SOLID *Embedded Engine* ™

Administrator Guide

June, 2000
Version 3.51

Solid Information Technology Ltd.
www.solidtech.com
sales@solidtech.com;techsupp@solidtech.com

Document number SSAG-3.51-0600
Date: June 26, 2000

Contents

3 Using SOLID SQL for Data Management

4 Using SOLID Data Management Tools

5 Managing Network Connections

6 Configuring SOLID *Embedded Engine*

7 Performance Tuning

8 Diagnostics and Troubleshooting

A Error Codes

B Configuration Parameters

C Data Types

D SOLID SQL Syntax

Welcome

SOLID Embedded Engine™ is a data management product for today's smart networks.

SOLID *Embedded Engine* provides broad operating system support including popular Windows, Unix, and real time operating system. It provides the features you would expect to find in any industrial-strength database server—multithread architecture, stored procedures, row level transaction management—but it delivers them with the special needs of today's smart networks.

About This Guide

This **SOLID Administrator Guide** is designed to make the administration of SOLID *Embedded Engine* smoother. This guide provides quick instructions on basic administration and maintenance, tools and utilities, and also provides reference information.

Organization

This manual contains the following chapters:

- *Chapter 1, Introducing SOLID Embedded Engine,* familiarizes you with the background and components of your SOLID data management system.
- *Chapter 2, Administering SOLID Embedded Engine,* covers the typical administration tasks such as starting, connecting to, and shutting down servers. It also explains how to perform routine maintenance such as creating backups and checkpoints, and using timed commands.
- *Chapter 3, Using SOLID SQL for Data Management,* gives readers the information they need to manage users, tables, indexes, and transactions.
- *Chapter 4, Using SOLID Data Management Tools,* describes the available utilities for handling database administration, specifying SQL commands and queries, and performing specific database operations, such as loading and unloading databases.

- *Chapter 5, Managing Network Connections*, describes how to connect to SOLID *Embedded Engine* using different communication protocols.
- *Chapter 6, Configuring SOLID Embedded Engine*, describes how to set SOLID *Embedded Engine* parameters for customization to meet your own environment, performance, and operation needs.
- *Chapter 7, Performance Tuning*, describes how to optimize SOLID *Embedded Engine* to improve performance.
- *Chapter 8, Diagnostics and Troubleshooting*, describes tools to use for observing performance and tracing problems.

Appendixes

The *Appendixes* give you detailed information about error messages, configuration parameters, and SOLID SQL functionality.

Glossary

The *Glossary of Terms* explains some of the terminology used in SOLID documentation.

Audience

This manual assumes general DBMS knowledge, and a familiarity with SQL.

Conventions

Product Name

In version 3.0, SOLID *Server* or SOLID Web Engine is now known as SOLID *Embedded Engine.* This guide may still make reference to SOLID *Server.* Throughout this guide, "SOLID *Server"* and "SOLID *Embedded Engine"* are used synonymously.

In this guide, "Solid server" or *"*Solid database*"* is used synonymously to refer to the server or database used in SOLID *Embedded Engine*.

Typographic

This manual uses the following typographic conventions.

Format	Used for
WIN.INI	Uppercase letters indicate filenames, SQL statements, macro names, and terms used at the operating-system command level.
`RETCODE SQLFetch(hdbc)`	This font is used for sample command lines and program code.
argument	Italicized words indicate information that the user or the application must provide, or word emphasis.
SQLTransact	Bold type indicates that syntax must be typed exactly as shown, including function names.
[ ]	Brackets indicate optional items; if in bold text, brackets must be included in the syntax.
|	A vertical bar separates two mutually exclusive choices in a syntax line.
{}	Braces delimit a set of mutually exclusive choices in a syntax line; if in bold text, braces must be included in the syntax.
...	An ellipsis indicates that arguments can be repeated several times.
. . .	A column of three dots indicates continuation of previous lines of code.

Other SOLID Documentation

SOLID *Embedded Engine* documentation is distributed in an electronic format (PDF, HTML, or Windows Help files).

Solid Online Services on our Web server offer the latest product and technical information free of charge. The service is located at:

`http://www.solidtech.com/`

Electronic Documentation

- **Read Me** contains installation instructions and additional information about the specific product version. This `readme.txt` file is typically copied onto your system when you install the software.
- **Release Notes** contains additional information about the specific product version. This `relnotes.txt` file is typically copied onto your system when you install the software.
- **SOLID *SynchroNet* Guide** introduces you to synchronization concepts and architecture and describes how to set up, use and administer SOLID *SynchroNet*.
- **SOLID Programmer Guide** describes the interfaces (APIs and drivers) available for accessing SOLID *Embedded Engine* and how to use them with a Solid server database.

1

Introducing SOLID *Embedded Engine*

This chapter introduces you to SOLID *Embedded Engine*™, a data management product for today's smart networks. It describes its benefits, features, main components, and high performance architecture.

About SOLID *Embedded Engine*

SOLID *Embedded Engine*, developed for this new era of distributed computing systems, provides what developers need, data storage features that meet the demands and requirements of their application environments.

Application developers can rely on SOLID *Embedded Engine's* wide range of data types, volumes, and processing features, which include, multithreaded parallel processing, symmetric multiprocessing (SMP), automatic roll-forward recovery, and stored procedures. Furthermore, SOLID *Embedded Engine's* portability and ease of deployment are ideal in today's internetworked environments. SOLID *Embedded Engine* provides broad operating systems support in such infrastructure platforms including popular Windows, Unix, and real time operating systems. It is fully Year 2000 Compliant.

SOLID *Embedded Engine* delivers performance within SQL-92, scalability, and high availability; yet it is lightweight, flexible, easy-to-use, and maintenance free with automatic operations.

SOLID *Embedded Engine* Features

SOLID *Embedded Engine* is a secure, reliable, and accommodating solution to your data storage needs. This section includes some of its unique benefits and features.

SOLID Bonsai Tree™

SOLID *Embedded Engine* features a small, but efficient index, known as The Bonsai Tree. This index tree resides in the main memory and maintains multiversion information. The Bonsai Tree performs concurrency control, detecting if any operations conflict with each other. This minimizes the effort needed for validating transactions. Active new data is separated from older, more stable data, which is transferred to a storage server as a highly-optimized batch insert, thus minimizing the hard disk load. The Bonsai Tree offers:

- Both optimistic and pessimistic concurrency control
- Fully serializable transactions free from phantom updates
- Multi-versioning that allows a consistent view of the database without extra locking
- Row-level locking is available if needed for pessimistic or mixed concurrency control methods. It can be turned on table by table, and a single transaction can use both pessimistic and optimistic concurrency control methods simultaneously.
- Declarative referential integrity ensuring the validity of references between tables.

Wide range of data type support

SOLID *Embedded Engine* supports binary compatible databases across all platforms. This support includes:

- Binary Large Objects (BLOBs), such as a picture, video clip, sound excerpt, or a formatted text object.
- Data stored in a variable-length format.
- Practically unlimited amount of tables, columns, keys, etc.
- Unicode support for double-byte character sets.

Stored procedures, event alerts, triggers, and sequencer objects

SOLID *Embedded Engine* provides these active database objects for reduced overhead:

- Stored procedure are used to execute part of the application logic in the server and for optimizing queries. A stored procedure can contain several SQL statements or a whole transaction for execution with a single call statement.

- Event alerts are used with stored procedures to signal an event in the database, thereby freeing the stored procedure from conducting its own database polling.
- Triggers are used to activate a stored procedure, which a Solid server automatically executes when a user attempts to change the data in a table. When a user modifies data within the table, the trigger that corresponds to the command is activated. Triggers serve several purposes, such as ensuring database changes do not compromise database integrity, reducing network traffic by transferring logic processing to the server, etc.
- Sequence objects generate number sequences for objects stored in databases. Sequences have an advantage over separate tables. They are specifically fine-tuned for fast execution and result in less overhead than normal update statements.

For details on these active database objects, read the **SOLID Programmer Guide**.

Easy Administration

With SOLID *Embedded Engine,* all administrative operations, including backups are performed automatically or at the administrator's request. Built-in timers are available for various administrative tasks. For example, administrator's can specify automated daily or weekly backups.

SOLID *Embedded Engine* also features online concurrent backup, and automatic and roll-forward recovery. Automatic recovery returns the database to the state it was in at the moment it encountered the error. To guarantee database integrity, all committed transactions are read from the transaction log.

SOLID *Embedded Engine* data management tools let you perform local and remote administration tasks, interactive SQL queries, and ASCII data handling, such as loading character data from character format data files, exporting character data to character format files, and writing data dictionary definitions of a database. For brief description of these tools, read *System Tools and Utilities* in this chapter.

SOLID *Embedded Engine* Components

SOLID *Embedded Engine,* the local data storage system for complex distributed network environments, contains the components described in the following sections.

Programming interfaces (ODBC and JDBC)

SOLID provides ODBC and JDBC drivers for programming access to SOLID data. SOLID *ODBC Driver* conforms to the Microsoft ODBC 3.51 API standard. SOLID *ODBC Driver* supported functions are accessed with SOLID ODBC API, a Call Level Interface (CLI) for Solid databases, which is compliant with ANSI X3H2 SQL CLI. The SOLID *JDBC Driver*

allows for application development with a Java tool that accesses the database using JDBC. For more details on programming interfaces, read the **SOLID Programmer Guide**.

Network Communications Layer

SOLID *Embedded Engine* runs on all major network types and supports all of the main communication protocols. Developers can create distributed applications for use in heterogeneous computing environments. For more details on network communication, read *Chapter 6,"Configuring SOLID Embedded Engine,"* in this guide.

SQL Parser and Optimizer

The SQL syntax used is based on the ANSI X3H3-1989 Level 2 standard and ANSI X3H3-1992 (SQL2) extensions. SOLID *Embedded Engine* contains an advanced cost-based optimizer, which ensures that even complex queries can be run efficiently. The SQL Optimizer automatically maintains information about table sizes, the number of rows in tables, the available indices, and the statistical distribution of the index values.

Read *Chapter 8,"Diagnostics and Troubleshooting,"* for more details on the SOLID SQL Optimizer.

Optimizer Hints

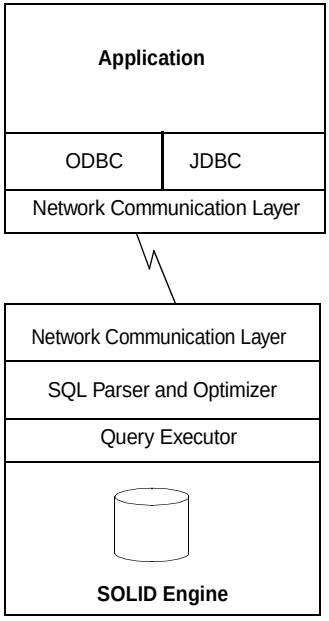
Optimizer hints (which is an extension of SQL) are directives specified through embedded pseudo comments within query statements. The Optimizer detects these directives or *hints* and bases its query execution plan accordingly. Optimizer hints allow applications to be optimized under various conditions to the data, query type, and the database. They not only provide solutions to performance problems occasionally encountered with queries, but shift control of response times from the system to the user.

Read *"Using Optimizer Hints"* on page 7-8 for more details on optimizer hints.

Engine

The SOLID engine is the core of the SOLID *Embedded Engine* product. It processes the data requests submitted via SOLID SQL. The engine stores data and retrieves it from the database.

Figure 1–1 SOLID Embedded Engine Components



System Tools and Utilities

SOLID *Embedded Engine* also includes the following tools for data management and administration:

SOLID *DBConsole*

SOLID *DBConsole* is an easy-to-use graphical user interface for administering and monitoring SOLID data management engines and executing SQL queries and commands. With SOLID *DBConsole*, you can:

- execute SQL commands
- administer all database servers in a network from a single workstation
- generate backups either on-line or as a timed command
- obtain server status information
- use either the interactive or batch mode operation
- have multiple active connections to various servers

- save or print query results

SOLID *Remote Contro*l (teletype) and SOLID *SQL Editor* (teletype) are also available to manage databases from the command line.

Tools for handling ASCII data

SOLID *Embedded Engine* provides the following tools for handling ASCII data:

- SOLID *Speedloader* (SOLLOAD) loads data from external ASCII files into a SOLID database. It is capable of inserting character data from character format. SOLID *Speed-Loader* bypasses the SQL parser and uses direct writes to the database file with loading, which allows for fast loading speed.
- SOLID *Export* (SOLEXP) writes from a SOLID database to character format files. It is capable of writing control files used by SOLID *SpeedLoader* to perform data load operations.
- SOLID *Data Dictionary* (SOLDD) writes the data dictionary of a database. This tool produces a SQL script that contains data definition statements describing the structure of the database.

Read *Chapter 4,"Using SOLID Data Management Tools,"* for details on system tools and utilities.

SOLID *Embedded Engine* High Performance Architecture

What differentiates SOLID *Embedded Engine* from other data management products is its high performance architecture. This section provides conceptual information, which can give you an understanding in configuring SOLID *Embedded Engine* to meet the needs of your own applications and platforms. It takes a behind the scenes look at the following processes:

- Bonsai Tree Multiversioning and Concurrency Control
- Dynamic SQL Optimization
- Network Services
- Multithread processing

SOLID Bonsai Tree Multiversioning and Concurrency Control

The Bonsai Tree is a small active index that efficiently stores new data (deletes, inserts, updates) in central memory, while maintaining multiversion information. Multiple versions of a row (old and new) can co-exist in the Bonsai Tree. Both the old and new data are used for concurrency control and for ensuring consistent read levels for all transactions without

any locking overhead. With the Bonsai Tree, the effort needed to validate transactions is significantly reduced.

When a transaction is started, it is given a transaction start number (TSN). The TSN is used as the read level of the transaction; all key values inserted later into the database from other connections are not visible to searches. This offers consistent index read levels that appear as if the read operation was performed atomically at the time the transaction was started. This guarantees read operations are presented with consistent view of the data without the need for locks.

Later the new committed data is merged to another indexing system, known as the storage server, and removed from the Bonsai Tree. The presorted key values are merged as a background operation concurrently with normal database operations. This offers significant I/O optimization and load balancing. During the merge, the deleted key values are physically removed.

Storage Server

The storage server uses a B-tree variation to store all permanent indices in the database file. It stores both secondary keys and the primary keys. Data rows are stored as the primary key values actually containing all the columns of the rows. There is no separate storage method for data rows, except for BLOBs and other long column values.

Indices are separated from each other by a system-defined index-identification inserted in front of every key value. This mechanism divides the index tree into several logical index subtrees, where the key values of one index are clustered close to each other. For details on data clustering, read *"Data clustering"* on page 3-10.

Index Compression

Two methods are used to store key values in the Bonsai Tree and storage server. First, only the information that differentiates the key value from the previous key value is saved. The key values are said to be prefix-compressed. Second, in the higher levels of the index tree, the key value borders are truncated from the end; that is, they are suffix-compressed.

SOLID SQL Optimizer

The SOLID SQL Optimizer, a cost-based optimizer, ensures that the execution of SQL statements is done efficiently. It uses the same techniques as a rules-based optimizer, relying on a preprogrammed set of rules in determining the shortest path to the results. For example, the SQL Optimizer considers whether or not an index exists, if it's unique, and over single or composite table columns. However, unlike a rule-based optimizer, its cost-based feature can adapt to the actual contents of the database; for example, the number of rows and the value distribution of individual columns.

SOLID *Embedded Engine* maintains the statistical information about the actual data automatically, ensuring optimal performance. Even when the amount and content of data changes, the optimizer can still determine the most effective route to the data.

Query Processing

Query processing is performed in small steps to ensure that one time-consuming operation does not block another application's request. A query is processed in a sequence containing the following phases.

Syntax analysis

A SQL query is analyzed and either a parse tree for the syntax or a syntax error is produced. When a statement is parsed, the information necessary for its execution is loaded into the statement cache. Statements are executed repeatedly without re-optimization, as long as its execution information remains in the statement cache

Creating the execution graph

The execution graph, with the following features, is created from the query parse tree.

- Complex statements are written to a uniform and more simple form
- If better performance will be realized, OR criteria is converted to UNION clauses
- Intelligent join constraint transfer is performed to produce intermediate join results that reduce the join process execution time.

Read "*The EXPLAIN PLAN Statement*" on page 8-2 for details on each operation or *unit* in the execution plan

Processing the execution graph

Processing of the execution graph is performed in three consecutive phases:

- Type-evaluation phase

  The data types of the columns of the result set are derived from the underlying table and view definitions

- Estimate-evaluation phase

  The cost of retrieving first rows and also entire result sets is evaluated, and an appropriate search strategy is selected dynamically based on the bound parameter values.

  The SQL Optimizer bases cost estimates on automatically maintained information of key value distribution, table sizes, and other dynamic statistical data. No manual updates to the index histograms or any other estimation information is required.

- Row-retrieval phase

  The result rows of the query are retrieved and returned to the client application

SOLID Network Services

SOLID Network Services are based on the remote procedure call (RPC) paradigm, which makes the communication interface simple to use. When a client sends a request to the server, it resembles calling a local function. The Network Services invisibly route the request and its parameters to the server, where the actual service function is called by the RPC Server. When the service function completes, the return parameters are sent back to the calling application.

In a distributed system, several applications may request a server to perform multiple operations concurrently. For maximum parallelism, SOLID Network Services use the operating system threads when available to offer a seamless multi-user support. On non-thread operating systems, the Network Services extensively use asynchronous operations for the best possible performance.

Communication Session Layer

SOLID communication protocol DLLs (or static libraries) offer a standard internal interface to each protocol. The protocol interface is BSD-socket like, containing methods, such as listen, accept, select, connect, read, write, disconnect, and control.

The lowest part of the communication session layer works as a wrapper that takes care of choosing the correct protocol DLL or library that relates with the given address information. After this point, the actual protocol information of the session is hidden.

The communication layer detects new communication messages (requests) from applications by selecting sessions that contain unread data. Where available, the process uses threads; elsewhere system-specific methods are used. The sessions that contain requests are written to a message queue (FIFO), where the RPC layer can find them in the order they arrived. If SOLID *Embedded Engine* is listening to many protocols simultaneously, the requests arriving through different protocols are all written to the same message queue.

RPC Session Layer

The RPC session layer contains services for typed parameter passing. Both application and server use these services for reading and writing all kinds of data, from standard C-types, up to the most complex internal data types.

For a server, the RPC session layer offers utilities for declaring the RPC service set that the server recognizes. The services are implemented as callback functions. The RPC level iden-

tifies the requested service, and the RPC server level is able to call the correct callback routine when requested.

For a client, the RPC session level also allows many simultaneously pending RPC requests. A client can in a single operation wait for any pending request to complete.

Multithread Processing

SOLID *Embedded Engine's* mutlithread architecture provides an efficient way of sharing the processor within an application. A thread is a dispatchable piece of code that merely owns a stack, registers, and its priority. It shares everything else with all other active threads in a process. Creating a thread requires much less system overhead than creating a process, which consists of code, data, an other resources such as open files and open queues.

Threads are loaded into memory as part of the calling program; no disk access is therefore necessary when a thread is invoked by another thread. Threads can communicate using global variables, events, and semaphores.

If the operating system supports symmetric multi-threading between different processors, SOLID Embedded Engine automatically takes advantage of the multiple processors.

Types of Threads

The SOLID *Embedded Engine* threading system consists of general purpose threads and dedicated thread.

General Purpose Threads

General purpose threads execute tasks from the server's tasking system. They execute such tasks as serving user requests, making backups, making checkpoints, making timed commands, and index merging.

General purpose threads take a task from the tasking system, execute the task step to completion and switch to another task from the tasking system. The tasking system works in a round-robin fashion distributing the client operations evenly between different threads.

The number of worker threads can be set in the `solid.ini` configuration file.

Dedicated Threads

Dedicated threads are dedicated to a specific operation. The following dedicated threads may exist in the server:

- I/O manager thread

  This thread is used for intelligent disk I/O optimization and load balancing. All I/O requests go through the I/O manager which determines whether to pass I/O requests to

the cache or to schedule it among other I/O requests. I/O requests are ordered by their logical file address. The ordering optimizes the file I/O since the file addresses accessed on the disk are in close range, reducing the disk read head movement.

- Communication read threads

  Applications always connect to a listener session that is running in the selector thread. After the connection is established, a dedicated read thread can be created for each client.

- One communication select thread per protocol (known as the selector thread)

  There is usually one communication selector thread per protocol. Each running sector thread writes incoming requests into a common message queue.

- Communication server thread (also known as the RPC server main thread)

  This thread reads requests from the common message queue and serves applications by calling the requested service functions.

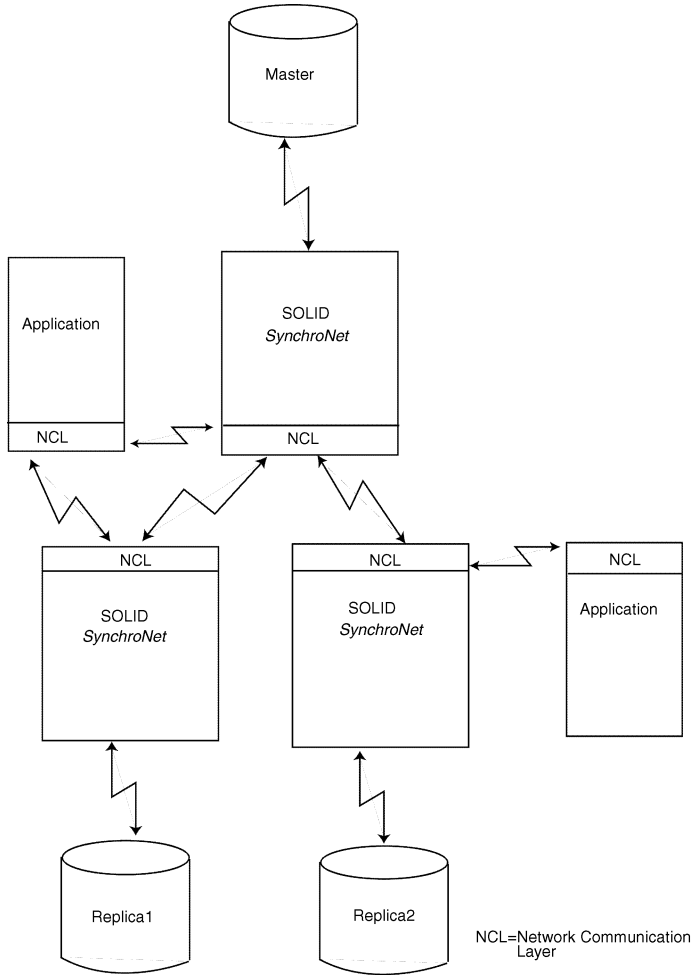
Distributed Data Management with SOLID *SynchroNet*

SOLID *SynchroNet* builds on the local data storage capabilities of SOLID *Embedded Engine*. It provides system-wide data sharing, which is particularly suited for applications in today's internetworked systems. With SOLID *SynchroNet's* asynchronous, bidirectional data synchronization, you can store data where it makes sense and distribute the data where it adds value.

SOLID *SynchroNet's* new approach to replication addresses the data reliablity shortcomings of traditional replication models. Its architecture builds data synchronization functionality inside a business application. Using SOLID *SynchroNet* SQL extensions and Intelligent Transactions*TM*, application developers, with minimal effort, can provide the logic to ensure data reliability within the context of their applications.

For details on SOLID *SynchroNet*, read the *SOLID SynchroNet* Guide.

Figure 1–2 System-wide sharing with SOLID SynchroNet



2

Administering SOLID *Embedded Engine*

This chapter covers the basics of SOLID *Embedded Engine* administration and maintenance. The administrative tasks covered in this chapter include such tasks as creating new databases, starting up and shutting down SOLID *Embedded Engine*, and connecting to servers. It also describes how to administer several servers on one computer.

For administrators, SOLID is practically maintenance free. The maintenance portion of this chapter covers way to automate tasks, such as backups and checkpoints, through timed commands. It also describes how to change database location, and perform SOLID *Embedded Engine* recovery basics, such as restoring databases, and specifying transaction logs.

What You Should Know

This section describes what you need to know about SOLID *Embedded Engine* before you begin administration and maintenance.

Installing SOLID *Embedded Engine*

If you have not yet installed SOLID *Embedded Engine*, refer to the **ReadMe** notice delivered with the software or included on the SOLID Website at:

`http://www.solidtech.com/`

The **ReadMe** contains a detailed description of the installation.

Using SOLID Databases 2.20 or Prior

Beginning with SOLID version 2.3 to the current version, the default collation sequence is set to the standard Latin-1. SOLID databases that were created with version 2.20 or prior do not match the Latin-1 collation sequence. To convert the data to Latin 1 in a version 2.20 database, you must export the database from its tables, extract data definitions, and load the

tables to the new database. Read *"Tools Sample: Reloading a Database"* on page 4-25 for details.

Special Roles for Database Administration

SOLID *Embedded Engine* has two special roles for administration and maintenance:

- SYS_ADMIN_ROLE

  This is the Database Administrator role and has privileges to all tables, indexes, and users, as well as the right to use SOLID *DBConsole* and SOLID *Remote Control* (teletype). This is also the role of the creator of the database.

- SYS_CONSOLE_ROLE

  This role has the right to use SOLID *Remote Control*, but has no other administration privileges.

You define these roles using the GRANT ROLE statement. For details, read *"Managing User Privileges and Roles"* on page 3-2.

Automated and Manual Administration

Embedded Engine is designed for continuous, unattended operation and ease of deployment. It requires minimal maintenance. Administrative operations, including backups can be performed programmatically using SQL extensions, which can run automatically or at an administrator's request.

Sometimes it makes sense to administer systems manually. This chapter refers you to the tools and methods available for performing manual administration. You can issue administrative commands equivalent to SOLID SQL's own ADMIN COMMANDs in SOLID *DBConsole* or in SOLID *Remote Control* (teletype). See *"Administrative Commands"* on page 4-6 for a commands list

Note that with SOLID *DBConsole*'s Administrative window, you can perform most of the SQL ADMIN COMMAND tasks that you execute on the command line with easy-to-use dialog boxes. For a description of SOLID *DBConsole*, read *"SOLID DBConsole"* on page 4-2. SOLID *SQL Editor* (teletype) also lets you enter administrative commands using the full SQL ADMIN COMMANDs syntax . See *"ADMIN COMMAND"* on page D-1 for a commands list.

Starting SOLID *Embedded Engine*

When SOLID *Embedded Engine* is started, it checks if a database already exists in the SOLID directory, that is, the directory where you installed SOLID executables. If a database

file is found, SOLID *Embedded Engine* will automatically open that database. If not, which is the case when you start the server for the first time, a new database will be created.

Operating System	To Start the Server...
UNIX	Enter the command `solid` at the command prompt. When you start the server for the first time, enter the command `solid -f` at the command prompt to force the server to run in the foreground.
Novell Netware	Enter the command `load solid.nlm` at the command prompt.
Open VMS	Enter the command `run solid` at the command prompt.
Windows	Click the icon labeled `SOLID Embedded Engine` in the SOLID *Embedded Engine* program group.

Creating a New Database

If a database does not exist, SOLID *Embedded Engine* will at startup automatically create a new database. In the Windows environment, creating the database begins with a dialog prompting for the database administrator's username, password, and a name for the default database catalog. For details, read *"Managing Database Objects"* on page 3-14.

In other environments, if you do not have an existing database, the following message appears:

```
Database does not exist. Do you want to create a new database (y/n)?
```

Answer y(es), and SOLID *Embedded Engine* prompts you for the database administrator's username, password, and a name for the default database catalog.

The username and password are case insensitive. The username requires at least two characters; the password at least three. For both the username and password, the maximum number of characters is 80. Use lower case letters from a to z, upper case letters from A to Z and the underscore character '_', and numbers from 0 to 9.

Note

You must remember your username and password to be able to connect to SOLID *Embedded Engine*. There are no default usernames; the username you enter when creating the database is the only username available for connecting to the new database.

After accepting the database administrator's username and password, SOLID *Embedded Engine* creates the new database.

By default the database will be created as one file (`solid.db`) in the SOLID directory, where the current working directory is located. An empty database containing only the system tables and views uses approximately 450 KB of disk space. The time it takes to create the database depends on the hardware platform you are using.

After the database has been created, SOLID *Embedded Engine* starts listening to the network for client connection requests. In the Windows environment, a SOLID *Embedded Engine* icon appears, but in most environments SOLID *Embedded Engine* runs invisibly in the background as a daemon process.

Windows only If in the Windows environment you double-click the icon of a running server, nothing will happen. SOLID *Embedded Engine* is a background process that only reacts to messages from clients through the communication interface.

About SOLID Databases

This section describes SOLID database structure and ways you can specify different values when creating SOLID databases.

Setting Database Size and location

By default, SOLID databases set a block size for the database file as 8192 bytes. SOLID *Embedded Engine* uses a multiple of 2 KB. The minimum block size is 2 KB and the maximum is 32 KB. The maximum size of the database is 64 TB.

If you want SOLID *Embedded Engine* to create a database with a different block size, you have to set a new constant value before creating a new database. If you have an existing database, be sure to move the old database and log files.

To modify the constant value for the new database, go to the SOLID directory and add the following lines in the `solid.ini` file, providing the size in bytes:

```
[Indexfile]
Blocksize=size_in_bytes
```

After you save the file and start SOLID *Embedded Engine*, it creates a new database with the new constant values from the `solid.ini` file.

Similarly, you can also modify the The `FileSpec` parameter to define the following:

- location of the database file to change the default of `solid.db` in the SOLID directory
- maximum size (in bytes) the database can reach to change the default of value of 2147483647, which equals 2GB.

You can also use the FileSpec parameter to divide the database file into multiple files and onto multiple disks.

For details on configuration with the FileSpec parameter, read *"Managing Database Files and Caching (IndexFile section)"* on page 6-2.

Defining Database Objects

SOLID database objects include catalogs, schemas, tables, views, indexes, stored procedures, triggers, and sequences. By default, database object names are qualified with the object owner's user id and a system catalog name that you specify when creating a database for the first time or converting an old database to a new format. You can also specify that database objects be qualified by a schema name. For details, read *"Managing Database Objects"* on page 3-14.

SOLID *Embedded Engine* supports a practically unlimited number of tables, rows, and indexes. Character strings and binary data are stored in variable length format. This feature saves disk space because no extra data is stored in the database. It also makes programming easier on developers since the length of strings or binary fields do not have to be fixed. The maximum size for a single attribute is 2GB.

By configuring the MaxBlobExpression parameter, you can set the maximum size of LONG VARCHAR columns in KBs that are used in string functions. By default, the size is 65KB. When BLOBs (Binary Large Objects), such as objects, images, video, graphics, or binary fields, are larger than the configured limit, SOLID *Embedded Engine* automatically detects this and stores the objects to a a special file area that has optimized block sizes for large files. No administrative action is required.

Connecting to SOLID *Embedded Engine*

After starting SOLID *Embedded Engine*, you can test the configuration by connecting to the server from your workstation using SOLID *DBConsole* or the SOLID teletype tools, *SQL Editor* or *Remote Control*. Read *Chapter 4,"Using SOLID Data Management Tools"* for details on these utilities which are part of the SOLID Data Management tools.

Note

You need to have SYS_ADMIN_ROLE or SYS_CONSOLE_ROLE privilege to be able to connect to a server using SOLID *DBConsole*. For details on creating these roles, read *"Managing User Privileges and Roles"* on page 3-2.

To connect to SOLID *Embedded Engine*:

1. View the `solmsg.out` file in your database directory for valid network names that you can use to connect to SOLID *Embedded Engine*.

   The following messages indicate what names you can use.

   ```
   Listening of 'ShMem Solid' started.
   Listening of 'TCP/IP 1313' started.
   ```

2. Start one of the following applications and give the network name of the server as a command line parameter:

Tool	Command
SOLID *DBConsole*	java DBConsole **-D***databasename* **-U***url* **-u***userid* **-p***password* For example: `java DBConsole -Dsolid -Ujdbc:solid:// localhost:1313 -udba -pdba` Alternatively, you can start *DBConsole* without any command line option. You are then prompted for the database connection information.
SOLID *Remote Control* (teletype)	solcon "*networkname*" For example: `solcon "tcp hobbes 1313"` When prompted, enter the database administrator's user name and password.
SOLID *SQL Editor* (teletype)	solsql "*networkname*" For example: `solsql "tcp hobbes 1313"` When prompted, enter the database administrator's user name and password.

After a while you will see a message indicating that a connection to the server has been established.

Viewing the SOLID *Embedded Engine* Message Log

Ensure the database started without errors by checking the message log `solmsg.out`, located in the SOLID directory. You can view this file in SOLID *DBConsole*'s Messages page from the Administration window.

SOLID *Embedded Engine* maintains the following message log files:

- `The solmsg.out` log file contains normal informational events, such as connects, disconnects, checkpoints, backups, etc. If an internal error occurs, the error is written to the `solmsg.out` file.
- If the error is fatal, the `solerror.out` file contains more detail about the error.

For troubleshooting purposes, SOLID *Embedded Engine* can also produce optional trace files that contain information for diagnostics. Monitoring the trace files is not necessary for everyday operation of the server. The trace files are primarily needed for troubleshooting of exceptional events. Refer to *Chapter 8,"Diagnostics and Troubleshooting,"* for more details on SOLID diagnostics.

Monitoring SOLID *Embedded Engine*

The following sections describe the methods used for querying the status of a SOLID database.

Checking overall database status

You can select the **Status** option from the SOLID *DBConsole* Administration window. The status page displays the following information as shown below:

You can also issue the following command in SOLID *DBConsole* or SOLID *Remote Control* (teletype):

```
status
```

or in SOLID *SQL Editor* (teletype):

```
ADMIN COMMAND 'status';
```

The command provides the following statistics information:

```
RC TEXT
-- ----
0 SOLID Embedded Engine started at Thu May 27 16:11:20 1999
0 Current directory is D:\solid
0 Using configuration file D:\solid\solid.ini
0 Memory statistics:
0  1400 kilobytes
0 Transaction count statistics:
0      Commit Abort Rollback    Total Read-only  Trxbuf  Active Validate
0      27     0       18       45        45       0       1        0
0 Cache count statistics:
0     Hit rate      Find      Read     Write
0     93.5          445       29        0
0 Database statistics:
0     Index writes            0  After last merge      0
0     Log writes              0  After last cp         0
0     Active searches         0  Average               1
0     Database size       1232 kilobytes
0     Log size             274 kilobytes
0 User count statistics:
0     Current Maximum Total
0           1       1     2
```

Following is a description of the result set fields:

- Memory statistics show the amount of memory SOLID has allocated from the operating system. This number does not include the size of the executable itself.
- Transaction count statistics show the number of different transaction operations since startup.
- Cache count statistics show cache hit rate and number of cache operations since startup. Cache hit rate typically is above 95 per cent.
- Database statistics show a number of the most important database operations since startup. "Index writes after last merge" is an important figure here. It reveals the size of the multi-versioning storage tree of SOLID, known as the "Bonsai Tree." The smaller this value is, the better the server performance. A large value indicates that there is a long-running transaction active in the engine.
- User count statistics shows current and maximum number of concurrent users.

