files is the table are the names with extension equals a number of LOB files which were created for this table before.

Step 6 - Specify Import Options

On this page you specify options for the import stage.

SQLWays allows you to perform export and import on different systems and platforms.

- If the **Import is executed on the local system (remote target database)** is clicked, select the **Start import automatically** check box to start import automatically**.**
- If the **Import is executed on a remote system** (using this option SQLWays can properly generate import scripts that contain file references) is clicked, you need to specify the operation system (Microsoft Windows or Unix system) where import to the target database will be executed. This option also defines rules for the path and new line delimiters that are used in the scripts and files generated by SQLWays.

  •If **Windows** is selected in the **System** box, SQLWays generates command files for importing using the Windows command interpreter. Windows uses a backslash symbol as the directory delimiter (d:\temp\dir\subdir1). In Windows a new line is defined by carriage return and new line characters (0x0D0A or \r\n).The default system is Windows.

  •If **Unix** is selected in the **System** box, SQLWays generates Bourne shell scripts. You can also use these scripts with the Korn shell, because it is upward compatible with the Bourne shell. Unix uses a slash symbol (tmp/dir1/subdir1). In Unix a new line is represented by new line character only (0x0A or \n).

  •By default, SQLWays specifies that **shell** scripts are executed by /bin/sh. Using this option you can specify another shell location, /usr/bin/sh e.g. This option is disabled if import is not executed on a Unix system.

  •In the **Directory** box, enter the path to the folder from which import is executed or click (...) to gain access to **Browse for folder** dialog box to indicate this folder. If you execute import on a Unix system, you can specify Unix paths in this option, like */usr/reports/data/*.

  •In the **LOB directory** box, enter the path to the folder where the data of **LOB** objects (large objects) are located at the import stage or click (...) to indicate this folder. It is used only for migration to IBM DB2 or Oracle and only in cases when you transfer export files to another location before import and LOB files are not located in the subdirectory LOBS of /IMPDIR. Using this option SQLWays can properly generate import scripts that contain file references. The default value is the subdirectory of the import directory.

- In the **Migration sequence for tables**, select the order of table constraints, importing data and creating indexes creating. Available values are *Clean* (Create constraints before importing data); *Fast* (Import data before creating constraints and indexes); *Ready* (Create constraints and indexes before importing data). Depending on this option SQLWays creates different command files to import tables. Default value is *Clean*.

SQLWays Command Line

SQLWays allows you to perform all migration tasks in an operating system command line or in a shell script.

You can specify options in the command line or in an initialization file (sqlways.ini). The command line is used for specifying only general options while an initialization file is used for specifying both general and specific migration project options.

Command line options have priority over initialization file options. If a specific option is specified both in the command line and the initialization files, it is the value of the command line option that is used.

Command Line Options

Before using SQLWays in the command line, open a command line window (cmd.exe on Windows NT/2000/XP or command.exe on Windows 9.x). Then set options and run SQLWays.exe.

The following section describes the available command line options.

SQLWays.exe /D = DataSourceName [/U = UserName] [/P = Password] [/T = TableName] [/V = ViewName] [/SP = StoredProcName] [/FN = FunctionName] [/TG = TriggerName] [/PKG = PackageName] [/S = SelectStatement] [/SF = Filename] [/TF = Filename] [/EXC = Column1] [/SROW = StartRow] [/CNROWS = CountOfRows] [/TARGET = TargetDBMS] [/TPROD = TargetDBMSProduct] [/TVER = TargetDatabaseVersion] [/TD = TargetDatabaseName] [/TU = TargetUserName] [/TP = TargetUserPassword] [/MIGS = MigrationSequence] [/IMPS = ImportSystem] [/DIR = ExportDir] [/LOBDIR = LOBFilesDir] [/IMPDIR = ImportDir] [/IMPLOBDIR = ImportLOBDir] [/OSN = OutSchemaName] [/OTN = OutTableName] [/OFN = OutFilesName] [/OTF = OutTextFileName] [/EMPS] [/DDL] [/OF = OutFormat] [/CDEL = ColumnDelimiter] [/LDEL = LineDelimiter] [/DECPT = DecPoint] [/STDEL = StatementDelimiter] [/LOBIN] [/TABLST = ListFileName] [/R = RowsPrefetch] [/INI = InitializationFile] [/NSTOP] [/GCMD = GenCmdFile] [/NODDL] [/NOCMD] [/RPT = ReportFileName] [/LOG = LogFileName] [/TRACE] [/REG = RegInfo] [/UNREG]

The command line options are not case sensitive and can be specified in any order.

- /D - Data source name (ODBC alias)
- /U - User name
- /P - Password
- /T - Table name, list or template (like *.*)
- /V - View name, list or template (like *.*)
- /SP - Stored procedure name, list or template (like *.*)
- /FN - Function name, list or template (like *.*)
- /TG - Trigger name, list or template (like *.*)
- /PKG - Package name, list or template (like *.*)
- /F - Script files path, name, list or template
- /FF - Text file with the list of script files path, name, list or template
- /S - SQL SELECT statement
- /SF - File containing SQL SELECT statement
- /EXC - List of columns to be excluded from converting (default none)
- /SROW - Start row (default 1)
- /CNROWS - Number of rows to be exported (default all rows)
- /TARGET - Type of the target database (default none)
- /TPROD - Target database product

- /TVER - Target database version
- /TD - Target database name
- /TU - User name for the target database
- /TP - User password for the target database
- /MIGS - Migration sequence (default Clean)
- /IMPS - Import system (default Windows)
- /DIR - Directory for export files (default current)
- /LOBDIR - Directory for LOB files (default subdirectory LOBS of /DIR)
- /IMPDIR - Directory where import will be executed (default /DIR)
- /IMPLOB - Import directory for LOB files (default subdirectory LOBS of /IMPDIR)
- /OSN - Output schema (owner) name (default the source schema (owner) name)
- /OTN - Output table name (default name of the source table)
- /OFN - Output files name (default name of the output table)
- /OTF - Output text file name (possibly with an extension)
- /EMPS - Omit the schema (owner) name in the DDL statements
- /DDL - Generate DDL statements only
- /OF - Output format (default depends on /TARGET)
- /CDEL - Column delimiter (default comma)
- /LDEL - Line delimiter (default carriage return and new line symbols)
- /DECPT - Decimal point character (default period)
- /STDEL - Statement termination string (default semicolon)
- /LOBIN - Write the LOB data inside the text file
- /TABLST - Generate list of tables available by using template /T
- /R - Prefetch count (default 1000)
- /INI - Initialization file (default sqlways.ini)
- /NSTOP - Continue when an error occurs
- /GCMD - General command file name (default depends on /IMPS)
- /NODDL - Not generate DDL scripts
- /NOCMD - Not generate OS command files
- /RPT - Report file name (default sqlways_report.html)

- /LOG - Log file name (default sqlways.log)
- /TRACE - Run in trace mode
- /REG - Run for registration
- /UNREG - Run for unregistration

/D - Data source name (ODBC alias)

Syntax: /D=DataSourceName

This option specifies the name of the source database. This is an ODBC alias for the database.

The option allows you to specify any data source type supported by ODBC: System, User or File DSN. For example, to specify a System or User DSN set */D=DsnName*, to specify a File data source set */D=C:\PathToFileDSN\FileDsn.dsn*.

The option is mandatory. You can specify this option in the initialization file. See Initialization File Options for details.

/U - User name

Syntax: **/U=UserName**

Specifies the user name (login name) used for connecting to the source database. The user must have the rights for accessing the exported database objects and their properties.

You can specify this option in the initialization file. See Initialization File Options for details.

/P - Password

Syntax: **/P=Password**

Specifies the password used for connecting to the source database. SQLWays does not log the password anywhere.

You can specify this option in the initialization file. See Initialization File Options for details.

/T - Table name, list or template

Syntax: /**T=[SchemaName.]TableName [, NextTable]**

This option specifies the exported tables.

If the database supports schemas (table owners) you can either specify a schema (owner) name for the table explicitly or use the default schema (owner) name for the user connected to the database.

You can provide a comma-separated list of tables or table templates. For example, */T=*.** or */T=%.%* will export all tables available in the database, */T=schema1.*, schema2.** or */T=schema1.%*, schema2.% will export all tables of the schemas *schema1* and *schema2*.

Note: Some databases don't support the schemas. For example, to export all tables of an Access database you have to use */T=** or */T=%*.

When this option is specified all the rows and columns of each table are exported. You can define a subset of the table using options /EXC to exclude columns or /SROW and /CNROWS to specify the start row and number of rows. To define more complex subsets of the table or subsets based on queries or multiple tables use options for defining queries /S and /SF.

/V - View name, list or template

Syntax: **/V=[SchemaName.]ViewName [, NextView]**

Using this option, you can specify the views that you want to export and convert.

If the database supports schemas (object owners) you can either specify a schema (owner) name for the view explicitly or use the default schema (owner) name for the user connected to the database.

You can specify a comma-separated list of views or view templates. For example, */V=*.** or */V=%.%* will extract all views accessible in the database, */V=schema1.*, schema2.** or */V=schema1.%, schema2.%* will extract all views of the schemas *schema1* and *schema2*.

/SP - Stored procedure name, list or template

Syntax: **/SP=[SchemaName.]StoredProcName [, NextStoredProc]**

Using this option, you can specify the stored procedures that you want to export and convert.

If the database supports schemas (object owners) you can either specify a schema (owner) name for the stored procedures explicitly or use the default schema (owner) name for the user connected to the database.

You can specify a comma-separated list of stored procedures or templates. For example, */SP=*.* or /SP=%.%* will extract all stored procedures accessible in the database, */SP=schema1.*, schema2.** or */SP=schema1.%*, *schema2.%* will extract all stored procedures of the schemas *schema1* and *schema2*.

/FN - Function name, list or template

Syntax: **/FN=[SchemaName.]FunctionName [, NextFunction]**

Using this option, you can specify the functions that you want to export and convert.

If the database supports schemas (object owners) you can either specify a schema (owner) name for the functions explicitly or use the default schema (owner) name for the user connected to the database.

You can specify a comma-separated list of functions or templates. For example, */FN=*.** or */FN=%.%* will extract all functions accessible in the database, */FN=schema1.*, schema2.** or */FN=schema1.%, schema2.%* will extract all functions of the schemas *schema1* and *schema2*.

/TG - Trigger name, list or template

Syntax: **/TG=[SchemaName.]TriggerName [, NextTrigger]**

Using this option, you can specify the triggers that you want to export and convert.

If the database supports schemas (object owners) you can either specify a schema (owner) name for the triggers explicitly or use the default schema (owner) name for the user connected to the database.

You can specify a comma-separated list of triggers or templates. For example, */TG=*.** or */TG=%.%* will extract all triggers accessible in the database, */TG=schema1.*, schema2.** or */TG=schema1.%, schema2.%* will extract all triggers of the schemas *schema1* and *schema2*.

/PKG - Package name, list or template

Syntax: **/PKG=[SchemaName.]PackageName [, NextPackage]**

Using this option, you can specify the packages (collections of sub-programs) that you want to export and convert. Packages are available in Oracle only.

If the database supports schemas (object owners) you can either specify a schema (owner) name for the packages explicitly or use the default schema (owner) name for the user connected to the database.

You can specify a comma-separated list of packages or templates. For example, */PKG=*.** or */PKG=%.%* will extract all packages accessible in the database, */PKG=schema1.*, schema2.** or */PKG=schema1.%*, *schema2.%* will extract all packages of the schemas *schema1* and *schema2*.

/F - Script files path, name, list or template

Syntax: **/F=[template of filename].[template of extension]**

Using this option, you can specify the list of files that you want to convert.

Example of using this option:
/F=*.sql; /F=?.*; etc.

/FF - Text file with the list of script files path, name, list or template

Syntax: **/FF=[name of the file]**

Using this option, you can specify the file that contains list of scripts that should be converted.
List of scripts should look in next way:
S:\POC\A1.p
S:\POC\A1.p

Example of using this option:
sqlways.exe /D=FIXED /SOURCE=Progress 4GL /TARGET=CS /FF=S:\Sqlways\Test\Auto\Progress 4GL\For Converter\CS\Ispirer\Option_FF_absolute\Objects.txt /INI=sqlways.ini /DIR=rslt

The contents of the Objects.txt can be as follows:
S:\Sqlways\Progress 4GL\CS\Ispirer\Option_FF_absolute\src\bo\AbcCode\AbcCode.p
S:\Sqlways\Progress 4GL\CS\Ispirer\Option_FF_absolute\src\bo\zKey\zKey.p
S:\Sqlways\Progress 4GL\CS\Ispirer\Option_FF_absolute\src\lib\GetLangDesc.i

/S - SELECT statement

Syntax: **/S=SelectStatement**

This option specifies a SQL SELECT statement that defines the data to export.

You can use this option to export data based on a subset of the table, multiple tables or complex queries.

You can optionally enclose the statement in double quotes. If you need to exclude some columns from the table, consider using options /T and /EXC options. If the statement is too large, consider using the /SF option that allows you to specify a file containing the statement.

/SF - File containing SELECT statement

Syntax: **/SF=Filename**

This option specifies the file containing a SQL SELECT statement that defines the data to export. In all other respects, this option is the same as the /S option. You can use new line characters in the SQL SELECT statement.

The file name can include the directory path to the file and have any extension.

For example, */SF=emp.txt* specifies the file from the current directory, */SF=d:\scripts\emp.txt* specifies the file located in the *d:\scripts* directory.

/EXC - List of columns to be excluded from converting

Syntax: **/EXC=Column1 [,ColumnNext]**

This option specifies the list of columns that must be excluded from converting. Columns must be separated by commas.

You can use the /S option to to specify the columns you want to export. However, in some cases especially when the column list is too long, it is easier to specify the columns you want to exclude. In this case it is beter to use the /T and /EXC options instead of /S.

/SROW - Start row

Syntax: **/SROW=StartRow**

This option specifies the number of the start row to convert. The default value is 1.

If you want to use this option to split converting of a large table into parts, you should be careful, as in some cases it is not a safe way. If possible, use the SQL WHERE clause in the /S option to split the table.

/CNROWS - Number of rows to be exported

Syntax: **/CNROWS=CountOfRows**

This option specifies the number of the exported rows. The default value is all rows matching the query condition.

/TARGET - Type of the target database

Syntax: **/TARGET=TargetDBMS**

This option specifies the type of the target database. You can specify the following values:

- **Oracle** - for Oracle database
- **IBM DB2** - for IBM DB2 database
- **MSSQL** - for Microsoft SQL Server database
- **Sybase** - for Sybase database
- **MySQL** - for MySQL database
- **Pervasive** - for Pervasive.SQL database

You can specify this option in the initialization file. See Initialization File Options for details. If this option is not specified, SQLWays does not generate import scripts.

/TPROD - Target database product

Syntax: **/TPROD=TargetDBMSProduct**

Some vendors offer several database products. Vendors try to simplify migration between their own products and, as a result, products have many similarities in SQL in general. However, certain differences can still be observed between the databases of one vendor.

This option is used in conjunction with the /TARGET option and allows you to specify to which particular database product the source database is migrated.

If /TARGET is IBM DB2, the following values can be specified for the /TPROD option:

- **DB2LUW** - IBM DB2 for Linux, Unix and Windows (Default)
- **DB2zOS** - IBM DB2 for z/OS and OS/390
- **DB2iSeries** - IBM DB2 for iSeries and AS/400

If /TARGET is Sybase, the following values can be specified for the /TPROD option:

- **Sybase ASE** - Sybase Adaptive Server Enterprise (Default)
- **Sybase ASA** - Sybase Adaptive Server Anywhere

You can specify this option in the initialization file. See Initialization File Options for details. If this option is not specified, SQLWays uses the default database product for /TARGET.

/TVER - Target database version

Syntax: **/TVER=TargetDatabaseVersion**

This option specifies the version of the target database.

You can specify this option to adjust SQL and import script generations for particular version of the target database

The version option allows to define the features and options supported by the database. The default values are *9.0.1* for Oracle, *7.2* for IBM DB2, *2000* for Microsoft SQL Server, *12.5* for Sybase and *3.23* for MySQL.

You can specify an earlier version of the database, so that the features not supported by this version are not used in the SQL and import scripts.

You can specify this option in the initialization file. See Initialization File Options for details.