Obtaining Currently Connected Users

To obtain a list of currently connected users:

1. Select the **Status** option from the SOLID *DBConsole* Administration window or menu.
2. On the Status page, click the **Users** icon.

A Users dialog box displays each user's name, user id, type, machine id, and login time.

You can also obtain a listing of connected users by entering command `userlist` in SOLID *DBConsole* or SOLID *Remote Control* (teletype) or enter the following SOLID SQL syntax in SOLID *SQL Editor* (teletype):

```
ADMIN COMMAND 'userlist' ;
```

The command provides the following kind of result set:

```
RC TEXT
-- ----
0 User name:     User id: Type:  Machine id:      Login time:
0 DBA            1        SQL    Local           27.05 16:13:22
```

Throwing out a connected Embedded Engine user

To disconnect a single user from the server, you can:

- Select the **Status** option from the SOLID *DBConsole* Administration window or menu, click the **Users** icon, and drop a selected user from the Users dialog box.

  or

- Enter command `throwout` *`user_id`* in SOLID *DBConsole* or SOLID *Remote Control* (teletype) or enter the following SOLID SQL syntax in SOLID *SQL Editor* (teletype):

  ```
  ADMIN COMMAND 'throwout user_id'
  ```

Querying the status of the last backups

To obtain a status of the most recently run backup:, you can:

- Select the **Status** option from the SOLID *DBConsole* Administration window, and click the **Backup** icon to view the backup status on the Backup dialog box.

  or

- Enter command `status backup` in SOLID *DBConsole* or SOLID *Remote Control* (teletype) or enter the following SOLID SQL syntax in SOLID *SQL Editor* (teletype):

  ```
  ADMIN COMMAND 'status backup'
  ```

If the last backup is successful, the result set looks as follows:

```
RC TEXT
-- ----
0  SUCCESS
```

If the latest backup has failed, then the RC column returns an error code. Return code 14003 with text "ACTIVE" means that the backup is currently running.

Detailed DBMS monitoring and troubleshooting

Besides checking the SOLID *DBConsole* Status page, you can also take a snapshot that provides additional information on *Embedded Engine* performance. Enter the `perfmon` command in SOLID *DBConsole* or SOLID *Remote Control* (teletype) or enter the following SOLID SQL syntax in SOLID *SQL Editor* (teletype):

```
ADMIN COMMAND 'perfmon'
```

The command returns a result set where each column represents a snapshot of the performance information that reflects the most recent few minutes.

The first column shows average performance information from a period of 55 seconds. The "Total" column shows average information since *Embedded Engine* was started. Most numbers are events/second. Those numbers that cannot be expressed as events/second (for example, database size) are expressed as absolute values.

The output information is categorized as follows:

- File operations
- Cache operations
- RPC and communications operations
- SQL operations
- SA (table-level db-operations) operations
- Transaction operations
- Index write (that is, database file write) operations
- Miscellaneous operations

Producing a status report

To create a report about the current status of SOLID *Embedded Engine*, enter the command `report report_filename` in SOLID *DBConsole* or SOLID *Remote Control* (teletype) or enter the following SOLID SQL syntax in SOLID *SQL Editor* (teletype):

```
ADMIN COMMAND 'report report_filename'
```

This report is primarily meant for SOLID internal use only because it contains information that requires very detailed understanding about the internals of SOLID *Embedded Engine*. End users sometimes are requested to produce the report for troubleshooting purposes.

Shutting Down SOLID *Embedded Engine*

You can shut down SOLID *Embedded Engine* in these ways:

- Programmatically from an application such as SOLID *DBConsole*, SOLID *Remote Control*, or SOLID *SQL Editor* (teletype)*.

  To do this, perform the following steps:

  1. To prevent new connections to *Embedded Engine*, close the database(s) by entering the following command:

     ```
     close
     ```

  2. Exit all users of *Embedded Engine* by entering the following command:

     ```
     throwout all
     ```

  3. Stop *Embedded Engine* by entering the following command:

     ```
     shutdown
     ```

     * Note that when using SOLID *SQL Editor* (teletype) for steps 1-3, you enter the full SQL Syntax, ADMIN COMMAND '*command_name*' (for example, ADMIN COMMAND '`close`').

- Clicking the server icon and selecting **Close** from the menu appearing in the Windows environment.
- Remotely, using the command `'net stop'` through the Windows NT system services. Note that you may also start up SOLID *Embedded Engine* remotely, using the `'net start'` command.

All the shutdown mechanisms will start the same routine, which writes all buffered data to the database file, frees cache memory, and finally terminates the server program. Shutting down a server may take a while since the server must write all buffered data from main memory to the disk.

Performing Backup and Recovery

This section describes how to back up your databases and recover from system failure.

Making Backups

Backups are made to secure the information stored in your database files. If you have lost your database files because of a system failure, you can continue working with the backup database.

You can initiate a backup in the following ways:

- Automate the backup using a timed command that initiates the backup according to a pre-defined schedule. Read “Entering Timed Commands” in this chapter for details.
- Select the **Status** option from the SOLID *DBConsole* Administration window, and click the **Backup** icon to initiate the backup from the Backup dialog box.
- Issuing the following command in SOLID *DBConsole* or SOLID *Remote Control* (teletype):

  ```
  backup
  ```

- Issue the following command in SOLID *SQL Editor* (teletype):

  ```
  ADMIN COMMAND 'backup'
  ```

Note

Be sure you have enough disk space in the backup directory for your database and log files.

Viewing SOLID Messages in the Backup Directory

The system copies the SOLID messages file (solmsg.out) file to the backup directory (parameter `BackupCopySolmsgout` in the `General` section of `solid.ini` is set to `yes` by default). This provides a convenient way to view what operations were performed with a Solid server before performing a backup. This also allows the SOLID messages file to exist in the backup directory for viewing before restoring the database from a corresponding backup file.

Backup Procedure

SOLID *Embedded Engine* uses a multiversioning technique allowing backups to be made on-line. There is no need to close the database file or shut down the server. However, it is advisable that you automate backups to be run at non-busy hours. After completing the

backup, copy your backup files on tape using your backup software for protection against disk crashes.

Notes

1. You can query programmatically the status of the most recently started backup in SOLID *DBConsole* or SOLID *Remote Control* (teletype) by using the command `status backup`. To query the list of all completed backups and their success status, use the command `backuplist`. To use these same commands in SOLID *SQL Editor*, enter the SOLID SQL syntax (for example, `ADMIN COMMAND 'backuplist'`).

   You can also query backup status in SOLID *DBConsole* by selecting the **Status** option in the Administration window or menu and clicking the **Backup** icon. A backup status listing is displayed in a dialog box.

2. The backup directory you enter must be a valid path name in the server operating system. For example, if the server runs on a UNIX operating system, path separators must be slashes, not backslashes.

3. The time needed for making a backup is the time that passed between the messages `Backup started` and `Backup completed successfully`, which is written to the `solmsg.out` log files. These messages are displayed on the SOLID *DBConsole* Messages page.

Before starting the backup process, a checkpoint is created automatically. This guarantees that the state of a backup database is from the moment the backup process was started. The following files are then copied to the backup directory:

- database file(s)
- configuration file (`solid.ini`)
- log file(s) modified or created after the previous backup (parameter `BackupCopyLog` in the `General` section of `solid.ini` is set to `yes` by default)
- backup of SOLID messages file `solmsg.out` (parameter `BackupCopySolmsgout` in the `General` section of `solid.ini` is set to `yes` by default).

The unnecessary log files are deleted from the original directory after successful backup (parameter `BackupDeleteLog` in the `General` section of `solid.ini` is set to `yes` by default).

Correcting a Failed Backup

When SOLID *Embedded Engine* is performing a backup, the `ADMIN COMMAND 'status backup'` command returns the value 'ACTIVE'. Once the backup is completed, the command returns either 'OK' or 'FAILED'. You can also query this information using SOLID *DBConsole*.

If the backup failed, you can find the error message that describes the reason for the failure from the `solmsg.out` file in the database directory or in the SOLID *DBConsole* Messages page (accessed through the Administration window or menu). Correct the cause of the error and try again. The most common causes for failed backups are:

- the backup media is out of disk space
- the backup directory does not exist
- a database directory is defined as the backup directory

Restoring Backups

There are two alternative ways to restore a backup. You can either:

- Return to the state when the backup was created, or
- Revive a backup database to the current state by using log files to add data inserted or updated data after the backup was made.

To Return to the State when the Backup was Made

1. Shut down SOLID *Embedded Engine*, if it is running.
2. Delete all log files from the log file directory. The default log file names are `sol00001.log`, `sol00002.log`, etc.
3. Copy the database file(s) from the backup directory to the database file directory.
4. Start SOLID *Embedded Engine*.

This method will not perform any recovery because no log files exist.

To Revive a Backup Database to the Current State

1. Shut down SOLID *Embedded Engine*, if it is running.
2. Copy the database file(s) from the backup directory to the database file directory.
3. Copy the log files from the backup directory to the log file directory.
4. Start SOLID *Embedded Engine*.

SOLID *Embedded Engine* will automatically use the log files to perform a roll-forward recovery.

Recovering from Abnormal Shutdown

If the server was closed abnormally, that is, if it was not shut down using the procedures described earlier, SOLID *Embedded Engine* automatically uses the log files to perform a roll-forward recovery during the next start up. No administrative procedures are required to start the recovery.

The message `Starting roll-forward recovery` appears. After the recovery is completed, a message indicates how many transactions were recovered. If no transactions were made since the last checkpoint, this is indicated by the following message:

```
Recovery successfully completed
```

Transaction Logging

The SOLID *Embedded Engine* transaction log manager ensures that transaction results are written to permanent storage immediately at commit time. Transaction logging guarantees that no committed operations are lost in case of a system failure. When an operation is executed in the server, the operation is also saved to a transaction log file. The log file is used for recovery in case the server is shut down abnormally.

A backup operation copies the log and database files to the backup directory and deletes the log files from the database directory. You may change this default behavior by changing the values to "no" in the following parameters: `BackupCopyLog` and `BackupDeleteLog` in the `General` section of `solid.ini`.

Tip

For both security and performance reasons, it is a good idea to keep log files and database files on different physical disk devices. If one disk drive is damaged, you will lose either your database files or log files but not both.

Creating Checkpoints

Checkpoints are used to store a consistent state of the database onto the database file. Checkpoints are needed to provide a starting point for the roll-forward recovery after a system failure. In the roll-forward recovery, the database will start recovering transactions from the last

successful checkpoint. The longer it has been since the last checkpoint was created, the more operations are recovered from the log file(s).

To speed up recoveries, create checkpoints frequently; note, however, that the server performance is reduced during the creation of a checkpoint. Furthermore, the speed of checkpoint creation depends on the amount of database cache used; the more database cache is used, the longer the checkpoint creation will take. You need to consider these issues when deciding the frequency of checkpoints. See *Appendix B, "Configuration Parameters"* for a description of the use of `CacheSize` parameter.

SOLID *Embedded Engine* has an automatic checkpoint creation daemon, which creates a checkpoint after a certain number of writes to the log files. The default checkpoint interval is every 5000 log writes. You may change the value of the parameter `CheckpointInterval` in the `General` section of parameters. To learn how to change a parameter value, see *"Managing Parameters"* on page 6-7 in this guide.

Before and after a database operation, you may want to create a checkpoint manually. You can do this programmatically from your application with SQL command `ADMIN COMMAND 'makecp'`. You can also force a checkpoint using a timed command. Read the section "Entering Timed Commands" in this chapter for details.

Note

There can be only one checkpoint in the database at a time. When a new checkpoint is created successfully, the older checkpoint is automatically erased. If the server process is terminated in the middle of checkpoint creation, the previous checkpoint is used for recovery.

Closing a Database

You can close the database which means no new connections to the database are allowed. To do this, issue the following command in SOLID *DBConsole* or SOLID *Remote Control* (teletype):

```
close
```

or in SOLID *SQL Editor* (teletype):

```
ADMIN COMMAND 'status';
```

In some cases you may want to prevent users from connecting to the database. For example, when you are shutting down SOLID *Embedded Engine*, you need to prevent new users from connecting to it. After closing the database, only connections from SOLID *DBConsole* or

SOLID *Remote Control* (teletype) will be accepted. Closing the database does not affect existing user connections.

When the database is closed no new connections are accepted (clients will get SOLID Error Message 14506).

SOLID *DBConsole* provides a user interface for managing database connections. For details, For details, refer to *DBConsole* Online Help available by selecting **Help** on the menu bar.

Changing Database Location

Changing a database location in SOLID *Embedded Engine* is as easy as copying a file from one directory to another.

Note

To copy a database file, you need to shut down SOLID *Embedded Engine* to release the operating system file locks on the database file and log files.

To Change Database Location

1. Verify that SOLID *Embedded Engine* is not running.
2. Copy the database and log files to the target directory.
3. Copy the `solid.ini` file to the target directory. Check that the database file directory, log file directory, and backup directory are correctly defined in the configuration file `solid.ini`.
4. Start SOLID *Embedded Engine* using the target directory as the current working directory using the command line option `-c` *`directory_name`*.

Running Several Servers on One Computer

In some cases, you may want to run two or more databases on one computer. For example, you may need a configuration with a production database and a test database running on the same computer.

SOLID *Embedded Engine* is able to use one database per database server, but you can start several engines each using its own database file. To make these engines use different databases, either start the engine processes from the directories your databases are located in or give the locations of configuration files by using the command line option `-c`

directory_name to change the working directory. Remember to use different network names for each database.

Entering Timed Commands

SOLID *Embedded Engine* has a built-in timer, which allows you to automate your administrative tasks. You can use timed commands to execute system commands, to create backups, checkpoints, and database status reports, to open and close databases, and to disconnect users and shut down servers.

To Enter a Timed Command Manually

Edit the `At` parameter of the `[Srv]` section in the `solid.ini` file. The syntax is:

At string := *timed_command*[, *timed_command*]

timed_command := [*day*] *HH:MM* command argument

day := **sun | mon | tue | wed | thu | fri | sat**

If the day is not given, the command is executed daily. For details on valid commands, refer to the table at the end of this section.

Example:

```
[Srv]
At=20:30 makecp,21:00 backup,sun 23:00 shutdown
```

Note

The format used is HH:MM (24-hour format).

To Enter a Timed Command in SOLID *DBConsole*

Select the **Status** option from the SOLID *DBConsole* Administration window or menu, click the **Scheduler** icon, then click **New** to enter a timed command in the Scheduler dialog box.

In the Scheduler dialog box provide the command, day, time, and arguments in each of the applicable fields. For syntax details, refer to the previous section. Refer to the following section for a list of valid commands.

Arguments and Defaults for the Different Timed Commands

Command	Argument	Default
`backup`	backup directory	the default backup directory that is set in the configuration file
`throwout`	user name, all	no default, argument compulsory
`makecp`	no arguments	no default
`shutdown`	no arguments	no default
`report`	report file name	no default, argument compulsory
`system`	system command	no default
`open`	no argument	no default
`close`	no argument	no default

3

Using SOLID SQL for Data Management

You manage SOLID databases as well as its users and schema using SOLID SQL statements. This chapter describes the management tasks you perform with SOLID SQL. These tasks include managing roles and privileges, tables, indexes, transactions, catalogs, and schemas.

Using SOLID SQL Syntax

The SQL syntax is based on the ANSI X3H2-1989 level 2 standard including important ANSI X3H2-1992 (SQL2) extensions. User and role management services missing from previous standards are based on the ANSI SQL3 draft. Refer to *Appendix D, "SOLID SQL Syntax"* for a more formal definition of the syntax.

SQL statements must be terminated with a semicolon (;) *only* when using SOLID *SQL Editor* or SOLID *DBConsole*. Otherwise, terminating SQL statements with a semicolon leads to a syntax error.

You can use SOLID *DBConsole* (as well as SOLID *SQL Editor* and ODBC compliant tools) to execute SQL statements. To automate the tasks, you may want to save the SQL statements to a file. You can use these files for rerunning your SQL statements later or as a document of your users, tables, and indexes.

SOLID SQL Data Types

SOLID SQL supports data types specified in the SQL2 Standard Entry Level specifications, as well as important Intermediate Level enhancements. Refer to *Appendix C, "Data Types"* for a complete description of the supported data types.

You can also define some data types with the optional length, scale, and precision parameters. In that case, the default properties of the corresponding data type are not used.

SOLID SQL Extensions

SOLID SQL provides the extension ADMIN COMMAND '*command*[*command_args*]' to perform basic administrative tasks, such as backups, performance monitoring, and shutdown.

You can use SOLID *SQL Editor* (teletype) to execute the command options provided by ADMIN COMMAND. To access a short description of available ADMIN COMMANDs, execute `ADMIN COMMAND 'help'`. For a formal definition of the syntax of these statements, refer to *Appendix D, "SOLID SQL Syntax"* in this guide.

Note

ADMIN COMMAND tasks are also available as administrative commands in SOLID *DBConsole* and SOLID *Remote Control* (teletype). Read *Chapter 3,"Using SOLID SQL for Data Management,"* for details.

Managing User Privileges and Roles

You can use SOLID *DBConsole*, SOLID teletype tools, and many ODBC compliant SQL tools to modify user privileges. Users and roles are created and deleted using SQL statements or commands. A file consisting of several SQL statements is called a SQL script.

In the `\samples\procedures` directory, you will find a SQL script called `Sample.sql`, which gives an example of creating users and roles. You can run it using SOLID *DBConsole*. To create your own users and roles, you can make your own script describing your user environment.

User Privileges

When using SOLID databases in a multi-user environment, you may want to apply user privileges to hide certain tables from some users. For example, you may not want an employee to see the table in which employee salaries are listed, or you may not want other users to mess with your test tables.

You can apply five different kinds of user privileges. A user may be able to view, delete, insert, update or reference information in a table or view. Any combination of these privileges may also be applied. A user who has none of these privileges to a table is not able to use the table at all.

Note

Once user privileges are granted, they take effect when the user who is granted the privi-

leges logs on to the database. If the user is already logged on to the database when the privileges are granted, they take effect only if the user:

- accesses the table or object on which the privileges are set for the first time
-or disconnects and then reconnects to the database.

User Roles

Privileges can also be granted to an entity called a role. A role is a group of privileges that can be granted to users as one unit. You can create roles and assign users to certain roles.

Privileges can also be granted to an entity called a role. A role is a group of privileges that can be granted to users as one unit. You can create roles and assign users to certain roles.

Note

1. The same string cannot be used both as a user name and a role name.
2. Once a user role is granted, it takes effect when the user who is granted the role logs on to the database. If the user is already logged on to the database when the role is granted, the role takes effect when the user disconnects and then reconnects to the database.

The following user names and roles are reserved:

Reserved Names	Description
PUBLIC	This role grants privileges to all users. When user privileges to a certain table are granted to the role `PUBLIC`, all current and future users have the specified user privileges to this table. This role is granted automatically to all users.
SYS_ADMIN_ROLE	This is the default role for the database administrator. This role has administration privileges to all tables, indexes and users, as well as the right to use SOLID *DBconsole* and SOLID *Remote Control* (teletype). This is also the role of the creator of the database.
_SYSTEM	This is the schema name of all system tables and views.
SYS_CONSOLE_ROLE	This role has the right to use SOLID *DBConsole*, but does not have other administration privileges.

Examples of SQL Statements

Below are some examples of SQL commands for administering users, roles, and user privileges.

Creating Users

CREATE USER *username* IDENTIFIED BY *password*;

Only an administrator has the privilege to execute this statement. The following example creates a new user named `CALVIN` with the password `HOBBES`.

```
CREATE USER CALVIN IDENTIFIED BY HOBBES;
```

Deleting Users

DROP USER *username*;

Only an administrator has the privilege to execute this statement. The following example deletes the user named `CALVIN`.

```
DROP USER CALVIN;
```

Changing a Password

ALTER USER *username* IDENTIFIED BY *new password*;

The user *`username`* and the administrator have the privilege to execute this command. The following example changes `CALVIN's` password to `GUBBES`.

```
ALTER USER CALVIN IDENTIFIED BY GUBBES;
```

Creating Roles

CREATE ROLE *rolename*;

The following example creates a new user role named `GUEST_USERS`.

```
CREATE ROLE GUEST_USERS;
```

Deleting Roles

DROP ROLE *role_name*;

The following example deletes the user role named `GUEST_USERS`.

```
DROP ROLE GUEST_USERS;
```

Granting Privileges to a User or a Role

GRANT *user_privilege* ON *table_name* TO *username* or *role_name*;

The possible user privileges on tables are SELECT, INSERT, DELETE, UPDATE, REFERENCES and ALL. ALL provides a user or a role all five privileges mentioned above. A new user has no privileges until they are granted.

The following example grants `INSERT` and `DELETE` privileges on a table named `TEST_TABLE` to the `GUEST_USERS` role.

```
GRANT INSERT, DELETE ON TEST_TABLE TO GUEST_USERS;
```

The EXECUTE privilege provides a user the right to execute a stored procedure:

GRANT *user_privilege* ON *procedure_name* TO *username* or *role_name*;

The following example grants EXECUTE privilege on a stored procedure named SP_TEST to user CALVIN.

```
GRANT EXECUTE ON SP_TEST TO CALVIN;
```

Note

Newly granted or updated privileges for users do not always take effect immediately. For example, if an administrator grants privileges to a user on a specific table, the user only sees the effect of the privileges if the user:

- logs on to the database
- or is already logged on to the database and is now accessing the table for the first time.

In other words, if the user is logged on and has already accessed the table, the new privileges the administrator has granted on the table do not take effect until the user disconnects from the database and then reconnects.

Granting Privileges to a User by Giving the User a Role

GRANT *role_name* TO *username*;

The following example gives the user `CALVIN` the privileges that are defined for the `GUEST_USERS` role.

```
GRANT GUEST_USERS TO CALVIN;
```

Revoking Privileges from a User or a Role

REVOKE *user_privilege* ON *table_name* FROM *username* or *role_name*;

The following example revokes the `INSERT` privilege on the table named `TEST_TABLE` from the `GUEST_USERS` role.

```
REVOKE INSERT ON TEST_TABLE FROM GUEST_USERS;
```

Revoking Privileges by Revoking the Role of a User

REVOKE *role_name* FROM *username*;

The following example revokes the privileges that are defined for the `GUEST_USERS` role from `CALVIN`.

```
REVOKE GUEST_USERS FROM CALVIN;
```

Granting Administrator Privileges to a User

GRANT SYS_ADMIN_ROLE TO *username*;

The following example grants administrator privileges to `CALVIN`, who now has all privileges to all tables.

```
GRANT SYS_ADMIN_ROLE TO CALVIN;
```

Note

If the autocommit mode is set OFF, you need to commit your work. To commit your work use the following SQL statement: `COMMIT WORK;` If the autocommit mode is set ON, the transactions are committed automatically.

Managing Tables

A Solid server has a dynamic data dictionary that allows you to create, delete and alter tables on-line. Solid database tables are managed using SQL commands.

In the SOLID directory, you can find a SQL script named `sample.sql`, which gives an example of managing tables. You can run the script using SOLID *DBConsole*.

Below are some examples of SQL statements for managing tables. Refer to *Appendix D, "SOLID SQL Syntax"* for a formal definition of the SOLID SQL statements.

Tip

If you want to see the names of all tables in your database, issue the SQL statement `SELECT * FROM TABLES` or use predefined command TABLES from SOLID *DBConsole*. The table names can be found in the column `TABLE_NAME`.

Examples of SQL Statements

Below are some examples of SQL commands for administering tables.

Creating Tables

CREATE TABLE *table_name* (*column column type*

[,*column column type*]...);

All users have privileges to create tables.

The following example creates a new table named `TEST` with the column `I` of the column type `INTEGER` and the column `TEXT` of the column type `VARCHAR`.

```
CREATE TABLE TEST (I INTEGER, TEXT VARCHAR);
```

Removing Tables

DROP TABLE *table_name*;

Only the creator of the particular table or users having `SYS_ADMIN_ROLE` have privileges to remove tables.

The following example removes the table named `TEST`.

```
DROP TABLE TEST;
```

Adding Columns to a Table

ALTER TABLE *table_name* ADD COLUMN *column_name*

column_type;

Only the creator of the particular table or users having `SYS_ADMIN_ROLE` have privileges to add or delete columns in a table.

The following example adds the column `C` of the column type `CHAR(1)` to the table `TEST`.

```
ALTER TABLE TEST ADD COLUMN C CHAR(1);
```

Deleting Columns from a Table

ALTER TABLE *table_name* DROP COLUMN

column_name;

A column cannot be dropped if it is part of a unique or primary key. For details on primary keys, read *"Managing Indexes"* on page 3-8.

The following example statement deletes the column `C` from the table `TEST`.

```
ALTER TABLE TEST DROP COLUMN C;
```

Note

If the autocommit mode is set OFF, you need to commit your work before you can modify the table you altered. To commit your work after altering a table, use the following SQL statement: `COMMIT WORK;`. If the autocommit mode is set ON, transactions are committed automatically.

Managing Indexes

Indexes are used to speed up access to tables. The database engine uses indexes to access the rows in a table directly. Without indexes, the engine would have to search the whole contents of a table to find the desired row. For details on creating indexes to improve performance, read *"Using Indexes to Improve Query Performance"* on page 7-2.

There are two kinds of indexes: non-unique indexes and unique indexes. A unique index is an index where all key values are unique. You can create as many indexes as you like on a single table; however, adding indexes does slow down updates, such as inserts, deletes, and updates on that table.

You can create and delete indexes using the following SQL statements. Refer to *Appendix D, "SOLID SQL Syntax"* for a formal definition of the syntax for these statement.

Examples of SQL Statements

Below are some examples of SQL commands for administering indexes.

Creating an Index on a Table

CREATE [UNIQUE] INDEX *index_name* ON *base_table_name*

[*column_identifier* [ASC | DESC]

[, *column_identifier* [ASC | DESC]] ...

Only the creator of the particular table or users having SYS_ADMIN_ROLE have privileges to create or delete indexes.

The following example creates an index named `X_TEST` on the table `TEST` to the column `I`.

```
CREATE INDEX X_TEST ON TEST (I);
```

Creating a Unique Index on a Table

CREATE UNIQUE INDEX *index_name* ON *table_name*

(*column_name*);

The following example creates a unique index named `UX_TEST` on the table `TEST` to the column `I`.

```
CREATE UNIQUE INDEX UX_TEST ON TEST (I);
```

Deleting an Index

DROP INDEX *index_name*;

The following example deletes the index named `X_TEST`.

```
DROP INDEX X_TEST;
```

Note

If the autocommit mode is set OFF, you need to commit your work before you can modify the table on which you altered the indexes. To commit your work after modifying indexes, use the following SQL statement: `COMMIT WORK;`. If the autocommit mode is set ON, the transactions are committed automatically.

Primary Keys

A primary key is a column or combination of columns that uniquely identify each record in a table. Primary keys like indexes speed up access to tables.

In a Solid server, the difference between primary keys and indexes is that the primary key clusters data in the database according to the key values. This clustering process is described

in the following section. Without a primary key defined to a table, rows are ordered on disk according to the time in which they were inserted into the database.

Data clustering

The storage server, part of a Solid server's indexing system, is used to store both secondary keys and primary keys (containing the actual data values). By defining a primary key for a table, you allow a Solid server to use the key to physically cluster the data rows to the order given by the index.

The set of columns used for clustering is called the row reference. The row reference uniquely identifies the data row. If the user-defined columns for the clustering key are not unique, the system ensures that the reference is unique by adding a unique row number to the reference columns. The row reference is also known as the "row identifier."

The row reference can be any combination of one or more columns. Each table has a different set of columns that are used for the unique row reference.

Secondary Keys

Some tables also have a secondary key to implement primary key referencing. In this case, a secondary key value refers to a data row using the row reference. If all the requested data is found from the secondary key, no search on the clustering key is performed. Otherwise, the data is searched from the clustering key using the row reference as the search argument.

Foreign Keys

A foreign key is a column or group of columns within a table that refers to, or relates to, some other table through its values. The foreign key must always include enough columns in its definition to uniquely identify a row in the referenced table. A foreign key must contain the same number of columns as the primary key and be in the same order; however, a foreign key can have different column names and default values than the primary key.

The main reason for defining foreign keys is to ensure the validity of references between tables. Rows in one table must always have corresponding rows in another table, thereby maintaining referential integrity,

You define the rules for referential integrity as part of the CREATE TABLE statement through primary and foreign keys. For example:

```
CREATE TABLE DEPT (
      DEPT INTEGER NOT NULL,
      DNAME VARCHAR,
      PRIMARY KEY (DEPTNO));
```

```
CREATE TABLE EMP (
      DEPTNO INTEGER,
      ENAME VARCHAR,
      FOREIGN KEY (DEPTNO) REFERENCES DEPT (DEPTNO));
```

Refer to *Appendix D, "SOLID SQL Syntax"* for CREATE TABLE syntax detail.

Managing Transactions

A transaction is a group of SQL statements treated as a single unit of work; either all the statements are executed as a group, or none are executed. This section assumes you know the fundamentals for creating transactions using standard SQL statements. It describes how SOLID SQL lets you handle transaction behavior, concurrency control, and isolation levels.

Defining Read-only or Read-write Transactions

To define a transaction to be read-only or read-write, use the following SQL commands:

SET TRANSACTION READ ONLY | READ WRITE

The following options are available with this command.

- READ ONLY

  Use this option for a read only transaction.

- READ WRITE

  Use this option for a read and write transaction. This option is the default.

Note

To detect conflicts between transactions, use the standard ANSI SQL command SET TRANSACTION ISOLATION LEVEL to define the transaction with a Repeatable Read or Serializable isolation level. For details, read *"Choosing Transaction Isolation Levels"* on page 3-13.

Transactions are ended with the COMMIT WORK or ROLLBACK WORK commands.

Setting Concurrency Control

The primary model used for concurrency is a multiversioning and optimistic concurrency control method. Multiversioning means that multiple versions of the same row can co-exist in the database. This way, users are able to concurrently access the database at the same time, and the view of the data that they access is consistent throughout the transaction. Data is always available to users because locking is not used; access is improved since deadlocks no longer apply. For details, read *"SOLID Bonsai Tree Multiversioning and Concurrency Control"* on page 1-6. The optimistic concurrency control is automatically set for all tables.

Setting Pessimistic and Mixed Concurrency Control

When necessary, you can use pessimistic (row-level locking) or mixed concurrency control methods. There are situations when pessimistic concurrency control is more appropriate. For example, in some applications there are small areas that are very frequently updated. In the case of these so-called *hotspots*, conflicts are so probable that optimistic concurrency control wastes effort in rolling back conflicting transactions.

You can also use mixed concurrency control, a combination of row-level locking and optimistic concurrency control. By turning on row-level locking table-by-table, you can specify that a single transaction use both concurrency control methods simultaneously. This functionality is available for both read-only and read-write transactions.

Note that since pessimistic ordering of SOLID *Embedded Engine* is managed on the row level, there is no need to manage page or table level locking.

To set individual tables for optimistic or pessimistic concurrency, use the following SQL command:

ALTER TABLE *base_table_name* SET {OPTIMISTIC | PESSIMISTIC}

Note that by default all tables are set for optimistic.

You can also set a database-wide default in the `General` section of the configuration file with the following parameter:

```
Pessimistic = yes
```

Locking

To control the level of consistency and concurrency in the application, locks are placed on rows when users are submitting queries or updates to rows. The following lock modes are used only for pessimistic tables:

- SHARED

Multiple users can hold shared locks on the same row simultaneously. Shared locks are used on queries.

- UPDATE

  When a user accesses a row with the SELECT... FOR UPDATE statement, the row is locked with an update mode lock. This means that no other user can read or update the row, and ensures the current user can later update the row.

- EXCLUSIVE

  Only one user has an exclusive lock on a row at any given time. Exclusive locks are used on insert, update, and delete operations.

Setting Lock Timeout

The lock timeout setting is the time in seconds that SOLID *Embedded Engine* waits for a lock to be released. By default, lock timeout is set to 30 seconds. When the timeout interval is reached, *Embedded Engine* terminates the timed out transaction. For example, if a user is querying a specific row in a table and the second user is inserting data into the same row, the insert will not go through until the first user's query is completed or times out. Once completed, a lock is then issued for the second user's insert transaction.

You can set the lock time out with the following SQL command:

SET LOCK TIMEOUT *timeout_in_seconds*

Setting Lock Timeout for Optimistic Tables

When you use SELECT FOR UPDATE, the selected rows are locked also for tables with optimistic concurrency control. To set the lock timeout separately for optimistic tables per connection, use the following SQL command:

SET OPTIMISTIC LOCK TIMEOUT *seconds*

Choosing Transaction Isolation Levels

Concurrency control is based on an applications requirements. Some applications need to execute as if they had the exclusive ownership of the database. Other applications can tolerate some degree of interference from other applications running simultaneously. To meet the needs of different applications, the SQL2 standard defines four levels of isolation for transactions.

- Read Uncommitted

  This isolation level allows transactions to read data modified by other transactions that have not yet committed.

- Read Committed

  This isolation level allows a transaction to read only committed data. Still, the view of the database may change in the middle of a transaction when other transactions commit their changes. Read Committed does not prevent phantom updates, but it does ensure that the results set returned by a single query is consistent by setting the read level to the latest committed transaction when the query is started.

- Repeatable Read

  This isolation level is the default isolation level for SOLID databases. It allows a transaction to read only committed data and guarantees that read data will not change until the transaction terminates. SOLID *Embedded Engine* additionally ensures that the transaction sees a consistent view of the database. Conflicts between transactions are detected by using transaction write-set validation. Still, phantom updates may occur.

- Serializable

  This isolation level allows a transaction to read only committed data with a consistent view of the database. Additionally, no other transaction may change the values read by the transaction before it is committed because otherwise the execution of transactions cannot be serialized in the general case.

  SOLID *Embedded Engine* can provide serializable transactions by detecting conflicts between transactions. It does this by using both write-set and read-set validations. Because no locks are used, all concurrency control anomalies are avoided, including the phantom updates.

Setting the Isolation Level

To set the isolation level, use the following SQL command:

```
SET TRANSACTION ISOLATION LEVEL
        READ COMMITTED | REPEATABLE READ | SERIALIZABLE
```

For example:

```
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;
```

Managing Database Objects

In keeping with ANSI SQL and ISO standards, schema and catalog support are provided for Solid database objects. Catalogs allow you to logically partition databases so you can organize your data to meet the needs of your business or application.

A catalog can qualify one or more schema names. A schema is a persistent database object that provides a definition for the entire database. It represents a collection of database objects associated with a specific schema name. These objects include tables, views, indexes, stored procedures, triggers, and sequences.

The catalog name is used to qualify a database object name. They are qualified in all DML statements as:

catalog_name.schema_name.database_object

or

catalog_name.user_id.database_object

You can qualify a schema with one or more database objects. To use a schema name with a database object, create the schema first.

schema_name.database_object_name

or

user_id.database_object_name

By default, database objects that are created without schema names are qualified using the user ID of the database object's creator. For example:

user_id.table_name

Catalog and schema contexts are set using the SET CATALOG or SET SCHEMA statement.

If a catalog context is not set using SET CATALOG, then all database object names are resolved always using the default catalog name.

Note

When creating a new database or converting an old database to a new format, the user is prompted to specify a default catalog name for the database system catalog. Users can access the system catalog name without knowing this specified default catalog name. For example, users can specify the following syntax to access the system catalog:

""._SYSTEM.*table*

SOLID *Embedded Engine* translates the empty string ("") specified as a catalog name to the default catalog name. *Embedded Engine also* provides for automatic resolution of _SYSTEM schema to the system catalog, even when users provide no catalog name.

The following SQL statement provide examples of creating catalogs and schemas. Refer to *Appendix D, "SOLID SQL Syntax"* for a formal definition of the SOLID SQL statements.

Examples of SQL Statements

Below are some examples of SQL commands for managing database objects.

Creating a Catalog

CREATE CATALOG *catalog_name*

Only the creator of the database or users having SYS_ADMIN_ROLE have privileges to create or delete catalogs.

The following example creates a catalog named `C` and assumes the userid is `SMITH`

```
CREATE CATALOG C;
SET CATALOG C;
CREATE TABLE T;
SELECT * FROM T;
--The name T is resolved to C.SMITH.T
```

Setting a Catalog and Schema Context

The following example sets a catalog context to `C` and the schema context to `S`.

```
SET CATALOG C;
SET SCHEMA S;
CREATE TABLE T;
SELECT * FROM T;
-- the name T is resolved to C.S.T
```

Deleting a Catalog

DROP CATALOG *catalog_name*;

The following example deletes the catalog named `C`.

```
DROP CATALOG C;
```

Creating a Schema

CREATE SCHEMA *schema_name*

Any database user can create a schema; however, the user must have permission to create the objects that are pertain to the schema (for example, CREATE PROCEDURE, CREATE TABLE, etc.).

The following example creates a schema named `FINANCE` and assumes the user id is SMITH.

The following example creates a schema named `FINANCE` and assumes the user id is SMITH:

```
CREATE SCHEMA FINANCE;
CREATE TABLE EMPLOYEE (EMP_ID INTEGER);
-- This table is qualified to SMITH.EMPLOYEE
SET SCHEMA FINANCE;
CREATE TABLE EMPLOYEE (ID INTEGER);
SELECT ID FROM EMPLOYEE;
-- In this case, the table is qualified to FINANCE.EMPLOYEE
```

Deleting a Schema

DROP SCHEMA *schema_name*;

The following example deletes the schema named `FINANCE`.

```
DROP SCHEMA FINANCE;
```

4

Using SOLID Data Management Tools

This chapter describes SOLID Data Management Tools, a set of utilities for performing various database tasks. These tools include:

- SOLID *DBConsole*, an easy-to-use graphical user interface for administration and configuration tasks, monitoring local and remote Solid servers, issuing SQL queries and statements, and executing SQL script files.
- SOLID *Remote Contro*l (teletype) and SOLID *SQL Editor* (teletype) for command line sessions at the operating system prompt.
- SOLID *SpeedLoader* for loading data from external ASCII files into a SOLID database.
- SOLID *Export* is a product for unloading data from a SOLID database to ASCII files.
- SOLID *Data Dictionary* for retrieving data definition statements from a SOLID database.

Note

Not all SOLID Tools are necessarily part of the standard product delivery, and their availability on some platforms may be limited. For information about SOLID data management tools, contact your SOLID sales representative or SOLID Online Services at the SOLID Web site:

`http://www.solidtech.com/`

SOLID *DBConsole*

SOLID *DBConsole* is a java-based, graphical user interface for managing, administering, and querying local and remote Solid servers. Designed for intuitive and efficient ease-of-use, it allows you to create and manipulate database schemas, browse data, monitor and manage both local and remote databases, and configure Solid server parameters.

With SOLID *DBConsole* you can use an Administration window, which features an intuitive interface to perform the basic administration tasks described in this manual. You can also use a Query window to issue administrative commands (equivalent to SOLID SQL ADMIN COMMANDs) and create and execute script files.

Note

To perform administration operations in SOLID *DBConsole* requires SYS_ADMIN_ROLE rights.

Starting DBConsole

To start SOLID *DBConsole*:

- Enter the following command at your operating system prompt:

  ```
  java DBConsole
  ```

  -or-

- When using Windows, start *DBConsole* from the icon in your Program Group.

This displays the *DBConsole* interface, where you perform database tasks by using the menubar, toolbar, or right-click mouse menus that apply to particular items in the work-space.

Note

Ensure that the SOLID database server is running before establishing a database connection. Use the Add Database dialog box to add additional databases and the Connect dialog box to connect to the databases. For details, refer to *DBConsole* Online Help available by selecting

Help on the menu bar.

You can also start *DBConsole* by including one or more of these optional command line arguments:

java DBConsole *options*

where *options* can be:

-M*mode*	*mode* = BATCH; specifies that *DBConsole* run in Batch Mode without showing the user interface.
-D*databasename*	Specifies a database for connection.
-U*url*	Specifies the JDBC URL required for *DBConsole* to connect to a Solid server. The format of the JDBC URL is: JDBC:SOLID://*machine_name*:*port_number* For example: `jdbc:solid://localhost:1313`
-u*userid*	Specifies the user ID for accessing the database
-p*password*	Specifies the user's password for accessing the database
-f*queryfile*	Executes the SQL statements contained in the script file.
-e*sql_strings*	Executes the SQL strings.
-o*Outputfile*	Specifies the file where resultsets are stored.
-O*Outputfile*	Specifies the file (which will be opened for writing in append mode) where resultsets are stored.
-a	All transactions are autocommitted.
-h	Help Usage

DBConsole Interface Features

DBConsole opens each new database connection with three separate windows: a Browse window, a Query window, and an Administration window. You can move from one window to another to manage different databases simultaneously.

Note

The features of each window are described briefly in the following sections. For details on usage, refer to the *DBConsole* Online Help available by selecting **Help** on the menu bar.

Query Window

With the Query window, you can issue administrative commands (equivalent to SOLID SQL ADMIN COMMANDs), SQL queries and statements, or execute a script file that contains these items. For a list of administrative commands, see *"Administrative Commands"* on page 4-6.

A results section in the Query window displays error messages and the result set, which you can print or save to a text file. If needed, you can cancel execution of a current SQL statement and specify transaction commits and rollbacks. Settings are also available to enable autocommit and the transaction isolation level for a connection.

Administration Window

With the Administration window, you can monitor server status (including messages) and control all Solid servers in a network from a single workstation. From the Administration window, you can perform the following local and remote operations:

- Control user access to databases
- Control network protocol connections
- Generate backups and checkpoints
- Create timed commands to automate administration
- Configure a Solid server's parameters

Browse Window

With the Browse window, you can browse database objects, which include tables, columns, views, indexes, stored procedures, sequences, roles, and users. A database workspace gives you a quick view of database connections, databases, and their objects in a tree format. You can click on a node in the tree to browse an object, which is displayed in table format. For easier viewing, you can rearrange data columns by moving and resizing table headers.

SOLID *Remote Control* (teletype)

With SOLID R*emote Control* (teletype), you can execute administrative commands (equivalent to the SOLID SQL ADMIN COMMANDs), at the command line, command prompt, or by executing a script file that contains the commands. For a list of these commands, see *"Administrative Commands"* on page 4-6.

Note

The user performing the administration operation must have SYS_ADMIN_ROLE or

SYS_CONSOLE_ROLE rights, or the connection will be refused.

Starting SOLID *Remote Control* (teletype)

Start SOLID *Remote Control* by issuing the command `solcon` or `load solcon` (Novell Netware) at the operating system prompt.

You can also specify the following syntax and include these optional command line arguments:

solcon *options servername username password*

where *options* can be:

-c*dir*	Change working directory
-e*command string*	Execute the specified *Remote Control* command
-f*filename*	Execute command string from a script file
-h, **-?**	Help = Usage

Servername is the network name of a Solid server that you are connected to. Logical Data Source Names can also be used with tools; Refer to *Chapter 5,"Managing Network Connections,"* for further information. The given network name must be enclosed in quotes.

Username is required to identify the user and to determine to determine the user's authorization. Without appropriate rights, execution is denied.

Password is the user's password for accessing the database.

SOLID *Remote Control* connects to the first server specified in the `Connect` parameter in the `solid.ini` file. If you specify no arguments, you are prompted for the database administrator's user name and password. You can give connection information at the command line to override the connect definition in `solid.ini`.

To exit *Remote Control*, enter the command `exit`.

Examples

Start up Remote Control with the servername and the administrator's username and password:

```
solcon "tcp localhost 1313" admin iohi4y
```

Start up Remote Control to back up a specific database:

```
solcon -ebackup "ShMem SOLID" dbadmin password
```

Entering SOLID *Remote Control* (teletype) Commands

After the connection to the server is established, the command prompt appears.

You can execute all commands at the command line with the **-e** option or in a text file with the **-f** option. You can also execute all SOLID *Remote Control* commands programmatically from an application using options of the SQL command "ADMIN COMMAND". For example, you can start a backup with the SQL command ADMIN COMMAND 'backup'.

When there is an error in the command line, SOLID *Remote Control* gives you a list of the possible options as a result. Please be sure to check the command line you entered.

Administrative Commands

Command	Abbreviation	Explanation
backup [*backup_directory*]	bak	Makes a backup of the database. The default backup directory is the one defined in parameter BackupDirectory in the General section of the configuration file. The backup directory may also be given as an argument. For example, **backup abc** creates backup on directory 'abc'. All directory definitions are relative to the SOLID *Embedded Engine* working directory.
backuplist	bls	Displays a status list of last backups.
close	clo	Closes server from new connections; no new connections are allowed.
describe parameter *param*	des par	Returns description text of parmeter. The following example describes parameter [Com]Trace=y/n. `describe parameter com.trace`
errorcode *SOLID_error_code*	ec	Displays a description of an error code. Provide the error code as an argument. For example: **errorcode 10033**
exit	ex	Exits SOLID *Remote Control.*

Command	Abbreviation	Explanation
info *options*	in	Returns server information. Options are one or more of the following values, each separated by a space: ■ **Numusers** - number of current users ■ **Maxusers** - maximum number of users ■ **Sernum** - Server serial number ■ **Dbsize** - database size ■ **Logsize** - size of log files ■ **Uptime** - server up since ■ **Bcktime** - timestamp of last successfully completed backup ■ **Cptime** - timestamp of last successfully completed checkpoint ■ **tracestate** - Current trace statem ■ **monitorstate** - Current monitor state, number of users withmonitor enabled, or -1 if all ■ **openstate** - Current open or close state Values are returned in the same order as requested, one row for each value. Example: `ADMIN COMMAND 'info dbsize logsize'`
help	?	Displays available commands.
makecp	mcp	Makes a checkpoint.
messages [**-n**] [**warnings** | **errors**] [*count*]	mes	Displays server messages. Optional severity and message numbers can also be defined. For example: `messages warning 100` The above example displays the last 100 warnings.
monitor {**on** | **off**} **[user** *username* | *userid*]	mon	Sets server monitoring on and off. Monitoring logs user activity and SQL calls to `soltrace.out` file

Command	Abbrevia-tion	Explanation
notify {**user** *username* | *user id* | **ALL** } *message*	not	This command sends an event to a given user with event identifier NOTIFY. This identifier is used to cancle an event waiting thread when the statement timeout is not long enough for a disconnect or to change the event registration. The following example sends a notify message to a user with user id 5; the event then gets the value of the message parameter. `notify user 5 Canceled by admin`
open	ope	Opens server for new connections; new connections are allowed.
parameter [*option*][*name*[=*value*]]	par	Displays server parameter values. For example: ■ **parameter** used alone displays all parameters ■ **parameter genera**l displays all parameters from section “general.” ■ **parameter general.readonly** displays a single parameter “readonly” from section “general.” ■ **parameter com.trace=yes** sets communication trace on If **-r** is used, then only the current parameter values are returned.

Command	Abbreviation	Explanation

perfmon [*options*][*subsystem prefix*]	pmon	Returns performance statistics from the server. Options are:
		■ **-c** returns all values as the counter
		■ **-d** returns short descriptions
		■ **-v** returns current values
		■ **-t** returns total values
		By default, some values are averages/second.
		The subsystem prefix is used to find matches to value names. Only those values are returned that match the subsystem prefix.
		The following example returns all information:
		`perfmon`
		The following example returns all values whose name starts with prefix **File** as counters.
		`perfmon-c`

pid	pid	Returns server process id
protocols	prot	Returns list of available communication protocols, one row for each protocol.
report *filename*	rep	Generates a report of server info to a file given as an argument.
shutdown	sd	Stops SOLID *Embedded Engine*.
status	sta	Displays server statistics.
status backup	sta bak	Displays status of the last started backup. The status can be one of the following: ■ If the last backup was successful or any backups have not been requested, the output is 0 SUCCESS. ■ If the backup is in process; for example, started but not ready yet, the output is 14003 ACTIVE. ■ If the last backup failed, the output is: *errorcode* ERROR where the *errcode* shows the reason for the failure.

Command	Abbreviation	Explanation
throwout {*username* | *userid* | **all }**	to	Exits users from SOLID *Embedded Engine*. To exit a specified user, give the user id as an argument. To throw out all users, use the keyword ALL as an argument.
trace {**on** | **off**} **sql** | **rpc** | **sync**	tra	Sets server trace on or off. This command is similar to the **monitor** command, but traces different entities and a different levels. By default, the output is written to the SOLTRACE.OUT file.
userid	uid	Returns user identification number of the current connection.
userlist [**-l**] [*name* | *id*]	ul	Displays a list of users. option -l displays more detailed output.
version	ver	Displays server version info.

SOLID *SQL Editor* (teletype)

With SOLID *SQL Editor* (teletype), SQL statements (including the SQL ADMIN COMMANDs) can be issued at the command line, command prompt, or by executing a script file that contains the SQL statements. For a formal definition of SQL statements and a list of ADMIN COMMANDs, refer to *Appendix D, "SOLID SQL Syntax"* in this guide. To access a short description of available ADMIN COMMANDs, including short abbreviations, execute:

```
ADMIN COMMAND 'help'.
```

Note

The user performing SQL statements must have appropriate user rights on the corresponding tables, or the connection will be refused.

Starting SOLID *SQL Editor* (teletype)

Start SOLID *SQL Editor* by issuing the command `solsql` or `load solsql` (Novell Netware) at the operating system prompt.

You can also specify the following syntax and include these optional command line arguments:

solsql *options servername username password*

where *options* can be:

-a	Auto commit every statement
-c*dir*	Change working directory
-e*sql-string*	Execute the SQL string; if used commit can only be done using -a.
-f*filename*	Execute SQL string from a script file
-h, -?	Help = Usage
-o*filename*	Write result set to this file
-O*filename*	Append result set to this file
-s*schema_name*	Use only this schema
-t	Print execution time per command
-u	Expect input in UTF-8 format
-x onlyresults	Print only rows

Note

If the user name and password are specified at the command line, the server name must also be specified. Also if the name of the SQL script file is specified at the command line (not with the **-f** option), the server name, user name, and password must also be specified. Remember to commit work at the end of the SQL script or before exiting *SQL Editor.*

Servername is the network name of a Solid server that you are connected to. Logical Data Source Names can also be used with tools; Refer to *Chapter 5,"Managing Network Connections,"* for further information. The given network name must be enclosed in quotes.

Username is required to identify the user and to determine to determine the user's authorization. Without appropriate rights, execution is denied.

Password is the user's password for accessing the database.

SOLID *SQL Editor* connects to the first server specified in the `Connect` parameter in the solid.ini file. If you specify no arguments, you are prompted for the database administrator's user name and password.

When there is an error in the command line, the SOLID *SQL Editor* gives you a list of the possible options as a result. Please be sure to check the command line you entered.

To exit *SQL Editor,* enter the command `exit`.

Examples

Assuming that a database connection is established, this example at the command prompt, executes the SQL statements terminated by a colon:

```
create table testtable (value integer, name varchar);
```

commit work;

Start *SQL Editor* and execute the tables.sql script:

```
solsql “tcp localhost 13113” admin iohe47 tables.sql
```

Executing SQL Statements with SOLID *SQL Editor* (teletype)

After the connection to the server has been established a command prompt appears. SOLID *SQL Editor* (teletype) executes SQL statements terminated by a semicolon.

Example:

```
create table testtable (value integer, name varchar);
commit work;

insert into testtable (value, name) values (31, 'Duffy Duck');
select value, name from testtable;
commit work;

drop table testtable;
commit work;
```

Executing a SQL Script from a File

To execute a SQL script from a file, the name of the script file must be given as a command line parameter:

solsql *servername username password filename*

All statements in the script must be terminated by a semicolon. SOLID *SQL Editor* (teletype) exits after all statements in the script file have been executed.

Example:

```
solsql "tcp localhost 1313" admin iohe4y tables.sql
```

Note

Remember to commit work at the end of the SQL script or before exiting SOLID *SQL Editor* (teletype). If a SQL string is executed with the option **-e**, commit can only be done using the **-a** option.

SOLID *SpeedLoader*

SOLID *SpeedLoader* is a tool for loading data from external ASCII files into a SOLID database. SOLID *SpeedLoader* can load data in a variety of formats and produce detailed information of the loading process into a log file. The format of the import file, that is, the file containing the external ASCII data, is specified in a control file.

The data is loaded into the database through the SOLID *Embedded Engine* program. This enables online operation of the database during the loading. The data to be loaded does not have to reside in the server computer.

Control File

The control file provides information on the structure of the import file. It gives the following information:

- name of the import file
- format of the import file
- table and columns to be loaded

Note

Each import file requires a separate control file. SOLID *SpeedLoader* loads data into one table at a time.

For details about control file format, read *"Control File Syntax"* on page 4-16. Please note the following:

- The table must exist in the database in order to perform data loading.
- Catalog support is available in SOLID *Speedloader*. The following syntax is supported:

  catalog_name.schema_name.table_name

Import File

The import file must be of ASCII type. The import file may contain the data either in a fixed or a delimited format:

- In fixed-length format data records have a fixed length, and the data fields inside the records have a fixed position and length.
- In delimited format data records can be of variable length. Each data field and data record is separated from the next with a delimiting character such as a comma (this is what SOLID *Export* produces). Fields containing no data are automatically set to NULL.

Data fields within a record may be in any order specified by the control file. Please note the following:

- Data in the import file must be of a suitable type. For example, numbers that are presented in a float format cannot be loaded into a field of integer or smallint type.
- Data of varbinary and long varbinary type are hexadecimal encoded in the import file.

Message Log File

During loading, SOLID *SpeedLoader* produces a log file containing the following information:

- Date and time of the loading
- Loading statistics such as the number of rows successfully loaded, the number of failed rows, and the load time if it has been specified with the option
- Any possible error messages

If the log file cannot be created, the loading process is terminated. By default the name of the log file is generated from the name of the import file by substituting the file extension of the import file with the file extension `.log`. For example, `my_table.ctr` creates the log file `my_table.log`. To specify another kind of file name, use the option **-l**.

Configuration File

A configuration file is not required for SOLID *SpeedLoader.* The configuration values for the server parameters are included in the SOLID *Embedded Engine* configuration file `solid.ini`.

Client copies of this file can be made to provide connection information required for SOLID *Speedloader.* If no server name is specified in the command line, SOLID *SpeedLoader* will choose the server name it will connect to from the server configuration file. For example to

connect to a server using the NetBIOS protocol and with the server name SOLID, the following lines should be included in the configuration file:

```
[Com]
Connect=netbios SOLID
```

Starting SOLID *SpeedLoader*

Start SOLID *SpeedLoader* with the command `solload` followed by various argument options. If you start SOLID *SpeedLoader* with no arguments, you will see a summary of the arguments with a brief description of their usage. The command line syntax is:

solload *options servername username password control_file*

where *options* can be:

-C*catalog_name*	Set the default catalog from where data is read from or written to
-s*schema_name*	Set the default schema
-b*records*	Number of records to commit in one batch
-c*dir*	Change working directory
-l*filename*	Write log entries to this file
-L*filename*	Append log entries to this file
-*n**records*	Insert array size (network version)
-t	Print load time
-x **emptytable**	Load data only if there are no rows in the table
-x **errors**:*count*	Maximum error count
-**x nointegrity**	No integrity checks during load (standalone version)
-**x skip**:*records*	Number of records to skip
--?	Help = Usage

For details on *control_file*, read following section.

Servername is the network name of a Solid server that you are connected to. Logical Data Source Names can also be used with tools; Refer to *Chapter 5,"Managing Network Connections,"* for further information. The given network name must be enclosed in quotes.

Username is required to identify the user and to determine the user's authorization. Without appropriate rights, execution is denied.

Password is the user's password for accessing the database.

When there is an error in the command line, the SOLID *SpeedLoader* gives you a list of the possible options as a result. Please be sure to check the command line you entered.

Control File Syntax

The control file syntax has the following characteristics:

- keywords must be given in capital letters
- comments can be included using the standard SQL double-dash (--) comment notation
- statements can continue from line to line with new lines beginning with any word

SOLID *SpeedLoader* reserved words must be enclosed in quotes if they are used as data dictionary objects, that is, table or column names. The following list contains all reserved words for the SOLID *SpeedLoader* control file:

AND	ANSI
APPEND	BINARY
BLANKS	BY
CHAR	CHARACTERSET
DATA	DATE
DECIMAL	DOUBLE
ENCLOSED	ERRORS
FIELDS	FLOAT
IBMPC	INFILE
INSERT	INTEGER
INTO	LOAD
LONG	MSWINDOWS
NOCNV	NOCONVERT
NULLIF	NULLSTR
NUMERIC	OPTIONALLY
OPTIONS	PCOEM
POSITION	PRECISION
PRESERVE	REAL
REPLACE	SCAND7BIT
SKIP	SMALLINT
TABLE	TERMINATED

TIME	TIMESTAMP
TINYINT	VARBIN
VARCHAR	WHITESPACE

The control file begins with the statement LOAD DATA followed by several statements that describe the data to be loaded. Only comments or the OPTIONS statement may optionally precede the LOAD DATA statement.

Full Syntax of the Control File

Syntax Element	Definition
control_file	::= [*option_part*] *load_data_part into_table_part*
option_part	::= OPTIONS (*options*)
options	::= *option* [, *option*]
option	::= [SKIP = '*int_literal*'] | [ERRORS = '*int_literal*']
load_data_part	::= LOAD [DATA] [*characterset_specification*] [DATE *date_mask*] [TIME *time_mask*] [TIMESTAMP *timestamp_mask*] [INFILE *filename*] [PRESERVE BLANKS]
characterset_specification	::= CHARACTERSET { NOCONVERT | NOCNV | ANSI | MSWINDOWS | PCOEM | IBMPC | SCAND7BIT }
into_table_part	::= INTO TABLE *tablename* [APPEND | INSERT | REPLACE] [FIELDS TERMINATED BY { WHITESPACE | *hex_literal* |'char']} [FIELDS [OPTIONALLY] ENCLOSED BY {"char""| *hex_literal*} [AND "char" | *hex_literal*]] (*column_list*)
hex_literal	::= X'hex_byte_string'
column_list	::= *column* [, *column*]
column	::= *column_name datatype_spec* [POSITION ('*int_literal*' {: | -} '*int_literal*')] [DATE *date_mask*] [TIME *time_mask*] [TIMESTAMP *timestamp_mask*] [NULLIF BLANKS | NULLIF NULLSTR| NULLIF '*string*' | NULLIF (('*int_literal*' {: | -} '*int_literal*') = '*string*')]

Syntax Element	Definition
datatype_spec	::= {BINARY | CHAR [*length*] | DATE | DECIMAL [(*precision* [, *scale*])] | DOUBLE PRECISION | FLOAT [(precision)] | INTEGER | LONG VARBINARY | LONG VARCHAR | NUMERIC [(*precision* [, *scale*])] | REAL | SMALLINT | TIME | TIMESTAMP [(*timestamp precision*)] | TINYINT | VARBINARY | VARCHAR [(*length*)] }

The following paragraphs explain syntax elements and their use is in detail.

CHARACTERSET

The CHARACTERSET keyword is used to define the character set used in the input file. If the CHARACTERSET keyword is not used or if it is used with the parameter NOCONVERT or NOCNV, no conversions are made. Use the parameter ANSI for the ANSI character set, MSWINDOWS for the MS Windows character set, PCOEM for the ordinary PC character set, IBMPC for the IBM PC character set, and SCAND7BIT for the 7-bit character set containing Scandinavian characters.

DATE, TIME, and TIMESTAMP

These keywords can be used in two places with different functionality:

- When one of these keywords is used as a part of the load-data-part element, it defines the format used in the import file for inserting data into any column of that type.
- When a keyword appears as a part of a column definition it specifies the format used when inserting data into that column.

Notes

1. Masks used as part of the load-data-part element must be in the following order: DATE, TIME, and TIMESTAMP. Each is optional.
2. Data must be of the same type in the import-file, the mask, and the column in the table into which the data is loaded.

Data Masks

Data Type	Available Data Masks
DATE	YYYY/YY-MM/M-DD/D
TIME	HH/H:NN/N:SS/S
TIMESTAMP	YYY/YY-MM/M-DD/D HH/H:NN/N:SS/S

In the above table, year masks are YYYY and YY, month masks MM and M, day masks DD and D, hour masks HH and H, minute masks NN and N, and second masks SS and S. Masks within a date mask may be in any order; for example, a date mask could be 'MM-DD-YYYY'. If the date data of the import file is formatted as 1995-01-31 13:45:00, use the mask YYYY-MM-DD HH:NN:SS.

PRESERVE BLANKS

The PRESERVE BLANKS keyword is used to preserve all blanks in text fields.

into_table_part

The *into_table_part* element is used to define the name of the table and columns that the data is inserted into.

FIELDS TERMINATED BY

The FIELDS TERMINATED BY keyword is used to define the character used to distinguish where fields end in the input file.

The ENCLOSED BY keyword is used to define the character that precedes and follows data in the input file.

POSITION

The POSITION keyword is used to define a field's position in the logical record. Both start and end positions must be defined.

NULLIF

The NULLIF keyword is used to give a column a NULL value if the appropriate field has a specified value. An additional keyword specifies the value the field must have. The keyword BLANKS sets a NULL value if the field is empty; the keyword NULL sets a NULL value if the field is a string 'NULL'; the definition '*string*' sets a NULL value if the field matches the string '*string*'; the definition '((start : end) = '*string*')' sets a NULL value if a specified part of the field matches the string '*string'*.

Loading Fixed-format Records

Examples of the control file when loading data from a fixed-format import file:

```
-- EXAMPLE 1
LOAD DATA
INFILE 'EXAMP1.DAT'
INTO TABLE SUPPLIERS (
NAME       POSITION(01:19) CHAR,
ADDRESS    POSITION(20:40) VARCHAR,
ID         POSITION(41:48) INTEGER )

-- EXAMPLE 2
OPTIONS (SKIP = 10, ERRORS = 5)
-- Skip the first ten records. Stop if
-- errorcount reaches five.
LOAD DATA
INFILE 'sample.dat'
-- import file is named sample.dat
INTO TABLE TEST1 (
ID         INTEGER POSITION(1-5),
ANOTHER_ID INTEGER POSITION(8-15),
DATE1      POSITION(20:29) DATE 'YYYY-MM-DD',
DATE2      POSITION(40:49) DATE 'YYYY-MM-DD' NULLIF NULL)
```

Loading Variable-length Records

Examples of the control file when loading data from a variable-length import file:

```
-- EXAMPLE 1
LOAD DATA
INFILE 'EXAMP2.DAT'
INTO TABLE SUPPLIERS
FIELDS TERMINATED BY ','
(NAME VARCHAR, ADDRESS VARCHAR, ID INTEGER)
-- EXAMPLE 2
OPTIONS (SKIP=10, ERRORS=5)
-- Skip the first ten records. Stop if
-- errorcount reaches five.
LOAD
DATE 'YYYY-MM-DD HH:NN:SS'
-- The date format in the import file
INFILE 'sample.dat'
```

```
-- The import file
INTO TABLE TEST1
-- data is inserted into table named TEST1
FIELDS TERMINATED BY X'2C'
-- Field terminator is HEX ',' == 2C
-- This line could also be:
-- FIELDS TERMINATED BY ','
OPTIONALLY ENCLOSED BY '[' AND ')'
-- Fields may also be enclosed
-- with '[' and ')'
(
ID INTEGER,
ANOTHER_ID DECIMAL(2),
DATE1 DATE(20) DATE 'YYYY-MM-DD HH:NN:SS',
DATE2 NULLIF NULL
)
-- ID is inserted as integer
-- ANOTHER_ID is a decimal number with 2
-- digits.
-- DATE1 is inserted using the datestring
-- given above
-- The default datestring is used for DATE2.
-- If the column for DATE2 is 'NULL' a NULL is
-- inserted.
```

Running a Sample Load Using Solload

To Run a Sample Load Using Solload

1. Start SOLID *Embedded Engine*.
2. Create the table using the `sample.sql` script and your SOLID *SQL Editor.*
3. Start loading using the following command line:

   ```
   solload "shmem solid" dba dba delim.ctr
   ```

   The user name and password are assumed to be 'dba'. To use the fixed length control file, use the following command line:

   ```
   solload "shmem solid" dba dba fixed.ctr
   ```

The output of a successful loading using `delim.ctr` will be:

```
SOLID Speed Loader v.3.00.00xx
```

```
(C) Copyright Solid Information Technology Ltd 1992-2000
Load completed successfully, 19 rows loaded.
```

The output of a successful loading using `fixed.ctr` will be:

```
SOLID Speed Loader v.3.00.00xx
(C) Copyright Solid Information Technology Ltd 1992-2000
Load completed successfully, 19 rows loaded.
```

Hints to Speed up Loading

The following hints can be used to ensure that loading is done with maximum performance:

- It is faster not to load data over the network, that is, connect locally if possible.
- Increasing the number of records committed in one batch speeds up the load. By default, commit is done after each record.
- Disable transaction logging.

To disable logging the LogEnabled parameter needs to be used. The following lines in the `solid.ini` file will disable logging:

```
[Logging]
LogEnabled=no
```

After the loading has been completed, remember to enable logging again. The following line in the `solid.ini` file will enable logging:

```
[Logging]
LogEnabled=yes
```

Note

Running the server with logging disabled is strongly discouraged. If logs are not written, no recovery can be made if an error occurs due to power failure, disk error etc.

SOLID *Export*

SOLID *Export* is a product for unloading data from a SOLID database to ASCII files. SOLID *Export* produces both the import file, that is, the file containing the exported ASCII

data, and the control file that specifies the format of the import file. SOLID *SpeedLoader* can directly use these files to load data into a SOLID database.

Note

The user name used for performing the export operation must have select rights on the table exported. Otherwise no data is exported.

Starting SOLID *Export*

Start SOLID *Export* with the command `solexp`. If you start `solexp` with no arguments, you'll see a summary of the arguments with a brief description. The command line syntax is:

solexp *options servername username password tablename* | *

where *options* argument can be:

-C*catalog_name*	Set the default catalog from where data is read from or written to
-c*dir*	Change working directory
-e*sql-string*	Execute SQL string for export
-f*filename*	Execute SQL string from file for export
-h, **-?**	Help = Usage
-l*filename*	Write log entries to this file
-L*filename*	Append log entries to this file
-o*filename*	Write exported data to this file
-s*schema_name*	Use only this schema for export

Notes

1. The symbol * can be used to export all tables with one command. However, it cannot be used as a wildcard.
2. The -t*tablename* (Export table) option is still supported in order to keep old scripts valid.

Servername is the network name of a Solid server that you are connected to. Logical Data Source Names can also be used with tools; Refer to *Chapter 5,"Managing Network Connections,"* for further information. The given network name must be enclosed in quotes.

Username is required to identify the user and to determine to determine the user's authorization. Without appropriate rights, execution is denied.

Password is the user's password for accessing the database.

When there is an error in the command line, the SOLID *Export* gives you a list of the possible options as a result. Please be sure to check the command line you entered.

SOLID *Data Dictionary*

SOLID *Data Dictionary* is a product for retrieving data definition statements from a SOLID database. SOLID *Data Dictionary* produces a SQL script that contains data definition statements describing the structure of the database. The generated script contains definitions for tables, views, procedures, sequences, and events.

Notes

1. User and role definitions are not listed for security reasons.
2. The user name used for performing the export operation must have select right on the tables. Otherwise the connection is refused.

Starting SOLID *Data Dictionary*

Start SOLID *Data Dictionary* with the command `soldd`. If you invoke `soldd` with no arguments, you'll see a summary of the arguments with a brief description. The command line syntax is:

soldd *options servername username password* [*tablename*]

where *options* can be:

-C*catalog_name*	Set the default catalog from where data definitions are read from or written to
-c*dir*	Change working directory
-h, -?	Help = Usage
-o*filename*	Write data definitions to this file
-O*filename*	Append data definitions to this file
-s*schema_name*	List definitions from this schema only
-x eventonly	List event definitions only
-x indexonly	List index definitions only
-x procedureonly	List procedure definitions only
-x sequenceonly	List sequence definitions only
-x tableonly	List table definitions only
-x viewonly	List view definitions only

Servername is the network name of a Solid server that you are connected to. Logical Data Source Names can also be used with tools; Refer to *Chapter 5,"Managing Network Connections,"* for further information. The given network name must be enclosed in quotes.

Username is required to identify the user and to determine to determine the user's authorization. Without appropriate rights, execution is denied.

Password is the user's password for accessing the database.

When there is an error in the command line, the SOLID *Data Dictionary* gives you a list of the possible options as a result. Please be sure to check the command line you entered.

Examples:

```
soldd -odatabase.sql "tcp database_server 1313" dbadmin f1q32j4
```

Print the definition of procedure TEST_PROC:

```
soldd -x procedureonly " " dba dba TEST_PROC
```

Notes

1. If no table name is given, all definitions are listed to which the user has rights.
2. If the *objectname* parameter is provided with one of the **-x** options, the name is used to print only the definition of the named object.
3. The **-t***tablename* option is still supported in order to keep old scripts valid.

Tools Sample: Reloading a Database

This example demonstrates how a SOLID *Embedded Engine* database can be reloaded to a new one. At the same time the use of each SOLID tool is introduced with an example. This reload is a useful procedure since it shrinks the size of the database file `solid.db` to a minimum.

To Reload the Database:

1. Extract data definitions from the old database.
2. Extract data from the old database.
3. Replace the old database with a new one.
4. Load data definitions into a new database.
5. Load data into the new database.

Walkthrough

In this example the server name is SOLID and the protocol used for connections is Shared Memory. Therefore, the network name is “ShMem SOLID”. The database has been created with the user name “dbadmin” and the password “password”.

1. Data definitions are extracted with SOLID *Data Dictionary*. Use the following command line to extract a SQLscript containing definitions for all tables, views, procedures, sequences, and events. The default for the extracted SQL file is `soldd.sql`.

   ```
   soldd "ShMem SOLID" dbadmin password
   ```

   With this command all data definitions are listed into one file, `soldd.sql` (the default name). As mentioned earlier, user and role definitions are not listed for security reasons. If the database contains users or roles, they need to be appended into this file.