/TD - Target database name

Syntax: **/TD=TargetDatabaseName**

This option specifies the name of the target database. For Oracle, this is the service name.

If specified, this value is used in SQL and import scripts generated by SQLWays.

You can specify this option in the initialization file. See Initialization File Options for details.

/TU - User name for the target database

Syntax: **/TU=TargetUserName**

This option specifies the user name (login name) for the target database.

If specified, this value is used in SQL and import scripts generated by SQLWays.

You can specify this option in the initialization file. See Initialization File Options for details.

/TP - User password for the target database

Syntax: **/TP=TargetUserPassword**

This option specifies the password of the user for the target database.

If specified, this value will be used in the generated command scripts for creating DDL statements and importing data.

You can also set the user password for the target database in the corresponding database section of the SQLWays.ini file. See Initialization File Options for details.

/MIGS - Migration sequence

Syntax: **/MIGS=MigrationSequence**

This option specifies the migration sequence for tables. Using this option you can choose the order of creating table constraints, importing data and creating indexes.

Available values are:

- **Clean** - Create constraints before importing data
- **Fast** - Import data before creating constraints and indexes
- **Ready** - Create constraints and indexes before importing data

Depending on this option SQLWays creates different command files to import tables.

You can specify this option in the initialization file. See Initialization File Options for details.

/IMPS - Import system

Syntax: **/IMPS=ImportSystem**

This option specifies the system where import to the target database will be executed. SQLWays allows you to perform import on a Microsoft Windows or Unix system.

Possible values are *Windows* and *Unix*. The default value is *Windows*. When */IMPS=Windows* is specified, SQLWays generates command files for importing using the Windows command interpreter.

When */IMPS=Unix* is specified, SQLWays generates Bourne shell scripts. You can also use these scripts with the Korn shell, because it is upward compatible with the Bourne shell.

This option also defines rules for the path and new line delimiters that are used in the scripts and files generated by SQLWays. Windows uses a backslash symbol as the directory delimiter (d:\temp\dir\subdir1) whereas Unix uses a slash symbol (tmp/dir1/subdir1). In Windows a new line is defined by carriage return and new line characters (*0x0D0A or \r\n*), while in Unix a new line is represented by new line character only (*0x0A or \n*).

You can specify this option in the initialization file. See Initialization File Options for details.

/DIR - Export directory

Syntax: **/DIR=ExportDir**

This option specifies the export directory. SQLWays uses this directory to save export data files, SQL and import scripts and log files. The default value is the current directory. If the directory does not exist, SQLWays creates it.

You can specify this option in the initialization file. See Initialization File Options for details.

/LOBDIR - Directory for LOB files

Syntax: **/LOBDIR=LOBFilesDir**

This option specifies the directory where the data of LOB objects (large objects) are exported. It is used only for migration to IBM DB2 or Oracle.

For each LOB value, SQLWays creates one file in the LOB directory, specifying the file name in the ASCII file containing the table rows.

The default value is the subdirectory \LOBS of the directory specified by the option /DIR.

LOB objects are **CLOB** and **BLOB** data types in **Oracle**, **IBM DB2**; **TEXT** and **IMAGE** data types in **Microsoft SQL Server**, **Sybase**; **MEMO** data types in **Access** and **FoxPro**.

/IMPDIR - Directory where import will be executed

Syntax: **/IMPDIR=ImportDir**

This option specifies the directory, from which import is executed. It should be used only if you transfer export files to another location before import. Using this option SQLWays can properly generate import scripts that contain file references.

If you execute import on a Unix system, you can specify Unix paths in this option, like */usr/reports/data/*. The default value is the export directory specified by the DIR option.

You can specify this option in the initialization file. See Initialization File Options for details.

/IMPLOB - Import directory for LOB files

Syntax: **/IMPLOBDIR=ImportLOBDir**

This option specifies the directory where the data of LOB objects (large objects) are located at the import stage. It is used only for migration to IBM DB2 or Oracle and only in cases when you transfer export files to another location before import and LOB files are not located in the subdirectory LOBS of /IMPDIR.

Using this option SQLWays can properly generate import scripts that contain file references. The default value is the subdirectory \LOBS of the import directory specified by the /IMPDIR option.

You can specify this option in the initialization file. See Initialization File Options for details.

/OSN - Output schema (owner) name

Syntax: **/OSN=OutSchemaName**

This option allows you to change the schema (owner) name of the exported database objects. If a value is specified, SQLWays changes the source schema (owner) name to the specified value. If *no* value is specified, the schema (owner) name is not changed and the source value is used.

If you want to omit the schema (owner) name in SQL scripts, use the /EMPS option. This allows you to use the default schema (owner) name for the user.

If you want to change both schema (owner) and table names, consider using the /OTN option.

/OTN - Output table name

Syntax: **/OTN=[OutSchemaName.]OutTableName**

This option specifies the name of the output table. This table name is used in the SQL and import scripts. The default value is the name of the source table.

If you want to change only the schema (owner) name for the table, consider using the option /OSN.

/OFN - Output files name

Syntax: **/OFN=OutFilesName**

This option specifies the name of the output files of the exported tables. You should use this option only in case of exporting one table. The output files are text, log and scripts files. The default value is the name of the output table.

The file extensions like *.txt, .sql, .cmd* are added to this file name depending on the output file type.

/OTF - Output text file name

Syntax: **/OTF=OutTextFileName**

This option specifies the output file name and possibly a file extension for the text file. You should use this option only in case of exporting one table. By default, the file name is taken from the table name or from the /OFN option and *.txt* file extension is added.

When this option is specified *.txt* file extension is not added and any other file extension can be specified.

/EMPS - Omit the schema (owner) name in DDL statements

Syntax: **/EMPS**

This option specifies that schema (owner) names be omitted in SQL scripts for DDL statements. This allows you to use the default schema (owner) name for the user.

When the target database is MySQL or Pervasive.SQL, schema names are always omitted even if the /EMPS option is not specified since MySQL and Pervasive.SQL do not support schemas (logical grouping of objects in the database).

If you want to change the schema (owner) name for database objects, consider using the /OSN option.

You can specify this option in the initialization file. See Initialization File Options for details.

/DDL - Generate DDL statements only

Syntax: **/DDL**

This option specifies that only SQL scripts for database structure (tables, constraints, views, stored procedures etc.) be generated. When this option is specified, the data are not exported and import scripts are not generated.

/OF - Output format

Syntax: **/OF=OutFormat**

This option specifies the output format for the text file.

Available values are:

- **CSV** - Column-delimited (CSV) output format
- **FIX** - Fixed length output format
- **TAB** - TAB-delimited output format
- **INS** - SQL INSERT statements
- **XML** - XML output format
- **BTR** - Btrieve output format

The default value depends on the /TARGET option. If the /TARGET option is Oracle, IBM DB2 or not specified the default value is the CSV format. If the /TARGET option is Microsoft SQL Server, Sybase or MySQL the default value is the TAB format. This format is more suitable for transferring data into Microsoft SQL Server, Sybase and MySQL databases using the loading tools supplied with these databases.

If the /TARGET option is Pervasive, the default value is BTR. This is the Btrieve ASCII file format for the BUTIL utility that loads data into a Pervasive.SQL database.

/CDEL - Column delimiter

Syntax: **/CDEL=ColumnDelimiter**

This option specifies a column delimiter used in the text files. This option can be used for the CSV format only. The default value is a comma. It is possible to specify one or multiple characters as a column delimiter.

To specify the hexadecimal value of the delimiter, use the following syntax: *0xhh*. For example, */CDEL=0x2c* specifies a comma as the column delimiter. Non-printing characters like tab, carriage return and new line can be represented by \t, \r and *n* accordingly. To specify a backslash character use \\.

You can combine different character representations to specify a column delimiter. For example, */CDEL=|||0x2c0x2c|||*

Note. To use a TAB character as a column delimiter use the TAB output format.

You can specify this option in the initialization file. See Initialization File Options for details.

/LDEL - Line delimiter

Syntax: **/LDEL=LineDelimiter**

This option specifies a line delimiter used in the text files. The default value is carriage return and new line characters (*0x0D0A* or \r\n) unless /IMPS=Unix is specified. If /IMPS=Unix is specified, the default value is a new line character (*0x0A* or \n).

Using this option you can specify any string as a line delimiter. The non-printing characters like tab, carriage return and new line can be represented by \t, \r and \n accordingly.

In most cases, if there are new line characters in the data, you have to change the default value to any other string that does not exist in the data. This will allow you to process rows properly by import/load utilities.

You can specify this option in the initialization file. See Initialization File Options for details.

/DECPT - Decimal point character

Syntax: **/DECPT=DecPoint**

This option specifies a single character, which is used instead of a decimal point character for numeric values. The default value depends on the *Regional Settings*.

To specify the hexadecimal value of a decimal point character use the following syntax: 0xhh. For example, */DECPT=0x2c* specifies a comma as a decimal point character.

You can specify this option in the initialization file. See Initialization File Options for details.

/STDEL - Statement termination string

Syntax: **/STDEL=StatementDelimiter**

This option specifies a string that is used to terminate SQL statements in SQL scripts. The default value is a semicolon (;) unless the target database (/TARGET) is Sybase or Microsoft SQL Server. If the target database is Sybase or Microsoft SQL Server the default value is GO.

To specify the hexadecimal value of a character belonging to the statement delimiter, use the following syntax: *0xhh*. For example, */STDEL=0x40* specifies the AT symbol (@) as the statement termination string.

You can specify this option in the initialization file. See Initialization File Options for details.

/LOBIN - Write the LOB data inside the text file

Syntax: **/LOBIN**

This option specifies that the data of LOB columns (CLOB, BLOB, TEXT, IMAGE and MEMO) be written inside the text file.

By default, if /TARGET is Oracle, IBM DB2 or not specified, each LOB value is written into a separate file. This allows the LOB data to be transferred into Oracle using SQL Loader and IBM DB2 IMPORT/LOAD utilities.

If /TARGET is Microsoft SQL Server, Sybase, MySQL or Pervasive.SQL, the LOB data are written inside the text file even if the /LOBIN option is not specified. This allows the LOB data to be transferred into Microsoft SQL Server and Sybase using the BCP utility, into MySQL using LOAD DATA INFILE command and into Pervasive.SQL using the BUTIL utility.

/TABLST - Generate list of tables available by using template

Syntax: **/TABLST=ListFileName**

When this option is specified, SQLWays only generates the file containing the list of tables matching the template in /T option.

For example, the command *SQLWays /t=dept.* /tablst=dept.lst* generates the list of tables belonging to the DEPT schema into the file dept.lst. Then you can comment tables that you don't want to convert (using a semicolon in the first line position) and run *SQLWays /tf=dept.lst*.

You can also use this option to see which tables are available in the source database.

If the file name is not explicitly specified, the default name is sqlways.lst. SQLWays creates the file in the directory specified by the /DIR option.

/R - Prefetch count

Syntax: **/R=RowsPrefetch**

This option specifies the number of rows that are read before being written to the text file. The default value is 1000 rows.

Also this value is used to specify internal buffers and commit points in scripts generated for import utilities. Specifying this option, you can significantly speed up data exporting and especially data importing.

/INI - Initialization file

Syntax: **/INI=InitializationFile**

This option specifies the path and name of the initialization file that is used by SQLWays. For more information about this file options and structure, see Initialization File Options.

By default, if the /INI option is not specified, SQLWays uses SQLWays.ini file. SQLWays searches for the file in the following sequence:

- The current directory.
- The directory from which the application loaded.

This option can be used to specify different initialization files to be used by SQLWays. This allows you to implement various migration routines that involve different database versions.

For example, if */ini=d:\ini\sqlways.ini* is specified, then SQLWays uses the *sqlways1.ini* file from the directory *d:\ini* as the initialization file.

/NSTOP - Continue when an error occurs

Syntax: **/NSTOP**

This option specifies continuing processing when an error occurs. By default this option is *Yes* and when an error occurs, SQLWays does not stop processing and tries to perform the next task.

You can specify this option in the initialization file. See Initialization File Options for details.

/GCMD - General command file name

Syntax: **/GCMD=GenCmdFile**

This option specifies the general command file that contains import scripts for all the exported database objects (tables, views etc).

SQLWays creates one command file for each exported object. This command file contains DDL statements and import scripts for one object only. This allows you to import each database object separately.

To simplify the import of large number of database objects, you can use the general command file that makes possible importing all objects immediately. The tables in the general command file are ordered by referential constraints, the tables that are referenced by the other tables are the first to be imported.

The default general command file is *sqlways_all.cmd* (Windows batch file) if /IMPS=Windows is specified, and sqlways_all.sh (Bourne shell script) if /IMPS=Unix is specified.

The file is saved in the directory specified in the option /DIR.

SQLWays always creates general command file despite the number of exported objects.

/NODDL - Not generate DDL scripts

Syntax: **/NODDL**

This option specifies that DDL scripts not be created when exporting tables. This option can be used when the data are imported into existing tables.

When this option is not specified and the target database is defined, SQLWays creates DDL scripts that can be used to recreate table definitions in a newly created database.

/NOCMD - Not generate OS command files

Syntax: **/NOCMD**

This option specifies that OS command files (import scripts) not be created. SQLWays creates OS command files to invoke database utilities to execute DDL statements and to import data to the target database.

/RPT - Report file name

Syntax: **/Rpt =ReportFileName**

This option specifies the report file in the HTML format that contains the results of the database conversion.

The default report file is *sqlways_report.html*.

The file is created in the directory specified in the /DIR option.

/LOG - Log file name

Syntax: **/LOG=LogFileName**

This option specifies the log file that will be used to store the logging information about the export process.

The default log file is *sqlways.log*. The file is created in the directory specified in the /DIR option.

/TRACE - Run in trace mode

Syntax: **/TRACE**

When this option is specified, detailed information is written to the log file. This information is typically requested by the technical support staff and can help to solve database export and conversion problems.

/REG - Run for registration

Syntax: **/REG=RegName[;RegNumber]**

Use this option to enter the registration information. You should enter the registration name and number. See **license.html** file supplied with the install package for the license agreement.

If you omit the registration name and number in the command line (e.g. running *SQLWays.exe /REG or SQLWays.exe /REG=CompanyName*), SQLWays asks you to enter the missing information.

/UNREG - Run for unregistration

Syntax: **/UNREG**

You can use this option to unregister SQLWays on the computer. See **license.html** file supplied with the install package for the license agreement.

Initialization File Options

The initialization file is used for specifying particular migration project options and adjusting script generations. By default, this is the SQLWays.ini file, located in the current directory. To specify any other file, use the command line /INI option.

The initialization file options section consists of several subsections:

- [Common]
- [Data]
- [DDL]
- [Windows]
- [Unix]
- [Oracle]
- [IBM DB2]
- [MSSQL]
- [Sybase]
- [MySQL]
- [Pervasive]
- [Formatting]

Each subsection contains options and their values.

Syntax: **Option=Value**

If *no* value is specified for an option, the default is used.

[Common] Subsection

This subsection is used for specifying the general options of the migration project. Options from this subsection can also be specified in the command line. When any of these options is not specified in the command line, its value is taken from the initialization file.

- **DSN** - for the /D option (Data source name)
- **USER** - for the /U option (User name)
- **PWD** - for the /P option (Password)
- **TARGET** - for the /TARGET option (Target database)
- **TARGET_PRODUCT** - for the /TPROD option (Target database product)
- **TARGET_VERSION** - for the /TVER option (Target database version)
- **MIGRATION_SEQUENCE** - for the /MIGS option (Migration sequence). The default value is *Clean*.
- **IMPORT_SYSTEM** - for the /IMPS option (Import system). The default value is *Windows*.
- **DIR** - for the /DIR option (Export directory)
- **LOBDIR** - for the /LOBDIR option (Directory for LOB files)
- **IMPORT_DIR** - for the /IMPDIR option (Import directory)
- **IMPORT_LOBDIR** - for the /IMPLOB option (Import directory for LOB files)
- **OUTFORMAT** - for the /OF option (Output text format)
- **PREFETCHROWS** - for the /R option (Prefetch count). The default value is 1000.
- **NONSTOP** - for the /NSTOP option (Continue when an error occurs). The default value is *Yes*. Possible values - *Yes*, *No*.

[Data] Subsection

This subsection is used for customizing data conversion and specifying data format in export files.