2. All data is extracted with SOLID *Export*. The export results in control files (files with the extension `.ctr`) and data files (files with the extension `.dat`). The default file name is the same as the exported table name. In 16-bit environments, file names longer than eight letters are concatenated. Use the following command line to extract the control and data files for all tables.

   ```
   solexp "ShMem SOLID" dbadmin password *
   ```

   With this command data is exported from all tables. Each table’s data is written to an import file named `table_name.dat`. A separate control file `table_name.ctr` is written for each table name.

3. A new database can be created to replace the old one by deleting the `solid.db` and all `sol####.log` files from the appropriate directories. When SOLID *Embedded Engine* is started for the first time after this, a new database is created.

Note

It is recommended that a backup is created of the old database before it is deleted. This can be done using SOLID *Remote Control* (teletype).

4. Use the following command line to create a backup using SOLID *Remote Control* (teletype):

   ```
   solcon -eBACKUP "ShMem SOLID" dbadmin password
   ```

   With this command, a backup is created. The option -e precedes an administration command.

5. Load data definitions into the new database. This can be done using SOLID *SQL Editor* (teletype). Use the following command line to execute the SQL script created by SOLID *Data Dictionary*.

   ```
   solsql -fSOLDD.SQL "ShMem SOLID" dbadmin password
   ```

   With this command, data definitions are loaded into the new, empty database. Definitions are retrieved with the option -f from the file `soldd.sql`. Connection parameters are the same as in the earlier examples.

   The previous two steps can be performed together by starting SOLID *Embedded Engine* with the following command line. The option -x creates a new database, executes commands from a file, and exits. User name and password are defined as well.

   ```
   solid -Udbadmin -Ppassword -x execute:soldd.sql
   ```

6. Load data into the new database. This is be done SOLID *Speedloader.* To load several tables into the database a batch file containing a separate command line for each table is recommended. In Unix-based operating systems, using the wildcard symbol * is possible. Use either of the following command lines to load data into the new database.

   ```
   solload "ShMem SOLID" dbadmin password table_name.ctr
   ```

7. With this command, data for one table is loaded. The server is online.

   Batch files that can be used are:

   - Shell scripts in Unix environments
   - `.com` -scripts in VMS
   - `.bat` -scripts in Windows 95/98/2000 and NT

5
Managing Network Connections

As a true client/server DBMS, SOLID *Embedded Engine* provides simultaneous support for multiple network protocols and connection types. Both the database server and the client applications can be simultaneously connected to multiple sites using multiple different network protocols.

This chapter describes how to set up network connections for each of the supported platforms.

Note

Some platforms may limit the number of concurrent users to a single SOLID server process even if the SOLID license accepts higher limits. Refer to the Release Notes for details that apply to your specific operating system.

Communication between Client and Server

The database server and client transfer information between each other through the computer network using a communication protocol.

When a database server process is started, it will publish at least one network name that distinguishes it in the network. The server starts to *listen* to the network using the given network name. The network name consists of a communication protocol and a server name.

To establish a connection from a client to a server they both have to be able to use the same communication protocol. The client has to know the network name of the server and often also the location of the server in the network. The client process uses the network name to specify which server it will *connect* to.

This chapter will give you information on how to administer network names.

Managing Network Names

The network name of a server consists of a *communication protocol* and a *server name*. This combination identifies the server in the network. The network names are defined with the `Listen` parameter in the `[Com]` section of the configuration file. The `solid.ini` file should be located in a SOLID *Embedded Engine* program's working directory or in the directory set by the `SOLIDDIR` environment variable.

A server may use an unlimited number of network names. Note that all components of network names are case insensitive.

Network names are managed in the following ways:

- Using the Protocols page in the SOLID *DBConsole* accessed through the Administration window or menu.
- Using the command `protocols` in SOLID *Remote Control* (teletype) or *SOLID DBConsole*.
- Using the SOLID SQL syntax ADMIN COMMAND 'protocols' in SOLID *SQL Editor* (teletype).
- Directly, by editing the server configuration file `solid.ini`.

An example of an entry in `solid.ini` is:

```
[Com]
Listen = tcpip 1313, nmpipe solid
```

The example contains two network names which are separated by a comma. The first one uses the protocol TCP/IP and the service port 1313, the other one uses the Named Pipes protocol with the name 'solid'. In our example the 'tcpip' and 'nmpipe' are communication protocols while '1313' and 'solid' are server names.

If the `Listen` parameter is not set in the `solid.ini` file, the environment dependent defaults are used.

Notes

1. When a database server process is started it publishes the network names it starts to listen to. This information is also written to a file named `solmsg.out` located in the same directory as the `solid.ini` file.
2. Network names must be unique within one host computer. For example, you cannot have two database servers running, both listening to the same TCP/IP port in one host, but it is possible that the same port number is in use in different hosts. Exceptions to this

are the NetBIOS and IPX/SPX protocols, which require that used server names are unique throughout the whole network.

Viewing Supported Protocols for the Server

Because not all protocols are supported in all environments and operating systems, you can view the protocol options available for your server.

To view supported protocols for a server:

Enter the command `protocols` in SOLID *DBConsole* or SOLID *Remote Control* (teletype).

-or-

Enter the following SOLID SQL syntax in SOLID *SQL Editor* (teletype):

```
ADMIN COMMAND 'protocols'
```

A list of all available communication protocols is displayed. The command provides the following kind of result set, which contains one row for each supported communication protocol:

```
RC        TEXT
--        ----
0         tc TCP/IP
0         nb NetBIOS
```

Viewing Network Names for the Server

Following are ways that you can view network names for the server:

- Select the **Status** option from the SOLID *DBConsole* Administration window or menu and click the **Protocols** icon to view the network names listed in the Protocols dialog box.
- View the parameter `Listen` in the `[Com]` section in the `solid.ini` file.
- Enter the command `parameter com.listen` in SOLID *DBConsole* or SOLID *Remote Control* (teletype).

  -or-

  Enter the following SOLID SQL syntax in the SOLID *SQL Editor* (teletype):

  ```
  ADMIN COMMAND 'parameter com.listen'
  ```

A list of all network names for the server is displayed. The command provides the following kind of result set, which contains one row for each supported communication protocol:

```
RC      TEXT
--      ----
0       TCP/IP 1313
0       NetBIOS Solid
```

Adding and Modifying a Network Name for the Server

Following are ways you can add and edit network names for a server, which consists of a *communication protocol* and a *server name*; for example, `nmpipe solid`.

- Select the **Status** option from the SOLID *DBConsole* Administration window or menu and click the **Protocols** icon to add or modify the network names in the Protocols dialog box.
- To add network names for the server:

  Enter the command `parameter com.listen=`*`network_name`* in SOLID *DBConsole* or SOLID *Remote Control* (teletype).

  -or-

  Enter the following SOLID SQL syntax in SOLID *SQL Editor* (teletype):

  `ADMIN COMMAND 'parameter com.listen=`*`network_name`*`'`

  The command returns the new value as the resultset. If the network name entered is invalid, the ADMIN COMMAND statement returns an error.
- In `solid.ini`, locate the working directory of your SOLID *Embedded Engine* process and add a new network name or edit an existing one as a part of the `Listen` parameter entry in the `[Com]` section.

  Use a comma (,) to separate network names. For example:

  ```
  [Com]
  Listen = tcpip 1313, nmpipe solid
  ```

  Be sure to save the changes and to restart the SOLID *Embedded Engine* process to activate the changes.

To Remove a Network Name from the Server

Following are ways you can remove network names for a server, which consists of a *communication protocol* and a *server name*; for example, `nmpipe solid.`

- Select the **Status** option from the SOLID *DBConsole* Administration window or menu and click the **Protocols** icon to remove the network name in the Protocols dialog box.
- In `solid.ini`, locate the working directory of your SOLID *Embedded Engine* process and remove the network name in the `Listen` parameter entry in the `[Com]` section.

  Be sure to save the changes and to restart the SOLID *Embedded Engine* process to activate the changes.

Note

The modifications to network names do not become active immediately after editing the `solid.ini` file. You must restart the SOLID *Embedded Engine* process.

Tip

You can disable a network name using option **-d** after the protocol name in the network name:

```
tcp -d hobbes 1313, shmem -d solid
```

Network Name for Clients

The network name of a client consists of a *communication protocol*, an optional *host computer name* and a *server name.* By this combination the client specifies the server it will establish a connection to. The communication protocol and the server name must match the ones that the server is using in its network listening name. In addition, most protocols need a specified host computer name if the client and server are running on different machines. All components of the client's network name are case insensitive.

The client's network names are defined in the configuration file `solid.ini` in the `[Com]` section with the `Connect` parameter. The `solid.ini` file should be located in the application program's working directory or in the directory set by the `SOLIDDIR` environment variable.

The following connect line in the `solid.ini` of the application workstation will connect an application (client) using the TCP/IP protocol to a SOLID server running on a host computer named 'spiff' and listening with the name (port number in this case) '1313'.

```
[Com]
Connect = tcpip spiff 1313
```

If the `Connect` parameter is not found in the configuration file `solid.ini` the client uses the environment dependent default instead. The defaults for the `Listen` and `Connect` parameters are selected so that the application (client) will always connect to a local SOLID server listening with a default network name. So the local communication (inside one machine) does not necessarily need a configuration file for establishing a connection.

Notes

1. When the connection is requested by client application using the `SQLConnect` function the network name of the server is given as a Data Source Name parameter for that function. If the given name is not an empty string, its contents are used as a network name and the `Connect` parameter in the configuration file is omitted. If an empty string is passed, the possibly existing `Connect` parameter is used.
2. In the Windows (95, 98, 2000, NT) operating system, the connection can be made by using the SOLID *ODBC Driver*. When a client program is using the SOLID *ODBC Driver*, the network name of the server can be used as the ODBC Data Source Name and the `Connect` parameter in the configuration file is not used

---

Communication Protocols

A client process and SOLID *Embedded Engine c*ommunicate with each other by using computer networks and network protocols. A network operating system—for example, IBM LAN Server or Novell NetWare—is not necessarily needed. You only need a functioning communication protocol for both ends. Supported communication protocols depend on the type of computer and network you are using.

The following paragraphs describe the supported communication protocols and common environments that may be used and also show the required forms of network names for the various protocols.

Note

Depending on your network protocol, there may be relevant communication parameters

associated with the protocol. To find the communication parameters in use, be sure to execute the command `parameter` in the SOLID *DBConsole* or SOLID *Remote Control* (teletype), or `ADMIN COMMAND 'parameter'` in SOLID *SQL Editor*. Then you can use the command `describe parameter` or ADMIN COMMAND 'describe parameter' to view details on the specific communication parameter. See *"Managing Parameters"* on page 6-7 for details on these commands.

Shared Memory

Usually the fastest way two processes can exchange information is to use Shared Memory. This can be used only when the embedded engine and application processes are both running in the same computer. The Shared Memory protocol uses a shared memory location for moving data from one process to another.

To use the Shared Memory protocol in SOLID *Embedded Engine*, select `ShMem` from the list of protocols in SOLID *DBConsole* and enter server name. The server name has to be unique only in this computer.

Format Used in the solid.ini File

Server `Listen = shmem` *`servername`*

Client `Connect = shmem` *`servername`*

Note

Server names must be character strings less than 128 characters long.

TCP/IP

The TCP/IP protocol is typically used for communicating to a server process running under a UNIX operating system. When starting a server using the TCP/IP protocol, you must reserve a port number for it. You will find reserved port numbers in the `/etc/services` file of your system. Select a free number greater than 1024 since smaller numbers are usually reserved for the operating system.

To use the TCP/IP protocol, select `TCP/IP` in the list of protocols in SOLID *DBConsole* and enter a non-reserved port number.

Format Used in the solid.ini File

Server `Listen = tcpip` *`server_port_number`*

Client `Connect = tcpip [`*`host_computer_name`*`]`
`server_port_name`

Notes

1. If the server is running in the same computer with the client program, the host computer name need not be specified. The client computer has to have the used host name listed in its `etc/hosts` file or it must be recognized by the DNS (Domain Name Server). You can also give the host computer's TCP/IP address in dotted decimal format (for example, 194.53.94.97) instead of its host name.
2. On Windows (95, 98, 2000, NT) and UNIX the TCP/IP protocol is usually included in the operating system. On other environments (like VAX/VMS) the TCP/IP software needs to be installed on the system. For a list of supported TCP/IP software, contact Solid Information Technology Ltd. at: `http://www.solidtech.com/`
3. The local loopback interface address, 127.0.0.1, is the default address when a client attempts to open a TCP/IP connection without specifying a hostname.
4. Using option **`-i`***`ip_address`* or **`-i`***`host_name`* SOLID *Embedded Engine* listens only to the specified IP-address or host name. This is useful in multi-homed systems that support many TCP/IP interfaces (or have multiple ip-addresses). For example, a server with the following setting in `solid.ini` accepts connection requests only from inside the same machine, either referred by IP-address 127.0.0.1 or with the name 'localhost', if the DNS is correctly configured:

   ```
   [com]
   Listen = tcp -i127.0.0.1 1313
   ```

   Note that DNS entries can be used instead of IP-addresses, for example:

   ```
   [com]
   Listen = tcp -ilocalhost 1313
   ```
5. Using option -i127.0.0.1, which starts the server to listen only to a local loopback connection, allows TCP/IP listening with a desktop license. To enable TCP/IP usage with desktop licenses, all entries in `solid.ini` have to be edited to include **-i**. Note that default listening of port 1313 (without `solid.ini`) works automatically.

UNIX Pipes

The UNIX domain sockets (UNIX Pipes, Named Pipes, portals) are typically used when communicating between two processes running in the same UNIX machine. UNIX Pipes usually have a very good throughput. They are also more secure than TCP/IP since Pipes can only be accessed from applications that run on the computer where the server executes.

When starting a server using UNIX Pipes, you must reserve a unique listening name (inside that machine) for the server, for instance, 'solid'. Because UNIX Pipes handle the UNIX domain sockets as standard file system entries, there is always a corresponding file created for every listened pipe. In SOLID *Embedded Engine's* case, the entries are created under the path '/tmp'. Our example listening name 'solid' creates the directory '/tmp/solunp_SOLID' and shared files into that directory. The '/tmp/solunp_' is a constant prefix for all created objects while the latter part ('SOLID' in this case) is the listening name in upper case format.

Format Used in the solid.ini File

Server `Listen = upipe server_name`

Client `Connect = upipe server_name`

Notes

1. Server and client processes must run in the same machine in order to use UNIX Pipes for communication.
2. The server process must have a "write" permission to the directory '/tmp'.
3. The client accessing UNIX Pipes must have an "execute" permission to the directory '/tmp'.
4. The directory '/tmp' must exist.
5. UNIX Pipes cannot be used in SCO UNIX.

NetBIOS

The NetBIOS protocol is commonly used in the Windows (95, 98, 2000, NT) operating systems.

To use NetBIOS protocol, select `NetBIOS` in the list of available protocols in SOLID *DBConsole* and enter a non-reserved server name.

Format Used in the solid.ini File

Server `Listen = netbios [a`*`LANA_NUMBER`*`] `*`server_name`*

Client `Connect = netbios [a`*`LANA_NUMBER`*`]`*`server_name`*

Notes

1. The server name must be a character string at most 16 characters long. It may not begin with an asterisk (*).
2. In the above format the optional -aLANA_NUMBER is used to override the default value of the LANA number.
3. In Windows NT the available LANA numbers can be checked using the Network Setup found in the Control Panel. The default value 0 may not be generally very good. You should choose the one(s) where the protocol stack matches the other computers you are using. The LANA number (Network Route: Nbf->Elnk3->Elnk31) that uses NetBEUI as a transport usually functions quite smoothly when used for SOLID communication.
4. The server names have to be unique in the whole network. Establishing a connection or starting the listener using the NetBIOS protocol may be somewhat slow because of the checks needed for uniqueness.
5. SOLID *Embedded Engine* and SOLID Client versions 2.2 and later use all available LANA numbers by default. This makes it unnecessary to specify explicitly which LANA number the application or embedded engine should use. For backward compatibility the parameter '-aLANA_NUMBER' remains available.

Named Pipes

Named Pipes is a protocol commonly used in the Windows (95, 98, 2000, NT) operating systems.

Windows (95/98) support Named Pipes only in client end communication. Windows NT and 2000 supports Named Pipes both in server and client communication.

Format Used in the solid.ini File

Server `Listen = nmpipe` *`server_name`*

Client `Connect = nmpipe [`*`host_computer_name`*`]`
`server_name`

Notes

1. The server names must be character strings at most 50 characters long.
2. If the server is running in the same computer with the application program, the host computer name should not be specified.
3. In order to connect to the SOLID *Embedded Engine* for Windows NT through Named Pipes, the user must have at least the same rights as the user, who started the server. For example if an administrator starts the server only, users with administrator's rights are able to connect to the server through Named Pipes. Similarly if a user with normal user's rights starts the server all users with greater rights are able to connect the server through Named Pipes. If a user does not have proper rights, SOLID Communication Error 21306 message will be given.
4. The use of Named Pipes communication from SOLID *Remote Control* is not recommended. The asynchronous nature of SOLID *Remote Control* communication may cause problems with Named Pipes.

DECnet

The DECnet protocol is used to connect to an embedded engine running on an OpenVMS system. To use this protocol in Windows (95, 98, 2000, NT) you need to have PATHWORKS 32 installed to your client computer.

To use the DECnet protocol, select `DECnet` in the list of protocols in SOLID *DBConsole* and enter a non-reserved server name.

Format Used in the solid.ini File

Server `Listen = decnet` *`server_name`*

Client `Connect = decnet` *`node_name server_name`*

Note

To establish a connection the DECnet node name of the server machine is configured to your node database. The node name can be given either as a node number such as '1.1' or as a node name such as 'VAX1'.

IPX/SPX

The IPX/SPX protocol is used to communicate with SOLID *Embedded Engine* for Novell Netware.

SOLID *Embedded Engine* for Novell Netware starts listening with the default listening name `SOLID` if no listening name is specified in the configuration file `solid.ini`. When SOLID *Embedded Engine* starts, it prints out the network and node information of the server machine.

The SOLID server listening name can be given as a character string or as a socket number. If the given network name is a valid socket number, that is, hex number with exactly 4 characters (for example, 400F) SOLID *Embedded Engine* starts listening in the given port. If the network name could not be interpreted as a socket number it is treated as a server name character string and is published using Novell NetWare SAP (Service Advertising Protocol).

Connecting to a SOLID *Embedded Engine* using SAP requires that you specify only the correct server name in the `Connect` parameter. If the server is listening using some given port, the full NLM server info (see comment below) has to be given.

To use the IPX/SPX protocol, select `IPX/SPX` in the list of protocols in SOLID *DBConsole* and enter a non-reserved server name.

Format Used in the solid.ini File

Server `Listen = spx {servername | socket_number}`

Client `Connect = spx {NLM server_info | server_name}`

Notes

1. The server names must be less than 48 characters long.
2. In the above format, NLM *server_info* stands for a string containing the network number, the node number, and the socket number separated by colons. For example, NLM *server_info* for network 1, node 1, socket number 1313 is 00000001:000000000001:1313. You can abbreviate the information by removing the leading zeros. The previous *Embedded Engine* information could thus also be written as 1:1:1313. The *server_name* stands for an alphanumeric string.
3. The possibility to use socket numbers as the listening name is supported mainly for historical reasons. SAPing is intended to be the primary method.
4. After removing a network name or shutting down SOLID *Embedded Engine* using SOLID *DBConsole* or SOLID teletype tools the server name used may still remain reserved for up to one minute although everything completes successfully. The error 'network name in use' is displayed if SOLID *Embedded Engine* is restarted immediately. This is a 'normal' NetWare SAP feature and happens more often if your network consists of more than one NetWare server. Propagating the SAP cancellation packets to every network node may take a while.

A Summary of Protocols

The following tables summarize the possible operating systems and required forms for network names for the various communication protocols.

Note

The following tables contain the protocols and operating systems that were available when this guide was printed. For an updated list, refer to the SOLID Website at: `http://www.solidtech.com/`.

***Embedded Engine* Protocols and Network Names**

Protocol	Server OS	Network name in solid.ini file
Shared Memory	Windows 95 Windows 98 Windows 2000 Windows NT	Listen = shmem *server*
NetBIOS	Windows 95 Windows 98 Windows 2000 Windows NT	Listen = netbios *server*
Named Pipes	Windows 2000 Windows NT	Listen = nmpipe *server*
IPX/SPX	Novell Netware	Listen = spx *server* Listen = spx *socket number*
TCP/IP	Windows 95 Windows 98 Windows 2000 Windows NT UNIX	Listen = tcpip *port*
UNIX Pipes	UNIX	Listen = upipe *server*

Application Protocols and Network Names

Protocol	Server OS	Network name in solid.ini file
Shared Memory	Windows 95 Windows 98 Windows 2000 Windows NT	Connect = shmem *server*
NetBIOS	Windows 95 Windows 98 Windows 2000 Windows NT	Connect = netbios *server*
Named Pipes	Windows 95 Windows 98 Windows 2000 Windows NT	Connect = nmpipe [host] *server*
IPX/SPX	Novell Netware Windows 95 [1] Windows 98 [1] Windows 2000 [1] Windows NT [1]	Connect = spx *server* Connect = spx NLM *server_nfo*
TCP/IP	Windows 95 Windows 98 Windows 2000 Windows NT UNIX	Connect = tcpip [host] *port*
UNIX Pipes	UNIX	Connect = upipe *server*
DECnet	Windows 95 [2] Windows 98 [2] Windows 2000 [2] Windows NT [2]	Connect = decnet *host server*

1) requires Novell's Netware Client for Windows 95, 98, 2000, and NT

2) requires Digital PATHWORKS 32 for Windows 95, 98, 2000, and NT

Logical Data Source Names

SOLID Clients support Logical Data Source Names. These names can be used for giving a database a descriptive name. This name can be mapped to a network name in three ways:

1. Using the parameter settings in the application's `solid.ini` file.

2. Using the Windows operating systems registry settings.
3. Using settings in a solid.ini file located in the Windows directory.

This feature is available on all supported platforms. However, on non-Windows platforms, only the first method is available.

A SOLID Client attempts to open the file `solid.ini` first from the directory set by the `SOLIDDIR` environment variable. If the file is not found from the path specified by this variable or if the variable is not set, an attempt is made to open the file from the current working directory.

To define a Logical Data Source Name using the `solid.ini` file, you need to create a `solid.ini` file containing the section `[Data Sources]`. In that section you need to enter the 'logical name' and 'network name' pairs that you want to define. The syntax of the parameters is the following:

```
[Data Sources]

logical_name = network_name, Description
```

In the description field, you may enter comments on the purpose of this logical name.

If, for example, you want to define a logical name for the application 'My_application', and the database is located in a UNIX server that you want to connect to by using TCP/IP. You should include the following lines to the solid.ini file, which you need to place in the working directory of your application:

```
[Data Sources]

My_application = tcpip irix 1313, Sample data source
```

When your application now calls the Data Source 'My_application', the SOLID Client maps this to a call to 'tcpip irix 1313'.

On Windows platforms (95, 98, 2000, NT), the registry can be used to map Data Sources. These follow the standards of mapping ODBC Data Sources on a system.

In Windows 95, 98, 2000, and NT, a Data Source may be defined in the Windows Registry. The entry is searched from the path "software\odbc\odbc.ini"

1. first under the root HKEY_CURRENT_USER and if not found,
2. under the root HKEY_LOCAL_MACHINE.

The order of resolving a Data Source name in Windows systems is the following:

1. Look for the Data Source Name from the `solid.ini` file in the current working directory, under the section `[Data Source]`

2. Look for the Data Source Name from the following registry path
   `HKEY_CURRENT_USER\software\odbc\odbc.ini\DSN`
3. Look for the Data Source Name from the following registry path
   `HKEY_LOCAL_MACHINE\software\odbc\odbc.ini\DSN`

In case an application uses normal ODBC Data Sources, the network name is mapped normally using the methods that are provided in the ODBC Driver Manager.

6

Configuring SOLID *Embedded Engine*

This chapter describes how to configure SOLID *Embedded Engine* to meet your environment, performance, and operation needs. It includes the most important parameters and their settings. The section *"Managing Parameters"* on page 6-7 in this chapter gives you step-by-step instructions on how to view and set the parameter values in SOLID *DBConsole,* SOLID *Remote Control* (teletype), or *SQL Editor* (teletype).

Configuration File and Default Settings

When SOLID *Embedded Engine* is started, it attempts to open the configuration file `solid.ini` first from the directory set by `SOLIDDIR` environment variable. If the file is not found from the path specified by this variable or if the variable is not set, an attempt is made to open the file from the current working directory.

The configuration file contains settings for the SOLID *Embedded Engine* parameters. If the file does not exist, SOLID *Embedded Engine* uses default settings for the parameters. Also, if a value for a specific parameter is not set in the `solid.ini` file, SOLID *Embedded Engine* will use a default value for the parameter. The default values depend on the operating system you are using.

Generally, default settings offer good performance and operability, but in some cases modifying some parameter values can improve performance.

Most Important Parameters

This section describes the most important SOLID *Embedded Engine* parameters and their default settings. Parameters are grouped according to section categories in the configuration file. See *Appendix B, "Configuration Parameters"* of this manual for a quick overview of the section categories and all available parameters.

Defining Network Names (Com section)

When a server is started, it will start listening to one or more protocols with network names that distinguish it in the network. A client application uses a similar network name to specify which protocol to use and which server to connect to.

Connect parameter

The `Connect` parameter in the `[Com]` section defines the network name for a client; this is the protocol and name that a client application uses for a server connection. Its default is Operating System dependent. Refer to *Chapter 5, "Managing Network Connections"* for details on the parameter format.

When an application program is using a SOLID *ODBC driver* the ODBC Data Source Name is used and the `Connect` parameter has no effect. The `solid.ini` file, which includes the `Connect` parameter, must be located in the application program's working directory or in the directory set by `SOLIDDIR` environment variable.

The following connect line will connect a client program using the TCP/IP protocol to a SOLID server running in a computer named 'spiff' and server port number '1313'.

```
connect = tcpip spiff 1313
```

Listen parameter

The `Listen` parameter in the `[Com]` section defines the network name for the server; this is the protocol and name that a SOLID server uses when it starts to listen to the network. Its default is Operating System dependent. Refer to *Chapter 5, "Managing Network Connections"* for details on the parameter format.

Managing Database Files and Caching (IndexFile section)

In SOLID *Embedded Engine* data and indexes are stored in the same logical files. The term 'index file' is used as a synonym for the term 'database file'. You also control the caching-related parameters in this section.

FileSpec_[1-N] parameter

The `FileSpec` parameter describes the location and the maximum size of the index file (database file). To define the location and maximum value the database file may reach, the `FileSpec` parameter accepts the following three arguments:

- database file name
- max filesize
- device number (optional)

You can also use the `FileSpec` parameter to divide the database file into multiple files and onto multiple disks. To do this, specify another `FileSpec` parameter identified by the number `2`. The index file will be written to the second file if it grows over the maximum value of the first `FileSpec` parameter. The default value for this parameter is solid.db, 2147483647 (which equals 2 GB expressed in bytes):

```
FileSpec_1=SOLID.DB 2147483647
```

In the following example, the parameters divide the database file on the disks C:, D: and E: to be split after growing larger than 1 GB (=1073741824 bytes).

```
FileSpec_1=c:\soldb\solid.1 1073741824  1
FileSpec_2=D:\soldb\solid.2 1073741824  2
FileSpec_3=G:\soldb\solid.3 1073741824  3
```

Note

The index file locations entered must be valid path names in the server's operating system. For example, if the server runs on a UNIX operating system, path separators must be slashes instead of backslashes.

Although the database files reside in different directories, the file names must be unique. In the above example, it is assumed that C:, D: and E: partitions reside on separate physical disks.

Splitting the database file on multiple disks will increase the performance of the server because multiple disk heads will provide parallel access to the data in your database. There is no limit to the number of database files you may use.

If the database file is split into multiple physical disks, then multithreaded SOLID *Embedded Engine* is capable of assigning a separate disk I/O thread for each device. This way the

server can perform database file I/O in a parallel manner. Read *"Dedicated Threads"* on page 1-10 for more details.

CacheSize

The `CacheSize` parameter defines the amount of main memory the server allocates for the cache. The default value depends on the server operating system; the minimum size is 512 KB. Although SOLID *Embedded Engine* is able to run with a small cache size, a larger cache size speeds up the server. The cache size needed depends on the size of the index file, the number of connected users, and the nature of the operations executed against the server. For example:

```
CacheSize=512
```

ExtendIncrement

The `ExtendIncrement` parameter defines the number of blocks that is allocated at one time when SOLID *Embedded Engine* needs to allocate more space for the database file. The default is 50 blocks. For example:

```
ExtendIncrement=50
```

Specifying the Backup Directory (General section)

Backups of the database, log files and the configuration file `solid.ini` are copied to the backup directory. If you are not using the default 'backup' directory, the backup directory must exist and it must have enough disk space for the backup files. It can be set to any existing directory except the database file directory, the log file directory or the working directory.

BackupDirectory parameter

The `BackupDirectory` parameter in the `[General]` section defines a name and location for your backup directory. Note that default 'backup' is a directory relative to your SOLID directory. For example if the parameter is:

```
BackupDirectory=backup
```

then the backup will be written to a directory that is a sub-directory of the SOLID directory.

Note

The backup directory entered must be a valid path name in the server's operating system. For example if the server runs on a UNIX operating system, path separators must be slashes

instead of backslashes.

Specifying the Transaction Log Files Directory (Logging section)

At commit time, transaction results are written immediately to a specified directory. This file must be stored to a local drive using local disk names to avoid problems with network I/O and to achieve better performance. The default log file directory is the SOLID working directory.

FileNameTemplate

The parameter `FileNameTemplate` in the `Logging` section defines a filename structure for the transaction log files. For example, the following setting

```
FileNameTemplate = d:\logdir\sol#####.log
```

instructs SOLID *Embedded Engine* to create log files to directory `d:\logdir` and to name them sequentially starting from `sol00001.log`.

Specifying a Directory for the External Sorter Algorithm (Sorter section)

The external sorter algorithm is used for sorting processes that do not fit in main memory. When the `TmpDir_[1...N]` is specified in the configuration file, the external sorter algorithm is enabled. All temporary files used by the external sort are created in a specified directory (or directories) and are automatically deleted.

TmpDir_[1...N]

The `TmpDir[1...N]` parameter in the `Sorter` section defines the directory (or directories) that can be used for the external sorter algorithm. There is no default setting. For example:

```
TmpDir_1=c:\soldb\temp.1
TmpDir_2=d:\soldb\temp.2
TmpDir_3=g:\soldb\temp.3
```

ReadAhead

When the I/O manager is handling a long sequential search, it enters a read ahead operation mode. This ensures that the next file blocks of the search in question are read in the cache in advance. This naturally improves the overall performance of sequential searches. The `ReadAhead` sets the number of prefetched index leafs during long sequential searches. The default is 4. For example:

```
ReadAhead=4
```

PreFlushPercent

The preflush operations prepare the cache for the allocation of new blocks. The blocks are written onto the disk from the tail of the cache based on a Least Recently Used (LRC) algorithm. Therefore, when the new cache blocks are needed, they can be taken immediately without writing the old contents onto the disk. The `PreFlushPercent` parameter defines the percentage of page buffer which is kept clean by the preflush thread. The default is 5. For example:

```
PreFlushPercent=5
```

Setting Threads for Processing (Srv section)

In addition to the communication, I/O, and log manager threads, SOLID *Embedded Engine* can start general purpose threads to execute tasks from the server's tasking system. Read *"Multithread Processing"* on page 1-10 for more details.

The optimum number of threads depends on the number of processors the system has installed. Usually it is most efficient to have between two and eight threads per processor.

Finding the value that provides the best performance requires experimentation. A good formula to start with is:

threads= (2 x number of processors) + 1

Threads

The `Threads` parameter in the `[Srv]` section defines the amount of general purpose threads used by SOLID *Embedded Engine*. The default value is two threads for SOLID *Embedded Engine* use. For example:

```
Threads=8
```

Setting SQL Trace Level (SQL section)

The SQL Info facility lets you specify a tracing level on the SQL Parser and Optimizer. For details on each level, read *"SQL Info Facility"* on page 8-1.

Info

The SQL Info facility is turned on by setting the `Info` parameter to a non-zero value in the `[SQL]` section of the configuration file. The output is written to a file named `soltrace.out` in the SOLID directory.

Specifying Network Communication Tracing (Com section)

The communication tracing facility is necessary, for instance, if the network hardware is not functioning properly. By turning the tracing on, the communication layer is capable of logging even the system specific errors and may help in diagnosing the real problem in the network. The following parameters control the outputting of network trace information.

Trace

By changing `Trace` parameter default setting No to the value **Yes**, SOLID *Embedded Engine* starts logging trace information on network messages on established network connection to the default trace file or to the file specified in the `TraceFile` parameters.

TraceFile

If `Trace` parameter is set to yes, trace information on network messages is written to a file specified with this parameter. The default if no file is specified is `soltrace.out`, written to the current working directory of the server or client depending on which end the tracing is started.

Specifying the Character Set for an Application (Client section)

By default, the `CharacterSet` parameter is set to the standard ISO Latin-1 (8859-1). If you are running an old SOLID client version (prior to version 3.0), the setting for the character set may be obsolete. In this case you need to include the `CharacterSet` parameter in the `solid.ini` file and set it to ISO 8859-1 or a setting suitable to your application.

Managing Parameters

You can view and modify SOLID *Embedded Engine* parameters and their values in the following ways:

- Using the Configuration page in SOLID *DBConsole*

  The *DBConsole* Configuration page lets you display a parameters listing in a tree node format and change configuration settings through a dialog box. For details, refer to *DBConsole* Online Help available by selecting **Help** on the menu bar.

- Entering the command `parameter` in SOLID *DBConsole* or SOLID *Remote Control* (teletype).

- Entering the SQL extension ADMIN COMMAND '*parameter*' in SOLID *SQL Editor.*
- Directly, by editing the `solid.ini` file in the SOLID directory.

Instructions for using managing parameters with ADMIN COMMAND and in `solid.ini` is described in the following sections.

Note

For details on viewing and setting server communication protocol parameters only, read *"Managing Network Connections"* on page 5-1.

Viewing and Setting Parameters with ADMIN COMMAND

With the SOLID SQL extension ADMIN COMMAND, you can change the parameters remotely through a Solid server without restarting it. All parameters are accessible even if they are not present in the `solid.ini` configuration file. If the parameter is not present, the default value is used.

Viewing All Parameters

To view all parameters, enter the following command in SOLID *DBConsole* or SOLID *Remote Control* (teletype):

```
parameter
```

or in SOLID *SQL Editor*:

```
ADMIN COMMAND 'parameter';
```

A list of all parameters with current and default values is returned. If you need to view communication protocol settings, see *"Managing Network Connections"* on page 5-1.