- **COLUMN_DELIMITER** - for the /CDEL option (Column delimiter).
- **LINE_DELIMITER** - for the /LDEL option (Line delimiter).
- **DECIMAL_POINT** - for the /DECPT option (Decimal point character).
- **DATETIME_FORMAT** - This option specifies the format of the datetime datatypes in the text files.

Datetime data consist of valid date and time combinations. Depending on the database, this is the DATE data type in Oracle, TIMESTAMP data type in IBM DB2 and DATETIME data type in Microsoft SQL Server and Sybase.

A datetime format can be composed of one or more datetime format elements as listed below.

- **YYYY** - 4-digit year
- **YY** - Last 2 digits of year
- **MM** - 2-digit numeric abbreviation of month (01-12)
- **MON** - 3-symbol abbreviated name of month (JAN, FEB etc)
- **DD** - 2-digit day of month (01-31)
- **HH** - 2-digit hour of day (01-12)
- **HH12** - 2-digit hour of day (01-12)
- **HH24** - 2-digit hour of day (00-23)
- **MI** - 2-digit minute (00-59)
- **SS** - 2-digit second (00-59)

F - Fraction of second. The number of F symbols represents the precision. For example, FFF for accuracy to the milliseconds and FFFFFF for accuracy to the microseconds.

For example, to get datetime values like 21-DEC-2002 21-21-00 set the datetime format DD-MON-YYYY HH24-MI-SS.

If *no* value is specified in this option, the default datetime format is used. By default, ODBC drivers convert datetime values using the ISO format YYYY-MM-DD HH24:MI:SS.FFF (the part FFF depends on database).

- **DATE_FORMAT** - This option specifies the format of the date datatypes in the text files.

A date is a three-part value (year, month and day). Not all databases have a data type that allows you to keep date values without a time part. For example, such data type exists in IBM DB2 and MySQL (the DATE data type).

A date format can be composed of one or more date format elements as listed below.

- **YYYY** - 4-digit year
- **YY** - Last 2 digits of year
- **MM** - 2-digit numeric abbreviation of month (01-12)
- **MON** - 3-symbol abbreviated name of month (JAN, FEB etc)
- **DD** - 2-digit day of month (01-31)

For example, to get date values like 21-DEC-2002, set the datetime format DD-MON-YYYY.

If *no* value is specified in this option, the default date format is used. By default, ODBC drivers convert date values using the ISO format YYYY-MM-DD.

- **TIME_FORMAT** - This option specifies the format of the time datatypes in the text files.

A time is a three-part value (hour, minute and second). Not all databases have a data type that allows you to keep time values without a date part. For example, such data type exists in IBM DB2 and MySQL (the TIME data type).

A time format can be composed of one or more time format elements as listed below.

- **HH** - 2-digit hour of day (01-12)
- **HH12** - 2-digit hour of day (01-12)
- **HH24** - 2-digit hour of day (00-23)
- **MI** - 2-digit minute (00-59)
- **SS** - 2-digit second (00-59)

For example, to get time values like 21-21-00 set the time format HH24-MI-SS.

If *no* value is specified in this option, the default time format is used. By default, ODBC drivers convert time values using the ISO format HH24:MI:SS.

- **DATETIME_FRACTION** - Specifies that the datetime columns can contain the fraction part in the text file. Datetime data consist of valid date and time combinations. Time can contain a fraction part. For example, IBM DB2 TIMESTAMP data type can contain microseconds, Microsoft SQL Server and Sybase DATETIME can keep milliseconds.

Possible values - *Yes*, *No*. *Yes* means that datetime values can contain the fraction part in the text files, *No* means that if the fraction part exists, it is not saved in the text file.

The default value is *Yes* unless the target database is Oracle.

If the target database is Oracle and the datetime datatypes are converted to the Oracle DATE data type, the default value is *No*. The Oracle DATE data type doesn't support fractional seconds.

If the target database is Oracle and the datetime datatypes are converted to the Oracle TIMESTAMP data type, the default value is *Yes*. The Oracle TIMESTAMP data type supports fractional seconds.

Note. This option is ignored if any value is specified in the option DATETIME_FORMAT (see earlier).

- **REPLACE_NEWLINE** - Specifies a character string that replaces newline characters in the data. Newline characters are 0x0D0A or \r\n for Windows and 0x0A or \n for Unix.

Character columns like CHAR and VARCHAR e.g. can contain newline characters in the data. By default, SQLWays saves them in the export files. Using this option you can specify a string that will replace newline characters.

For example, to replace newlines with *zzz*, set *REPLACE_NEWLINE=zzz*, to replace newlines with a blank, set *REPLACE_NEWLINE=" "*.

To specify the hexadecimal value of a character use the following syntax: *0xhh*. For example, *0x2c* specifies a comma character. Non printing characters like tab, carriage return and new line can be represented by \t, \r and \n accordingly. To specify a backslash character use \\. You can combine different character representations to specify the string. For example, *REPLACE_NEWLINE=|||0x2c0x2c|||*

When you need to remove newlines from the data, use the *REMOVE_NEWLINE* option. The *REPLACE_NEWLINE* option is ignored when the *REMOVE_NEWLINE* option is set to *Yes* (see below).

In most cases, when you need to save newline characters in the data you have to change the line delimiter using the /LDEL option. This makes possible properly processing rows by import/load utilities.

- **REMOVE_NEWLINE** - If *Yes* is specified, newline characters are removed from the data. Newline characters are 0x0D0A or \r\n for Windows and 0x0A or \n for Unix.

Character columns like CHAR and VARCHAR e.g. can contain newline characters in the data. By default, SQLWays saves them in the export files. Using this option, you can specify that newline characters be removed from the data. The default is *No*. Possible values - *Yes*, *No*.

If you need to replace newline characters with another string, use the REPLACE_NEWLINE option (see earlier).

In most cases, when you need to save newline characters in the data, you have to change the line delimiter using the /LDEL option. This makes possible properly processing rows by import/load utilities.

- **ODBC_FLOAT_CONVERSION** - If *Yes* is specified, numeric data (REAL, FLOAT, DOUBLE datatypes) are converted to the text representation by ODBC driver. Otherwise, SQLWays performs the conversion. The default is *Yes*. Possible values - *Yes*, *No*.

- **LOBS_INFILE** - for the /LOBIN option. (Place the LOB data inside the text file). Possible values - *Yes*, *No*.

- **LOBS_DIR_TAB** - If *Yes* is specified, SQLWays creates a subdirectory in the LOB directory for each table and writes LOB data for each table to the LOB subdirectories. The name of the subdirectory is the table name. A file is created for each LOB value, and the file name is the table name with the extension equal to the sequence number of the LOB value.

If *No* is specified, SQLWays does not create LOB subdirectories for each table and writes all LOB files to the LOB directory.

The default is *Yes*. Possible values - *Yes*, *No*.

- **FIXFMT_COLUMN_GAPS** - This option specifies the number of blanks that are placed between columns for FIX (fixed length) output format. The default value is 1.

- **DATAFILE_PART_SIZE** - By default this option specifies that the text file with exported table is not divided into parts and it is not possible to define the largest possible file size for the file in the edit box.

The option can be defined. 2G value (two gigabytes) is specified by default. When this limit is exceeded SQLWays divides the text file into parts. Each file contains a whole number of rows. SQLWays creates the files: table_name.txt, table_name2.txt, ... table_nameN.txt.

Note: The minimal part size must exceed the maximum row size (without LOB data) multiplied by the prefetch row count.

To specify Data Export options using the SQLWays wizard

- SQLWays Wizard - Data Options Page

[DDL] Subsection

This subsection is used to adjust DDL and SQL script generations.

- **GENERATE_DROP_TABLE** - If *Yes* is specified, the DROP TABLE statement is generated before each CREATE TABLE statement. The default value is *No*. Possible values - *Yes*, *No*.

- **DROP_TABLE_CASCADE_CONSTRAINTS** - If *Yes* is specified, the CASCADE CONSTRAINTS option is generated in the DROP TABLE statement. Currently this option is supported by Oracle only. For more information, see Dropping Tables.

This option is only available if the target database is Oracle and the GENERATE_DROP_TABLE is set to *Yes*. The default value is *No*.

- **GENERATE_DROP_INDEX** - If *Yes* is specified, the DROP INDEX statement is generated before each CREATE INDEX statement. The default value is *No*. Possible values - *Yes*, *No*.

This option can be helpful when the scripts for indexes are re-executed without recreating the table.

- **OUTSCHEMA** - for the /OSN option (Output schema name).

- **EMPTY_SCHEMA** - for the /EMPS option. Possible values - *Yes, No.*

- **COLUMN_NAME_CASE** - This option specifies the case of column names in the SQL statements. Possible values - Upper, Lower. If *no* value is specified, the case of the column names is not changed and column names are used as they are provided by source database.

- **USE_CONSTRAINT_NAMES** - If *Yes* is specified, constraint names of the source database will be used in generated DDL scripts. Otherwise constraint names will be skipped. The default value is *No*. Possible values - *Yes, No*.

- **PK_COLS_NOTNULL** - This option specifies to force NOT NULL constraints in the CREATE TABLE statement for columns making up a primary key. The default value is *Yes*. Possible values - *Yes, No.*

Some databases (Oracle, Sybase, Access e.g.) allow not specifying NOT NULL constraints for primary key columns explicitly when creating the table, and they change columns to NOT NULL when adding a primary key. Other databases (IBM DB2, Microsoft SQL Server, MySQL e.g.) require primary key columns to be created with NOT NULL constraints before adding a primary key.

The default value is *Yes* (all primary key columns will be created with NOT NULL constraints). Possible values - *Yes*, *No*.

- **CONVERT_IDENTITY_COLUMNS** - If *Yes* is specified, SQLWays converts the identity properties of columns. Otherwise, SQLWays does not extract the identity properties. The default value is *Yes*. Possible values - *Yes, No*.

- **REMOVE_NOTNULL_EXCEPT_PK** - If *Yes* is specified, the NOT NULL constraint for all columns except primary key columns in the target database are removed to avoid their conversion between databases by SQLWays. The default value is *No*. Possible values - Yes, No.

Identity columns are commonly used in conjunction with primary key constraints to serve as the unique row identifier for the table.

- **INSERT_COMMIT_COUNT** - This option is only valid for INS output format and is used to generate a COMMIT statement after the specified number of INSERT statements are generated. The default value is 0 (*no* commits are generated).

- **STATEMENT_DELIMITER** - for the /STDEL option (Statement termination character).

- **STATISTICS_STATEMENTS** - This option specifies whether or not to generate statements to calculate statistics on tables in the DDL scripts for table indexes.The default value is *Yes*. Possible values - *Yes, No*. Currently SQLWays generates statistics statements for IBM DB2 only.

- **PK_UNIQUE_INDEXES** - This option specifies whether to generate indexes on primary keys and unique constraints. The default value is *No*. This means *no* index scripts are generated for primary keys and unique constraints. In this case the database creates unique indexes for primary keys and unique constraints implicitly.

If the unique indexes with the same definition as required for primary keys and unique constraints were created before creating primary keys and unique constraints, the database uses these indexes for primary keys and unique constraints.

When *PK_UNIQUE_INDEXES=ALWAYS* is specified, the index scripts for primary and unique constraints are always created. This allows creating indexes before creating primary keys and unique constraints.

- **REPLACE_NOT_ALLOWED_CHARS** - This option specifies a character or a string that replaces not allowed characters in identifiers in the target database.

The source database may allow users to use characters in identifiers (table and column names etc.) that are not allowed in the target database.

For example, users can use @ in identifiers in Microsoft SQL Server, but this character is not allowed in identifiers in Oracle.

If no value is specified in this option and "remove_not_allowed_chars=no", then all not allowed characters are left in identifiers for the target database.

- REMOVE_NOT_ALLOWED_CHARS - This option is similar to the *replace_not_allowed_chars* option, except that it allows you to remove not allowed characters in identifiers in the target database.

When Yes is specified, all not allowed characters are removed in identifiers in the target database.

The default value is Yes. Possible values - Yes, No.

- **REPLACE_NOT_ALLOWED_CHARS_FPOS** - This option specifies a character or a string that replaces not allowed characters at the first position of identifiers in the target database.

Characters that are allowed at the first position of identifiers in the source database may not be allowed in the target database.

For example, Microsoft SQL Server identifiers can begin with _ (underscore) while Oracle identifiers cannot.

If no value is specified in this option and "remove_not_allowed_chars_fpos=no", then all not allowed characters are left at the first position of identifiers for the target database.

- **REMOVE_NOT_ALLOWED_CHARS_FPOS** - This option is similar to the *replace_not_allowed_chars_fpos* option, except that it allows users to remove not allowed characters at the first position of identifiers in the target database.

When *Yes* is specified, all not allowed characters are removed at the first position of identifiers in the target database.

The default value is *Yes*. Possible values - *Yes*, *No*.

- _NAMES_EXCEEDING_MAX_LEN - Databases may have different identifier length limits. For example, the maximum identifier length in SQL Server is 128 characters while in Oracle - 30. An error occurs when a table with the name exceeding 30 characters is migrated from SQL Server to Oracle.

If *Yes* is specified, SQLWays trims off identifiers exceeding the maximum length for the target database. If during the trimming SQLWays gets identical identifiers, it replaces the last characters of these identifiers (except for the first identifier) with the numbers in the ascending order beginning with "2".

For example, if after trimming SQLWays gets two identical identifiers like "home_phone_numbers" etc., they are further converted to: "home_phone_numbers" and "home_phone_number2".

If *No* is specified, SQLWays does not trim identifiers exceeding the maximum length for the target database.

The default value is *Yes*. Possible values - *Yes*, *No*.

- **TRIM_RULE_ALPHANUM_ONLY** - This option is used in conjunction with the "trim_names_exceeding_max_len=yes" option.

If *Yes* is specified and an identifier exceeds the maximum length for the target database, SQLWays first of all deletes all non-alphanumeric characters and then, if the identifier's length is still more than the maximum length, it is trimmed. If No is specified, SQLWays immediately trims the identifiers without deleting the non-alphanumeric characters.

For example, if Yes is specified, SQL Server identifier like "regional_customer_account_number#" (33 characters) is converted to "regionalcustomeraccountnumber" (29 characters).

The default value is *Yes*. Possible values - *Yes*, *No*.

- **REPLACE_RESERVED_WORDS** - This option is used to change identifiers of the source database that served as reserved words in the target database. The option specifies a template for reserved words replacement. For example, if *%RWORD%_* is specified, underscore character is added to the right of all reserved words.

The default value of the template is *%RWORD%* which means that the reserved words are not changed and are delimited in SQL statements for the target database. The delimeter depends on the database, see Delimeted Identifies for more information.

- **ANSI_QUOTED_IDENTIFIERS** - When ANSI_QUOTED_IDENTIFIER is *ON*, identifiers can be delimited by double quotation marks, and literals must be delimited by single quotation marks.

When ANSI_QUOTED_IDENTIFIER is *OFF*, identifiers cannot be quoted and must follow all Transact-SQL rules for identifiers. Literals can be delimited by either single or double quotation marks.

The default value for ANSI_QUOTED_IDENTIFIER is *ON*. This option is applicable for MSSQL and Sybase ASE.

- **CONVERT_INDEXES** - Specifies whether to convert indexes or not. The default value is *All*, in this case all indexes will be converted. Possible values are *unique* and *none*.

- **VAR_PREFIX** - For variables, when some special characters are used to identify them in the source database, like in SQL Server (@), this option identifies symbols that substitute special symbol. Default value is v_

- **REMOVE_VAR_PREFIX** - Specifies whether to remove or not variable prefix (like '@' in SQL Server and Sybase) during the conversion. Default value is 'no'.

- **TARGET_FILES_GROUPING** - If the conversion of the SCHEMA only is made this option can control how to store the SQL converted, As the default sql for each object is stored in its own file. ALL - means that all the objects will be grouped into one file. OBJECT is the default.

- **REMOVE_SPECIAL_COLUMNS** - This option was created for the direction from MSSQL to Oracle. When this option is set to "yes" (default value is "no") while converting table with TIMESTAMP column, this column is removed from target table (not converted).

- **START_VALUE_EXTRACTION_WITH_SEQUENCE_INCREMENT** - This option was created for the direction from DB2 to Oracle. If this option is set to "yes" starting value for converted sequence would be the current value of source sequense plus the value specified in INCREMENT BY clause. Source: CREATE SEQUENCE ORDER_SEQ START WITH 1 INCREMENT BY 2 NO MAXVALUE NO CYCLE CACHE 24# (current sequence value is 3) Equivalent: CREATE SEQUENCE ORDER_SEQ INCREMENT BY 2 START WITH 5 MAXVALUE 2147483647 MINVALUE 1 NOCYCLE CACHE 24 NOORDER; When this option is set to "no" (default) Equivalent: CREATE SEQUENCE ORDER_SEQ INCREMENT BY 2 START WITH 1 MAXVALUE 2147483647 MINVALUE 1 NOCYCLE CACHE 24 NOORDER;

- **EXPORT_CONSTRAINTS_FOR_QUERIES** - If this option is set to "yes" (default) and objects (tables) are selected for conversion using query, along with the export of the object (table) itself it's constraints (primary, foreign keys, etc.) will be also exported. But if you set this option to "no" object (table) will be converted without it's constraints.

- **CONVERT_DATABASE_TO_SCHEMA** - This option was created for the direction from Sybase ASE to Oracle. The default value is "no". When this option is set to "yes" source DB name will be used as schema name in the target. Example: source: schema1.table1 target: db1.table1.

- **cd2s_sch_to_obj_name** - This option was created for direction from Sybase ASE to DB2. This option is used only when option CONVERT_DATABASE_TO_SCHEMA is set to "yes". Option cd2s_sch_to_obj_name has "no" as a default value. When cd2s_sch_to_obj_name=yes schema name would be padded to object name like in the example below Source; db1.sch1.tab2 Equivalent: db1.sch1_tab2 When cd2s_sch_to_obj_name=no schema name would be omitted: Source; db1.sch1.tab2 Equivalent: db1.tab2

To specify DDL options using the SQLWays wizard

- SQLWays Wizard - DDL Options Page

[Windows] Subsection

This subsection is used to adjust command file generations for Microsoft Windows. SQLWays generates command files for importing database structure and data to the target database.

- **GCMD_AFTER_IMP_CMD** - Specifies a command that will be executed after importing each database object (table, view etc). This command is used in the general command file only.

SQLWays creates the general command file to make possible the importation of large number of database objects (tables, views etc) at once. See the /GCMD option for more information.

The general command file invokes command files for each database object using the *CALL* command. The *GCMD_AFTER_IMP_CMD* option allows specifying a command that will be placed after each *CALL* command.

For example, when *no* value is specified in the *GCMD_AFTER_IMP_CMD* option, the general command file contains *CALL* commands only:

```
call table1.cmd
call table2.cmd
call table3.cmd
call view1.cmd
call view2.cmd
```

When you specify, for example, GCMD_AFTER_IMP_CMD=pause, the general command file will contain:

```
call table1.cmd
pause
call table2.cmd
pause
call table3.cmd
pause
call view1.cmd
pause
call view2.cmd
pause
```


[Unix] Subsection

This subsection is used to adjust shell script generations for Unix. SQLWays generates shell scripts for importing database structure and data to the target database.

- **SHELL_LOCATION** - Specifies a Unix shell that executes import scripts on a Unix system. If you specify that import is executed on a Unix system, SQLWays generates Bourne shell scripts for importing database structure and data.

By default, SQLWays specifies that shell scripts be executed by */bin/sh*. Using this option you can specify another shell location, */usr/bin/sh* e.g.

This option is disabled if import is not executed on a Unix system.

[Oracle] Subsection

This subsection is used to adjust script generations when Oracle is the target database, and to specify properties when it is the source database.

- **BIN** - Specifies the directory where the Oracle utilities like SQL Plus and SQL Loader are located.
- **SQLLOADER** - The name of the executable module of the Oracle SQL Loader. The default value is sqlldr.exe.
- **SQLPLUS** - The name of the executable module of the Oracle SQL Plus. The default value is sqlplus.exe.
- **USER** - Specifies the user name that is used in generated scripts for the Oracle SQL Plus and SQL Loader.
- **PWD** - Specifies the user password that is used in generated scripts for the Oracle SQL Plus and SQL Loader.
- **SERVICE_NAME** - Specifies the service name for the Oracle Net protocol that is used in generated scripts for the Oracle SQL Plus and SQL Loader.
- **SQLPLUS_EXIT_CMD** - If *Yes* is specified, the SQL Plus EXIT command is generated at the end of DDL scripts. This allows executing DDL scripts from multiple files in batch mode. Possible values - *Yes*, *No*. The default value is *Yes*.
- **SQLPLUS_TO_DATE** - If *Yes* is specified, the TO_DATE function with the corresponding date format is generated for DATE columns in SQL INSERT statements for SQL Plus. This allows inserting rows that contain DATE columns when the date format differs from the default date format for Oracle database.

If *No* is specified, the TO_DATE function is not generated in SQL INSERT statements. Possible values - *Yes*, *No*. The default value is *Yes*.

- **DIRECT** - Specifies the Oracle SQL Loader method to use either conventional or direct path. Possible values - *Yes, No*. *Yes* specifies a direct path load. *No* specifies a conventional path load. The default value is *No*.
- **LOAD_OPTION** - Specifies loading option in the generated SQL Loader control file. Possible values - Insert, Append, Replace, Truncate. The default value is Insert.
- **DATE** - Specifies the format string in SQL Loader control files for loading dates (year, month, day) into DATE columns from the text files. The default value is "YYYY-MM-DD".

If any value is specified in the DATE_FORMAT option in the [Data] subsection, it overrides the DATE option.

- **TIME** - Specifies the format string in SQL Loader control files for loading times (hours, minutes, seconds) into DATE columns from the text files. The default value is "HH24:MI:SS".

If any value is specified in the TIME_FORMAT option in the [Data] subsection, it overrides the TIME option.

- **TIMESTAMP** - Specifies the format string in SQL Loader control files for loading timestamps (year, month, day, hours, minutes, seconds) into DATE columns from the text files. The default value is "YYYY-MM-DD HH24:MI:SS".

If any value is specified in the DATETIME_FORMAT option in the [Data] subsection, it overrides the TIMESTAMP option.

- **TIMESTAMP9i** - Specifies the format string in SQL Loader control files for loading Oracle 9i TIMESTAMP data type (year, month, day, hours, minutes, seconds, fractional seconds precision) from the text files. The default value is "YYYY-MM-DD HH24:MI:SS.FF".

- **FIXFMT_DECODE_NULLCHAR** - Specifies generating DECODE function in the SQL Loader control file for CHAR NOT NULL columns for fixed-length text file. Possible values - *Yes*, *No*. The default value is *No*.

If you what to load fields that contain only blanks into CHAR NOT NULL columns and the text file has the fixed length format, you have to use DECODE function, because SQL Loader treats blanks as NULLs. Set this option to *Yes* and SQLWays will generate DECODE function to convert NULLs to blanks.

To specify Oracle options using the SQLWays wizard

- SQLWays Wizard - Oracle Advanced Options Page

[IBM DB2] Subsection

This subsection is used to adjust script generations when IBM DB2 is the target database, and to specify properties when it is the source database.

- **BIN** - Specifies the directory where the IBM DB2 utilities like CLP, IMPORT and LOAD are located.
- **DATABASE** - Specifies the database name that is used in generated scripts for the IBM DB2 utilities.
- **USER** - Specifies the user name that is used in generated scripts for the IBM DB2 utilities.
- **PWD** - Specifies the user password that is used in generated scripts for the IBM DB2 utilities.
- **USE_LOAD_COMMAND** - If *Yes* is specified, the IBM DB2 LOAD command is used to move data into the IBM DB2 database. Possible values - *Yes*, *No*. The default value is *No*, which means that the IBM DB2 IMPORT command is used.
- **LOAD_FROM_CLIENT** - If *Yes* is specified, the CLIENT option is used in the IBM DB2 LOAD command. This option specifies that the data can reside on a remote client. Otherwise, the data must reside on the IBM DB2 server.

Possible values - *Yes*, *No*. The default value is *No*. The CLIENT option in the LOAD command is available in IBM DB2 7.1 or later.

- **LOAD_NONRECOVERABLE** - If *Yes* is specified, the NONRECOVERABLE option is added to the IBM DB2 LOAD command. This option specifies that the load transaction is to be marked as non-recoverable, and that it will not be possible to recover it by a subsequent roll forward action.

With this option, when the database is configured to support online backups, table spaces are not put in backup pending state following the load operation.

Possible values - *Yes*, *No*. The default value is *No*.

- **IMPORT_OPTION** - Specifies a mode, under which the import utility will execute. Possible values are INSERT (Adds the imported data to the table without changing the existing table data), INSERT_UPDATE (Adds rows of imported data to the target table, or updates existing rows of the target table with matching primary keys) and REPLACE (Deletes all existing data from the table by truncating the data object, and inserts the imported data. The table definition and the index definitions are not changed). The default mode is INSERT.
- **LOAD_OPTION** - Specifies a mode, under which the load utility will execute. Possible values are INSERT (Adds the loaded data to the table without changing the existing table data) and REPLACE (Deletes all existing data from the table, and inserts the loaded data. The table definition and index definitions are not changed). The default mode is INSERT.
- **MODIFIEDBY_OPTIONS** - Specifies additional options that are added to the modified by option of the IBM DB2 IMPORT and LOAD utilities.

Some options like character, column delimiters and others are added to the IBM DB2 IMPORT/LOAD scripts by SQLWays. You can use this option if you need other options to be added to the modified by option.

For example, set modifiedby_options=usedefaults delpriority char and these options will be added to the modified by option in each script for the IBM DB2 IMPORT/LOAD utilities.

- **TIMESTAMP_FORMAT** - This option specifies the format of the timestamp columns in the text file when the target database is IBM DB2. Possible values - IBM DB2, ISO.
  The default value is IBM DB2 that means using the IBM DB2 native format of timestamps. The IBM DB2 native format is YYYY-MM-DD-HH.MI.SS.FFFFFF. The IBM DB2 IMPORT command requires this format to import text

files containing timestamps not enclosed by double quotes (TAB delimited output format e.g.). When ISO is specified, the ISO format of timestamps will be used. The ISO format is YYYY-MM-DD HH:MI:SS.FFFFFF. The IBM DB2 IMPORT command can import timestamp values in the ISO format, when they are enclosed by double quotes (CSV output format e.g.).

Note. This option is ignored when any value is specified in the option DATETIME_FORMAT (see earlier in the [DATA] subsection).

For example, when DATETIME_DATATYPE=DATE is specified, the datetime datatypes will be converted to DATE in IBM DB2.

- **TABLESPACE** - Specifies the table space in which the tables will be created. The table space must exist, and be a REGULAR table space. If *no* other table space is specified, all table parts will be stored in this table space. The default value is USERSPACE1.
- **INDEX_TABLESPACE** - Specifies the table space in which any indexes on the tables will be created. The specified table space must exist, be a REGULAR DMS table space.
- **LONG_TABLESPACE** - Specifies the table space in which the values of any long columns (LONG VARCHAR, LONG VARGRAPHIC, LOB data types) will be stored. The table space must exist, be a LONG DMS table space.
- **SELECT_EXACT_ORDER**- This option was created for direction Sybase ASE - DB2 to get exactly the same order of output rows as in the source. Available values are 'yes' and 'no' (default). When set to 'yes' SQLWays explicitly adds all the columns listed in SELECT clause to ORDER BY clause as Sybase sorts by all columns implicitly.
- **DECLARE_TEMP2CREATE_TEMP**- This option was created for direction Sybase ASE - DB2. Available values are 'yes' and 'no' (default). When set to 'yes' SQLWays converts Sybase SELECT INTO tempdb and CREATE TABLE tempdb to DB2 CREATE GLOBAL TEMPORARY TABLE. Otherwise these statements are converted to DECLARE GLOBAL TEMPORARY TABLE.
- **DYNAMIC_RESULT_SETS**- This option was created for DB2 as a target database. Available values are 'yes' and 'no' (default). When set to 'yes' SQLWays converts CREATE PROCEDURE statement to DB2 procedure with DYNAMIC RESULT SETS clause.
- **DYNAMIC_RESULT_SETS_VALUE**- This option is used together with the option DYNAMIC_RESULT_SETS. Default value is 100.

To specify IBM DB2 options using the SQLWays wizard

- SQLWays Wizard - IBM DB2 Advanced Options Page

[MSSQL] Subsection

This subsection is used to adjust script generations when Microsoft SQL Server is the target database, and to specify properties when it is the source database.

- **BIN** - Specifies the directory where the Microsoft SQL Server utilities like BCP and ISQL are located.
- **SERVER_NAME** - Specifies the server name that is used in generated scripts for the BCP and ISQL utilities (Parameter -S of BCP).
- **DATABASE** - Specifies the database name that is used in generated scripts for the BCP and ISQL utilities.
- **USER** - Specifies the user name that is used in generated scripts for the BCP and ISQL utilities (Parameter -U of BCP). To use this option the option TRUSTED_CONNECTION (see below) must be set to *No*.
- **PWD** - Specifies the user password that is used in generated scripts for the BCP and ISQL utilities (Parameter -P of BCP). To use this option the option TRUSTED_CONNECTION (see below) must be set to *No*.
- **TRUSTED_CONNECTION** - Specifies that a trusted connection to SQL Server is used in generated scripts for the BCP and ISQL utilities (Parameters -T of the BCP and -E of the ISQL). When this option is set to *Yes* the security credentials of the network user are used and the user name (login_id) and password are not required. Possible values - *Yes*, *No*. The default value is *No*.
- **MAX_ERRORS** - Specifies the maximum number of errors that can occur before the BCP utility is canceled. Each row that cannot be copied by BCP is ignored and counted as one error (Parameter -m of BCP). The default value is 10.
- **CODE_PAGE** - Specifies the code page of the data in the data file for the BCP utility (Parameter -C of BCP). The following values can be specified for the CODE_PAGE option:
  - **ACP** - ANSI/Microsoft Windows (ISO 1252).
  - **OEM** - Default code page used by the client. This is the default code page used by bcp if -C is not specified.
  - **RAW** - No conversion from one code page to another is taking place.
  - <*value*> - Specific code page number, for example, 437.
- **USE_CONVERT_CHAR_TO_VARCHAR**- Specifies what CHAR should be converted to. If set to 'yes' and the length of CHAR is equal or greater than the number specified in CONVERT_CHAR_TO_VARCHAR option CHAR datatype should be converted to VARCHAR. If set to 'no' (default) CHAR is converted to CHAR regardless the length.
- **CONVERT_CHAR_TO_VARCHAR**- Specifies the length for CHAR to be converted to VARCHAR if USE_CONVERT_CHAR_TO_VARCHAR option is set to 'yes'.

[Sybase] Subsection

This subsection is used to adjust script generations when Sybase is the target database, and to specify properties when it is the source database.

- **BIN** - Specifies the directory where the Sybase utilities like BCP and ISQL are located.
- **SERVER_NAME** - Specifies the server name that is used in generated scripts for the BCP and ISQL utilities (Parameter -S).
- **DATABASE** - Specifies the database name that is used in generated scripts for the BCP and ISQL utilities.
- **USER** - Specifies the user name that is used in generated scripts for the BCP and ISQL utilities (Parameter -U).
- **PWD** - Specifies the user password that is used in generated scripts for the BCP and ISQL utilities (Parameter -P).

[MySQL] Subsection

This subsection is used to adjust script generations when MySQL is the target database, and to specify properties when it is the source database.

- **BIN** - Specifies the directory where the MySQL utilities are located.
- **HOST** - Specifies a remote host name where the target MySQL database resides.
- **DATABASE** - Specifies the database name that is used in generated scripts for MySQL.
- **USER** - Specifies the user name that is used in generated scripts for MySQL.
- **PWD** - Specifies the user password that is used in generated scripts for MySQL.
- **TABLE_TYPE** - This option specifies a table type that is used when creating tables in the MySQL database. MySQL supports two different kinds of tables: transaction-safe tables (InnoDB and BDB) and not transaction-safe tables (HEAP, ISAM, MERGE, and MyISAM).

For example, when table_type=InnoDB is selected, SQLWays generates the TYPE=InnoDB clause in the CREATE TABLE statements for MySQL.

- **IMPORT_FROM_CLIENT** - If *Yes* is specified, the LOCAL keyword is generated for the MySQL LOAD DATA INFILE command that is used for importing data to MySQL. When LOCAL is specified, data files can be located on the client host.