If desired, you can also qualify this command with a **-r** option to display only the current values. For example:

```
parameter -r
```

The command provides each parameter in one row. Note that there are four parts to the resultset:

- **Section name** identifies the parameter category
- **Parameter name** is the name of the parameter

- **Value** displays the parameters current value from the solid.ini file if it is present; if the parameter is not present, it displays the system default value.
- **System** default displays the default value.

Viewing the Value of a Specific Parameter

To view the value of a specific parameter, enter the following command using SOLID *DBConsole* or SOLID *Remote Control* (teletype):

parameter -r *section_name.parameter_name*

or in SOLID *SQL Editor*:

ADMIN COMMAND 'parameter -r *section_name.parameter_name*';

where:

section_name is the category name where the parameter is located in `solid.ini`

Example:

- see if communication trace is on

```
parameter -r com.trace
```

-or-

```
ADMIN COMMAND 'parameter -r com.trace';
```

If you need to view a communication protocol settings, see *"Managing Network Connections"* on page 5-1.

Viewing the Description of a Specific Parameter

You can also view a description of a specific parameter, which includes valid parameter types and access modes. This is useful information, especially because parameters may need to be handled dynamically; parameter support may vary between products, platforms, or releases.

To view a parameter's description, enter the following command using SOLID *DBConsole* or SOLID *Remote Control* (teletype):

describe parameter *section_name.parameter_name*

or in SOLID *SQL Editor*:

ADMIN COMMAND 'describe parameter *section_name.parameter_name*';

where:

section_name is the category name where the parameter is located in `solid.ini`

Example:

```
describe parameter com.trace
```

-or-

```
ADMIN COMMAND 'describe parameter com.trace';
```

The command provides a resultset with the following four rows:

- Row 1 is a short description of the parameter
- Row 2 shows the valid types for the parameter
- Row 3 show the parameter access modes, which can be one of the following:
  - READ - the parameter is read only and its value cannot be altered.
  - WRITE - the parameter value can be altered and the new value is used.
  - CREATE - the parameter value is used only when a new database is created and cannot be altered.
  - STARTUP - the parameter value can be altered and the new value is used only when the server is started.
- Row 4 shows the current value of the parameter.

Setting a Parameter Value

To set a value for a specific parameter, enter the following command using SOLID *DBConsole* or SOLID *Remote Control* (teletype):

parameter *section_name.parameter_name=value*

or in SOLID *SQL Editor*:

ADMIN COMMAND 'parameter *section_name.parameter_name=value*';

where:

section_name is the category name where the parameter is located in `solid.ini`

value is a valid parameter value. If no value is specified, this sets the parameter with a default (or unset) value.

Example:

-set communication trace is on

```
parameter com.trace=yes
```

-or-

```
ADMIN COMMAND 'parameter com.trace=yes';
```

Note

Parameter management operations are not transactional and cannot be rolled back.

The commands return the new value as the resultset. If the parameter's access mode is READ (read-only) or the value entered is invalid, the ADMIN COMMAND statement returns an error.

Viewing and Setting Parameters in SOLID.INI

1. Open the `solid.ini` file located in the working directory of your SOLID *Embedded Engine* process.
2. View the value of the parameter.
3. If necessary add the section, parameter and parameter's value.
4. Save the changes.

You need to restart the SOLID *Embedded Engine* process to activate the changes.

The parameters displayed are the parameters currently active in the server. If you have not set a parameter value, the displayed value is the default value for the parameter. The default values are set at start-up and depend on the operating system SOLID *Embedded Engine* runs on.

Note

To force a parameter value change to take effect you must shut down and restart the SOLID *Embedded Engine* process.

Caution

The new parameter values are not checked by the server. Setting an unreasonable value for a parameter may result in an operation failure the next time the server process is started. Do not set a parameter to a random value unless you know what you are doing. Use the default

parameter values as an indication on the value range.

Constant Parameter Values

The parameter access mode for the `Blocksize` parameter in both the `IndexFile` and `Logging` sections of the configuration file are CREATE. This means that the parameter is set when the database is created and cannot be modified afterwards.

If you want to use a different constant value, you have to create a new database. Before creating a new database, set the new parameter constant value by editing the `solid.ini` file in the SOLID directory.

The following example sets a new block size for the index file by adding the following lines to the `solid.ini` file:

```
[Indexfile]
Blocksize=4096
```

After editing and saving the `solid.ini` file, move or delete the old database and log files, and start SOLID *Embedded Engine*.

Note

If you have changed the constant value for the log file block size only, you are only required to move or delete the log files, not the database file.

The server program will create a new database with the new constant value from the `solid.ini` file.

7

Performance Tuning

This chapter discusses techniques that you can use to improve the performance of SOLID *Embedded Engine*. The topics included in this chapter are:

- Tuning SQL statements and applications
- Using indexes to improve query performance
- Optimizing batch inserts and updates
- Tuning memory allocation
- Tuning CPU concurrency load
- Tuning I/O
- Tuning checkpoints
- Using optimizer hints for performance

Tuning SQL Statements and Applications

Tuning the SQL statements, especially in applications where complex queries are involved, is generally the most efficient means of improving the database performance.

Be sure to tune your application *before* tuning the RDBMS because:

- during application design you have control over the SQL statements and data to be processed
- you can improve performance even if you are unfamiliar with the internal working of the RDBMS you are going to use
- if your application is not tuned well, it will not run well even on a well-tuned RDBMS

You should know what data your application processes, what are the SQL statements used, and what operations the application performs on the data. For example, you can improve

query performance when you keep SELECT statements simple, avoiding unnecessary clauses and predicates.

Evaluating Application Performance

To isolate areas where performance is lacking in your application, SOLID *Embedded Engine* provides the following diagnostic tools for observing database performance:

- SQL info facility
- EXPLAIN PLAN statement

These tools are helpful in tuning your application and identifying any insufficient SQL statements in it. Read *Chapter 8, "Diagnostics and Troubleshooting,"* for additional information on how to use these tools.

Using Indexes to Improve Query Performance

You can use indexes to improve the performance of queries. A query that references an indexed column in its WHERE clause can use the index. If the query selects only the indexed column, the query can read the indexed column value directly from the index, rather than from the table.

If a table has a primary key, SOLID *Embedded Engine* orders the rows on disk in the order of the values of the primary key. Otherwise the rows are ordered using the ROWID, that is, the rows are stored on disk in the order they are inserted into the database. Read *"Primary Keys"* on page 3-9.

Searches with row value constructor constraints are optimized to use an index (if one is available). For efficiency, SOLID *Embedded Engine* uses an index to resolve constraints of the form (A, B, C) >= (1, 2, 3). The constraints <, <=, >= and > can use the index. Note that if several row value constraints are defined for one table, only the first one is optimized to use an index.

Indexes improve the performance of queries that select a small percentage of rows from a table. You should consider using indexes for queries that select less than 15% of table rows.

Full table scan

If a query does not use an index, SOLID *Embedded Engine* must perform a full table scan to execute the query. This involves reading all rows of a table sequentially. Each row is examined to determine whether it meets the criteria of the query's WHERE clause. Finding a single row with an indexed query can be substantially faster than finding the row with a full table scan. On the other hand, a query that selects more than 15% of a table's rows may be performed faster by a full table scan than by an indexed query.

To perform a full table scan, every block in the table is read. For each block, every row stored in the block is read. To perform an indexed query the rows are read in the order in which they appear in the index, regardless of which blocks contain them. If a block contains more than one selected row it may be read more than once. So, there are cases when a full table scan requires less I/O than an indexed query.

Concatenated indexes

An index can be made up of more than one column. Such an index is called a concatenated index. It is recommended to use concatenated indexes when possible.

Whether or not a SQL statement uses a concatenated index is determined by the columns contained in the WHERE clause of the SQL statement. A query can use a concatenated index if it references a leading portion of the index in the WHERE clause. A leading portion of an index refers to the first column or columns specified in the CREATE INDEX statement.

Example:

```
create index job_sal_deptno on emp(job, sal, deptno);
```

This index can be used by these queries:

```
select * from emp where job = ‘clerk’ and sal =
   800 and deptno = 20;
select * from emp where sal = 1250 and job = salesman;
select job, sal from emp where job = ‘manager’ ;
```

The following query does not contain the first column of the index in its WHERE clause and cannot use the index:

```
select * from emp where sal = 6000;
```

Choosing columns to index

The following list gives guidelines in choosing columns to index:

- index columns that are used frequently in WHERE clauses
- index columns that are used frequently to join tables
- index columns that are used frequently in ORDER BY clauses
- index columns that have few of the same values or unique values in the table.
- do not index small tables (tables that use only a few blocks) because a full table scan may be faster than an indexed query

- if possible choose a primary key that orders the rows in the most appropriate order
- if only one column of the concatenated index is used frequently in WHERE clauses, place that column first in the CREATE INDEX statement
- if more than one column in concatenated index is used frequently in WHERE clauses, place the most selective column first in the CREATE INDEX statement.

Optimizing Batch Inserts and Updates

You can optimize the speed for large batch inserts and updates to SOLID *Embedded Engine*. Following are guidelines for increasing speed:

1. Check that you are running the application with the AUTOCOMMIT mode set off.

   SOLID ODBC Driver's default setting is AUTOCOMMIT. This is the standard setting according to the ODBC specification. To set your application with AUTOCOMMIT off, call the SQLSetConnectOption function as in the following example:

   ```
   rc = SQLSetConnectOption
   (hdbc, SQL_AUTOCOMMIT, SQL_AUTOCOMMIT_OFF);
   ```

2. Do not use large transactions.

   It is recommended to COMMIT transactions after every 100-200 rows inserted.

3. Use stored procedures to further increase the speed of inserts.

   The use of stored procedures provides an additional 10-20% speed increase.

4. Lower the `MergeInterval` parameter.

   The `MergeInterval` parameter determines the frequency of writing data from SOLID *Embedded Engine*'s cache to disk. For large batch operations, set the value to a lower level. The parameter should be set to approximately 100 when running large inserts.

Number 1 and 2 of these guidelines are the most important actions you can take to increase the speed of batch inserts. The actual rate of insertions also depends on your hardware, on the amount of data per row and on the existing indices for the table.

Tuning Memory Allocation

Main memory is allocated dynamically according to system usage and the operating system environment. The basic element of the memory management system is a pool of central memory buffers of equal size. You can configure the amount and size of memory buffers to meet the demands of different application environments.

Tuning Your Operating System

Your operating system may store information in:

- real memory
- virtual memory
- expanded storage
- disk

Your operating system may also move information from one location to another. Depending on your operating system, this movement is called paging or swapping. Many operating systems page and swap to accommodate large amounts of information that do not fit into real memory. However, this takes time. Excessive paging or swapping can reduce the performance of your operating system and indicates that your system's total memory may not be large enough to hold everything for which you have allocated memory. You should either increase the amount of total memory or decrease the amount of database cache memory allocated.

Database Cache

The information used by SOLID *Embedded Engine* is stored either in memory or on disk. Since memory access is faster than disk access, it is desirable for data requests to be satisfied by access to memory rather than access to disk.

Database cache uses available memory to store information that is read from the hard disk. When an application next time requests this information, the data is read from memory instead of from the hard disk. The default value of cache depends on the platform used and can be changed by changing the `CacheSize` parameter. Increasing the value is recommended when there are several concurrent users.

The following values can be used as a starting point:

- a dedicated server with 16 MB RAM: Cachesize 4 MB
- a dedicated server with 32 MB RAM: Cachesize 10 MB
- a dedicated server with 64 MB RAM: Cachesize 30 MB

Note

You should increase the value of `Cachesize` carefully. If a value is too large, it leads to poor performance.

Sorting

SOLID *Embedded Engine* does all sorting by default in memory. The amount of memory used for sorting is determined by the parameter `SORTARRAYSIZE` in the `[SQL]` section. If the amount of data to be sorted does not fit into the allocated memory, you may want to increase the value of the parameter `SORTARRAYSIZE`. If there is not enough memory to increase the value of `SORTARRAYSIZE` you should activate external sort that stores intermediate information to disk.

The external disk sort is activated by adding the following section and parameters in the configuration file `solid.ini`:

```
[sorter]
TmpDir_1 = c:\tmp
```

Additional sort directories are added with similar definitions:

```
[sorter]
TmpDir_1 = c:\tmp
TmpDir_2 = d:\tmp
TmpDir_3 = e:\tmp
```

Defining more than one sorter temporary directory on separate physical disks significantly improves sort performance by balancing the I/O load to multiple disks.

Tuning CPU Concurrency Load

SOLID *Embedded Engine* contains shared data structures. Some actions on them are protected by mutual exclusion semaphores (mutexes) which allow only one thread access to a resource at a time. In a multi-CPU machine, it is possible that at some configuration and workload, one or more mutexed sections can become a bottleneck that limits the maximum CPU load.

To improve concurrency you can do the following:

- Run the server with more general purpose (worker) threads (for example, 40-80 threads may be reasonable on an NT machine running 400 clients). For details on setting threads, read *"Setting Threads for Processing (Srv section)"* on page 6-6.
- Increase the log file size from the 2048 default. A large blocksize for a log file speeds up logging considerably (especially under NT and UW-SCSI disks). A recommended setting is:

  ```
  [logging]
  BlockSize=32728
  ```

Note

Be sure to shut down the server and delete all old log files before changing the blocksize value.

Tuning I/O

The performance of many software systems is inherently limited by disk I/O. Often CPU activity must be suspended while I/O activity completes.

Distributing I/O

Disk contention occurs when multiple processes try to access the same disk simultaneously. To avoid this, move files from heavily accessed disks to less active disks until they all have roughly the same amount of I/O.

Follow these guidelines:

- use a separate disk for log files
- divide your database into several files and place each of these database files on a separate disk. Read *"Managing Database Files and Caching (IndexFile section)"* on page 6-2.
- consider using a separate disk for the external sorter

Setting the MergeInterval Parameter

SOLID *Embedded Engine*'s indexing system consists of two storage structures, the Bonsai Tree which stores new data in central memory and the storage server which stores more stable data. As the Bonsai Tree performs concurrency control, storing delete, insert, and update operations, as well as key values, it merges new commit data to the storage server as a highly-optimized batch insert. This offers significant I/O optimization and load balancing.

You can adjust the number of index inserts made in the database that causes the merge process to start by setting the following parameter in the General section of the `solid.ini file`. For example:

```
MergeInterval = 100
```

Normally the recommended setting is the default value, which is cache size dependent. The default is calculated dynamically from the cache size, so that only part of the cache is used

for the Bonsai Tree. If you change the merge interval, be sure that the cache is large enough to accommodate the Bonsai Tree.

Note

If the merge interval setting does not allow the Bonsai Tree to fit into cache, then it is flushed partially to the disk; this has an adverse affect on performance.

Tuning Checkpoints

Checkpoints are used to store a consistent state of the database quickly onto the disk.

Checkpoints affect:

- recovery time performance
- runtime performance

Frequent checkpoints can reduce the recovery time in the event of a system failure. If the checkpoint interval is small, then relatively few changes to the database are made between checkpoints and relatively few changes must be recovered.

Checkpoints cause SOLID *SynchroNet* to perform I/O, so they momentarily reduce the runtime performance. This overhead is usually small.

Using Optimizer Hints

Due to various conditions with the data, user query, and database, the SQL Optimizer is not always able to choose the best possible execution plan. For more efficiency, you may want to force a merge join because you know, unlike the Optimizer, that your data is already sorted.

Or sometimes specific predicates in queries cause performance problems that the Optimizer cannot eliminate. The Optimizer may be using an index that you know is not optimal. In this case, you may want to force the Optimizer to use one that produces faster results.

Optimizer hints is a way to have better control over response times to meet your performance needs. Within a query, you can specify directives or *hints* to the Optimizer, which it then uses to determine its query execution plan. Hints are detected through a pseudo comment syntax from SQL2.

Hints are available for:

- Selecting merge or nested loop join

- Using a fixed join order as given in the from list
- Selecting internal or external sort
- Selecting a particular index
- Selecting a table scan over an index scan
- Selecting sorting before or after grouping

You can place a hint(s) in a SQL statement as a static string, just after a SELECT, UPDATE, or DELETE keyword. Hints are not allowed after the INSERT keyword. The hint always follows the SQL statement that applies to it.

Table name resolution in optimizer hints is the same as in any table name in a SQL statement. When there is an error in a hint specification, then the whole SQL statement fails with an error message.

Hints are enabled and disabled using the following configuration parameter in `solid.ini`:

```
[Hints]
EnableHints=YES | NO
```

The default is set to **YES**.

Optimizer Hints Syntax

The syntax to specify an optimizer hint is:

--(* vendor (SOLID), product (Engine), option(hint)

--*hint* ***)--**

hint:=

```
[MERGE JOIN |
LOOP JOIN |
JOIN ORDER FIXED |
INTERNAL SORT |
EXTERNAL SORT |
INDEX [REVERSE] table_name.index_name |
PRIMARY KEY [REVERSE] table_name
FULL SCAN table_name |
[NO] SORT BEFORE GROUP BY]
```

Following is a description of the keywords and clauses used in the syntax:

Pseudo comment identifier

The pseudo comment prefix is followed by identifying information. You must specify the vendor as SOLID, product as Engine, and the option, which is the pseudo comment class name, as hint.

Hint

Hints always follow the SELECT, UPDATE, or DELETE keyword that applies to it.

Note

Hints are not allowed after the INSERT keyword.

Each subselect requires its own hint; for example, the following are valid uses of hints syntax:

INSERT INTO ... SELECT *hint* FROM ...

UPDATE *hint* TABLE ... WHERE *column* = (SELECT *hint* ... FROM ...)

DELETE *hint* TABLE ... WHERE *column* = (SELECT hint ... FROM ...)

Example 1

```
SELECT
--(* vendor(SOLID), product(Engine), option(hint)
--MERGE JOIN
--JOIN ORDER FIXED *)--
*
FROM TAB1 A, TAB2 B;
WHERE A.INTF = B.INTF;
```

Example 2

```
SELECT
--(* vendor(SOLID), product(Engine), option(hint)
--INDEX TAB1.INDEX1
--INDEX TAB1.INDEX1 FULL SCAN TAB2 *)--
*
FROM TAB1, TAB2
```

```
WHERE TAB1.INTF = TAB2.INTF;
```

Hint is a specific semantic, corresponding to a specific behavior. Following is a list of possible hints:

Hint	Definition
MERGE JOIN	Directs the Optimizer to choose the merge join access plan in a select query for all tables listed in the FROM clause. Use this hint when the data is sorted by a join key and the nested loop join performance is not adequate. Note that when data is not sorted before performing the merge operation, the SOLID query executor sorts the data. When considering the usage of this hint, keep in mind that the merge join with a sort is more resource intensive than the merge join without the sort.
LOOP JOIN	Directs the Optimizer to pick the nested loop join in a select query for all tables listed in the FROM clause. By default, the Optimizer does not pick the nested loop join. Using the loop join when tables are small and fit in memory may offer greater efficiency than using other join algorithms.
JOIN ORDER FIXED	Specifies that the Optimizer use tables in a join in the order listed in the FROM clause of the query. This means that the Optimizer does not attempt to rearrange any join order and does not try to find alternate access paths to complete the join. Before using this hint, be sure to run the EXPLAIN PLAN to view the associated plan. This gives you an idea on the access plan used for executing the query with this join order.
INTERNAL SORT	Specifies that the query executor use the internal sort. Use this hint if the expected result set is small (100s of rows as opposed to 1000s of rows); for example, if you are performing some aggregates, ORDER BY with small result sets, or GROUP BY with small result sets, etc. This hint avoids the use of the more expensive external sort.
EXTERNAL SORT	Specifies that the query executor use the external sort. Use this hint when the expected result set is large and does not fit in memory; for example, if the expected result set has 1000s of rows. In addition, specify the SORT working directory in the `solid.ini` before using the external sort hint. If a working directory is not specified, you will receive a run-time error.

Hint	Definition
INDEX {REVERSE] *table_name.index_name*	Forces a given index scan for a given table. In this case, the Optimizer does not proceed to evaluate if there are any other indexes that can be used to build the access plan or whether a table scan is better for the given query. Before using this hint, it is recommended that you run the EXPLAIN PLAN output to ensure that the plan generated is optimal for the given query. The optional keyword **REVERSE** returns the rows in the reverse order. In this case, the query executor begins with the last page of the index and starts returning the rows in the descending (reverse) key order of the index. Note that in *tablename.indexname*, the *tablename* is a fully qualified table name which includes the *catalogname* and *schemaname*.
PRIMARY KEY [REVERSE] *tablename*	Forces a primary key scan for a given table. The optional keyword **REVERSE** returns the rows in the reverse order. If the primary KEY is not available for the given table, then you will receive a run-time error.
FULL SCAN *table_name*	Forces a table scan for a given table. In this case, the optimizer does not proceed to evaluate if there are any other indexes that can be used to build the access plan or whether a table scan is better for the given query. Before using this hint, it is recommended that you run the EXPLAIN PLAN output to ensure that the plan generated is optimal for the given query. In this FULL SCAN, the query executor tries to use the PRIMARY KEY, if one is available. If not, then it uses the SYSTEM KEY.
[NO] SORT BEFORE GROUP BY	Indicates whether the SORT operation occurs before the result set is grouped by the GROUP BY columns. If the grouped items are few (100s of rows) then use NO SORT BEFORE. On the other hand, if the grouped items are large (1000s of rows), then use SORT BEFORE.

Optimizer Hint Examples

```
SELECT
--(* vendor(SOLID), product(Engine), option(hint)
-- INDEX TAB1.IDX1 *)--
```

```
* FROM TAB1 WHERE I > 100

SELECT
--(* vendor(SOLID), product(Engine), option(hint)
-- INDEX MyCatalog.mySchema.TAB1.IDX1 *)--
* FROM TAB1 WHERE I > 100

SELECT
--(* vendor(SOLID), product(Engine), option(hint)
-- JOIN ORDER FIXED *)--
* FROM TAB1, TAB2 WHERE TAB1.I >= TAB2.I

SELECT
--(* vendor(SOLID), product(Engine), option(hint)
-- LOOP JOIN *)--
* FROM TAB1, TAB2 WHERE TAB1.I >= TAB2.I

SELECT
--(* vendor(SOLID), product(Engine), option(hint)
-- INDEX REVERSE MyCatalog.mySchema.TAB1.IDX1 *)--
* FROM TAB1 WHERE I > 100

SELECT
--(* vendor(SOLID), product(Engine), option(hint)
-- SORT BEFORE GROUP BY *)--
AVG(I) FROM TAB1 WHERE I > 10 GROUP BY I2

SELECT
--(* vendor(SOLID), product(Engine), option(hint)
-- INTERNAL SORT *)--
* FROM TAB1 WHERE I > 10 ORDER BY I2
```

8

Diagnostics and Troubleshooting

This chapter provides information on the following SOLID *Embedded Engine* diagnostic tools:

- SQL info facility and the EXPLAIN PLAN statement used to tune your application and identify inefficient SQL statements in your application.
- Network trace facility used to trace the server communication
- Ping facility used to trace client communication

You can use these facilities to observe performance, troubleshooting, and produce high quality problem reports. These reports lets you pinpoint the source of your problems by isolating them under product categories (such as SOLID *ODBC API*, SOLID *ODBC Driver*, SOLID *JDBC Driver*, etc.).

Observing Performance

You can use the SQL Info facility to provide information on a SQL statement and the SQL statement EXPLAIN PLAN to show the execution graph that the SQL optimizer selected for a given SQL statement. Typically, if you need to contact SOLID Technical support, you will be asked to provide the SQL statement, EXPLAIN PLAN output, and SQL Info output from the EXPLAIN PLAN run with info level 8 for more extensive trace output.

SQL Info Facility

Run your application with the SQL Info facility enabled. The SQL Info facility generates information for each SQL statement processed by SOLID *Embedded Engine*.

The `Info` parameter in the `[SQL]` section specifies the tracing level on the SQL parser and optimizer as an integer between 0 (no tracing) and 8 (SOLID info from every fetched row). Trace information will be output to the file named `soltrace.out` in the SOLID directory.

Example:

```
[SQL]
info = 1
```

SQL Info levels

Info value	Information
0	no output
1	table, index, and view info in SQL format
2	SQL execution graphs (for SOLID technical support use only)
3	some SQL estimate info, SOLID selected key name
4	all SQL estimate info, SOLID selected key info
5	SOLID info also from discarded keys
6	SOLID table level info
7	SQL info from every fetched row
8	SOLID info from every fetched row

The SQL Info facility can also be turned on with the following SQL statement (this sets SQL Info on only for the client that executes the statement):

`SET SQL INFO ON LEVEL` *`info_value`* `FILE` *`file_name`*

and turned off with the following SQL statement:

```
SET SQL INFO OFF
```

Example:

```
SET SQL INFO ON LEVEL 1 FILE 'my_query.txt'
```

The EXPLAIN PLAN Statement

The syntax of the EXPLAIN PLAN statement is:

`EXPLAIN PLAN FOR` *`sql_statement`*

The EXPLAIN PLAN statement is used to show the execution plan that the SQL optimizer has selected for a given SQL statement. An execution plan is a series of primitive operations, and an ordering of these operations, that SOLID *Embedded Engine* performs to execute the statement. Each operation in the execution plan is called a unit.

Unit	Description
JOIN UNIT*	Join unit joins two or more tables. The join can be done by using loop join or merge join.
TABLE UNIT	Table unit is used to fetch the data rows from a table. Table unit is always the last unit in the chain, since it is responsible for fetching the actual data from the index or table.
ORDER UNIT	Order unit is used to order rows for grouping or to satisfy ORDER BY. The ordering can be done in memory or using an external disk sorter.
GROUP UNIT	Group unit is used to do grouping and aggregate calculation (SUM, MIN, etc.).
UNION UNIT*	Union unit performs the UNION operation. The unit can be done by using loop join or merge join.
INTERSECT UNIT*	Intersect unit performs the INTERSECT operation. The unit can be done by using loop join or merge join.
EXCEPT UNIT*	Except unit performs the EXCEPT operation. The unit can be done by using loop join or merge join.

*This unit is generated also for queries that reference only a single table. In that case no join is executed in the unit; it simply passes the rows without manipulating them.

Explain Plan Table Columns

The table returned by the EXPLAIN PLAN statement contains the following columns.

Column Name	Description
ID	The output row number, used only to guarantee that the rows are unique.
UNIT_ID	This is the internal unit id in the SQL interpreter. Each unit has a different id. The unit id is a sparse sequence of numbers, because the SQL interpreter generates unit ids also for those units that are removed during the optimization phase. If more than one row has the same unit id it means that those rows belong to the same unit. For formatting reasons the info from one unit may be divided into several different rows.
PAR_ID	Parent unit id for the unit. The parent id number refers to the id in the UNIT_ID column.

Column Name	Description
JOIN_PATH	For join, union, intersect, and except units there is a join path which specifies which tables are joined in the unit and the join order for tables. The join path number refers to the unit id in the UNIT_ID column. It means that the input to the unit comes from that unit. The order in which the tables are joined is the order in which the join path is listed. The first listed table is the outer-most table in a loop join.
UNIT_TYPE	Unit type is the execution graph unit type.
INFO	Info column gives additional info. It may contain, for example, index usage, the database table name and constraints used in the database engine to select rows. Note that the constraints listed here may not match those constraints given in the SQL statement.

The following texts may exist in the INFO column for different types of units.

Unit type	Text in Info column	Description
TABLE UNIT	*tablename*	The table unit refers to table *tablename*.
TABLE UNIT	*constraints*	The constraints that are passed to the database engine are listed. If for example in joins the constraint value is not known in advance, the constraint value is displayed as NULL.
TABLE UNIT	SCAN TABLE	Full table scan is used to search for rows.
TABLE UNIT	SCAN *indexname*	Index *indexname* is used to search for rows. If all selected columns are found from an index, sometimes it is faster to scan the index instead of the entire table because the index has fewer disk blocks.
TABLE UNIT	PRIMARY KEY	The primary key is used to search rows. This differs from SCAN in that the whole table is not scanned because there is a limiting constraint to the primary key attributes.
TABLE UNIT	INDEX *indexname*	Index *indexname* is used to search for rows. For every matching index row, the actual data row is fetched separately.

Unit type	Text in Info column	Description
TABLE UNIT	INDEX ONLY *indexname*	Index *indexname* is used to search for rows. All selected columns are found from the index, so the actual data rows are not fetched separately
JOIN UNIT	MERGE JOIN	Merge join is used to join the tables.
JOIN UNIT	3-MERGE JOIN	A 3-merge join is used to merge the tables.
JOIN UNIT	LOOP JOIN	Loop join is used to join the tables.
ORDER UNIT	NO ORDERING REQUIRED	No ordering is required, the rows are retrieved in correct order from the database engine.
ORDER UNIT	EXTERNAL SORT	External sorter is used to sort the rows. To enable external sorter, the temporary directory name must be specified in the Sorter section of the configuration file.
ORDER UNIT	FIELD *n* USED AS PARTIAL ORDER	For distinct result sets, an internal sorter (in-memory sorter) is used for sorting and the rows retrieved from the database engine are partially sorted with column number *n*. The partial ordering helps the internal sorter avoid multiple passes over the data.
ORDER UNIT	*n* FIELDS USED FOR PARTIAL SORT	An internal sorter (in-memory sorter) is used for sorting and the rows retrieved from the database engine are partially sorted with n fields. The partial ordering helps the internal sorter to avoid multiple passes over the data.
ORDER UNIT	NO PARTIAL SORT	Internal sorter is used for sorting and the rows are retrieved in random order from the database engine.
UNION UNIT	MERGE JOIN	Merge join is used to join the tables.
UNION UNIT	3-MERGE JOIN	A 3-merge join is used to merge the tables.
UNION UNIT	LOOP JOIN	Loop join is used to join the tables.
INTERSECT UNIT	MERGE JOIN	Merge join is used to join the tables.
INTERSECT UNIT	3-MERGE JOIN	A 3-merge join is used to merge the tables.

Unit type	Text in Info column	Description
INTERSECT UNIT	LOOP JOIN	Loop join is used to join the tables.
EXCEPT UNIT	MERGE JOIN	Merge join is used to join the tables.
EXCEPT UNIT	3-MERGE JOIN	A 3-merge join is used to merge the tables.
EXCEPT UNIT	LOOP JOIN	Loop join is used to join the tables.

Example 1

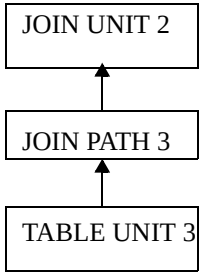
```
EXPLAIN PLAN FOR SELECT * FROM TENKTUP1 WHERE UNIQUE2_NI BETWEEN 0 AND
99;
```

ID	UNIT_ID	PAR_ID	JOIN_PATH	UNIT_TYPE	INFO
1	2	1	3	JOIN UNIT	
2	3	2	0	TABLE UNIT	TENKTUP1
3	3	2	0		FULL SCAN
4	3	2	0		UNIQUE2_NI <= 99
5	3	2	0		UNIQUE2_NI >= 0
6	3	2	0		

Execution graph:

JOIN UNIT 2 gets input from TABLE UNIT 3

TABLE UNIT 3 for table TENKTUP1 does a full table scan with constraints UNIQUE2_NI <= 99 and UNIQUE2_NI >= 0



Example 1. Execution graph

Example 2

```
EXPLAIN PLAN FOR SELECT * FROM TENKTUP1, TENKTUP2 WHERE TENKTUP1.UNIQUE2
> 4000 AND TENKTUP1.UNIQUE2 < 4500 AND TENKTUP1.UNIQUE2 =
TENKTUP2.UNIQUE2;
```

ID	UNIT_ID	PAR_ID	JOIN_PATH	UNIT_TYPE	INFO
1	6	1	9	JOIN UNIT	MERGE JOIN
2	6	1	10		
3	9	6	0	ORDER UNIT	NO ORDER-ING REQUIRED
4	8	9	0	TABLE UNIT	TENKTUP2
5	8	9	0		PRIMARY KEY
6	8	9	0		UNIQUE2 < 4500
7	8	9	0		UNIQUE2 > 4000
8	8	9	0		
9	10	6	0	ORDER UNIT	NO ORDER-ING REQUIRED
10	7	10	0	TABLE UNIT	TENKTUP1
11	7	10	0		PRIMARY KEY
12	7	10	0		UNIQUE2 < 4500
13	7	10	0		UNIQUE2 > 4000
14	7	10	0		

Execution graph:

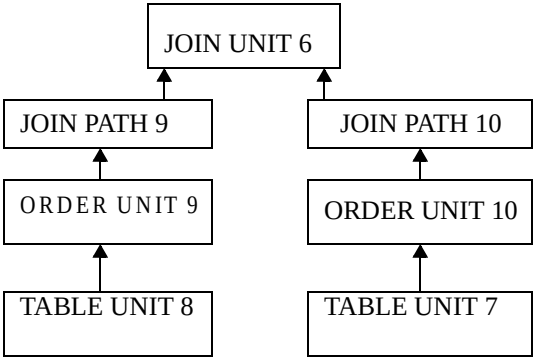
JOIN UNIT 6 the input from order units 9 and 10 are joined using merge join algorithm

ORDER UNIT 9 orders the input from TABLE UNIT 8. Since the data is retrieved in correct order, no real ordering is needed

ORDER UNIT 10 orders the input from TABLE UNIT 7. Since the data is retrieved in correct order, no real ordering is needed

TABLE UNIT 8: rows are fetched from table TENKTUP2 using primary key. Constraints UNIQUE2 < 4500 and UNIQUE2 > 4000 are used to select the rows

TABLE UNIT 7: rows are fetched from table TENKTUP1 using primary key. Constraints UNIQUE2 < 4500 and UNIQUE2 > 4000 are used to select the rows



Example 2. Execution graph

Tracing Communication between Client and Server

SOLID *Embedded Engine* provides the following tools for observing the communication between an application and a database server:

- the Network Trace facility
- the Ping facility

You can use these tools to analyze the functionality of the networking between an application and an embedded engine. The network trace facility should be used when you want to know why a connection is not established to an embedded engine. The ping facility is used to determine how fast packets are transferred between an application and a database server.

The Network Trace Facility

Network tracing can be done on the *Embedded Engine* computer, on the application computer or on both computers concurrently. The trace information is written to the default trace file or file specified in the `TraceFile` parameter.

The default name of the output file is `soltrace.out`. This file will be written to the current working directory of the server or client depending on which end the tracing is started.

The file contains information about:

- loaded DLLs
- network addresses
- possible errors

The Network Trace facility is turned on by editing the configuration file:

```
[Com]
Trace ={Yes|No}
; default No
TraceFile = file_name
; default soltrace.out
```

or by using the environment variables `SOLTRACE` and `SOLTRACEFILE` to override the definitions in the configuration file. Setting of `SOLTRACE` and `SOLTRACEFILE` environment variables have the same effect as the parameters `Trace` and `TraceFile` in the configuration file.

Note

Defining the `TraceFile` configuration parameter or the `SOLTRACEFILE` environment variable automatically turns on the Network trace facility.

A third alternative to turn on the Network trace facility is to use the option -t and/or -o*filename* as a part of the network name. The option -t turns on the Network trace facility. The option -o turns on the facility and defines the name of the trace output file.