If *No* is specified, the LOCAL keyword is not generated and data files must be located on the server before importing.

Possible values - *Yes*, *No*. The default value is *Yes*.

Note. There are some issues regarding using the LOCAL keyword in MySQL 4.0.x. See LOCAL keyword - The used command is not allowed with this MySQL version.

- **DATA_LOAD_OPTION** - If *Replace* is specified, existing rows are updated in the existing table basing on primary or unique key information or added to the table if primary key is not matched. If *Ignore* is specified the existing rows in the table are not updated, when primary or unique key values are equivalent to the existing in the table. The rows from the text file which do not match primary or unique key in the table are inserted.

To specify MySQL options using the SQLWays wizard

- SQLWays Wizard - MySQL Advanced Options Page

[Pervasive] Subsection

This subsection is used to adjust script generations when Pervasive.SQL is the target database, and to specify properties when it is the source database.

- **BIN** - Specifies the directory where the Pervasive.SQL utilities like BUTIL are located.
- **DATA_DIR** - Specifies the directory where the database data files (.mkd files) are located. This option is used for the BUTIL utility.
- **DATABASE** - Specifies the database name (ODBC alias) that is used in generated scripts for Pervasive.SQL.
- **USER** - Specifies the user name that is used in generated scripts for Pervasive.SQL.
- **PWD** - Specifies the user password that is used in generated scripts for Pervasive.SQL.

[Formatting] Subsection

This subsection is used to apply formatting and bring better readability to converted script.Works for the direction Sybase ASE - DB2.

- **FORMATTING** - Specifies whether the formatting would be applied or not. Available values 'yes' and 'no' (default).
- **LENGTH**- Specifies max number of symbols in a line.
- **CAPS** - Specifies whether all key words would be converted in capital letters or not. Available values 'yes' and 'no' (default).
- **EMPTY_LINE4DML** - Specifies whether empty line spaces would be added after each DML statement. Available values 'yes' and 'no' (default).
- **NEW_LINE** - Specifies whether all key words would start from a new line. Available values 'yes' and 'no' (default).

Command Line Tips

Return Codes

If the exporting was successful, the SQLWays return code is 0. Otherwise, the return code is -1. You can use this feature in batch files.

For example:

```
@echo off
cls
SQLWays /d=datasource /u=user /p=pwd /t=table
if errorlevel 0 goto ok
echo.
echo Converting error
goto ex
:ok
echo.
echo no errors
:ex
```

SQLWays can also return 1. This is possible when the /NSTOP option is used, and it means that the last operation was successful but there has been an intermediate error (for example, processing of one of the previous tables failed).

Using OS Special Characters

- Microsoft Windows interprets such characters as %,| as the OS special characters. Thus, if you want to use them in the command line or batch files, you have to duplicate them.

Use:
SQLWays /d=dsn /s=select * from t1 where field1 like '%%123%%'

instead of
SQLWays /d=dsn /s=select * from t1 where field1 like '%123%'

- Microsoft Windows treats the character > as a redirection character. If you want to use this character in the SELECT statements, you have to enclose the statement in double quotes:

SQLWays /d=dsn /s="select * from t1 where field2 > 123"

Setting Up ODBC Data Sources

ODBC (Open Database Connectivity) is a standard application programming interface (API) used to access data in relational databases. Database vendors as well as third-party vendors offer ODBC drivers to enable connecting to all popular databases using ODBC. SQLWays can use the ODBC interface to access source databases.

In order to use ODBC to access databases, you have to create a data source that associates a particular ODBC driver with a database and contains information how to connect to the database.

ODBC data sources can be created by using the **ODBC Data Source Administrator** application of the Control Panel under **Administrative Tools** as Data Sources (ODBC) (for information about how add and configure the drivers, click Help in the ODBC Data Source Administrator dialog box) or by SQLWays Wizard clicking **New** in the **Choose a Source Database** page.

Data source information must be configured for each database system that is to be accessed using the ODBC Driver for a database.

The following steps describe how to create and configure data sources for use with the ODBC Driver for a database.

- Run the **ODBC Data Source Administrator**.
- Select the ODBC data source type to be created or modified. An ODBC data source name (**DSN**) can be one of the following types:

  •**File** - A data source stored in a file that can be shared among all users who have the same ODBC drivers installed. These data sources need not be dedicated to a specific user or local to a computer.

  •**User** - A data source local to a computer and accessible only by the current user that created the data source.

  •**System** - A data source local to a computer but not dedicated to a specific user, so any user with appropriate privileges can access a system DSN. A System data source is visible to all users on a computer, including Windows NT services.

User and System data sources are stored in the registry. File data sources are stored as files with a file extension of dsn. File DSNs can be stored in any location on the file system including remotely-mounted shares. By default, File DSNs are stored in the following location:

C:\Program Files\Common Files\ODBC\Data Sources

- Select the Driver for a database from the list of drivers, set **Advanced File DSN Creation Settings** (driver-specific keywords) for a File data source type and click the Next button.
- Specify the name and the location of **ODBC Data Sources file** and click **Next**.
- View the configured data source and click **Finish**.

- Configuring ODBC Connection to Sybase Adaptive Server Anywhere
- Configuring ODBC Connection to Sybase Adaptive Server Enterprise
- Configuring Connection to MySQL using MyODBC

Configuring ODBC Connection to Sybase Adaptive Server Anywhere

This section provides a description of the ODBC Configuration Options dialogs that are provided with Adaptive Server Anywhere to save a set of its connection parameters as an ODBC data source.

- Sybase Adaptive Server Anywhere Driver (ASA version 6.x)
- Sybase Adaptive Server IQ Driver (ASIQ version 12.x)
- Sybase Adaptive Server Enterprise Drivers (ASE versions 11.x and 12.0)

Sybase Adaptive Server Anywhere Driver

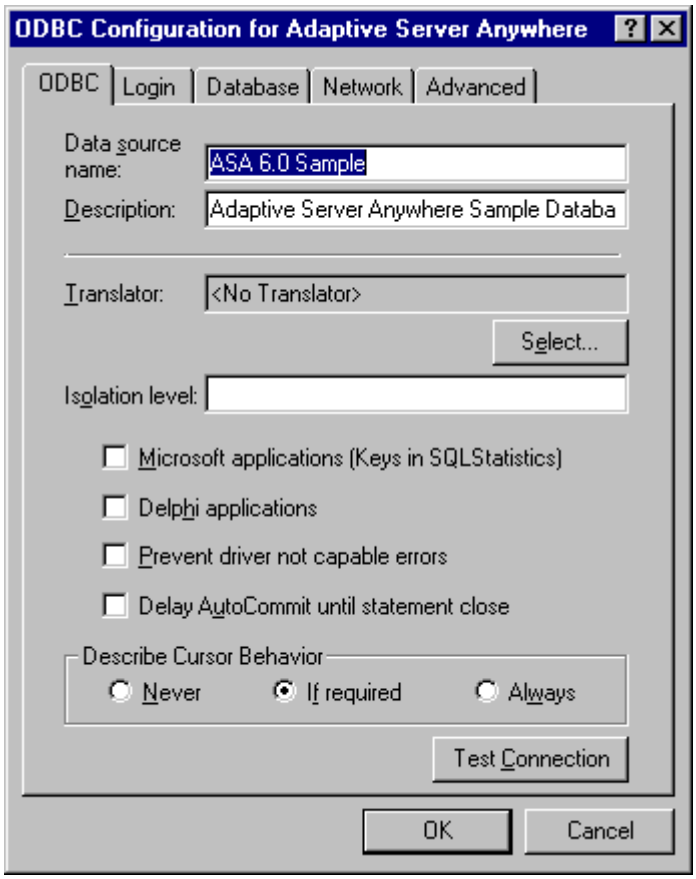
This section provides connectivity information for the ODBC driver for Adaptive Server Anywhere (hereafter called 'ASA') version 6.0.1 (driver version 6.0). This driver allows client applications to connect to an ASA server. Unlike the ODBC driver for Adaptive Server Enterprise, the ASA driver does not require that Open Client be installed on the machine with the ODBC driver. For more information about the Adaptive Server Enterprise driver, see the section Sybase Adaptive Server Enterprise Drivers.

The ODBC Configuration dialog consists if six tabs:

- ODBC tab
- Login tab
- Database tab
- Network tab
- Advanced tab
- Certicom Encryption Options dialog

ODBC Sybase ASA Driver Setup Dialog: ODBC Tab

The following shows the **ODBC** tab in the driver configuration. It is where you name the ODBC data source (DSN) you are configuring. This tab also contains the **Test Connection** button. This button allows you to test the connection to ASA, after you finish the configuring the DSN and before you exit out of the driver configuration utility. The following screen shows the sample ASA ODBC DSN created with the installation of the ASA network server.



- In the **Data source name box** type a name to identify this ODBC data source. You can use any descriptive name for the data source (spaces are allowed) but it is recommended that you keep the name short, as you may need to enter it in connection strings.
- In the **Description** box you can provide a description of the data source. This description may help you or your end users to identify this data source from among their list of available data sources. This is an optional field.
- In the **Isolation level** box type a numerical value to specify the initial isolation level for this data source:

  •**0** This is also called the read uncommitted isolation level. This is the default isolation level. It provides the maximum level of concurrency, but dirty reads, non-repeatable reads, and phantom rows may be observed in result sets.

  •**1** This is also called the read committed level. This provides less concurrency than level 0, but eliminates some of the inconsistencies in result sets at level 0. Non-repeatable rows and phantom rows may occur, but dirty reads are prevented.

- **2** This is also called the repeatable read level. Phantom rows may occur. Dirty reads and non-repeatable rows are prevented.
- **3** This is also called the serializable level. This provides the least concurrency, and is the strictest isolation level. Dirty reads, non-repeatable reads, and phantom rows are prevented.

- If the **Microsoft applications (Keys in SQLStatistics)** check box is selected, so the SQLStatistics function returns foreign keys. The ODBC specification states that SQLStatistics should not return primary and foreign keys; however, some Microsoft applications (such as Visual Basic and Access) assume that primary and foreign keys are returned by SQLStatistics.

- If the **Delphi applications** check box is selected the Borland Delphi application development tool is used to create applications that produce your data source.

When this option is selected, one bookmark value is assigned to each row instead of the two that are otherwise assigned (one for fetching forwards and a different one for fetching backwards).

Delphi cannot handle multiple bookmark values for a row. If the option is cleared, scrollable cursor performance can suffer since scrolling must always take place from the beginning of the cursor to the row requested in order to get the correct bookmark value.

- If the **Suppress fetch warnings** check box is selected warning messages that are returned from the database server on a fetch are suppressed.

Versions 8.0 and later of the database server return a wider range of fetch warnings than earlier versions of the software. For applications that are deployed with an earlier version of the software, you can select this option to ensure that fetch warnings are handled properly.

- **Prevent driver not capable errors** The Adaptive Server Anywhere ODBC driver returns a Driver not capable error because it does not support qualifiers. Some ODBC applications do not handle this error properly. Select this option to prevent this error code from being returned, allowing these applications to work.

- **Delay AutoCommit until statement close** Select this option to delay the commit operation until a statement closes.

- **Describe Cursor Behavior** Choose how often you want a cursor to be redescribed when a procedure is executed. The default setting is If Required.

  - **Never** Select this option if you know that your cursors do not have to be redescribed. Redescribing cursors is expensive and can decrease performance.
  - **If required** When you select this option, the ODBC driver determines whether a cursor must be redescribed. The presence of a RESULT clause in your procedure prevents ODBC applications from redescribing the result set after a cursor is opened. This is the default.
  - **Always** The cursor is redescribed each time it is opened. If you use Transact-SQL procedures or procedures that return multiple result sets, you must redescribe the cursor each time it is opened.

- **Translator** The translator converts characters between ANSI and OEM code pages. Most databases do not require a translator because the ODBC Driver Manager automatically performs a conversion between the client's character set and the database's character set. If your database uses an ANSI code page (the default) do not select a translator.

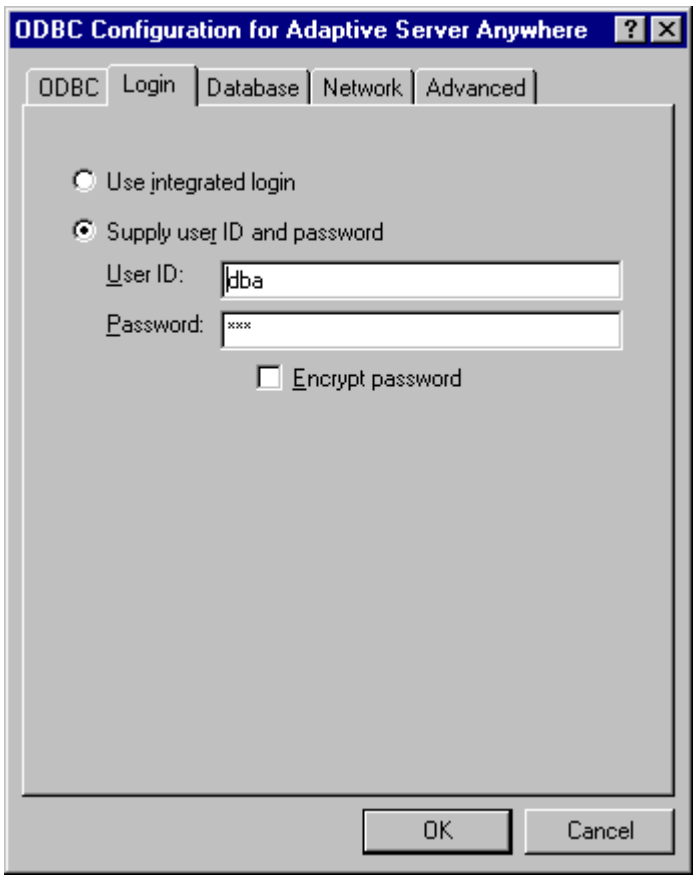
If you require a translator, click Select and choose the translator from the list of installed translators.

- **Select Translator** Click Select Translator to choose the ODBC translator you require from a list of installed translators.

- **Test Connection** Tests whether the information provided results in a proper connection. In order for the test to work, a user ID and password must be specified on the Login tab.

ODBC Sybase ASA Driver Setup Dialog: Login Tab

The **Login** tab, shown below, is the next tab in the driver configuration. It is where you specify the login information that the ODBC driver will use when it makes a connection.



The Login tab of the ODBC Configuration dialog Login tab has the following components:

- **Use integrated login** Select this option to connect using an integrated login.

If you select this option, the user ID and password do not need to be specified: instead, your operating system user ID and password are supplied to the Adaptive Server Anywhere integrated login mechanism.

To use integrated logins, users must have been granted integrated login permission and the database you are connecting to must also be set up to accept integrated logins. Only users with DBA access may administer integrated login permissions.

- **Supply user ID and password** Select this option if you want to specify the user ID and password for the connection.

User ID Type a user ID for the connection. The user ID you supply must have permissions on a database in order to connect.

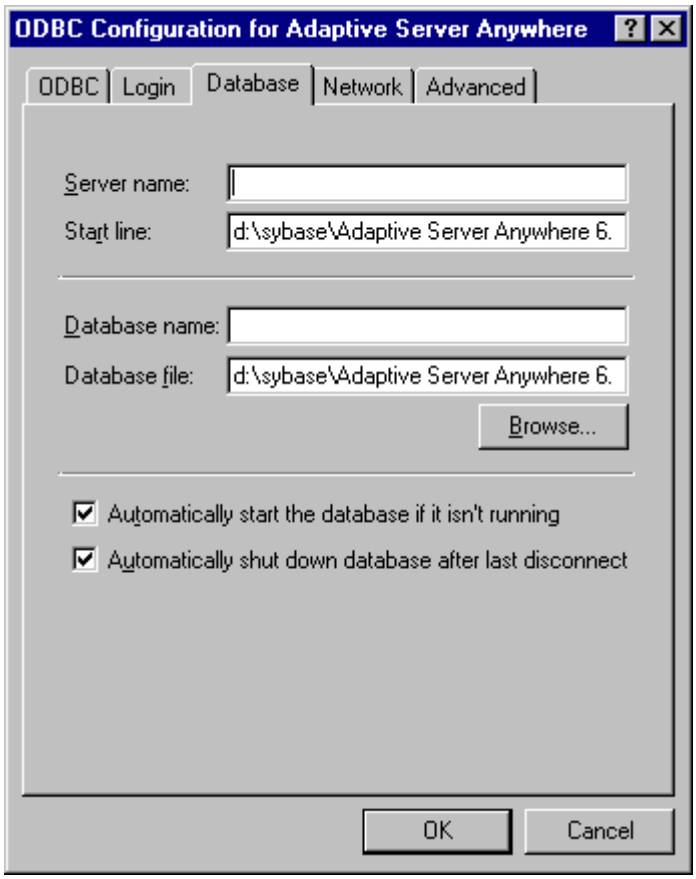
Password Type a password for the connection. The password must be the correct password for the user ID you supply.

Extended characters used in passwords are case-sensitive, regardless of the database sensitivity setting.

- **Encrypt password** Select this option if you want the password to be stored in encrypted form in the profile.

ODBC Sybase ASA Driver Setup Dialog: Database Tab

The third tab from the left is the **Database** tab. If you are configuring an ODBC DSN for an ASA installation on the same machine, you specify the name and location of the server file and database file as shown in the following screen. This screen also allows you to configure whether to start or shut down the database when connecting or disconnecting. This is definitely something to keep in mind when configuring and then using an ASA DSN.



The Database tab of the ODBC Configuration dialog has the following components:

- **Server name** Type the name of the Adaptive Server Anywhere personal server or network server. For example, **asademo**. You need to supply a server name if you want to connect to a network server.

Do not enter a Server Name if you want to connect to the default local personal server or if you want to start a database server from a database file on your local machine. If you do not have a default personal server and you omit the server name, the connection fails.

- **Start line** The start line is a command to start a personal database server or a network server on your machine. Enter a start line only if you want to connect to a local database server that is not currently running and if you want to set your own start parameters. You must enter the full path of the server, for example to start the personal database server enter, **c:\Program Files\Sybase\SQL Anywhere 9\win32\dbeng9.exe**.

You can also include options in the Start Line field. The start line and options are used when you want to:

•Employ any advanced server features.

- Control communications parameters.
- Provide diagnostic or troubleshooting messages.
- Set permissions.
- Set database parameters (including encryption).

- **Database name** Each database running on a server is identified by a database name. Type the name of the database you are connecting to.

- **Database file** Specify the database file when the database you want to connect to is not currently running on a server. It is recommended that you type the full path and name of the database file, for example, C:\sample.db. Otherwise, the path of the file is relative to the working directory of the server.

  - **Browse** Click Browse to select the database file from the file directory.

- **Encryption key** If the database file is encrypted, you must supply a key to the database server every time the database server starts the database.

If the database is strongly encrypted using AES or MDSR, you must specify a key in this field to access the database.

You can also supply encryption options in the Start Line field.

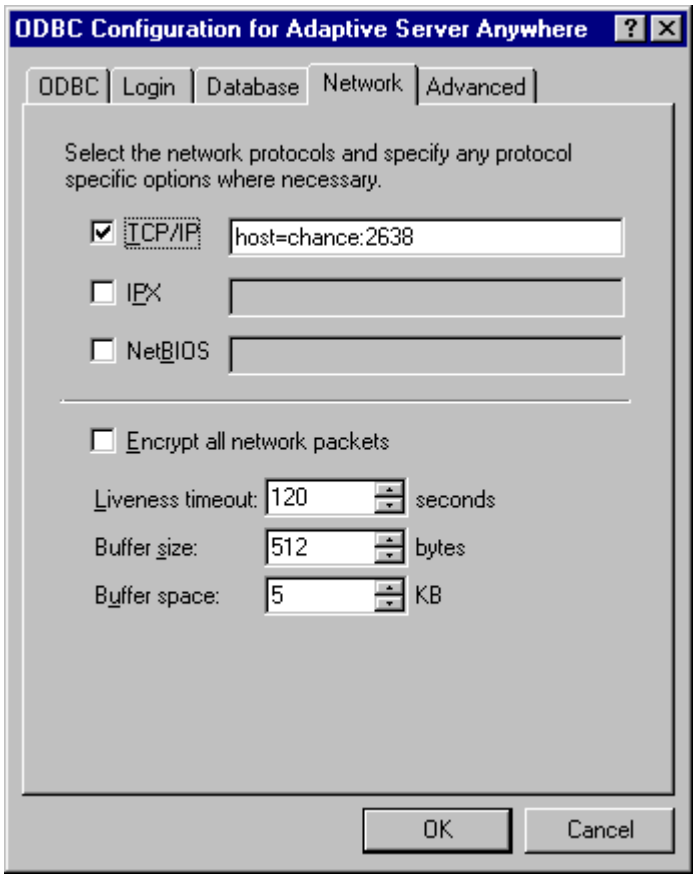
- **Start database automatically** Select this option to start the database specified in the Database File field before you connect to it.

You should clear the Start Database Automatically option if you want to ensure that you connect only to a running database.

- **Stop database after last disconnect** Select this option to automatically shut down the database after the last user disconnects.

ODBC Sybase ASA Driver Setup Dialog: Network Tab

If you are configuring an ODBC DSN for an ASA installation on a remote machine, then you need to fill in connectivity information on the **Network** tab. The network connection information must be in the format: host=*machine_name:port_number*, where *machine_name* is the name of the machine on which the ASA you want to connect runs and *port_number* is the port number for that ASA. For example:
host=chance:2638



The Network tab of the ODBC Configuration dialog has the following components:

- **TCP/IP** If you want to use ECC_TLS (formerly Certicom) or RSA_TLS encryption for network packets, you must select the TCP/IP protocol to access the network database server. In the adjacent field, you may enter communication parameters that establish and tune the connection from your client application to a database.

For example, to look on the machine server1 on port 4436 for a database server, you would enter **HOST=server1;PORT=4436**.

Contact your network administrator if you are unsure which protocol to use.

- **SPX** You can select the SPX protocol to connect to databases on Novell NetWare networks. NetWare also supports the TCP/IP protocol. In the adjacent field, you may type communication parameters that establish and tune the connection from your client application to a database.

For example, you could enter the following communication parameters for an SPX connection **HOST=0:0:0:0:0:1/4:236:121:215;PORT=2369**.

Contact your network administrator if you are unsure which protocol to use.

- **Named pipes** The Named Pipes protocol is used for client/server communication on the same machine. If you want to run under a certified security environment, you can use the named pipes protocol. It is only provided on Windows NT.

Contact your network administrator if you are unsure which protocol to use.

- **Shared memory** The shared memory protocol is used for communication between a client and server running under the same operating system on the same machine.

Contact your network administrator if you are unsure which protocol to use.

- **Liveness timeout** A liveness packet is sent across a client/server to confirm that a connection is intact. If the client runs for the liveness timeout period without detecting a liveness packet, the communication is severed. This parameter only works with network server and TCP/IP or IPX communication protocols.

The default liveness timeout is 120 seconds.

- **Idle timeout** Set the amount of client idle time before the connection is terminated. If a client runs for the idle timeout period without submitting a request, the connection is severed.

The default client idle time is 240 minutes.

- **Buffer size** Set the maximum size of communication packets, in bytes. You should set the buffer size to be smaller than that allowed by your network because the network software may add information to each buffer before sending it over the network.

The default buffer size is 1460 bytes.

- **Compress network packets** Select this option to turn on compression for the connection. Using compression for a connection can significantly improve Adaptive Server Anywhere performance under some circumstances.

- **Select the method for encryption of network packets** Allows the encryption of packets transmitted from the client machine over the network.

  •**None** Communication packets transmitted from the client are not encrypted. This is the default setting.

  •**Simple** Communication packets transmitted from the client are encrypted with Simple encryption. Simple encryption is supported on all platforms, as well as on previous versions of Adaptive Server Anywhere. Simple encryption is not as strong as ECC_TLS or RSA_TLS encryption.

  •**ECC TLS** Select this option to enable ECC_TLS (formerly Certicom) encryption. ECC_TLS encryption protects the confidentiality and integrity of network packets as they pass between the client and the server. It is only available over the TCP/IP protocol.

In the adjacent field you must supply a trusted certificate value. You may also click Edit to supply this value.

  •**RSA TLS** Select this option to enable RSA_TLS encryption. RSA_TLS encryption protects the confidentiality and integrity of network packets as they pass between the client and the server. It is only available over the TCP/IP protocol.

In the adjacent field you must supply a trusted certificate value. You may also click Edit to supply this value.

- **Edit** Click Edit to supply Certicom encryption values for trusted certificates, certificate company, certificate unit, and certificate name in the Certicom Encryption Options dialog.

Note that this is unlike the format of the connection string for Adaptive Server Enterprise and the DirectConnects, which use the Open Client format of *machine_name,port*. For example: chance,5000

Testing the Connection

As noted previously in this section, the ASA driver configuration utility provides a **Test Connection** button on the **ODBC** tab. This allows you to verify connectivity while you are still in the driver configuration utility.

ODBC Sybase ASA Driver Setup Dialog: Advanced Tab

The Advanced tab of the ODBC Configuration dialog has the following components:

- **Connection name** Type a name to identify the connection. This field is optional.
- **Character set** Type a character set name. By default, the client's ANSI character set is used, for example cp1252 for English systems. You can also supply an OEM character set instead of an ANSI character set.
- **Allow multiple record fetching** Select this option to retrieve multiple records at one time, instead of individually, for improved performance.
- **Display debugging information in a log file** Select this option to record diagnostic information on communications links in the log file.

  •**Log file** Type the name of the log file where you want to save the debugging information.
- **Additional connection parameters** Type any additional connection parameters in this field in a semicolon separated list. For example,

DEBUG=YES;LOG=connection.logParameters set in this field take precedence over parameters set throughout the rest of this dialog. For example, if you enter the user ID DBA on the Identification tab, and set the connection parameter "uid=bsmith" in this field, a connection with the user ID bsmith is attempted.

Certicom Encryption Options Dialog

This dialog displays fields for the client's Certicom encryption settings.

The Certicom Encryption Options dialog has the following components:

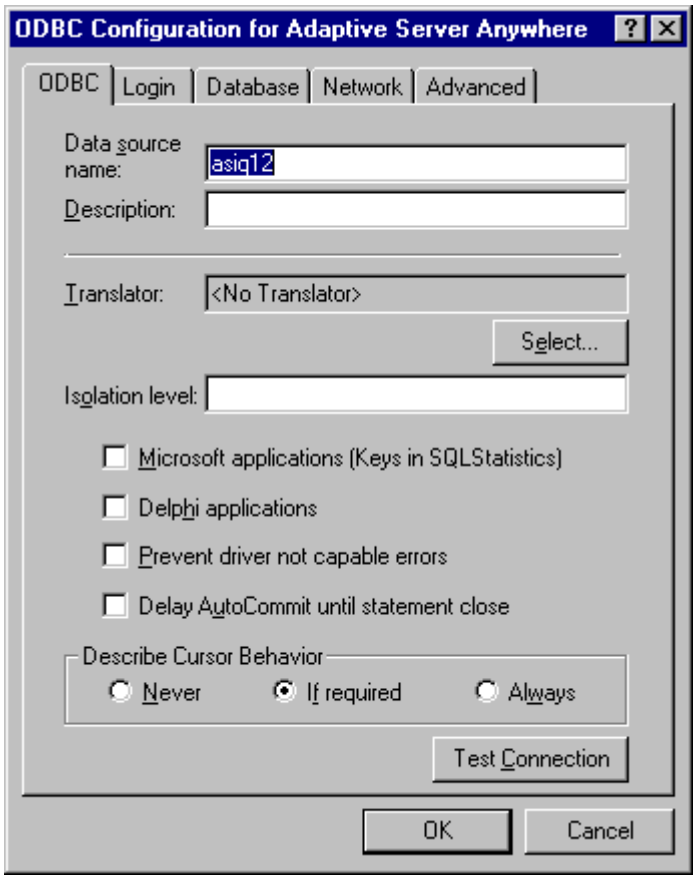
- **Trusted certificates** Type the name of the certificate file the client uses to authenticate the server. You can also click Browse to select the trusted certificate from the file directory. This is a required field.
  - **Browse** Click Browse to select the certificate from the file directory.
- **Certificate company** Type the name of the certificate authority or organization that issued the certificate. The server's and the client's values must match. This is an optional field.
- **Certificate unit** Type the certificate unit. This is also called the organizational unit. The server's and the client's values must match. This is an optional field.
- **Certificate name** Type the certificate's common name. The server's and the client's values must match. This is an optional field.

Sybase Adaptive Server IQ Driver

This section provides information about the ODBC driver for Adaptive Server IQ (hereafter called 'ASIQ') version 12. This driver allows client applications to connect to ASIQ version 12 servers. The configuration banner for the ASIQ driver actually reads "ODBC Configuration for Adaptive Server Anywhere," when it is, in fact, for ASIQ as shown in the following screen. The ASA and ASIQ 12 ODBC drivers are essentially the same. (ASIQ 12 is built on the ASA architecture.)

Because the drivers are essentially the same, the information that applies to the ASA driver also applies to the ASIQ 12 driver.

Please go to the previous section, Adaptive Server Anywhere Driver, more information about this driver.



Sybase Adaptive Server Enterprise Drivers

This section contains connection information about both the 11.x and 12.0 versions of the ODBC drivers for Adaptive Server Enterprise (hereafter called 'ASE'). Both of these drivers allow client applications to connect to ASE servers.

- ASE 12 ODBC Driver
- System 11 ODBC Driver

Note that both versions of the driver work with Open Client to make the ODBC connection to ASE. As you probably know, the ASE installation and configuration process automatically defines ASE to Open Client based on the server name specified during installation. The default is the name of the machine on which ASE is being installed. This definition involves creating an entry in the sql.ini file for ASE. The sql.ini entry is in the following format: *machine_name,port_number*. For example:
chance,5000

Note that this is a different connection string format than the ASA and ASIQ drivers use. They use the format host=*machine_name:port_number*.

If you are configuring an ODBC DSN on a machine that is remote from ASE, you must install the appropriate version of Open Client on the machine and define ASE to Open Client by creating a sql.ini file entry for it on that machine.

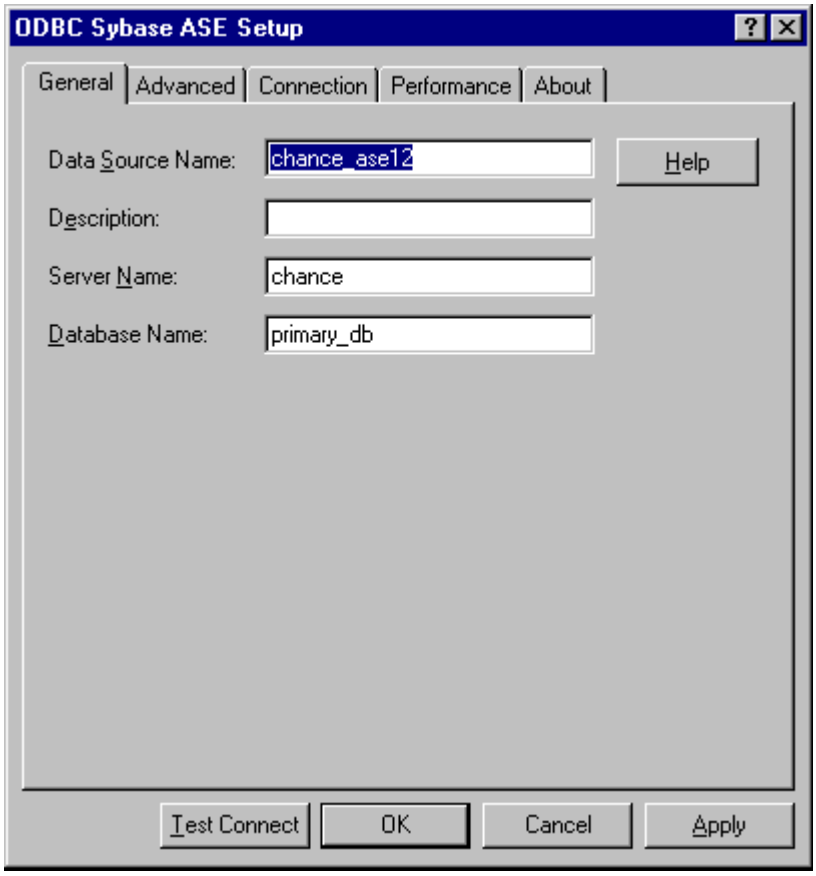
ASE 12 ODBC Driver

Configuring the ODBC DSN

The ASE ODBC Driver, which is new with ASE version 12, allows you to specify all the information you need on the **General** tab. It also now provides a connection test feature, so you can test the connection from within the driver configuration utility.