Example 1. Defining Parameter Trace in the Configuration File

```
[Com]
Connect = nmp SOLID
Listen = nmp SOLID
Trace = Yes
```

Example 2. Defining Environment Variables

```
set SOLTRACE = Yes
```

or

```
set SOLTRACEFILE = trace.out
```

Example 3. Using Network Name Options

```
[Com]
Connect  = nmp -t solid
Listen = nmp -t solid
```

or

```
[Com]
Connect = nmp -oclient.out solid
Listen = nmp -oserver.out solid
```

The Ping Facility

The Ping facility can be used to test the performance and functionality of the networking. The Ping facility is built in all SOLID client applications and is turned on with the network name option -p*level*.

The output file will be written to the current working directory of the computer where the parameter is given. The default name of the output file is `soltrace.out`.

Clients can always use the Ping facility at level 1. Levels 2, 3, 4 or 5 may only be used if the server is set to use the Ping facility at least at the same level.

The Ping facility levels are:

Setting	Function	Description
0	no operation	do nothing, default
1	check that server is alive	exchange one 100 byte message
2	basic functional test	exchange messages of sizes 0.1K, 1K, 2K..30K, increment 1K
3	basic speed test	exchange 100 messages of sizes 0.1K, 1K, 8K and display each sub-result and total time
4	heavy speed test	exchange 100 messages of sizes 0.1K, 1K, 2K, 4K, 8K, 16K and display each sub-result and total time

Setting	Function	Description
5	heavy functional test	exchange messages of sizes 1..30K, increment 1 byte

Note

If a SOLID client does not have an existing server connection, you can use the **SQLConnect()** function with the connect string **-p1** option (ping test, level 1) to check if SOLID is listening in a certain address. Without logging into SOLID, **SQLConnect()** can then check the network layer and ensure SOLID is listening. When used in this manner, **SQLConnect()** generates error code 21507, which means the server is alive.

Example 1

The client turns on the Ping facility by using the following network name:

```
nmp -p1 -oping.out SOLID
```

This runs the Ping facility at the level 1 into a file named SOLTRACE.OUT. This test checks if the server is alive and exchanges one 100 byte message to the server.

After the Ping facility has been run, the client exits with the following message:

```
SOLID Communication return code xxx: Ping test successful/failed,
results are in file FFF.XX
```

Example 2

If the server is using the following listen parameter, applications can run the Ping facility at levels 1,2 and 3, but not 4 and 5.

```
[Com]
Listen = nmp -p3 SOLID
```

Note

Ping clients running at level greater than 3 may cause heavy network traffic and may cause slowness of application using the network. They will also slow down ordinary SQL clients connected to the same SOLID *Embedded Engine*.

Problem Reporting

SOLID *Embedded Engine* offers sophisticated diagnostic tools and methods for producing high quality problem reports with very limited effort. Use the diagnostic tools to capture all the relevant information about the problem.

All problem reports should contain the following files and information:

- solid.ini
- license number
- solmsg.out
- solerror.out
- soltrace.out
- problem description
- steps to reproduce the problem
- all error messages and codes
- contact information, preferably email address of the contact person

Problem Categories

Most problems can be divided into the following categories:

- SOLID *ODBC API*
- SOLID *ODBC* or *JDBC Driver*
- UNIFACE driver for SOLID *Embedded Engine*
- Communication problems between the application and SOLID *Embedded Engine*

The following pages include a detailed instructions to produce proper problem report for each problem type. Please follow the guidelines carefully.

SOLID *ODBC API* Problems

If the problem concerns the performance of SOLID *ODBC API* or a specific SQL statement, you should run SQL info facility at level 4 and include the generated `soltrace.out` file into your problem report. This file contains the following information:

- create table statements
- create view statements

- create index statements
- SQL statement(s)

SOLID *ODBC Driver* Problems

If the problem concerns the performance of SOLID *ODBC Driver*, please include the following information:

- SOLID *ODBC Driver* name, version, and size
- ODBC Driver Manager version and size

If the problem concerns the cooperation of SOLID *Embedded Engine* and any third party standard software package, please include the following information:

- Full name of the software
- Version and language
- Manufacturer
- Error messages from the third party software package

Use ODBC trace option to get a log of the ODBC statements and include it to your problem report.

SOLID *JDBC Driver* Problems

If the problem is related to the SOLID *JDBC Driver*, please include the following information into your problem report:

- Exact version of JDK or JRK used
- Size and date of the SOLIDDriver class package
- Contents of DriverManager.setLogStream(someOutputStream) output, if available
- Call stack (that is, Exception.printStackTract() output) of the application, if an Exception has occurred n the application

UNIFACE Driver for SOLID *Embedded Engine* Problems

If the problem concerns the performance of SOLID UNIFACE Driver, please include following information:

- SOLID *UNIFACE Driver* version and size
- UNIFACE version and platform

- Contents of the UNIFACE message frame
- Error codes from the driver, $STATUS, $ERROR
- All necessary files to reproduce the problem (TRXs, SQL scripts, USYS.ASN etc.)

Communication between a Client and Server

If the problem concerns the performance of the communication between a client and server use the Network trace facility and include the generated trace files into your problem report. Please include the following information:

- SOLID communication DLLs used: version and size
- other communication DLLs used: version and size
- description of the network configuration

A
Error Codes

Error Categories

SQL Errors

These errors are caused by erroneous SQL statements and are detected by the SOLID SQL Parser. Administrative actions are not needed.

Database Errors

These errors are detected by the SOLID *Embedded Engine* and may demand administrative actions.

Executable Errors

These errors are caused by the failure of a SOLID *Embedded Engine* executable or a command line argument related error. They enable implementing intelligent error handling logic in system startup scripts.

System Errors

These errors are detected by the operating system and demand administrative actions.

Table Errors

These errors are caused by erroneous SQL statements and detected by SOLID *Embedded Engine*. Administrative actions are not needed.

Server Errors

These errors are caused by erroneous administrative actions or client requests. They may demand administrative actions.

Communication Errors

These errors are caused by network errors or faulty configuration of the SOLID *Embedded Engine* software. These errors demand administrative actions.

Procedure Errors

These errors are caused by errors in the definition or execution of a stored procedure. Administrative actions are not needed.

Sorter Errors

These errors are caused by external sorter algorithm errors when solving queries that require ordering rows.

SOLID SQL Errors

Error code	Description
SQL Error 1	Parsing error ‘syntax error’ The SQL parser could not parse the SQL string. Check the syntax of the SQL statement and try again.
SQL Error 2	Table *table* can not be opened You may not have privileges to access the table and its data.
SQL Error 3	Table *table* can not be created Table can not be created. You may not have privileges for this operation.
SQL Error 4	Illegal type definition *column* A column type in your CREATE TABLE statement is illegal. Use a legal type for the column.
SQL Error 5	Table table can not be dropped Table can not be dropped. Only the owner (that is, the creator) can drop it.
SQL Error 6	Illegal value specified for column *column* The value specified for column is invalid. Check the value for the column.

Error code	Description
SQL Error 7	Insert failed The server failed to do the insertion. You may not have INSERT privilege on the table or it may be locked.
SQL Error 8	Delete failed The server failed to do the deletion. You may not have DELETE privilege on the table or the row may be locked.
SQL Error 9	Row fetch failed The server failed to fetch a row. You may not have SELECT privilege on the table or there may be an exclusive lock on the row.
SQL Error 10	View *view* can not be created You cannot create this view. You may not have SELECT privilege on one or more tables in the query-specification of your CREATE VIEW statement.
SQLError11	View *view* cannot be dropped. You cannot drop this view. Only the owner (i.e. the creator) of the view can drop it.
SQLError12	Illegal view definition *view* The view definition is illegal. Check the syntax of the definition.
SQLError13	Illegal column name *column* Column name is illegal. Check that the name is not a reserved name.
SQL Error 14	Call to function *function* failed Function call to function failed. Check the arguments and their types.
SQL Error 15	Arithmetic error An arithmetic error occurred. Check the operators, values and types.
SQL Error 16	Update failed The server failed to update a row. There may a lock on a row.

Error code	Description
SQL Error 17	View is not updatable This view is not updatable. UPDATE, INSERT and DELETE operations are not allowed.
SQL Error 18	Inserted row does not meet check option condition You tried to insert a row, but one or more of the column values do not meet column constraint definition.
SQL Error 19	Updated row does not meet check option condition You tried to update a row, but one or more of the column values do not meet column constraint definition.
SQL Error 20	Illegal CHECK constraint A check constraint given to the table is illegal. Check the types of the check constraint of this table.
SQL Error 21	Insert failed because of CHECK constraint You tried to insert a row, but the values do not meet the check option conditions.
SQL Error 22	Update failed because of CHECK constraint You tried to update a row, but the values do not meet the check option conditions.
SQL Error 23	Illegal DEFAULT value The DEFAULT value for the column given is illegal.
SQL Error 25	Duplicate columns in INSERT column list You have included a column in column list twice. Remove duplicate columns.
SQL Error 26	At least one column definition required in CREATE TABLE You need to specify at least one column definition in a CREATE TABLE statement.
SQL Error 27	Illegal REFERENCES column list There are wrong number of columns in your REFERENCES list.

Error code	Description
SQL Error 28	Only one PRIMARY KEY allowed in CREATE TABLE You can use only one PRIMARY KEY in CREATE TABLE.
SQL Error 29	GRANT failed Granting privileges failed. You may not have privileges for this operation.
SQL Error 30	REVOKE failed Revoking privileges failed. You may not have privileges for this operation.
SQL Error 31	Multiple instances of a privilege type You tried to grant privileges to a role or a user. You have included multiple instances of a privilege type in the list of privileges.
SQL Error 32	Illegal constant *constant* Illegal constant was found. Check the syntax of the statement.
SQL Error 33	Column name list of illegal length You have entered different number of columns in CREATE VIEW statement to the view and to the table.
SQL Error 34	Conversion between types failed An expression in UPDATE statement has illegal type for a column.
SQL Error 35	Column names not allowed in ORDER BY for UNION You can not use column name in an ORDER BY for UNION statement.
SQL Error 36	Nested aggregate functions Nested aggregate functions can not be used. For example: SUM(AVG(*column*)).
SQL Error 37	Aggregate function with no arguments An aggregate function was entered with no arguments. For example: SUM().

Error code	Description
SQL Error 38	Set operation between different row types You have tried to execute a set operation of tables with incompatible row types. The row types in a set operation must be compatible.
SQL Error 39	COMMIT WORK failed Committing a transaction failed.
SQL Error 40	ROLLBACK WORK failed Rolling back a transaction failed.
SQL Error 41	Savepoint could not be created A savepoint could not be created.
SQL Error 42	Could not create index *index* An index could not be created. You may not have privileges for this operation. You need to be an owner of the table or have SYS_ADMIN_ROLE to have privileges to create index for the table.
SQL Error 43	Could not drop index *index* An index could not be dropped. You may not have privileges for this operation. You need to be an owner of the table or have SYS_ADMIN_ROLE to have privileges to drop index from the table.
SQL Error 44	Could not create schema *schema* A schema could not be created.
SQL Error 45	Could not drop schema *schema* A schema could not be dropped.
SQL Error 46	Illegal ORDER BY specification You tried to use an ORDER BY column that does not exist. Refer to an existing column in the ORDER BY specification.
SQL Error 47	Maximum length of identifier is 31 You have exceeded the maximum length for the identifier.

Error code	Description
SQL Error 48	Subquery returns more than one row You have used a subquery that returns more than one row. Only subqueries returning one row may be used in this situation.
SQL Error 49	Illegal expression *expression* You tried to insert or update a table using an aggregate function (SUM, MAX, MIN or AVG) as a value. This is not allowed.
SQL Error 50	Ambiguous column name *column* You have referenced a column which is exists in more than one table. Use syntax *table.column* to indicate which table you want to use.
SQL Error 51	Non-existent function *function* You tried to use a function which does not exist.
SQL Error 52	Non-existent cursor *cursor* You tried to use a cursor which is not created.
SQL Error 53	Function call sequence error A function was called in wrong order. Check the sequence and success of the function calls.
SQL Error 54	Illegal use of a parameter A parameter was used illegally. For example: SELECT * FROM TEST WHERE ? < ?;
SQL Error 55	Illegal parameter value A parameter has an illegal value. Check the type and value of the parameter.
SQL Error 56	Only ANDs and simple condition predicates allowed in UPDATE CHECK All search condition predicates are not supported.
SQL Error 57	Opening the cursor did not succeed Server failed to open a cursor. You may not have cursor open at this moment.

Error code	Description
SQL Error 58	Column *column* is not referenced in group-by-clause You tried to group rows using *column*. All columns in *group_by_clause* must be listed in your *select_list*. A star ('*') notation is not allowed with GROUP BY.
SQL Error 59	Comparison between incompatible types You tried to compare values which have incompatible types. Incompatible types are for example an integer and a date value.
SQL Error 60	Reference to the insert table not allowed in the source query You have referenced in subquery a table where you are inserting values. This is not allowed.
SQL Error 61	Reference to the update table not allowed in subquery You have referenced in subquery a table where you are updating values. This is not allowed.
SQL Error 62	Reference to the delete table not allowed in subquery You have referenced in subquery a table where you are deleting values. This is not allowed.
SQL Error 63	Subquery returns more than one column You have used a subquery that returns more than one column. Only subqueries returning one column may be used.
SQL Error 64	Cursor *cursor* not updatable The cursor opened is not updatable.
SQL Error 65	Insert or update tried on pseudo column You tried to update a pseudo column (ROWID, ROWVER). Pseudo columns are not updatable.
SQL Error 66	Could not create user *user* A user could not be created. You may not have privileges for this operation.
SQL Error 67	Could not alter user *user* A user could not be altered. You may not have privileges for this operation.

Error code	Description
SQL Error 68	Could not drop user *user* A user could not be dropped. You may not have privileges for this operation.
SQL Error 69	Could not create role *role* A role could not be created. You may not have privileges for this operation.
SQL Error 70	Could not drop role *role* A role could not be dropped. You may not have privileges for this operation.
SQL Error 71	Grant *role* failed Granting role failed. You may not have privileges for this operation.
SQL Error 72	Revoke *role* failed Revoking role failed. You may not have privileges for this operation.
SQL Error 73	Comparison of vectors of different length You have tried to compare row value constructors that have different number of dimensions. For example you have compared (a,b,c) to (1,1).
SQL Error 74	Expression * not compatible with aggregate expression The aggregate expression can not be used with * columns. Specify columns using their names when used with this aggregate expression. This usually happens when GROUP BY expression is used with the * columns.
SQL Error 75	Illegal reference to table *table* You have tried to reference a table which is not in the FROM list. For example: SELECT T1.* FROM T2.
SQL Error 76	Ambiguous table name *table* You have used the syntax *table.column name* ambiguously. For example: SELECT T1.* FROM T1 A,T1 B WHERE A.F1=0;

Error code	Description
SQL Error 77	Illegal use of aggregate expression You tried to use aggregate expression illegally. For example: SELECT ID FROM TEST WHERE SUM(ID) = 3;
SQL Error 78	Row fetch failed The server failed to fetch a row. You may not have SELECT privilege on the table or there may be an exclusive lock on the row.
SQL Error 79	Subqueries not allowed in CHECK constraint You tried to use subquery in a check constraint.
SQL Error 80	Sorting failed External sorter is out of disk space or cache memory. Modify parameters in configuration file `solid.ini`.
SQL Error 81	SET syntax results in error
SQL Error 82	Improper type used with LIKE
SQL Error 83	Syntax error
SQL Error 84	Parser error *statement*
SQL Error 85	Incorrect number of values for INSERT
SQL Error 86	Illegal ROWNUM constraint
SQL Error 88	Subquery not allowed in UPDATE expression Subqueries cannot be used in UPDATE statements.
SQL Error 93	Illegal GROUP BY expression GROUP BY expression is illegal.

SOLID Database Errors

Error code	Description
Database Error 10001	Key value is not found. Internal error: a key value cannot be found from the database index.
Database Error 10002	Operation failed. This is an internal error indicating that the index of the table accessed is in inconsistent state. Try to drop and create the index again to recover the error.
Database Error 10004	Redefinition. Unexpected failure occurred in the database engine. This error may also occur during recovery: either an index or a view has been redefined during recovery. The server is not able to do the recovery. Delete log files and start the server again.
Database Error 10005	Unique constraint violation. You have violated a unique constraint. This happens when you have tried to insert or update a column which has a unique constraint and the value inserted or updated is not unique. This may also occur when you create users, tables or roles having same names in separate transactions.
Database Error 10006	Concurrency conflict, two transactions updated or deleted the same row. Two separate transactions have modified a same row in the database simultaneously. This has resulted in a concurrency conflict.
Database Error 10007	Transaction is not serializable. The transaction committed is not serializable.
Database Error 10010	No checkpoint in database. This error occurs when the server has crashed in the middle of creating a new database. Delete the database and log files and try to create the database again.

Error code	Description
Database Error 10011	Database headers are corrupted. The headers in the database are corrupted. This may be caused by a disk error or other system failure. Restore the database from the backup.
Database Error 10012	Node split failed. This is an internal error.
Database Error 10013	Transaction is read-only. You have tried to write inside a transaction that is set read-only. Remove the write operation or unset the read-only mode in the transaction.
Database Error 10014	Resource is locked. This error occurs when you are trying to use a key value in an index which has been concurrently dropped.
Database Error 10016	Log file is corrupted. One of the log files of the database is corrupted. You can not use these log files. Delete them and start the server again.
Database Error 10017	Too long key value. The maximum length of the key value has been exceeded. The maximum value is one third of the size of the index leaf.
Database Error 10019	Backup is active You have tried to start a backup when a backup process is already in progress.
Database Error 10020	Checkpoint creation is active. You have tried to start a checkpoint when a checkpoint creation is already in progress.
Database Error 10021	Failed to delete log file. The deletion of a log file in making a backup has failed. Reasons for the failure can be: ■ The log file has already been deleted from the operating system. ■ The log file has a read-only attribute.

Error code	Description
Database Error 10023	Wrong log file, maybe the log file is from another database. The log file in the database directory is from another SOLID *Embedded Engine* database. Copy the correct log files to the database directory. The log file in the database directory is from another SOLID *Embedded Engine* database. Copy the correct log files to the database directory.
Database Error 10024	Illegal backup directory. The backup directory is either an empty string or a dot indicating that the backup will be created in the current directory.
Database Error 10026	Transaction is timed out. An idle transaction has exceeded the maximum idle transaction time. The transaction has been aborted. The maximum value is set in parameter AbortTimeOut in SRV section. The default value is 120 minutes.
Database Error 10027	No active search. Internal error.
Database Error 10028	Referential integrity violation, foreign key values exist. You tried to delete a row that is referenced from a foreign key.
Database Error 10029	Referential integrity violation, referenced column values do not exist. The definition of a foreign key does not uniquely identify a row in the referenced table.
Database Error 10030	Backup directory '*directory name*' does not exist. Backup directory is not found. Check the name of the backup directory.
Database Error 10031	Transaction detected a deadlock, transaction is rolled back. Deadlock detected. If necessary, begin transaction again.

Error code	Description
Database Error 10032	Wrong database block size specified. The block size of the database file differs from the block-size given in the configuration file `solid.ini`.
Database Error 10033	Primary key unique constraint violation. Your primary key definition is not unique.
Database Error 10034	Sequence name *sequence* conflicts with an existing entity. Choose a unique name for a sequence. The specified name is already used.
Database Error 10035	Sequence does not exist. Check the name of the sequence.
Database Error 10036	Data dictionary operation is active for accessed sequence. Create or drop operation is active for the accessed sequence. Try again.
Database Error 10037	Can not store sequence value, the target data type is illegal. The valid target data types are INTEGER and BINARY.
Database Error 10038	Illegal column value for descending index. Corrupted data found in descending index. Drop the index and create it again.
Database Error 10040	Log file write failure, probably the disk containing the log files is full. Shut down the server and reserve more disk space for log files.
Database Error 10041	Database is read-only.
Database Error 10042	Database index check failed, the database file is corrupted.
Database Error 10043	Database free block list corrupted, same block twice in free list.
Database Error 10044	Primary key can not contain blob attributes.
Database Error 10046	Operation failed, data dictionary operation is active.
Database Error 10047	Replicated transaction is aborted.

Error code	Description
Database Error 10048	Replicated transaction contains schema changes, operation failed.
Database Error 10049	Slave server not available any more, transaction aborted
Database Error 10050	Replicated row contains BLOb columns that cannot be replicated.
Database Error 10054	Opening the database file failed. Probably another SOLID process is already running in the same directory.
Database Error 10055	Too little cache memory has been specified for the SOLID process.
Database Error 10056	Cannot open *database file*. *Error text (number)*. Most likely the SOLID process does not have correct access rights to the database file.
Database Error 10057	The database is irrevocably corrupted. Revert to the latest backup.
Database Error 10058	Database version (*number*) does not match with SOLID version. Possible causes for this error include: ■ a version of SOLID that is too old is used with this database ■ the database has been corrupted
Database Error 10059	Database version (*number*) does not match with SOLID version. Possible causes for this error include: ■ a version of SOLID that is too old is used with this database ■ the database has been corrupted

Error code	Description

Database error 10060	Cannot perform roll-forward recovery in read-only mode.
	
	Read-only mode can be specified in 3 ways. To restart SOLID in normal mode, verify that:
	
	■ SOLID process is not started with command-line option **-x** read only
	
	■ `solid.ini` does not contain the following parameter setting:
	
	`[General]`
	`ReadOnly=yes`
	
	■ license file does not have read-only limitation

Database error 10061	Out of database cache memory blocks. SOLID process cannot continue because there is too little cache memory allocated for the SOLID process. Typical cause for this problem is a heavy load from several concurrent users. To allocate more cache memory, set the following `solid.ini` parameter to a higher value: `[IndexFile]` `CacheSize=`*`cache size in bytes`* NOTE: Allocated cache memory size should not exceed the amount of physical memory.
Database error 10062	Failed to write to *log filename* at *offset.* Verify that the disk containing the log files is not full and is functioning properly. Also, log files should not be stored on shared disks over the network.
Database error 10063	Cannot create new *log filename* because such a file already exists in the log file directory. Probably your log file directory also contains logs from some other database. SOLID process cannot continue until invalid log files are removed from the log file directory. Remove *log filename* and all other log files with greater sequence numbers.

Error code	Description
Database error 10064	Illegal log file name template. Most likely, the log file name template specified in: `[Logging]` `FileNameTemplate=`*`name`* contains too few or too many sequence number digit positions. There should be at least 4 and at most 10 digit positions.
Database error 10066	Cannot open *log filename*. Check the following log file name template in `solid.ini`: `[Logging]` `FileNameTemplate=`*name* and verify that: ■ it can be expanded into a valid file name in this environment ■ SOLID process has appropriate privileges to the log files directory.
Database error 10067	Cannot create database because old *log filename* exists in the log files directory. Possibly the database has been deleted without deleting the log files or there are log files from some other database in the log files directory of the database to be created.

Error code	Description
Database error 10068	Roll-forward recovery cannot be performed because the configured log file *block size number* does not match with *block size number* of existing *filename.* To enable recovery, edit `solid.ini` to include parameter setting: `[Logging]` `BlockSize=`*`blocksize in bytes`* and restart the SOLID process. After successful recovery, you can change the log file block size by performing these steps: **1.** Shut down the SOLID process. **2.** Remove old log files. **3.** Edit new block size into `solid.ini` **4.** Restart SOLID.
Database error 10069	Roll-forward recovery failed because *relation id number* was not found. Database has been irrevocably corrupted. Please restore the database.
Database error 10070	Roll-forward failed because *relation id number* was not found. Database has been irrevocably corrupted. Please restore the database from the latest backup.
Database error 10073	Database is inconsistent. Illegal index block type *size*, *address*, *routine*, *reachmode*. Please restore the database from the latest backup.
Database error 10074	Roll-forward recovery failed. Please revert to the latest backup.
Database error 10075	The database you are trying to use has been originally created with different database block size settings than your current settings. Edit the `solid.ini` file to contain the following parameter setting: `[IndexFile]` `BlockSize=`*`blocksize in bytes`*

Error code	Description
Database error 10076	Roll-forward recovery failed because *tablename* or *viewname i*s redefined in the log *filename.* Possible causes for this error include: ■ another SOLID process is using the same log file directory ■ old log files are present in the log file directory SOLID process cannot use this corrupted log file to recover. In order to continue, you have the following alternatives: **1.** Revert to the last backup **2.** Revert to the last checkpoint **3.** Revert to the last committed transaction within the last valid log file
Database error 10077	No base catalog given for database conversion (use -C *catalogname)* A database's base catalog must be provided when converting the database to a new format.

SOLID Executable Errors

Error code	Description
Executable Error 10	Failed to open database
Executable Error 11	Failed to connect to database
Executable Error 12	Database test failed
Executable Error 13	Database fix failed
Executable Error 14	License error
Executable Error 15	Database must be converted
Executable Error 16	Database does not exist
Executable Error 17	Database exists
Executable Error 18	Database not created
Executable Error 19	Database create failed

Error code	Description
Executable Error 20	Communication init failed
Executable Error 21	Communication listen failed
Executable Error 22	Service operation failed
Executable Error 50	Illegal command line argument
Executable Error 51	Failed to change directory
Executable Error 52	Input file open failed
Executable Error 53	Output file open failed
Executable Error 54	Server connect failed
Executable Error 55	Operation init failed

SOLID System Errors

Error code	Description
System Error 11000	File open failure. The server is unable to open the database file. Reason for the failure can be: ■ The database file has been set read-only. ■ You do not have rights to open the database file in write mode. ■ Another SOLID *Embedded Engine* is using the database file. ■ Correct the error and try again.
System Error 11001	File write failure. Server is unable to write to the disk. The database files may have a read-only attribute set or you may not have rights to write to the disk. Add rights or unset read-only attribute and try again.
System Error 11002	File write failed, disk full. Server failed to write to the disk, because the disk is full. Free disk space or move the database file to another disk. You can also split the database file to several disks using the FileSpec_[1-N] parameter in IndexFile section.

Error code	Description
System Error 11003	File write failed, configuration exceeded. Writing to the database file failed, because the maximum database file size set in FileSpec_[1-N] parameter is exceeded.
System Error 11004	File read failure. An error occurred reading a file. This may indicate a disk error in your system.
System Error 11005	File read beyond end of file. Internal error.
System Error 11006	File read failed, illegal file address. An error occurred reading a file. This may indicate a disk error in your system.
System Error 11007	File lock failure. The server failed to lock the database file. This error occurs in the Windows version, if you do not have SHARE.EXE loaded. To correct the failure: 1. Exit Windows 2. Load SHARE.EXE 3. Delete the database file SOLID.DB and log files. 4. Start Windows and launch SOLID *Embedded Engine*.
System Error 11008	File unlock failure. Server failed to unlock a file.
System Error 11009	File free block list corrupted. internal error.
System Error 11010	Too long file name. Filename specified in parameter FileSpec_[1-N] is too long. Change the name to a proper file name.
System Error 11011	Duplicate file name specification. Filename specified in parameter FileSpec_[1-N] is not unique. Change the name to a proper file name.

Error code	Description
System Error 11012	License information not found, exiting from SOLID *Embedded Engine* Check the existence of your `solid.lic` file.
System Error 11013	License information is corrupted. Your `solid.lic` file has been corrupted.
System Error 11014	Database age limit of evaluation license expired.
System Error 11015	Evaluation license expired.
System Error 11016	License is for different CPU architecture.
System Error 11017	License is for different OS environment.
System Error 11018	License is for different version of this OS.
System Error 11019	License is not valid for this server version.
System Error 11020	License information is corrupted.
System Error 11021	Problem with Your license, please contact Solid Information Technology Ltd. immediately.
System Error 11022	Desktop license is only for local *protocol* communication, cannot use protocol *protocol* for listening.
System Error 11024	Desktop license is only for local communication, cannot use name *name* for listening.
System Error 11025	License file *filename* is not compatible with this server executable. Server has been started with an incompatible license file. You need to update your license file to match the server version.
System Error 11026	Backup directory contains a file which could not be removed. Some file could not be removed from the backup directory. The backup directory may point to a wrong location.
System Error 11027	No such parameter section *section.* Parameter was not found from the specified section in the `solid.ini` file.
System Error 11028	No such parameter *section.name.* Parameter does not exist.

Error code	Description
11029	Not allowed to set parameter value. User is not allowed to set the parameter value.
11030	Cannot set values to multiple parameters. Only one parameter can be set at one time.
11031	Illegal type for parameter. Parameter type is illegal.
11032	Cannot set new value for parameter *section.name*. A new value cannot be set for the parameter.

SOLID Table Errors

Error code	Description
Table Error 13001	Illegal character constant *constant.* An illegal character constant was found in the SQL statement.
Table Error 13002	Type CHAR not allowed for arithmetic. You have entered a calculation having a character type constant. Character constants are not supported in arithmetical.
Table Error 13003	Aggregate function *function* not available for ordinary call. Aggregate functions can not be used for ordinary function calls.
Table Error 13004	Illegal aggregate function *parameter* parameter. An illegal parameter has been given to an aggregate function. Aggregate function parameters can only be column names or numbers.
Table Error 13005	SUM and AVG not supported for CHAR type. Aggregate functions SUM and AVG are not supported for character type parameters.

Error code	Description
Table Error 13006	SUM or AVG not supported for DATE type. Aggregate functions SUM and AVG are not supported for date type parameters.
Table Error 13007	Function *function* is not defined. The function you tried to use is not defined.
Table Error 13009	Division by zero. A division by zero has occurred.
Table Error 13011	Table *table* does not exist. You have referenced a table which does not exist or you do not have REFERENCES privilege on the table.
Table Error 13013	Table name *table* conflicts with an existing entity. Choose a unique name for a table. The specified name is already used.
Table Error 13014	Index *index* does not exist. You have referenced an index which does not exist.
Table Error 13015	Column *column* does not exist on table *table.* You have referenced a column in a table which does not exist.
Table Error 13016	User does not exist. You have referenced a user which does not exist.
Table Error 13018	Join table is not supported Joined tables are not supported in this version of SOLID *Embedded Engine*.
Table Error 13019	Transaction savepoints are not supported. Transaction savepoints are not supported in this version of SOLID *Embedded Engine*.
Table Error 13020	Default values are not supported. Default column values are not supported in this version of SOLID *Embedded Engine*.

Error code	Description
Table Error 13021	Foreign keys are not supported. Foreign keys are not supported in this version of SOLID *Embedded Engine*.
Table Error 13022	Descending keys are not supported. Descending keys are not supported in this version of SOLID *Embedded Engine*.
Table Error 13023	Schema is not supported. Schema is not supported in this version of SOLID *Embedded Engine*.
Table Error 13025	Update through a cursor with no current row. You have tried to update using cursor, but you do not have current row in the cursor.
Table Error 13026	Delete through a cursor with no current row You have tried to delete using cursor, but you do not have current row in the cursor.
Table Error 13028	View *view* does not exist. You have referenced a view which does not exist.
Table Error 13029	View name *view* conflicts with an existing entity. Choose a unique name for a view. The specified name is already used.
Table Error 13030	No value specified for NOT NULL column *column*. You have not specified a value for a column which is defined NOT NULL.
Table Error 13031	Data dictionary operation is active for accessed table or key. You can not access the table or key, because a data dictionary operation is currently active. Try again after the data dictionary operation has completed.
Table Error 13032	Illegal type *type.* You have tried to create a table with a column having an illegal type.

Error code	Description
Table Error 13033	Illegal parameter *parameter* for type *type.* The type of the parameter you entered is illegal in this column.
Table Error 13034	Illegal constant *constant.* You have entered an illegal constant.
Table Error 13035	Illegal INTEGER constant *constant.* You have entered an illegal integer type constant. Check the syntax of the statement and try again.
Table Error 13036	Illegal DECIMAL constant *constant.* You have entered an illegal decimal type constant. Check the decimal number and try again.
Table Error 13037	Illegal DOUBLE PREC constant *constant.* You have entered an illegal double precision type constant. Check the number and try again.
Table Error 13038	Illegal REAL constant *constant.* You have entered an illegal real type constant. Check the real number and try again.
Table Error 13039	Illegal assignment. You have tried to assign an illegal value for a column.
Table Error 13040	Aggregate *function* function is not defined. The aggregate function you tried to use is not supported.
Table Error 13041	Type DATE not allowed for arithmetic. DATE type columns or constants are not allowed in arithmetical.
Table Error 13042	Power arithmetic not allowed for NUMERIC and DECIMAL data type. Decimal and numeric data types do not support power arithmetical.
Table Error 13043	Illegal date constant *constant.* A date constant is illegal. The correct form for date constants is: YYYY-MM-DD.

Error code	Description
Table Error 13045	Reference privileges are not supported. Reference privileges are not supported in this version of SOLID *Embedded Engine*.
Table Error 13046	Illegal user name *user.* User name entered is not legal. A legal user name is at least 2 and at most 31 characters in length. A user name may contain characters from A to Z, numbers from 0 to 9 and underscore character '_'.
Table Error 13047	No privileges for operation. You have no privileges for the attempted operation.
Table Error 13048	No grant option privilege for entity *name.* You have no privileges to grant privileges for the entity.
Table Error 13049	Column privileges cannot be granted WITH GRANT OPTION Granting column privileges WITH GRANT OPTION is not supported in this version of SOLID *Embedded Engine*.
Table Error 13050	Too long constraint value. Maximum constraint length has been exceeded. Maximum constraint length is 255 characters.
Table Error 13051	Illegal column name *column.* You have tried to create a table with an illegal column name.
Table Error 13052	Illegal comparison operator *operator* for a pseudo column *column.* You have tried to use an illegal comparison operator for a pseudo column. Legal comparison operators for pseudo columns are: equality '=' and non-equality '<>'.
Table Error 13053	Illegal data type for a pseudo column. You have tried to use an illegal data type for a pseudo column. Data type of pseudo columns is BINARY.

Error code	Description
Table Error 13054	Illegal pseudo column data, maybe data is not received using pseudo column. You have tried to compare pseudo column data with non-pseudo column data. Pseudo column data can only be compared with data received from a pseudo column.
Table Error 13055	Update not allowed on pseudo column. Updates are not allowed on pseudo columns.
Table Error 13056	Insert not allowed on pseudo column. Inserts are not allowed on pseudo columns.
Table Error 13057	Index name *index* already exists. You have tried to create an index, but an index with the same name already exists. Use another name for the index.
Table Error 13058	Constraint checks were not satisfied on column *column.* Column has constraint checks which were not satisfied during an insert or update.
Table Error 13059	Reserved system name *name.* You tried to use a name which is a reserved system name such as PUBLIC and SYS_ADMIN_ROLE.
Table Error 13060	User name *user* not found. You tried to reference a user name which is not created.
Table Error 13061	Role name *role* not found. You tried to reference a role name which is not created.
Table Error 13062	Admin option is not supported. Admin option is not supported in this version of SOLID *Embedded Engine*.
Table Error 13063	Name *name* already exists. You tried to use a role or user which already exists. User names and role names must all be different, that is, you can not have a user named HOBBES and a role named HOBBES.