This driver is compatible with Open Client version 12.0 and ASE version 12, as well as ASE 11.9.2. You cannot use the driver with Open Client 11.1.1. If you attempt to configure a DSN using a server defined with Open Client 11.1.1, you will see that you can configure the DSN and run the Test Connect utility successfully. However, you will not be able to save the DSN and/or exit the driver configuration utility.

This ASE ODBC driver works with Open Client version 12.0 to make an ODBC connection. Therefore, you must specify the ASE server name as it is defined to Open Client in the sql.ini file in the Server name field of the ODBC DSN. The following image shows the **General** tab from the ASE driver configuration.



It contains the following information:

Data Source Name =the arbitrary name you assign for your ASE ODBC DSN. This is a required field.

Description =(blank in this picture) any descriptive information you want to provide about the DSN. This is an optional field.

Server Name =the server name for ASE as it is defined to Open Client in the sql.ini file. This is a required field.

Database Name =the name of the specific ASE database to which you want the ODBC DSN to connect. This is an optional field. You can specify a database on this tab, or on the logon screen, or not at all, in which case you are connected by default to the Master database.

Testing the Connection

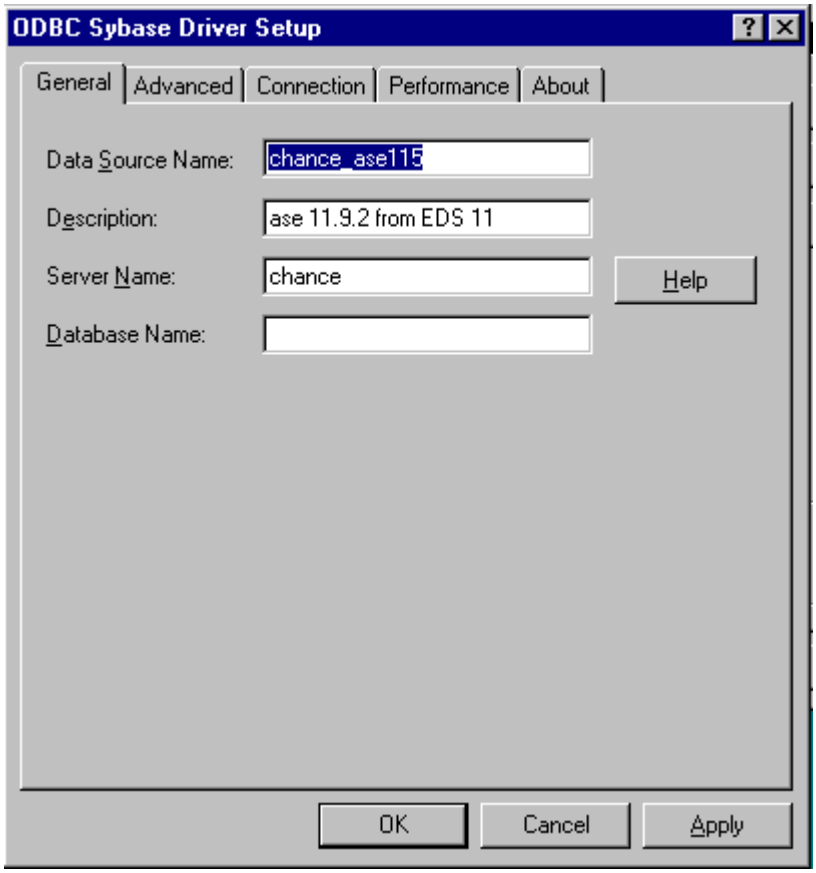
Use the Test Connect button to verify connectivity before you exit the driver configuration utility.

System 11 ODBC Driver

Configuring the ODBC DSN

The 11.x version of the ODBC driver for ASE has a name that is not exactly intuitive. It is called 'Sybase System 11.' This driver is compatible with Open Client 10.x and 11.1.1, SQL Server 10.x, ASE 11.5, ASE 11.9.2 and ASE 12.0.

This driver works with the Sybase Open Client protocol to make its ODBC connection. Therefore, you must specify the name of the ASE server in the Server Name field of the ODBC DSN definition as it is defined to Open Client in the sql.ini file. The following picture shows its **General** tab.



The information it contains is as follows:

Data Source Name =the arbitrary name you assign for your ASE ODBC DSN. This is a **required** field.

Description =any descriptive information you want to provide about the DSN. This is an optional field.

Server Name =the server name for ASE as it is defined to Open Client in the sql.ini file. This is a **required** field.

Database Name =the name of the specific ASE database to which you want the ODBC DSN to connect (In the picture, this is blank). This is an optional field. If you leave the Database Name field blank, the driver will connect by default to the Master database.

Testing the Connection

The Sybase System 11 driver configuration does not provide a connection test feature. To test connectivity for ODBC DSNs defined with this driver, you must try them with an ODBC application. If you have ODBC Test, which is provided with the Microsoft SDK, you can use it to verify connectivity to ASE.

Configuring ODBC Connection to Sybase Adaptive Server Enterprise

This section provides a description of the ODBC Configuration Options dialogs that are provided with Adaptive Server Enterprise to save a set of its connection parameters as an ODBC data source.

The ODBC Configuration dialog consists if four tabs:

- General Tab
- Advanced Tab
- Connection Tab
- Performance Tab

ODBC Sybase ASE Driver Setup Dialog : General Tab

- Use the ODBC Sybase ASE Driver Setup dialog to create new Sybase ASE data sources or configure existing data sources.

Data Source Name: Type a string that identifies this Sybase ASE data source configuration in the system information. Examples include "Accounting" or "Sys10-Serv1."

Description: Type an optional long description of a data source name. For example, "My Accounting Database" or "System 10 on Server number 1."

Network Library Name: Select the name of the network library. This specifies which network protocol to use. The values are Winsock and NamedPipes. The default is Winsock.

Network Address: Type the network address. The value you specify depends on which network protocol is chosen under Network Library Name and on the Sybase server. If you have chosen Winsock for the Network Library Name, specify an IP address as follows: "<servername or IP address>, <port number>". For example, if your network supports named servers, you may specify an address such as "Sybaseserver, 5000". You may also specify the IP address directly such as "199.226.224.34, 5000".

If you have chosen NamedPipes as the network protocol, you must specify the pipe address of the server. For example, "\\machine1\sybase\pipe\query".

Database Name: Type the name of the database to which you want to connect by default. If you do not specify a value, the default is the database defined by the system administrator for each user.

HA Failover Server Connection Information/Network Address: Type the network address of the High Availability (HA) Failover server to be used in the event of a connection loss. The driver detects the dropped connection and automatically reconnects to the HA Failover server specified by this attribute. This option is valid only for Sybase ASE version 12 servers that have the High Availability Failover feature enabled.

See the previous description of the Network Address option for an explanation of valid values.

ODBC Sybase ASE Driver Setup Dialog : Advanced Tab

Use the Advanced tab on the ODBC Sybase ASE Driver Setup dialog box to specify optional settings when you create new Sybase ASE data sources or configure existing data sources.

Cursor Positioning for raiserror: Select a value of 0 or 1 that specifies when the error is returned and where the cursor is positioned when raiserror is encountered.

When set to 0 (the default), raiserror is handled separately from surrounding statements. The error is returned when raiserror is processed via SQLExecute, SQLExecDirect, or SQLMoreResults. The result set is empty.

When set to 1 (MS compatible), raiserror is handled with the next statement. The error is returned when the next statement is processed; the cursor is positioned on the first row of subsequent result set. This could result in multiple raiserrors being returned on a single execute.

Default Buffer Size for Long Columns (in Kb): Type an integer value that specifies the maximum length of data fetched from a TEXT or IMAGE column. The value must be in multiples of 1024 (for example, 1024, 2048. The default is 1024 KB. You will need to increase this value if the total size of any long data exceeds 1 MB.

Distributed Transaction Model: Select a model to use for distributed transaction support—either XA Protocol or Native OLE.

Initialization String: Enter a semicolon-separated list of Sybase language commands that will be issued immediately after connection.

Enable Quoted Identifiers: Select this check box to allow support of quoted identifiers.

Enable Describe Parameter: Select this check box to enable the SQLDescribeParam function, which allows an application to describe parameters in SQL statements and in stored procedure calls. To use this option, Prepare Method must be set to 0 or 1, and the SQL statement must not include long parameters. This option should be selected when using Microsoft Remote Data Objects (RDO) to access data.

Application Using Threads: Select this check box to ensure that the driver works with multi-threaded applications. You can clear this check box when using the driver with single-threaded applications. Turning off this setting avoids additional processing required for ODBC thread-safety standards.

Tightly Coupled Distributed Transactions: Select this check box to use tightly coupled distributed transactions when connected to a Sybase ASE version 12 database and to ensure that multiple connections within the same distributed transaction do not obey each other's locks.

When this check box is not selected, the overall performance of the driver is better, but multiple connections within the same distributed transaction may hang each other because the connections do not obey each other's locks.

This option is valid only when the driver is enlisted in a distributed transaction and when it is connected to a Sybase ASE version 12 database. Otherwise, this option is ignored.

Translate Button

Displays the Select Translator dialog box, where you can translate your data from one character set to another. Choose the OEM to ANSI translator to translate your data from the IBM PC character set to the ANSI character set.

ODBC Sybase ASE Driver Setup Dialog : Connection Tab

Use the Connection tab on the ODBC Sybase ASE Driver Setup dialog box to specify optional settings when you create new Sybase ASE data sources or configure existing data sources.

Database List: Type the databases that appear in the logon dialog box. Separate the names with commas.

Default Logon ID: Type the default logon ID used to connect to your Sybase database. This ID is case-sensitive. A logon ID is required only if security is enabled for the database you are connecting to. Your ODBC application may override this value or you can override this value in the logon dialog box or connection string.

Workstation ID: Type the workstation ID used by the client.

Application Name: Type the name used by Sybase to identify your application.

Charset: Type the name of a character set. This character set must be installed on the Sybase server. The default is the setting on the Sybase server. For this driver to support Unicode, this attribute must be set to UTF-8. Refer to the Sybase server documentation for a list of valid character set names.

Language: Type the national language. This language must be installed on the Sybase server. The default is English.

ODBC Sybase ASE Driver Setup Dialog : Performance Tab

Use the Performance tab on the ODBC Sybase ASE Driver Setup dialog box to specify optional settings when you create new Sybase ASE data sources or configure existing data sources.

Select Method: Select a value of 0 or 1 that determines whether database cursors are used for Select statements. When set to 0, the default, database cursors are used; when set to 1, Select statements are run directly without using database cursors. A setting of 1 limits the data source to one active statement.

Prepare Method: Select a value of 0, 1, 2, or 3 that determines whether stored procedures are created on the server for calls to SQLPrepare.

When set to 0, stored procedures are created for every call to SQLPrepare. This setting can result in decreased performance when processing statements that do not contain parameters.

When set to 1 (the initial default), the driver creates stored procedures only if the statement contains parameters. Otherwise, the statement is cached and run directly at the time of SQLExecute.

When set to 2, stored procedures are never created. The driver caches the statement, executes it directly at the time of SQLExecute, and reports any syntax or similar errors at the time of SQLExecute.

When set to 3, stored procedures are never created. This is identical to value 2 except that any syntax or similar errors are returned at the time of SQLPrepare instead of SQLExecute. Use this setting only if you must have syntax errors reported at the time of SQLPrepare.

Fetch Array Size: Type the number of rows the driver retrieves when fetching from the server. This is not the number of rows given to the user. The default is 50 rows.

Packet Size: Type a value of -1, 0, or x that determines the number of bytes per network packet transferred from the database server to the client. The correct setting of this attribute can improve performance.

When set to -1, the driver computes the maximum allowable packet size on the first connect to the data source and saves the value in the system information.

When set to 0, the default, the driver uses the default packet size as specified in the Sybase server configuration.

When set to x, an integer from 1 to 1024, the driver uses a packet size represented by x times 512 bytes. For example, "6" means to set the packet size to 6 * 512 bytes (3072 bytes).

To take advantage of this connection attribute, you must configure the Sybase server for a maximum network packet size greater than or equal to the value you specified for PacketSize. For example:

sp_configure "maximum network packet size", 5120

reconfigure

Restart Sybase Server

NOTE: The ODBC specification identifies a connect option, SQL_PACKET_SIZE, that offers this same functionality. To avoid conflicts with applications that may set both the connection string attribute and the ODBC connect option, they have been defined as mutually exclusive. If PacketSize is specified, you will receive a message "Driver Not Capable" if you attempt to call SQL_PACKET_SIZE. If you do not set PacketSize, then application calls to SQL_PACKET_SIZE are accepted by the driver.

Connection Cache Size: Type a value that determines the number of connections that the connection cache can hold. The default Connection Cache setting is 1. To set the connection cache, you must set the Select Method option

to 1 - Direct. Increasing the connection cache may increase performance of some applications but requires additional database resources.

Configuring Connection to MySQL using MyODBC

MySQL ODBC driver prompts for more information about connection parameters in the **MySQL ODBC Driver - DSN Configuration** dialog box upon data source creation

- **DSN Information**:
  - **Data Source Name** - a name of the configured data source.
  - **Description** - a description of the configured data source.
- **MySQL Connection Parameters**:
  - **Host/Server Name** (or IP) - the hostname of the MySQL server.
  - **Database Name** - the name of the default database.
  - **User** - the user name used to connect to MySQL.
  - **Password** - the password for the server user combination.
  - **Port** (if not 3306) - the TCP/IP port to use if server is not localhost. If the port number is not given, the default port (3306) is used
  - **SQL command on connect** - a command that will be executed when connecting to MySQL.

SQLWays Studito

This chapter contains information on how the SQLWays Studio helps you to manage script conversion.

Next pages show how to use SQLWays Studio for conversion:

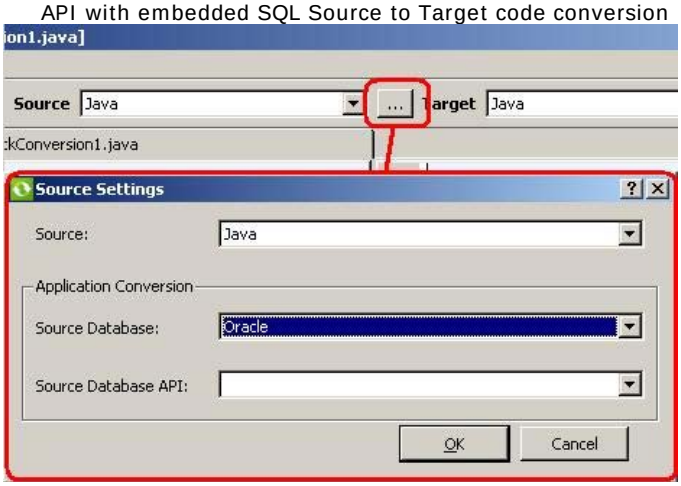
- “Choosing Source and Target”
  This page contains an instruction how to choose source and target for the convesrion.
- “Running Conversion”
  This page contains an instruction how to run script conversion.

Choosing Source and Target

Simple Source to Target code conversion

Before starting the conversion of scripts or application files, you need to specify the direction of the conversion - **Source** and **Target**, as specified in the screen shot above.

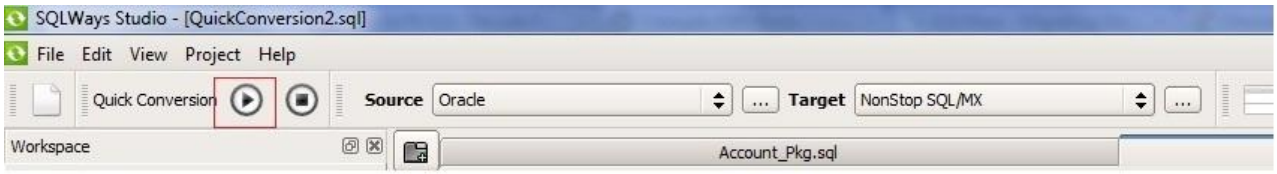
API with embedded SQL Source to Target code conversion



If you need to convert embedded SQL queries in the application, you have to specify Source and Target database information, so SQLWays will know what syntax to convert. To specify the database information, click "**...**" buttons. Please, see the screen-shot above.
Before the API code conversion you need to specify the **Source Database** from the **Application Conversion** group on the **Source Settings** page and perform the same actions for the **Target Database**.