Error code	Description
Table Error 13064	Not a valid user name *user.* You tried to create an invalid user name. A valid user name has at least 2 characters and at most 31 characters.
Table Error 13065	Not a valid role name *role.* You tried to create an invalid role name. A valid user name has at least 2 characters and at most 31 characters.
Table Error 13066	User *user* not found in role *role.* You tried to revoke a role from a user and the user did not have that role.
Table Error 13067	Too short password. You have entered a too short password. Password length must be at least 3 characters.
Table Error 13068	Shutdown is in progress. You are unable to complete this operation, because server shutdown is in progress.
Table Error 13070	Numerical overflow. A numerical overflow has occurred. Check the values and types of numerical variables.
Table Error 13071	Numerical underflow. A numerical underflow has occurred. Check the values and types of numerical variables.
Table Error 13072	Numerical value out of range. A numerical value is out of range. Check the values and types of numerical variables.
Table Error 13073	Math error. A mathematical error has occurred. Check the mathematics in the statement and try again.
Table Error 13074	Illegal password. You have tried to enter an illegal password.

Error code	Description
Table Error 13075	Illegal role name *role.* You have tried to enter an illegal role name. A legal role name is at least 2 and at most 31 characters in length. A user role may contain characters from A to Z, numbers from 0 to 9 and underscore character '_'.
Table Error 13076	NOT NULL must not be specified for added column *column.* You have tried to add a column to a table using ALTER TABLE statement. NOT NULL constraint is not allowed in ALTER TABLE statement when the table already includes data.
Table Error 13077	Last column can not be dropped. You have tried to drop the final column in a table. This is not allowed; at least one column must remain in the table.
Table Error 13078	Column already exist on table. You have tried to create a column which already exists in a table.
Table Error 13079	Illegal search constraint. Check the search engine. There may be mismatch between data types.
Table Error 13080	Incompatible types, can not modify column *column* from type *type* to type *type.* You have tried to modify column to a data type that is incompatible with the original definition, such as VARCHAR and INTEGER
Table Error 13081	Descending keys are not supported for binary columns. You can not define descending key for a binary column.
Table Error 13082	Function *function*: parameter * not supported. You can not use parameter star (*) with ODBC Scalar Functions.
Table Error 13083	Function *function*: Too few parameters. The function expects more parameters. Check the function call.

Error code	Description
Table Error 13084	Function *function*: Too many parameters. The function expects fewer parameters. Check the function call.
Table Error 13085	Function *function*: Run-time failure. An error was detected during the execution of the function. Check the parameters.
Table Error 13086	Function *function*: type mismatch in parameter *parameter number.* A erroneous type of parameter detected in the given position of the function call. Check the function call.
Table Error 13087	Function *function*: illegal value in parameter *parameter number.* An illegal value for a parameter detected in the given position of the function call. Check the function call.
Table Error 13090	Foreign key column *column* data type not compatible with referenced column data type. References specification error. Check that the column data type are compatible between referencing and referenced tables.
Table Error 13091	Foreign key does not match to the primary key or unique constraint of the referenced table. References specification error. Check that the column data type are compatible between referencing and referenced tables and that the foreign key is unique for the referenced table.
Table Error 13092	Event name *event* conflicts with an existing entity. Choose a unique name for an event. The specified name is already used.
Table Error 13093	Event *event* does not exist. You referenced to a nonexistent event. Check the name of event.
Table Error 13094	Duplicate column *column* in primary key definition. Duplicate columns are not allowed in a table-constraint-definition. Remove duplicate columns from the definition.

Error code	Description
Table Error 13095	Duplicate column *column* in unique constraint definition. Duplicate columns are not allowed in a table-constraint-definition. Remove duplicate columns from the definition.
Table Error 13096	Duplicate column *column* in index definition. Duplicate columns are not allowed in CREATE INDEX statement. Remove duplicate columns.
Table Error 13097	Primary key columns must be NOT NULL. Error in a *column_constraint_definition*. Define primary key columns NOT NULL. For example: CREATE TABLE DEPT (DEPTNO INTEGER NOT NULL, DNAME VARCHAR, PRIMARY KEY(DEPTNO));
Table Error 13098	Unique constraint columns must be NOT NULL. Error in a *column_constraint_definition*. Define unique columns NOT NULL. For example: CREATE TABLE DEPT4 (DEPTNO INTEGER NOT NULL, DNAME VARCHAR, UNIQUE(DEPTNO));
Table Error 13099	No REFERENCES privileges to referenced columns in table *table.* You do not have privileges to reference to the table.
Table Error 13100	Illegal table mode combination. You have defined illegal combination of locking. Check locking type of tables.
Table Error 13101	Only execute privileges can be used with procedures.
Table Error 13102	Execute privileges can be used only with procedures.
Table Error 13103	Illegal grant or revoke operation.
Table Error 13104	Sequence name *sequence* conflicts with an existing entity. Choose a unique name for a sequence. The specified name is already used.
Table Error 13105	Sequence *sequence* does not exist. You referenced a nonexistent sequence. Check the name of sequence.
Table Error 13106	Foreign key reference exists to table *table.*

Error code	Description
Table Error 13107	Illegal set operation. You tried to execute a non-existent set operation.
Table Error 13108	Comparison between incompatible types *datatype* and *datatype.*
Table Error 13109	There are schema objects for this user, drop failed
Table Error 13110	NULL values given for NOT NULL column *column*.
Table Error 13111	Ambiguous entity name *name*.
Table Error 13112	Foreign keys are not supported with main memory tables.
Table Error 13113	Illegal arithmetic between types *datatype* and *datatype*.
Table Error 13114	String operations are not allowed on values stored as BLOBs or CLOBs.
Table Error 13115	Function *function_name*: Too long value (stored as CLOB) in parameter *parameter.* The parameter value was stored as CLOB and cannot be used with a function.
Table Error 13116	Column *column_name* specified more than once. Column was specified more than once in the GRANT or REVOKE statement.
Table Error 13117	Wrong number of parameters Wrong number of parameters when converting subscription parameters to base publication parameter types.
Table Error 13118	Column privileges are supported only for base tables. Column privileges are allowed only for base tables; they cannot be used, for example, for views.
Table Error 13119	Types *column_type* and *column_type* are not union compatible. Column types are not union compatible.
Table Error 13120	Too long entity name '*entity_name*' Entity name is too long, maximum entity name is 254 characters.

Error code	Description

Table Error 13121	Too many columns, maximum number of columns is *value*.
	
	Too many columns; the default maximum number of columns is 1000. It can be changed using the following parameter in the `solid.ini` file:
	
	`[Srv]`
	`MaxOpenCursors=n`

Table Error 13122	Operation is not supported for a table with sync history. Operation is not supported because the table has synchronization history defined.
Table Error 13123	Table '*table_name*' is not empty. Some operations are allowed only for empty tables.
Table Error 13124	User id *user_id* not found. Internal user id was not found; the user may have been dropped.
Table Error 13125	Illegal LIKE pattern '*pattern*' Illegal like pattern was given as a search constraint.
Table Error 13126	Illegal type *datatype* for LIKE pattern. Only CHAR and WCHAR allowed for LIKE search constraints.
Table Error 13127	Comparison failed because at least one of the was too long. Comparison failed because at least one of the column values was stored as a BLOB or CLOB.
Table Error 13128	LIKE predicate failed because value is too long. LIKE predicate failed because the column value is stored as a CLOB.
Table Error 13129	LIKE Predicate failed because pattern is too long. LIKE predicate failed because pattern value is stored as a CLOB.
Table Error 13130	Illegal type *datatype* for LIKE ESCAPE character. Like ESCAPE character must be CHAR or WCHAR type.

Error code	Description
Table Error 13131	Too many nested triggers. Maximum number of nested triggers is reached. Triggers may be nested, for example, by activating other triggers from a trigger or causing recursive cycle when activating triggers. Default value for maximum allowed nested triggers is 16. It can be changed using a configuration parameter: `{SQL]` `MaxNestedTriggers=`*n*
Table Error 13132	Too many nested procedures. Maximum number of nested procedures is reached. Procedures may be nested, for example, by activating other procedures from a procedure or causing a recursive cycle when activating procedures. Default value for maximum allowed nested procedures is 16. It can be changed using a configuration parameter: `SQL` `MaxNestedTriggers=`*n*
Table Error 13133	Not a valid license for this product. The license file is for another SOLID product.
Table Error 13134	Operation is allowed only for base tables. Given operation is available only for base tables.
Table Error 13137	Illegal grant/revoke mode Grant or revoke mode is not allowed for given database objects.
Table Error 13138	Index *index_name* given in index hint does not exist. Index name given in optimizer hint is not found for a table.
Table Error 13139	Catalog *catalog_name* does not exist. Catalog name is not a valid catalog.
Table Error 13140	Catalog *catalog_name* already exists. Catalog name is an existing catalog.

Error code	Description
Table Error 13141	Schema *schema_name* does not exist. Schema name is not a valid schema.
Table Error 13142	Schema *schema_name* already exists. Schema name is an existing schema.
Table Error 13143	Schema *schema_name* is an existing user. Schema name specifies an existing user name.
Table Error 13144	Commit and rollback are not allowed inside trigger. Commit or rollback are not supported inside trigger execution. This error is also given if a trigger calls a procedure that tries to execute commit or rollback command.
Table Error 13145	Sync parameter not found. Parameter name given in command SET SYNC PARAMETER *name* NONE is not found.
Table Error 13146	There are schema objects for this catalog, drop failed. Catalog contains schema object and cannot be dropped. Schema objects like tables and procedures need to be dropped before catalog can be dropped.
Table Error 13151	Cannot drop a column that is part of primary or unique key. Table definition contains a column that is part of a primary or unique key in an index.

SOLID Server Errors

Error code	Description
Server Error 14501	Operation failed. This error occurs when a timed command fails. Check the arguments of timed commands.
Server Error 14502	RPC parameter is invalid. A network error has occurred.

Error code	Description
Server Error 14503	Communication error. A communication error has occurred.
Server Error 14504	Duplicate cursor name *cursor*. You have tried to declare a cursor with a cursor name which is already in use. Use another name.
Server Error 14505	Connect failed, illegal user name or password. You have entered either a user name or a password that is not valid.
Server Error 14506	Server is closed, no new connections allowed. You have tried to connect to a closed server. Connecting was aborted.
Server Error 14507	Maximum number of licensed user connections exceeded. You have tried to connect to a server which has all licenses currently in use. Connecting was aborted.
Server Error 14508	The operation has timed out. You have launched an operation that has been aborted.
Server Error 14509	Version mismatch. A version mismatch has occurred. The client and server are different versions. Use same versions in the client and the server.
Server Error 14510	Communication write operation failed. A write operation failed. This indicates a network problem. Check your network settings.
Server Error 14511	Communication read operation failed. A read operation failed. This indicates a network problem. Check your network settings.
Server Error 14512	There are users logged to the server. You can not shutdown the server now. There are users connected to the server.
Server Error 14513	Backup process is active. You can not shutdown the server now. The backup process is active

Error code	Description
Server Error 14514	Checkpoint creation is active. You can not shutdown the server now. The checkpoint creation is active.
Server Error 14515	Invalid user id. You tried to drop a user, but the user id is not logged in to the server.
Server Error 14516	Invalid user name. You tried to drop a user, but the user name is not logged in to the server.
Server Error 14517	Someone has updated the at commands at the same time, changes not saved. You tried to update timed commands at the same time another user was doing the same. Your changes will not be saved.
Server Error 14518	Connection to the server is broken, connection lost. Possible network error. Reconnect to the server.
Server Error 14519	The user was thrown out from the server, connection lost. Possible network error.
Server Error 14521	Failed to create a new thread for the client.
Server Error 14529	The operation timed out.
Server Error 14530	The connected client does not support UNICODE data types. Connected client is an old version client that does not support UNICODE data types. UNICODE data type columns cannot be used with old clients.

Server Error 14531	Too many open cursor, max limit is *value*.
	There are too many open cursors for one client; maximum number of open cursors for one connection is 1000. The value can be changed using a configuration value:
	`[Srv]`
	`MaxOpenCursors=n`

Error code	Description
Server Error 14533	Operation cancelled Operation was cancelled because client application called ODBC or JDBC cancel function.
Server Error 14534	Only administrative statements are allowed. Only administrative statements are allowed for the connection.

SOLID Communication Errors

Error code	Description
Communication Error 21300	Protocol *protocol* is not supported. Protocol is not supported.
Communication Error 21301	Cannot load the dynamic link library <library> or one of its components. The server was unable to load the dynamic link library or a component needed by this library. Check the existence of necessary libraries and components.
Communication Error 21302	Wrong version of dynamic link library *library.* The version of this library is wrong. Update this library to a newer version.
Communication Error 21303	Network adapter card is missing or needed <protocol> software is not running. The network adapter card is missing or not functioning.
Communication Error 21304	Out of *protocol* resources The network protocol is out of resources. Increase the protocols resources in the operating system.
Communication Error 21305	An empty or incomplete network name was specified. The network name specified is not legal. Check the network name.

Error code	Description
Communication Error 21306	Server *network name* not found, connection failed. The server was not found. 1) Check that the server is running. 2) Check that the network name is valid. 3) Check that the server is listening given network name.
Communication Error 21307	Invalid connect info *network name.* The network name given as the connect info is not legal. Check the network name.
Communication Error 21308	Connection is broken (*protocol read/write* operation failed with code *internal code*). The connection using the protocol is broken. Either a read or a write operation has failed with an internal error *internal code*.
Communication Error 21309	Failed to accept a new client connection, out of *protocol* resources. The server was not able to establish a new client connection. The protocol is out of resources. Increase the protocol's resources in the operating system.
Communication Error 21310	Failed to accept a new client connection, listening of *network name* interrupted. The server was not able to establish a new client connection. The listening has been interrupted.
Communication Error 21311	Failed to start a selecting thread for *network name.* A thread selection has failed for *network name*.
Communication Error 21312	Listening info *network name* already specified for this server. A network name has already been specified for this server. A server can not use a same network name more than once.
Communication Error 21313	Already listening with the network name *network name.* You have tried to add a network name to a server when it is already listening with that network name. A server can not use a same network name more than once.

Error code	Description
Communication Error 21314	Cannot start listening, network name *network name* is used by another process. The server can not start listening with the given network name. Another process in this computer is using the same network name.
Communication Error 21315	Cannot start listening, invalid listening info *network name.* The server can not start listening with the given listening info. The given network name is invalid. Check the syntax of the network name.
Communication Error 21316	Cannot stop the listening of *network name*. There are clients connected. You can not stop listening of this network name. There are clients connected to this server using this network name.
Communication Error 21317	Failed to save the listen information into the configuration file. The server failed to save this listening information to the configuration file. Check the file access rights and format of the configuration file.
Communication Error 21318	Operation failed because of an unusual *protocol* return code *code.* Possible network error. Create connection again.
Communication Error 21319	RPC request contained an illegal version number. Either the message was corrupted or there may be a mismatch between server and client versions.
Communication Error 21320	Called RPC service is not supported in the server. There maybe a mismatch between server and client versions.
Communication Error 21321	Protocol *protocol* is not valid, try using switch '-a' for specifying another adapter id instead of *switch*. This is returned if the NetBIOS LAN adapter id given in listen/connect string is not valid.

Error code	Description
Communication Error 21322	The host machine given in connect info '%s' was not found. This is returned in clients if the host machine name given in connect info is not valid.
Communication Error 21323	Protocol *protocol* can not be used for listening in this environment. This message is displayed if the server end communication using specified protocol is not supported.
Communication Error 21324	The process does not have the privilege to create a mailbox.
Communication Error 21325	Only one listening name is supported in this server. In some operating systems like Novell Netware only one listening name is supported.
Communication Error 21326	Failed to establish an internal *number* socket connection *code* number. SOLID uses one connect socket for internal use. Creation of this socket has failed; the local loopback may not be working correctly.

SOLID Communication Warnings

Error code	Description
Warning Code 21100	Illegal value *value* for configuration parameter *parameter*, using default. An illegal value was given to the parameter *parameter*. The server will use a default value for this parameter.
Warning Code 21101	Invalid protocol definition *protocol* in configuration file. The protocol is defined illegally in the configuration file. Check the syntax of the definition.

SOLID Procedure Errors

Error code	Description
Procedure Error 23001	Undefined symbol *symbol.* You have used a symbol that has not been defined in a procedure definition.
Procedure Error 23002	Undefined cursor *cursor.* You have used a cursor that has not been defined in a procedure definition.
Procedure Error 23003	Illegal SQL operation *operation.*
Procedure Error 23004	Syntax error: parse error, line *line number.* Check the syntax of your procedure.
Procedure Error 23005	Procedure *procedure* not found.
Procedure Error 23006	Wrong number of parameters for procedure *procedure.*
Procedure Error 23007	Procedure name *value* conflicts with an existing entity. Choose a unique name for a procedure. The specified name is already used.
Procedure Error 23009	Event *event* does not exist, line *line number.*
Procedure Error 23010	Incompatible event *event* parameter type, line *line number.*

Error code	Description
Procedure Error 23011	Wrong number of parameter for event *event*, line *line number.*
Procedure Error 23012	Duplicate wait for event *event*, line *line number.*
Procedure Error 23013	Undefined sequence *sequence.*
Procedure Error 23014	Duplicate sequence name *sequence.*
Procedure Error 23015	Sequence *sequence* not found.
Procedure Error 23016	Incompatible variable type in call to sequence *sequence*, line *line number.*
Procedure Error 23017	Duplicate symbol *symbol.* You have duplicate definitions for a symbol.
Procedure Error 23018	Procedure owner *owner* not found.
Procedure Error 23019	Duplicate cursor name *'cursor'*
Procedure Error 23020	Illegal option *option* for WHENEVER SQLERROR … statement.
Procedure Error 23021	RETURN ROW not allowed in procedure with no return type, line *line number.*
Procedure Error 23022	SQL String variable *variable* must be of character data type, line *line number.*
Procedure Error 23023	Call syntax error: *syntax*, line *line number.*
Procedure Error 23024	Trigger *trigger_name* not found. Trigger name not found.
Procedure Error 23025	Trigger name *trigger_name* conflicts with an existing entity. Trigger name conflicts with some other database object. Triggers share the same name space, as for example, in table and procedures.
Procedure Error 23026	Variable *variable* is ot of character type, line *line number.* A CHAR or WCHAR variable is required for the operations like RETURN SQLERROR *variable*.

Error code	Description
Procedure Error 23027	Duplicate reference to column *column_name* in trigger definition. One column can be reference only once in the trigger definition.
Procedure Error 23028	Commit and rollback are not allowed in triggers. Trigger body may not contain commit or rollback statements.
Procedure Error 23501	Cursor *cursor* is not open.
Procedure Error 23502	Illegal number of columns in EXECUTE ... *procedure* in cursor *cursor.*
Procedure Error 23503	Previous SQL operation *operation* failed in cursor *cursor.*
Procedure Error 23504	Cursor *cursor* is not executed.
Procedure Error 23505	Cursor *cursor* is not a SELECT statement.
Procedure Error 23506	End of table in cursor *cursor.*
Procedure Error 23507	Illegal type conversion in cursor *cursor* from type *data type* to type *data type.*
Procedure Error 23508	Illegal assignment, line *line number.*
Procedure Error 23509	In *procedure* line *line number* Stmt *statement* was not in error state in RETURN SQLERROR OF ...
Procedure Error 23510	In *procedure* line *line number* Transaction cannot be set read only, because it has written already.
Procedure Error 23511	In *procedure* line *line number* USING part is missing for dynamic parameters for *procedure.*
Procedure Error 23512	In *procedure* line *line number* USING list is too short for *procedure.*
Procedure Error 23513	In *procedure* line *line number* Comparison between incompatible types *data type* and *data type.*
Procedure Error 23514	In *procedure* line *line number* type *data type* is illegal for logical expression.
Procedure Error 23515	In *procedure* line *line number* assignment of parameter *parameter* in *list* list failed.

Error code	Description
Procedure Error 23516	In CALL *procedure* assignment of parameter *parameter* failed.
Procedure Error 23518	User error: *error_text* User generated error in a procedure or trigger. User can generate this error by using a statement RETURN SQLERROR *string* or RETURN SQLERROR *variable*. Variable must be of CHAR or WCHAR type.
Procedure Error 23519	Fetch previous is not supported for procedures. Fetch previous row does not work for result sets returned by a procedure.

SOLID Sorter Errors

Error code	Description
Sorter Error 24001	Sort failed due to insufficient configured TmpDir space
Sorter Error 24002	Sort failed due to insufficient physical TmpDir space
Sorter Error 24003	Sort failed due to insufficient sort buffer space
Sorter Error 24004	Sort failed due to too long row (internal failure)
Sorter Error 24005	Sort failed due to I/O error

B

Configuration Parameters

By managing the parameters of your SOLID *Embedded Engine*, you can modify the environment, performance, and operation of the server.

When SOLID *Embedded Engine* is started, it attempts to open the configuration file `solid.ini` in the current directory. The configuration values for the server parameters are included in this file. If the file does not exist, SOLID *Embedded Engine* will use the default settings for the parameters. Also, if a value for a parameter is not set in the `solid.ini` file, SOLID *Embedded Engine* will use a default value for the parameter. The default values depend on the operating system you are using.

Generally, the default settings offer the best performance and operability, but in some special cases modifying a parameter will improve performance. You can change the parameters in the following ways:

- Using the SOLID *DBConsole* Configuration page.
- Entering the command `parameter` in SOLID *DBConsole* (Query window or command line) or SOLID *Remote Control* (teletype).
- Entering `ADMIN COMMAND 'parameter'` in SOLID *SQL Editor*
- Manually editing the configuration file `solid.ini`.

General Section

[General]	Description	Default
MaxOpenFiles	the maximum number of files kept concurrently open during SOLID *Embedded Engine* sessions	OS depend.
BackupDirectory	makes a backup of the database if the default `'backup'` is used or may also be given as an argument. For example, backup abc, creates a backup on directory `'abc'`. All directory definitions are relative to the SOLID *Embedded Engine* working directory unless the full path is provided.	'backup' directory
BackupCopyLog	if set to `yes`, backup operation will copy log files to the backup directory	yes
BackupDeleteLog	if set to `yes`, old log files will be deleted after backup operation	yes
BackupCopyIniFile	if set to `yes`, `solid.ini` file will be copied to the backup directory	yes
BackupCopySolmsgout	If set to yes, `solmsg.out` file is copied to the backup directory	yes
Checkpoint Interval	the number of inserts made in the database that causes automatic checkpoint creation	5000
MergeInterval	the number of index inserts made in the database that causes the merge process to start	Cache size depend.
Readonly	if set to `yes`, database is set to read-only mode	no
LongSequential SearchLimit	the number of sequential fetches after which search is treated as long sequential search	500
SearchBuffer Limit	the maximum percentage of search buffers from the total buffered memory reserved for open cursors	50
Transaction HashSize	the hash table size for incomplete transactions	Cache size depend.

IndexFile Section

[IndexFile]	Description	Default
FileSpec_[1-N]	the file name followed with maximum size (in bytes) of that database file, for example: `c:\sol1.db 2000000` This parameter also has an optional parameter after the maxsize: physical drive number. The number value itself is not essential, but it is used as a hint for I/O threads on which I/O requests can be parallelized. This file must be stored to a local drive using local disk names to avoid problems with network I/O and to achieve better performance.	solid.db 2147483647
BlockSize	the block size of the index file in bytes; use multiple of 2 KB: minimum 2 KB, maximum 32 KB	8192
CacheSize	the size of database cache memory for the server in bytes; the minimum 512 KB	OS depend.
ExtendIncrement	the number of blocks that is allocated at one time when SOLID *Embedded Engine* needs to allocate more space for the database file	50
ReadAhead	sets the number of prefetched index leafs during long sequential searches	4
PreFlushPercent	Percentage of page buffer which is kept clean by preflush thread	5

Logging Section

[Logging]	Description	Default
LogEnabled	whether logging is enabled or not	yes
BlockSize	the block size of log files	2048
MinSplitSize	when this file size is reached, logging will be continued to the following log file after the next checkpoint	1 MB
FileNameTemplate	the path and naming convention used when creating log files; template characters are replaced with sequential numbering; for example: `c:\solid\log\sol#####.log` This file must be stored to a local drive using local disk names to avoid problems with network I/O and to achieve better performance.	sol#####.log
DigitTemplateChar	the template character that will be replaced in the name template of the log file	#

Communication Section

[Com]	Description	Default
Listen	the network name for server; the protocol and server name that SOLID *Embedded Engine* uses when it starts listening to the network	OS depend.
Connect	the network name for client; the protocol and name that a SOLID *Embedded Engine* client uses for server connection; in a Windows environment ODBC Data Source Name overrides the value of this parameter	OS depend.
MaxPhysMsgLen	the maximum length of a single physical network message in bytes; longer network messages will be split into smaller messages of this size	OS depend.
ReadBufSize	the buffer size in bytes for the data read from the network	OS depend.
WriteBufSize	the buffer size in bytes for the data written into the network	OS depend.
Trace	if this parameter is set to yes, trace information on network messages is written to a file specified with the TraceFile parameter	no
TraceFile	if this parameter is set to **yes**, trace information on network messages is written to a file specified with this parameter	`soltrace.out` (written to the current working directory of the server or client depending on which end the tracing is started)

Data Sources

[Data Sources]	Description	Default
<logical name> = <network name>, <Description>	These parameters can be used to give a logical name to a SOLID *Embedded Engine*.	

Server Section

[Srv]	Description	Default
At	The syntax is: *At* := *timed_command* [, *timed_command*] *timed_command* := [*day*] *HH:MM command argument* *day* := **sun | mon | tue | wed | thu | fri | sat** If entered, allows you specify a command to automate an administrative task, such as executing system commands, creating backups, checkpoints, and database status reports. For example: AT=20:30 makecp, 21:00 backup,sun 23:00 shutdown	If you specify backup, the default is the backup directory set with the BackupDirectory parameter in the General section. Also if the day is not given, the command is executed daily.
RowsPerMessage	the number of rows returned from the server in one network message	10
ConnectTimeOut	specifies the continuous idle time in minutes after that an connection is dropped; negative or zero value means infinite	480
AbortTimeOut	specifies the time in minutes after that an idle transaction is aborted; negative or zero value means infinite	120
Threads	the number of threads used for database access in SOLID *Embedded Engine*.	OS depend.
Echo	if set to `yes`, contents of `solmsg.out` file are displayed also at the server's command window	no
Name	the informal name of the server, equivalent to the -n command line option	
AllowConnect	if set to no only connections from *Remote Contro*l or *DBConsole* are allowed	yes
MessageLogSize	The maximum size of the solmsg.out file in bytes. The deafult is 60 KB.	OS depend.
MaxOpenCursors	The maximum number of cursors that a database client can have simultaneously open.	1000

SQL Section

[SQL]	Description	Default
Info	Set the level of informational messages [0-8] printed from the server (0=no info, 8=all info); information is written into the file defined by parameter InfoFileName.	0
SQLInfo	Set the level of informational SQL level messages [0-8] (0=no info, 8=all info); information is written into a file defined by parameter InfoFileName.	no default
InfoFileName	Default global info file name.	`soltrace.out`
InfoFileSize	Maximum size of the info file. The default is 1 MB.	no default
InfoFileFlush	If set to yes, flushes info file after every write operation	yes
SortArraySize	Size of the array that SQL uses when ordering result set; for optimal performance this should be as big as the biggest retrieved result set that cannot be ordered by key values; for large sorts use external sorter,	OS depend.
ProcedureCache	Size of cache memory for parsed procedures in number of procedures.	5
MaxNestedProcedures	Maximum number of allowed nested procedures. If this parameter is defined too high, the server stack may become insufficient depending on the operating system.	16
TriggerCache	Size of cache memory that each user has for triggers.	10
MaxNestedTriggers	Maximum number of allowed nested triggers. This maximum number includes both direct and indirect nesting, so both A->A->A and A->B->A are counted as three nested triggers.	16
MaxBlobExpression Size	Maximum size of LONG VARCHAR columns in KBs that can be used in string functions.	64
EmulateOldTIMESTAMPDIFF	If included in the `solid.ini` file and set to "Yes", the old TIMESTAMPDIFF behavior is emulated by the server. This old behavior returns the integer number of intervals of type *interval* by which *timestamp_exp2* is greater than *timestamp_exp1*. Otherwise, the default is the new behavior which returns the integer number of *interval* as the amount of full units between *timestamp_exp1* and *timestamp_exp2*.	The default "No" returns the integer number of *interval* as the amount of full units between *timestamp_exp1* and *timestamp_exp2*

Sorter Section

[Sorter]	Description	Default
MaxCacheUse Percent	maximum percentage of cache pages used for sorting; range from 10% to 50%	
MaxMemPerSort	maximum memory available in bytes for one sort	
MaxFilesTotal	maximum number of files used for sorting	
TmpDir_[1-N]	name of the directory that contains temporary files created during sorting	no default

Hints Section

[Hints]	Description	Default
EnableHints	If included in the `solid.ini` file and set to "Yes", all hints that are in the `solid.ini` file are enabled.	Yes

C
Data Types

Supported Data Types

The tables in this appendix list the supported data types by category. the following abbreviations are used in each table.

Abbreviation	Description
DEFLEN	the defined length of the column; for example, for CHAR(24) the precision and length is 24
DEFPREC	the defined precision; for example, for NUMERIC(10,3) it is 10
DEFSCALE	the defined scale; for example, for NUMERIC(10,3), it is 3
MAXLEN	the maximum length of column
N/A	not applicable

Character Data Types

Data type	Size	Precision	Scale	Length	Display size
CHAR, WCHAR	2 G*	DEFLEN	N/A	DEFLEN	DEFLEN
VARCHAR, WVARCHAR	2 G**	DEFLEN	N/A	DEFLEN	DEFLEN
LONG VARCHAR, LONG WVARCHAR	2 G	MAXLEN	N/A	MAXLEN	MAXLEN
* default is 1					
** default is 254					

Numeric Data Types

Data type	Range	Precision	Scale	Length	Display size
DECIMAL	±3.6e16	16	DEFSCALE	18	18
NUMERIC	±3.6e16	DEFPREC	DEFSCALE	DEFPREC +2	DEFPREC +2
TINYINT	[-128, 127] [0, 255]	3	0	1 (bytes)	4 (signed) 3 (unsigned)
SMALLINT	[-32768, 32767] [0, 65535]	5	0	2 (bytes)	6 (signed) 5 (unsigned)
INTEGER	$[-2^{31}, 2^{31-1}]$ $[0, 2^{32-1}]$	10	0	4 (bytes)	11 (signed) 10 (unsigned)
REAL	±1.7014117 e38	7	N/A	4 (bytes)	13
FLOAT	±8.9884657 e307	15	N/A	8 (bytes)	22
DOUBLE PRECISION	±8.9884657 e307	15	N/A	8 (bytes)	22

Binary Data Types

Data type	Size	Precision	Scale	Length	Display size
BINARY	2 G*	DEFLEN	N/A	DEFLEN	DEFLEN x 2
VARBINARY	2 G**	DEFLEN	N/A	DEFLEN	DEFLEN x 2
LONG VAR-BINARY	2 G	MAXLEN	N/A	MAXLEN	MAXLEN x 2

* default is 1

** default is 254

Date Data Type

Data type	Range	Precision	Scale	Length	Display size
DATE	N/A	10*	N/A	6**	10*

* the number of characters in the yyyy-mm-dd format

** the size of the DATE_STRUCT structure

Time Data Type

Data type	Range	Precision	Scale	Length	Display size
TIME	N/A	8*	N/A	6**	8*

* the number of characters in the hh:mm:ss format
** the size of the TIME_STRUCT structure

Timestamp Data Type

Data type	Range	Precision	Scale	Length	Display size
TIMESTAMP	N/A	19*	9	16**	19/29***

* the number of characters in the 'yyyy-mm-dd hh:mm:ss.fffffffff' format
** the size of the TIMESTAMP_STRUCT structure
*** size is 29 with a decimal fraction part

The Smallest Possible Non-zero Numbers

Data type	Value
DOUBLE	2.2250738585072014e-308
REAL	1.175494351e-38

Description of Different Column Values in the Tables

The range of a numeric column refers to the minimum and maximum values the column can store. The size of character columns refers to the maximum length of data that can be stored in the column of that data type.

The precision of a numeric column refers to the maximum number of digits used by the data type of the column. The precision of a non-numeric column refers to the defined length of the column.

The scale of a numeric column refers to the maximum number of digits to the right of the decimal point. Note that for the approximate floating point number columns, the scale is undefined, since the number of digits to the right of the decimal point is not fixed.

The length of a column is the maximum number of bytes returned to the application when data is transferred to its default C type. For character data, the length does not include the null termination byte. Note that the length of a column may differ from the number of bytes needed to store the data on the data source.

The display size of a column is the maximum number of bytes needed to display data in character form.

D

SOLID SQL Syntax

SOLID *Embedded Engine* SQL syntax is based on the ANSI X3H2-1989 level 2 standard including important ANSI X3H2-1992 (SQL2) extensions. User and role management services missing from previous standards are based on the ANSI SQL3 draft.

This appendix presents a simplified description of the SQL statements including some examples. The same information is included in the **SOLID Programmer Guide.**

ADMIN COMMAND

ADMIN COMMAND '*command_name*'

command_name ::= BACKUP | BACKUPLIST | CLOSE | DESCRIBE PARAMETER

| ERRORCODE | EXIT | HELP | INFO | MAKECP | MESSAGES

| SHUTDOWN | MONITOR | NOTIFY | OPEN | PARAMETERS | PERFMON

| PID | PROTOCOLS | REPORT | SHUTDOWN | STATUS

| STATUS BACKUP | THROWOUT | TRACE | USERID | USERLIST

| VERSION

Usage

This SQL extension executes administrator commands. The *command_name* in the syntax is a SOLID *DBConsole* or SOLID *Remote Control* (teletype) command string (without the quotes); for example, `backup`.

If you are entering these commands in the SOLID *SQL Editor* (teletype), be sure to use the full SOLID SQL syntax; for example, `ADMIN COMMAND 'backup'`. Abbreviations for ADMIN COMMANDs are also available; for example, `ADMIN COMMAND 'bak'.` To access a list of abbreviated commands, execute `ADMIN COMMAND 'help'`.

The result set contains two columns: RC INTEGER and TEXT VARCHAR(254). Integer column RC is a command return code (0 if success), varchar column TEXT is the command reply. The TEXT field contains the same lines that are displayed on SOLID *DBConsole* screen, one line per one result row.

Note that all options of the ADMIN COMMAND are not transactional and cannot be rolled back.

Following is a description of the syntax for each ADMIN COMMAND command option:

Option Syntax	Description
ADMIN COMMAND 'backup [*backup_directory*]'	Makes a backup of the database. The default backup directory is the one defined in the configuration parameter **General.BackupDirectory**. The backup directory may also be given as an argument. For example, **backup abc** creates backup on directory ‘abc’. All directory definitions are relative to the SOLID *Embedded Engine* working directory.
ADMIN COMMAND 'backuplist'	Displays a status list of last backups.
ADMIN COMMAND 'close'	Closes the server from new connections; no new connections are allowed.
ADMIN COMMAND 'describe parameter *param*'	Returns description text of parameter. The following example describes parameter [Com]Trace=y/n. `ADMIN COMMAND 'describe parameter com.trace'`
ADMIN COMMAND 'errorcode *SOLID_error_code'*	Displays a description of an error code. Gives the code number as an argument. For example, '**errorcode 10033**'
ADMIN COMMAND 'help'	Displays available commands.