Running Conversion

Simple code conversion



To Start conversion you should press the button “Run” as shown on the picture above.

SQLWays Troubleshooting Guide

There are different kinds of problems that you may encounter using SQLWays, and various techniques that you can use to deal with those problems. This part discusses these techniques.

- IBM DB2 Database
- Oracle Database
- MySQL Database
- Access Database
- Excel Files

IBM DB2 Database

This chapter presents troubleshooting tips for IBM DB2 databases. This chapter also describes how to deal with some frequently encountered problems with IBM DB2 utilities that are used by SQLWays to import databases to IBM DB2.

Importing to IBM DB2

This section has answers to the most frequently asked questions about importing data and database objects to IBM DB2.

- SQL0286N A default table space could not be found with a pagesize of at least "<pagesize>" that authorization ID "<user-name>" is authorized to use

SQL0286N A default table space could not be found with a pagesize of at least "<pagesize>" that authorization ID "<user-name>" is authorized to use

Software

IBM DB2 Database

Symptoms

When importing data to a DB2 database, the SQLSTATE0286N message is returned.

Actions

SQLSTATE0286N means that you are creating a table with the row length exceeding 4K. In order to create such tables, you have to create a tablespace with sufficient page size (at least "<pagesize>") in the database.

Exporting from IBM DB2

This section has answers to the most frequently asked questions about exporting data and database objects from IBM DB2.

- SQLSTATE 01517 - A character that could not be converted was replaced with a substitute character

SQLSTATE 01517 – A character that could not be converted was replaced with a substitute character

Software

IBM DB2 Database

Symptoms

- When exporting data from a DB2 database, the SQLSTATE 01517 message returned.

Actions

SQLSTATE 01517 means that you are using a database containing native language data (German or French e.g) and you have not configured the IBM DB2 client software to work with native language databases.

Set the appropriate value for the DB2CODEPAGE registry variable on the client workstation. Please refer to the IBM DB2 Documentation for the codepage values.

Oracle Database

This chapter presents troubleshooting tips for Oracle databases. This chapter also describes how to deal with some frequently encountered problems with the SQL Loader and SQL Plus utilities. These tools are used by SQLWays to import databases to Oracle.

Oracle SQL Loader does not terminate

Software

Windows 2000, Oracle client 8.0.5 (8.0.x).

Symptoms

- When data is imported SQLLoader does not return to the command line, so the user have to press Ctrl+C before he/she can start the next job.
- When SQL Loader is called from other applications, its window stays open.

Action

To solve the problem you need to upgrade your client to Oracle 8i (8.1.x). Oracle 8.0.x is not certified for Windows 2000 and unexpected things can happen. This might be one of them.

DROP TABLE Errors, ORA-02449: unique/primary keys in table referenced by foreign keys

When SQLWays executes the DROP TABLE statements before creating tables, Oracle may return the error ORA-02449. This means that the table cannot be dropped because it is referenced by a FOREIGN KEY constraint.

To drop tables referenced by child tables, you can the CASCADE CONSTRAINTS option in the DROP TABLE statement. For more information, see Dropping Tables.

To generate the CASCADE CONSTRAINTS option for Oracle

- SQLWays Wizard - DDL Options Page
- SQLWays command line - [DDL] subsection

SQL*Loader-350: Syntax error - found "TIMESTAMP"

Software

Oracle 8i

Symptoms

When data is imported SQL Loader generates an error indicating incorrect syntax near TIMESTAMP

Action

The TIMESTAMP data type is available since Oracle 9i. If you are migrating to Oracle 8i, please specify this version in SQLWays and it will use the DATE data type instead of TIMESTAMP.

MySQL Database

This chapter presents troubleshooting tips for MySQL databases. This chapter also describes how to deal with some frequently encountered problems with MySQL utilities that are used by SQLWays to import databases to MySQL.

- Importing data to MySQL 4.0.x using the LOCAL DATA INFILE command (LOCAL keyword - The used command is not allowed with this MySQL version)

Importing data to MySQL 4.0.x using the LOCAL DATA INFILE command (LOCAL keyword - The used command is not allowed with this MySQL version)

There are some specific issues regarding using the LOCAL keyword in LOAD DATA INFILE command in MySQL 4.0.x. The LOCAL keyword allows loading data from the client, otherwise the data must be located on the server.

In MySQL 4.0.x LOCAL will only work if MySQL server was started with --local-infile=1.

In the case that LOAD DATA LOCAL INFILE is disabled in the server or the client, you will get the error message (1148): "The used command is not allowed with this MySQL version".

Access Database

This chapter presents troubleshooting tips for Access databases. This chapter also describes how to deal with some frequently encountered problems with Access utilities that are used by SQLWays to import databases to Access.

Export from Access

When exporting from Access the following errors appear:

"Record(s) cannot be read; no read permission on 'MSysRelationships"

This error appears when SQLWays reads relationship from Access. SQLWays reads them from the 'MSysRelationships system table. To allow read this system table, open Access -> Tools -> Options -> View->Show and select System Objects.

After that restart export in SQLWays, and SQLWays will be able to export relationships.

Excel Files

This chapter presents troubleshooting tips for exporting data from Excel files.

- Syntax for specifying Excel table names

Syntax for specifying Excel table names

An Excel file contains one or more sheets. You can refer these sheets as tables using an Excel ODBC driver. But you have to use the Excel syntax to refer to Excel tables.

For example, an Excel file contains the Contacts sheet. To refer this table using the Excel ODBC driver, add a $ character to the table name. So you have to specify the table as *Contacts$* in queries.

In cases when you specify *Contacts$* and the Excel ODBC driver returns "Syntax error in from clause", enclose the table name in brackets. So you have to specify the table as *[Contacts$]* in queries.

Frequently Asked Questions

This FAQ contains answers to the most frequently asked questions about using SQLWays. The FAQ is regularly updated basing on the new questions recieved by the technical support.

- FAQ: Export Database Schema (DDL) only
- FAQ: Export Data only

If you do not find an answer to your question here, please send a message to the technical support at support@ispirer.com.

FAQ: Export Database Schema (DDL) only

This section contains the most frequently asked questions about how to export a source database schema only by setting DDL (Data Definition Language) options for Global Data Type Mapping, Column Name Mapping, SQL Scripts and Tables.

SQLWays creates SQL scripts when it exports the definitions of the source database objects. These scripts contain DDL statements that are used to create database objects in the target database.

- How to Omit Schema Names?

How to Omit Schema Names?

Q: How to omit schema names?

A: • SQLWays Wizard

Go over the wizard **Set DDL and Data Options** page**/ DLL Options/ SQL Scripts** and select *Yes* in the **Omit schema names in SQL scripts** box to omit schema (owner) names in SQL scripts for DDL statements. This allows you to use the default schema (owner) name for the user.

FAQ: Export Data only

This section contains the most frequently asked questions about how to export data only to the target database by setting general export options, modifying data export format and specifying how data is represented in output files.

- How to Change a Decimal Point Character?
- How to Change a Line Delimiter?
- What Export File Formats are Supported by SQLWays?

How to Change a Decimal Point Character?

Q: How to change a decimal point character?

A: SQLWays Wizard

Go over the wizard **Set DDL and Data Options** page**/ Data Options/ Formats** and change the character in the **Decimal point** box.

- ISQLWays **Command Line**.

Set a **DecPoint** value in the command line. Use the following **syntax**: /DECPT=DecPoint.
A **decimal_point=** value can be changed in the **sqlways.ini** initialization file in the **[Data] subsection**.

For example, 0x2c decimal point character specifies a comma.

How to Change a Line Delimiter?

Q: How to change a line delimiter?

A: •Line delimeter used in the text files may be specified and changed either in the SQLWays wizard, in the command line or in the initialization file.

- In the case of changing the value in the wizard, you need to go over the wizard **Set DDL and Data Options** page**/ Data Options/ Files** and change the character in the **Line delimeter** box.
- In the case of changing a value in the command line, open a command line window, and change a **LineDelimiter** value of the following **syntax**: /LDEL=LineDelimiter.
- In the case of changing a value in the initialization file, open Ispirer\SQLWays 3.7**sqlways.ini** file and change a **line_delimiter=** value of the **Data conversion options**.

What Export File Formats are Supported by SQLWays?

SQLWays exports data from the source database to the ASCII text files (export files). Depending on the selected target database, the most suitable export file formats for data transferring are:

Column-delimited (**CSV**) output format, Fixed length output format (**FIX**), TAB-delimited output format (**TAB**), SQL INSERT statements (**INS**), XML output format (**XML**) or Btrieve ASCII output format (**BTR**).

If the target database is **Oracle**, **IBM DB2** or not specified, by default, the output format of the text file is the **CSV** format.

If the target database is **Microsoft SQL Server**, **Sybase** or **MySQL**, by default, the output format of the text file is the **TAB** format.

If the target database is **Pervasive**, by default, the output format of the text file is **BTR**.

Ispirer Systems Resources and Contacts

A Guide to Ispirer Systems Customer Information Resources

Ispirer Systems Software make every effort to ensure that your experience with Ispirer Systems is successful. This title implies the resources and information available to you as a valued customer of Ispirer Systems.

The following variety of resources can help you get answers to your questions, troubleshoot problems, and interact with the Ispirer Systems team as well as with other customers:

- Online documentation
- Technical Support
- How to Order

Online Documentation

The latest versions of Ispirer Systems product manuals in HTML, and PDF format are available for download from the Ispirer Systems web site. These titles include:

- Product Overview
- SQLWays Documentation

http://www.ispirer.com/wiki/

Technical Support

Technical Support is available via e-mail at support@ispirer.com from Monday through Friday.

Our support staff cooperate closely with the development team and have the opportunity to test any issues on various database systems of our Test Center.

We provide free technical support to all users and do our best to reply as soon as possible.

Customers with Enhanced Support receive priority handling of all incoming requests.

How To Order

In addition to regular four support types listed below, Ispirer offers fixed price migration projects and services. Please contact us to get a FREE quote.

Support Types

- Standard (database migration only)
- Professional (database migration only)
- Enterprise (database migration only)
- Corporate (both database & application conversions)

For Commercial Use by system integrators, consultants, VARs, ISVs, resellers, developers, service providers or IT professionals who provide solutions based on the software to third parties, a license is required for each project in which the software applied (discounts provided for partners).

Customers receive free upgrades for one year. For more information, see SQLWays License Agreement

Accepted payment options: credit card, purchase order, check and wire transfer.

Please visit our site at http://www.ispirer.com/purchase/ for more information.

If you have any questions, feel free to contact us http://www.ispirer.com/contacts/

Legal Notices

SQLWays License Agreement

GENERAL TERMS

PLEASE READ THIS AGREEMENT CAREFULLY BEFORE USING SQLWAYS (THE "SOFTWARE"). ISPIRER SYSTEMS WILL LICENSE THE SOFTWARE TO YOU ONLY IF YOU FIRST ACCEPT THE TERMS OF THIS AGREEMENT. BY USING THE SOFTWARE YOU AGREE TO THESE TERMS. IF YOU DO NOT AGREE TO THE TERMS OF THIS AGREEMENT, YOU MUST REMOVE THE SOFTWARE FROM YOUR STORAGE DEVICES AND CEASE TO USE THE SOFTWARE.

The parties to this Agreement are the Ispirer Systems Ltd and the User. The "User" is the person or organization that orders and will pay the fee for the use of one or more products offered by Ispirer Systems Ltd. The Software is owned by Ispirer Systems Ltd, and is copyrighted and licensed, not sold. The term "Software" means the original Software and all whole or partial copies of it. The Software consists of machine-readable instructions, its components, data, visual content (such as images, text, or pictures), and related licensed materials.

DEMONSTRATION LICENSE

This is not free Software. The User is hereby licensed by Ispirer Systems Ltd to use one copy of the Software, on one (1) computer or workstation, for demonstration purposes without charge. To get a Demonstration license key The User should contact Ispirer Systems. If the User wishes to use this Software after the demonstration period they are required to contact Ispirer Systems.

The Demonstration version converts all tables, stored procedures and other database objects and exports all rows in each table and converts all lines in each stored procedure, trigger or function.

Unregistered use of the Software after the demonstration period is in violation of U.S. and international copyright laws.

LICENSE - USE OF THE SOFTWARE

Ispirer Systems Ltd grants the User a nonexclusive, non-transferrable license to use the Software.

SQLWays license copies may be used by multiple people on one or more computers working in different geographical locations on the migration project(s) specified in the license provided by Ispirer Systems Ltd. to the User.

LICENSE LIMITATIONS

The User is expected to use SQLWays in accordance with specific license limitations that restrict the total number of particular objects - tables, functions, stored procedures and triggers - available in each of the databases with which SQLWays is used (e.g. if one database has 100 objects and the other 300, you are to choose the 300-object license and so forth).

The number of converted indexes, constraints, views and other objects is not limited as well as the number of table rows and SQL code lines. Neither do any of the licenses stipulate any limitations to the number and type of databases.

TECHNICAL SUPPORT

Technical support is included in the cost of license and implies technical assistance, prompt answers to any product-related questions (provided within 12-24 hours maximum) and expert advice on database migration issues.

MAINTENANCE PACKAGE

The user acquiring a license with the maintenance package is entitled to the following benefits:

- Product customization to your particular needs
- Fixes within a couple of business days (up to 2 weeks for complex issues)
- Priority handling of all incoming requests

NO WARRANTY

The Software is being delivered to the User "AS IS" and Ispirer Systems Ltd makes no warranty as to its use or performance.

ISPIRER SYSTEMS LTD DOES NOT AND CANNOT WARRANT THE PERFORMANCE OR RESULTS THE CUSTOMER MAY OBTAIN BY USING THE SOFTWARE. EXCEPT FOR ANY WARRANTY, CONDITION, REPRESENTATION OR TERM TO THE EXTENT TO WHICH THE SAME CANNOT OR MAY NOT BE EXCLUDED OR LIMITED BY LAW APPLICABLE TO THE CUSTOMER IN THE CUSTOMER'S JURISDICTION, ISPIRER SYSTEMS LTD MAKES NO WARRANTIES CONDITIONS, REPRESENTATIONS, OR TERMS (EXPRESS OR IMPLIED WHETHER BY STATUTE, COMMON LAW, CUSTOM, USAGE OR OTHERWISE) AS TO ANY MATTER INCLUDING WITHOUT LIMITATION OF THIRD PARTY RIGHTS, MERCHANTABILITY, INTEGRATION, SATISFACTORY QUALITY, OR FITNESS FOR ANY PARTICULAR PURPOSE.

LIMITATION OF LIABILITY

IN NO EVENT WILL ISPIRER SYSTEMS LTD BE LIABLE TO THE CUSTOMER FOR ANY DAMAGES, CLAIMS OR COSTS WHATSOEVER OR ANY CONSEQUENTIAL, INDIRECT, INCIDENTAL DAMAGES, OR ANY LOST PROFITS OR LOST SAVINGS, EVEN IF AN ISPIRER SYSTEMS LTD REPRESENTATIVE HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH LOSS, DAMAGES, CLAIMS OR COSTS OR FOR ANY CLAIM BY ANY THIRD PARTY. THE FOREGOING LIMITATIONS AND EXCLUSIONS APPLY TO THE EXTENT PERMITTED BY APPLICABLE LAW IN THE CUSTOMER'S JURISDICTION. ISPIRER SYSTEMS LTD' AGGREGATE LIABILITY UNDER OR IN CONNECTION WITH THIS AGREEMENT SHALL BE LIMITED TO THE AMOUNT PAID FOR THE SOFTWARE, IF ANY.

APPLICABLE LAW AND SETTLEMENT OF DISPUTES

This agreement shall be governed by law of state of New York as well as by relevant international intellectual property law treaties. Any controversy or claim arising out of or relating to this licence agreement, or the breach thereof, shall be settled by arbitration administered by the American Arbitration Association under its Commercial Arbitration Rules, and judgment on the award rendered by the arbitrator(s) may be entered in any court having jurisdiction thereof. If the AAA is selected as the administrating agency, the claim is filed together with a filing fee based upon the amount of the claim.

###

Trademarks