Option Syntax	Description
ADMIN COMMAND 'info *options*'	Returns server information. *Options* are one or more of the following values, each separated by a space: ■ **Numusers** - number of current users ■ **Maxusers** - maximum number of users ■ **Sernum** - Server serial number ■ **Dbsize** - database size ■ **Logsize** - size of log files ■ **Uptime** - server up since ■ **Bcktime** - timestamp of last successfully completed backup ■ **Cptime** - timestamp of last successfully completed checkpoint ■ **tracestate** - Current trace statem ■ **monitorstate** - Current monitor state, number of users withmonitor enabled, or -1 if all ■ **openstate** - Current open or close state Values are returned in the same order as requested, one row for each value. Example: `ADMIN COMMAND 'info dbsize logsize'`
ADMIN COMMAND 'makecp'	Makes a checkpoint.
ADMIN COMMAND 'messages [**-n**] [**warnings** | **errors**] [*count*]'	Displays server messages. Optional severity and message numbers can also be defined. For example: ADMIN COMMAND 'messages warnings 100' displays last 100 warnings.
ADMIN COMMAND 'monitor {**on** | **off**} [**user** *username* | *user id*]'	Sets server monitoring on and off. Monitoring logs user activity and SQL calls to `soltrace.out` file

Option Syntax	Description
ADMIN COMMAND 'notify {**user** *username* | *user id* | **ALL** } *message*'	This command sends an event to a given user with event identifier NOTIFY. This identifier is used to cancel an event-waiting thread when the statement timeout is not long enough for a disconnect or to change the event registration. The following example sends a notify message to a user with user id **5**; the event then gets the value of the message parameter. ADMIN COMMAND 'notify user 5 Canceled by admin'
ADMIN COMMAND 'open'	Opens server for new connections; new connections are allowed.
ADMIN COMMAND 'parameter [*option*][*name*[=*value*]]'	Displays and sets server parameter values. For example: ■ **parameter** used alone displays all parameters. ■ **parameter genera**l displays all parameters from section "general." ■ **parameter general.readonly** displays a single parameter "readonly" from section "general." ■ **parameter com.trace=yes** sets communication trace on If **-r** is used, then only the current parameter values are returned.

ADMIN COMMAND 'perfmon [*options*] [*subsystem prefix*]'	Returns performance statistics from the server. Options are:
	■ **-c** returns all values as the counter
	■ **-d** returns short descriptions
	■ **-v** returns current values
	■ **-t** returns total values
	By default, some values are averages/second.
	The subsystem prefix is used to find matches to value names. Only those values are returned that match the subsystem prefix.
	The following example returns all information:
	`ADMIN COMMAND 'perfmon'`
	The following example returns all values whose name starts with prefix **File** as counters.
	`ADMIN COMMAND 'perfmon-c File'`

Option Syntax	Description
ADMIN COMMAND 'pid'	Returns server process id.
ADMIN COMMAND 'protocols'	Returns list of available communication protocols, one row for each protocol. Example: `ADMIN COMMAND 'protocols'`
ADMIN COMMAND 'report *filename'*	Generates a report of server info to a file given as an argument.
ADMIN COMMAND 'shutdown'	Stops SOLID *Embedded Engine*.
ADMIN COMMAND 'status'	Displays server statistics.
ADMIN COMMAND 'status backup'	Displays status of the last started backup. The status can be one of the following: ■ If the last backup was successful or any backups have not been requested, the output is 0 SUCCESS. ■ If the backup is in process; for example, started but not ready yet, the output is 14003 ACTIVE. ■ If the last backup failed, the output is: *errorcode* ERROR where the *errcode* shows the reason for the failure
ADMIN COMMAND 'throwout {*username* | *userid* | **all }**	Exits users from SOLID *Embedded Engine*. To exit a specified user, give the user id as an argument. To throw out all users, use the keyword ALL as an argument.
ADMIN COMMAND 'trace {**on** | **off**} **sql** | **rpc** | **sync'**	Sets server trace on or off. This command is similar to the **monitor** command, but traces different entities and a different levels. By default, the output is written to the `soltrace.out` file.
ADMIN COMMAND 'userid'	Returns user identification number of the current connection. Example: `ADMIN COMMAND 'userid'`
ADMIN COMMAND 'userlist [**-l**] [*name* | *id*]'	Displays a list of users. option -l displays more detailed output.
ADMIN COMMAND 'version'	Displays server version info.

ALTER TABLE

ALTER TABLE *base_table_name*

{ADD [COLUMN] *column_identifier data_type* |

DROP [COLUMN] *column_identifier* | RENAME [COLUMN]

column_identifier column_identifier |

MODIFY [COLUMN]

column_identifier data-type} | MODIFY SCHEMA *schema_name* |

SET {OPTIMISTIC | PESSIMISTIC}

Note

Keywords CASCADE and RESTRICT are not supported in the SQL grammar of SOLID *Embedded Engine*. Objects are always dropped with drop behavior RESTRICT.

Usage

The structure of a table may be modified through the `ALTER TABLE` statement. Within the context of this statement, columns may be added, modified, or removed.

The server allows users to change the width of a column using the ALTER TABLE command. A column width can be increased at any time (that is, whether a table is empty [no rows] or non-empty). However, the ALTER TABLE command disallows decreasing the column width when the table is non-empty; a table must be empty to decrease the column width.

Note that a column cannot be dropped if it is part of a unique or primary key.

The owner of a table can be changed using the ALTER TABLE *base_table_name* MODIFY SCHEMA *schema_name* statement. This statement gives all rights to the new owner of the table including creator rights. The old owner's access rights to the table, excluding the creator rights, are preserved.

Individual tables can be set to optimistic or pessimistic with the command ALTER TABLE *base_table_name* `SET {OPTIMISTIC | PESSIMISTIC}`. By default, all tables are optimistic. A database-wide default can be set in the `General` section of the configuration file with the parameter `Pessimistic = yes`.

Example

```
ALTER TABLE TEST ADD X INTEGER;

ALTER TABLE TEST RENAME COLUMN X Y;

ALTER TABLE TEST MODIFY COLUMN X SMALLINT;

ALTER TABLE TEST DROP COLUMN X;
```

ALTER TRIGGER

ALTER TRIGGER *trigger_name_attr* SET ENABLED | DISABLED

trigger_name_attr := [*catalog_name*.[*schema_name*]]*trigger_name* |

Usage

You can alter trigger attributes using the ALTER TRIGGER command. The valid attributes are ENABLED and DISABLED trigger.

The ALTER TRIGGER command causes a Solid server to ignore the trigger when an activating DML statement is issued. With this command, you can also enable a trigger that is currently inactive or disable a trigger that is currently active.

You must be the owner of a table, or a user with DBA authority to alter a trigger from the table.

Example

```
ALTER TRIGGER SET ENABLED trig_on_employee;
```

ALTER USER

ALTER USER *user_name* IDENTIFIED BY *password*

Usage

The password of a user may be modified through the `ALTER USER` statement.

Example

```
ALTER USER MANAGER IDENTIFIED BY O2CPTG;
```

CALL

CALL *procedure_name* [(*parameter* [, *parameter* ...])]

Usage

Stored procedures are called with statement `CALL`.

Example

```
CALL proctest;
```

COMMIT

COMMIT WORK

Usage

The changes made in the database are made permanent by `COMMIT` statement. It terminates the transaction.

Example

```
COMMIT WORK;
```

CREATE CATALOG

CREATE CATALOG *catalog_name*

Usage

Catalogs allow you to logically partition databases so you can organize your data to meet the needs of your business or application. A database can have one or more catalogs. Users are prompted for a default catalog name when creating a new database or converting an old database to a new format. This default catalog name allows for backward compatibility of Solid databases prior to version 3.5.

A catalog can have zero or more *schema_names*. The default schema name is *user_id*. A schema can have zero or more database object names. A database object can be qualified by a schema or user ID.

The catalog name is used to qualify a database object name. Database object names are qualified in all DML statements as:

catalog_name.schema_name.database_object

or

catalog_name.user_id.database_object

Only a user with DBA authority (SYS_ADMIN_ROLE) can create a catalog for a database.

To use schemas, a schema name must be created before creating the database object name. However, a database object name can be created without a schema name. In such cases, database objects are qualified using *user_id* only. For details on creating schemas, read *"CREATE SCHEMA"* on page D-21.

A catalog context can be set in a program using:

SET CATALOG *catalog_name*

A catalog can be dropped from a database using:

DROP CATALOG *catalog_name*

When dropping a catalog name, all objects associated with the catalog name must be dropped prior to dropping the catalog.

Following are the rules for resolving catalog names:

- A fully qualified name (*catalog_name.schema_name.database_object_name*) does not need any name resolution, but will be validated.
- If a catalog context is not set using SET CATALOG, then all database object names are resolved always using the default catalog name as the catalog name. The database object name is resolved using schema name resolution rules. For details on these rules, read *"CREATE SCHEMA"* on page D-21.
- If a catalog context is set and the catalog name cannot be resolved using the *catalog_name* in the context, then *database_object_name* resolution fails.
- To access a database system catalog, users do not need to know the system catalog name. Users can specify "".SYSTEM.*table. Embedded Engine t*ranslates the empty string "" used as a catalog name to the default catalog name. *Embedded Engine* also provides automatic resolution of _SYSTEM schema to the system catalog, even when the catalog name is not provided.

Examples

```
CREATE CATALOG C;
SET CATALOG C;
CREATE SCHEMA S;
SET SCHEMA S;
```

```
CREATE TABLE T;
SELECT * FROM T;
-- the name T is resolved to C.S.T

-- Assume the userid is SMITH
CREATE CATALOG C;
SET CATALOG C;
CREATE TABLE T;
SELECT * FROM T;
--The name T is resolved to C.SMITH.T

-- Assume there is no Catalog context set.
-- Meaning the default catalog name is BASE or the setting
-- of the base catalog.
CREATE SCHEMA S;
SET SCHEMA S;
CREATE TABLE T;
SELECT * FROM T;
--The name T is resolved to <BASE>.S.T

CREATE CATALOG C1;
SET CATALOG C1;
CREATE SCHEMA S1;
SET SCHEMA S1;
CREATE TABLE T1 (c1 INTEGER);

CREATE CATALOG C2;
SET CATALOG C2;
CREATE SCHEMA S2;
SET SCHEMA S2;
CREATE TABLE T1 (c2 INTEGER)

SET CATALOG BASE;
```

```
SET SCHEMA USER;
SELECT * FROM T1;
-- This select will give an error as it
-- cannot resolve the T1.
```

CREATE EVENT

CREATE EVENT *event_name* [(*parameter_definition* [, *parameter_definition* ...])]

Usage

Event alerts are used to signal an event in the database. Events are simple objects with a name. The use of event alerts removes resource consuming database polling from applications.

An event object is created with the SQL statement

CREATE EVENT *event_name* [*parameter_list*]

The name can be any user-specified alphanumeric string. The parameter list specifies parameter names and parameter types. The parameter types are normal SQL types.

Events are dropped with the SQL statement

DROP EVENT *event_name*

Events are triggered and received inside stored procedures. Special stored procedure statements are used to trigger and receive events.

The event is triggered with the stored procedure statement

POST EVENT *event_name* [*parameters*]

Event parameters must be local variables or parameters in the stored procedure where the event is triggered. All clients that are waiting for the posted event will receive the event.

To make a procedure wait for an event to happen, the WAIT EVENT construct is used in a stored procedure:

wait_event_statement ::=

```
   WAIT EVENT
      [event_specification ...]
   END WAIT
```

```
event_specification ::=
   WHEN event_name (parameters) BEGIN
      statements
   END EVENT
```

Each connection has its own event queue. To specify the events to be collected in the event queue command REGISTER EVENT *event_name* (*parameters*) is used. Events are removed from the event queue with command UNREGISTER EVENT *event_name* (*parameters*).

Example of a procedure that waits for an event:

```
"create procedure event_wait(i1 integer)
returns (result varchar)
begin
declare i integer;
declare c char(4);

i := 0;

wait event
   when test1 begin
      result := 'event1';
      return;
   end event

   when test2(i) begin
   end event

   when test3(i, c) begin
  end event
end wait
```

```
if i <> 0 then
   result := 'if';
   post event test1;
else
   result := 'else';
   post event test2(i);
   post event test3(i, c);
end if
end";
```

The creator of an event or the database administrator can grant and revoke access rights. Access rights can be granted to users and roles. The select access right gives waiting access to an event. The insert access right gives triggering access to an event.

Example

```
CREATE EVENT ALERT1(I INTEGER, C CHAR(4));
```

CREATE INDEX

CREATE [UNIQUE] INDEX *index_name*

ON *base_table_name*

(*column_identifier* [ASC | DESC]

[, *column_identifier* [ASC | DESC]] ...)

Usage

Creates an index for a table based on the given columns. Keyword `UNIQUE` specifies that columns being indexed must contain unique values. Keywords `ASC` and `DESC` specify whether the given columns should be indexed in ascending or descending order. If not specified ascending order is used.

Example

```
CREATE UNIQUE INDEX UX_TEST ON TEST (I);
```

```
CREATE INDEX X_TEST ON TEST (I, J);
```

CREATE PROCEDURE

CREATE PROCEDURE *procedure_name* [(*parameter_definition*
[, *parameter_definition* ...])]
[RETURNS (*parameter_definition* [, *parameter_definition* ...])]
BEGIN *procedure_body* END;

parameter_definition ::= *parameter_name data_type*

procedure_body ::= [*declare_statement*; ...]
procedure_statement; [*procedure_statement*; ...]

declare_statement ::= DECLARE *variable_name data_type*

procedure_statement ::= *prepare_statement* | *exec_statement* | *fetch_statement* |
control_statement | *post_statement* | *wait_event_statement* |
wait_register_statement

prepare_statement ::= EXEC SQL PREPARE *cursor_name sql_statement*

execute_statement ::=
EXEC SQL EXECUTE
cursor_name
[USING (*variable* [, *variable* ...])]
[INTO (*variable* [, *variable* ...])] |
EXEC SQL {CLOSE | DROP} *cursor_name* |
EXEC SQL {COMMIT | ROLLBACK} WORK |
EXEC SQL SET TRANSACTION {READ ONLY | READ WRITE} |
EXEC SQL WHENEVER SQLERROR {ABORT | ROLLBACK [WORK], ABORT}
EXEC SEQUENCE *sequence_name*.CURRENT INTO *variable* |

EXEC SEQUENCE *sequence_name*.NEXT INTO *variable* |
EXEC SEQUENCE *sequence_name* SET VALUE USING *variable*

fetch_statement ::= EXEC SQL FETCH *cursor_name*

post_statement ::= POST EVENT *event_name* [*parameters*]

wait_event_statement ::=
WAIT EVENT
[*event_specification* ...]
END WAIT

event_specification ::=
WHEN *event_name* (*parameters*) BEGIN
statements
END EVENT

wait_register-statement ::=
REGISTER EVENT *event_name* (*parameters*) |
UNREGISTER EVENT *event_name* (*parameters*)

control_statement ::=
SET *variable_name* = *value* | *variable_name* := *value* |
WHILE *expression*
LOOP *procedure_statement*... END LOOP |
LEAVE |
IF *expression* THEN *procedure_statement* ...
[ELSEIF *procedure_statement* ... THEN] ...

ELSE *procedure_statement* ... END IF |

RETURN | RETURN SQLERROR OF *cursor_name* | RETURN ROW

Usage

Stored procedures are simple programs, or procedures, that are executed in the server. The user can create a procedure that contains several SQL statements or a whole transaction and execute it with a single call statement. Usage of stored procedures reduces network traffic and allows more strict control to access rights and database operations.

Procedures are created with the statement

CREATE PROCEDURE *name body*

and dropped with the statement

DROP PROCEDURE *name*

Procedures are called with the statement

CALL *name* [*parameter* ...]

Procedures can take several input parameters and return a single row or several rows as a result. The result is built from specified output parameters. Procedures are thus used in ODBC in the same way as the SQL SELECT statement.

Procedures are owned by the creator of the procedure. Specified access rights can be granted to other users. When the procedure is run, it has the creator's access rights to database objects.

The stored procedure syntax is a proprietary syntax modeled from SQL3 specifications and dynamic SQL. Procedures contain control statements and SQL statements.

The following control statements are available in the procedures:

Control statement	Description
set *variable* = *expression*	Assigns a value to a variable. The value can be either a literal value (e.g., 10 or 'text') or another variable. Parameters are considered as normal variables.
variable := *expression*	Alternate syntax for assigning values to variables.

while *expr* loop *statement-list* end loop	Loops while expression is true.
leave	Leaves the innermost while loop and continues executing the procedure from the next statement after the keyword end loop.

if
expr
then
statement-list1
else
statement-list2
end if
	Executes *statements-list1* if expression *expr* is true; otherwise, executes *statement-list2*.

if *expr1* then *statement-list1* elseif *expr2* then *statement-list2* end if	If *expr1* is true, executes *statement-list1*. If *expr2* is true, executes *statement-list2*. The statement can optionally contain multiple *elseif* statements and also an *else* statement.
return	Returns the current values of output parameters and exits the procedure. If a procedure has a one *return row* statement, *return* behaves like *return norow.*
return sqlerror of *cursor-name*	Returns the sqlerror associated with the cursor and exits the procedure.
return row	Returns the current values of output parameters and continues execution.

return norow	Returns the end of the set and exits the procedure.

All SQL DML and DDL statements can be used in procedures. Thus the procedure can, for example, create tables or commit a transaction. Each SQL statement in the procedure is atomic.

Preparing SQL Statements

The SQL statements are first prepared with the statement

EXEC SQL PREPARE *cursor sql-statement*

The *cursor* specification is a cursor name that must be given. It can be any unique cursor name inside the transaction. Note that if the procedure is not a complete transaction, other open cursors outside the procedure may have conflicting cursor names.

Executing Prepared SQL Statements

The *SQL statement* is executed with the statement

EXEC SQL EXECUTE *cursor* [*opt-using*] [*opt-into*]

The optional *opt-using* specification has the syntax

USING (*variable_list*)

where *variable_list* contains a list of procedure variables or parameters separated by a comma. These variables are input parameters for the SQL statement. The SQL input parameters are marked with the standard question mark syntax in the prepare statement. If the SQL statement has no input parameters, the USING specification is ignored.

The optional *opt_nto* specification has the syntax

INTO (*variable_list*)

where *variable_list* contains the variables that the column values of the SQL SELECT statement are stored into. The INTO specification is effective only for SQL SELECT statements.

After the execution of UPDATE, INSERT and DELETE statements an additional variable is available to check the result of the statement. Variable SQLROWCOUNT contains the number of rows affected by the last statement.

Fetching Results

Rows are fetched with the statement

EXEC SQL FETCH *cursor*

If the fetch completed successfully, the column values are stored into the variables defined in the *opt_into* specification.

Checking for Errors

The result of each EXEC SQL statement executed inside a procedure body is stored into the variable SQLSUCCESS. This variable is automatically generated for every procedure. If the previous SQL statement was successful, a value one is stored into SQLSUCCESS. After a failed SQL statement, a value zero is stored into SQLSUCCESS.

EXEC SQL WHENEVER SQLERROR {ABORT | [ROLLBACK [WORK], ABORT}

is used to decrease the need for IF NOT SQLSUCCESS THEN tests after every executed SQL statement in a procedure. When this statement is included in a stored procedure all return values of executed statements are checked for errors. If statement execution returns an error, the procedure is automatically aborted. Optionally the transaction can be rolled back.

The error string of latest failed SQL statement is stored into variable SQLERRSTR.

Using Transactions

EXEC SQL {COMMIT | ROLLBACK} WORK

is used to terminate transactions.

EXEC SQL SET TRANSACTION {READ ONLY | READ WRITE}

is used to control the type of transactions.

Using Sequencer Objects and Event Alerts

Refer to the usage of the CREATE SEQUENCE and CREATE EVENT statements.

Procedure Stack Functions

The following functions may be used to analyze the current contents of the procedure stack: PROC_COUNT(), PROC_NAME(N), PROC_SCHEMA(N).

PROC_COUNT() returns the number of procedures in the procedure stack. This includes the current procedure.

PROC_NAME(N) returns the Nth procedure name is the stack. First procedure position is zero.

PROC_SCHEMA(N) returns the schema name of the Nth procedure in procedure stack.

Example 1

```
"create procedure test2(tableid integer)
   returns (cnt integer)
begin
   exec sql prepare c1 select count(*) from
      sys_tables where id > ?;
   exec sql execute c1 using (tableid) into
      (cnt);
   exec sql fetch c1;
end";
```

Example 2

```
"create procedure return_tables
returns (name varchar)
begin
        exec sql whenever sqlerror rollback, abort;
        exec sql prepare c1 select table_name
             from sys_tables;
        exec sql execute c1 into (name);
        while sqlsuccess loop
            exec sql fetch c1;
            if not sqlsuccess
            then leave;
            end if
            return row;
        end loop;
        exec sql close c1;
end";
```

CREATE ROLE

CREATE ROLE *role_name*

Usage

Creates a new user role.

Example

```
CREATE ROLE GUEST_USERS;
```

CREATE SCHEMA

CREATE SCHEMA *schema_name*

Usage

Creates a collection of database objects, such as tables, views, indexes, events, triggers, sequences, and stored procedures for a database user. The schema name is used to qualify a database object name. Database object names are qualified in all DML statements as:

catalog_name.schema_name.database_object_name

or

user_id.database_object_name

To logically partition a database, users can create a catalog before they create a schema. For details on creating a catalog, read CREATE CATALOG. Note that when creating a new database or converting an old database to a new format, users are prompted for a default catalog name.

To use schemas, a schema name must be created before creating the database object name (such as a table name or procedure name). However, a database object name can be created without a schema name. In such cases, database objects are qualified using *user_id* only.

You can specify the database object names in a DML statement explicitly by fully qualifying them or implicitly by setting the schema name context using:

SET SCHEMA *schema_name*

A schema can be dropped from a database using:

DROP SCHEMA *schema_name*

When dropping a schema name, all objects associated with the schema name must be dropped prior to dropping the schema.

A schema context can be removed using:

SET SCHEMA USER

Following are the rules for resolving schema names:

- A fully qualified name (*catalog_nameschema_name.database_object_name*) does not need any name resolution, but will be validated.
- If a schema context is not set using SET SCHEMA, then all database object names are resolved always using the user id as the schema name.
- If the database object name cannot be resolved from the schema name, then the database object name is resolved from all existing schema names.
- If name resolution finds either zero matching or more than one matching database object name, then a Solid server issues a name resolution conflict error.

Examples

```
-- Assume the userID is SMITH.
CREATE SCHEMA FINANCE;
CREATE TABLE EMPLOYEE (EMP_ID INTEGER);
SET SCHEMA FINANCE;
CREATE TABLE EMPLOYEE (ID INTEGER);
SELECT ID FROM EMPLOYEE;
-- In this case, the table is qualified to FINANCE.EMPLOYEE
SELECT EMP_ID FROM EMPLOYEE;
-- This will give an error as the context is with FINANCE and
-- table is resolved to FINANCE.EMPLOYEE

--The following are valid schema statements: one with a schema context,
--the other without.
SELECT ID FROM FINANCE.EMPLOYEE;
SELECT EMP_ID FROM SMITH.EMPLOYEE
--The following statement will resolve to schema SMITH without a schema
--context
SELECT EMP_ID FROM EMPLOYEE;
```

CREATE SEQUENCE

CREATE [DENSE] SEQUENCE *sequence_name*

Usage

Sequencer objects are objects that are used to get sequence numbers.

Using a dense sequence guarantees that there are no holes in the sequence numbers. The sequence number allocation is bound to the current transaction. If the transaction rolls back, also the sequence number allocations are rolled back. The drawback of dense sequences is that the sequence is locked out from other transactions until the current transaction ends.

Using a sparse sequence guarantees uniqueness of the returned values, but they are not bound to the current transaction. If a transaction allocates a sparse sequence number and later rolls back, the sequence number is simply lost.

The advantage of using a sequencer object instead of a separate table is that the sequencer object is specifically fine-tuned for fast execution and requires less overhead than normal update statements.

Sequence values can be incremented and used within SQL statements. These constructs can be used in SQL:

sequence_name.CURRVAL

sequence_name.NEXTVAL

Sequences can also be used inside stored procedures. The current sequence value can be retrieved using the following stored procedure statement:

EXEC SEQUENCE *sequence_name*.CURRENT INTO *variable*

The new sequence value can be retrieved using the following stored procedure statement:

EXEC SEQUENCE *sequence_name*.NEXT INTO *variable*

Sequence values can be set with the following stored procedure statement:

EXEC SEQUENCE *sequence_name* SET VALUE USING *variable*

Select access rights are required to retrieve the current sequence value. Update access rights are required to allocate new sequence values. These access rights are granted and revoked in the same way as table access rights.

Examples

```
CREATE DENSE SEQUENCE SEQ1;
INSERT INTO ORDER (id) VALUES (order_sequence.NEXTVAL);
```

CREATE TABLE

CREATE TABLE *base_table_name* (*column_element* [, *column_element*] ...)

base_table_name ::= *base_table_identifier* | *schema_name*.*base_table_identifier* |
catalog_name.*schema_name*.*base_table_identifier*

column_element ::= *column_definition* | *table_constraint_definition*

column-definition ::= *column_identifier*
data_type [*column_constraint_definition* [*column_constraint_definition*] ...]

column-constraint-definition ::= NOT NULL | NOT NULL UNIQUE |
NOT NULL PRIMARY KEY | REFERENCES *ref_table_name referenced_columns*
| CHECK (*check_condition*)

table_constraint_definition ::= UNIQUE (*column_identifier* [, *column_identifier*] ...) |
PRIMARY KEY (*column_identifier* [, *column_identifier*] ...) |
CHECK (*check-condition*) |

FOREIGN KEY (*column-identifier* [, *column-identifier]* ...)
REFERENCES *table-name* (*column-identifier* [, *column-identifier*] ...)

Usage

Tables are created through the CREATE TABLE statement. The CREATE TABLE statement requires a list of the columns created, the data types, and, if applicable, sizes of values within each column, in addition to other related alternatives (such as whether or not null values are permitted).

Example

```
CREATE TABLE DEPT (DEPTNO INTEGER NOT NULL, DNAME VARCHAR, PRIMARY
KEY(DEPTNO));
```

```
CREATE TABLE DEPT2 (DEPTNO INTEGER NOT NULL PRIMARY KEY, DNAME VARCHAR);

CREATE TABLE DEPT3 (DEPTNO INTEGER NOT NULL UNIQUE, DNAME VARCHAR);

CREATE TABLE DEPT4 (DEPTNO INTEGER NOT NULL, DNAME VARCHAR,
UNIQUE(DEPTNO));

CREATE TABLE EMP (DEPTNO INTEGER, ENAME VARCHAR, FOREIGN KEY (DEPTNO)
REFERENCES DEPT (DEPTNO));

CREATE TABLE EMP2 (DEPTNO INTEGER, ENAME VARCHAR, CHECK (ENAME IS NOT
NULL), FOREIGN KEY (DEPTNO) REFERENCES DEPT (DEPTNO));
```

CREATE TRIGGER

CREATE TRIGGER *trigger_name* ON *table_name time_of_operation*
triggering_event [*REFERENCING column_reference*] *trigger_body*

where:

trigger_name := *literal*

time_of_operation ::= BEFORE | AFTER

triggering_event :: = INSERT | UPDATE | DELETE

column_reference ::= OLD *old_column_name* [AS] *old_col_identifier*
[, *REFERENCING column_reference* |
NEW *new_column_name* [AS] *new_col_identifier*
[, *REFERENCING column_reference*]

trigger_body ::= *trigger_body*:= [*declare_statement*] *trigger_statement*
{, *trigger_statement*]}

old_column_name := *literal*

new_column_name := *literal*

old_col_identifier := *literal*

new_col_identifier := *literal*

Note

This appendix is intended to provide a quick reference to using SOLID SQL commands. For details on when and how to use triggers, read *Chapter 3, "Stored Procedures, Events, Triggers, and Sequences* in the **SOLID Programmer Guide**.

Usage

A trigger activates a stored procedure code, which a Solid server automatically executes when a user attempts to change the data in a table. You may create one or more triggers on a table, with each trigger defined to activate on a specific INSERT, UPDATE, or DELETE command. When a user modifies data within the table, the trigger that corresponds to the command is activated.

You can only use inline or stored procedures with triggers. In addition, you must first store the procedure with the CREATE PROCEDURE command. A procedure invoked from a trigger body can invoke other triggers.

To create a trigger, you must be a DBA or owner of the table on which the trigger is being defined.

Triggers are created with the statement

CREATE TRIGGER *name body*

and dropped from the system catalog with the statement

DROP TRIGGER *name*

Triggers are disabled by using the statement

ALTER TRIGGER *name*

When you disable a trigger defined on a table, a Solid server ignores the trigger when an activating DML statement is issued. With this command, you can also enable a trigger that is currently inactive.

Note

Following is a brief summary of the keywords and clauses used in the CREATE TRIGGER command. For more in depth information on usage, read Chapter 3 in the **SOLID Programmer Guide**.

Trigger name

The *trigger_name* identifies the trigger and can contain up to 254 characters.

Trigger body

The *trigger_body* is the statement or statements to be executed when a trigger fires. The trigger body definition is identical to the stored procedure definition. The trigger parameters are the table column values. For details on creating a trigger body, read *"CREATE PROCEDURE"* on page D-14.

A trigger body may also invoke any procedure registered with a Solid server. Solid procedure invocation rules follow standard procedure invocation practices.

In a procedure definition, you can use COMMIT and ROLLBACK statements. But, in a trigger body, you *cannot* use COMMIT (including AUTOCOMMIT and COMMIT WORK) and ROLLBACK statements; you can use only the WHENEVER SQLERROR ABORT statement.

You must explicitly check for business logic errors and raise an error.

Trigger Event Clauses

The BEFORE or AFTER clause specifies whether to execute the trigger before or after the the invoking DML statement, which modifies data.

The INSERT | UPDATE | DELETE clause indicates the trigger action when a user action is attempted (INSERT, UPDATE, DELETE).

Statements related to processing a trigger occurs first before commits and autocommits from the invoking DML (INSERT, UPDATE, DELETE) statements on tables. If a trigger body or a procedure called within the trigger body attempts to execute a COMMIT (including AUTOCOMMIT or COMMIT WORK) or ROLLBACK, than a Solid server returns an appropriate run-time error.

INSERT specifies that the trigger is activated by an INSERT on the table. Loading *n* rows of data is considered as *n* inserts.

Note

There may be some performance impact if you try to load the data with triggers enabled. Depending on your business need, you may want to disable the triggers before loading and enable them after loading. For details, see *"ALTER TRIGGER"* on page D-7.

DELETE specifies that the trigger is activated by a DELETE on the table.

UPDATE specifies that the trigger is activated by an UPDATE on the table. Note the following rules for using the UPDATE clause:

- The same column cannot be referenced by more than one UPDATE trigger.
- A Solid server allows for recursive update to the same table and does not prohibit recursive updates to the same row.

A separate trigger can be generated for each INSERT, DELETE, or UPDATE operation on a table. You can define, by default, up to one trigger for each combination of table, event (INSERT, UPDATE, DELETE) and time (BEFORE and AFTER). This means there can be a maximum of 6 triggers per table.

Note

The triggers are applied to each row. This means that if there are 10 inserts, a trigger is executed 10 times.

Referencing Clause

This clause is optional when creating a trigger on an INSERT/UPDATE/DELETE operation. It provides a way to reference the current column identifiers in the case of INSERT and DELETE operations, and both the old column identifier and the new updated column identifier by aliasing the table on which an UPDATE operation occurs.

You must specify the *old_column_identifier* or the *new_col_identifier* to access them before and after an UPDATE operation. A Solid server does not provide access to them unless you define them using the REFERENCING subclause.

OLD *old_column_name* AS *old_col_identifier* or
NEW *new_column_name* AS *new_col_identifier*

The subclause of the REFERENCING clause allows you to reference the values of columns both before and after an UPDATE operation.

It produces a set of old and new column values which can be passed to an inline or stored procedure; once passed, the procedure contains logic (for example, domain constraint checking) used to evaluate these parameter values.

Use the OLD AS clause to alias the table's old identifier as it exists before the UPDATE. Use the NEW AS clause to alias the table's new identifier as it exists after the UPDATE.

You cannot use the same name for the *old_column_name* and the *new_column_name*, or for the *old_column_identifier* and the *new_column_identifier.*

Each column that is referenced as NEW or OLD should have a separate REFERENCING subclause.

The statement atomicity in a trigger is such that operations made in a trigger are visible to the next SQL statements inside the trigger. For example, if you execute an INSERT statement in a trigger and then also perform a select in the same trigger, then the inserted row is visible.

In the case of AFTER trigger, an inserted row or an updated row is visible in the after insert trigger, but a deleted row cannot be seen for a select performed within the trigger. In the case of a BEFORE trigger, an inserted or updated row is invisible within the trigger and a deleted row is visible.

The table below summarizes the statement atomicity in a trigger, indicating whether the row is visible to the SELECT statement in the trigger body.

Operation	BEFORE TRIGGER	AFTER TRIGGER
INSERT	row is invisible	row is visible
UPDATE	previous value is invisible	new value is visible
DELETE	row is visible	row is invisible

Other Restrictions

- You cannot define triggers on a view (even if the view is based on a single table).
- You cannot alter a table that has a trigger defined on it when the dependent columns are affected.
- You cannot create a trigger on a system table.

- To use the stored procedure that a trigger calls, provide the catalog, schema/owner and name of the table on which the trigger is defined. and specify whether to enable or disable the triggers in the table. For more details on triggers and stored procedures, see Chapter 3 of the **SOLID Programmer Guide**.
- You cannot execute triggers that reference dropped or altered objects. To prevent this error:
  - Recreate any referenced object that you drop.
  - Restore any referenced object you changed back to its original state (known by the trigger).
- You can use reserved words in trigger statements if they are enclosed in double quotes. For example, the following CREATE TRIGGER statement references a column named "data" which is a reserved word.

```
"CREATE TRIGGER TRIG1 ON TMPT BEORE INSERT
REFERENCING NEW "DATA" AS NEW_DATA
BEGIN
END"
```

Setting the Maximum Number of Nested Triggers

Triggers can invoke other triggers or a trigger can invoke itself (or recursive triggers). You can nest triggers up to 16 levels deep. The maximum number of nested triggers is set in the `MaxNestedTriggers` parameter in the SQL section of the SOLID.INI configuration file:

`[SQL] MaxNestedTriggers=`*n*

where *n* is the maximum number of nested triggers.

The default is 16 triggers.

Setting the Triggers Cache

Triggers are cached on a separate cache in the Solid server; each user has a separate cache for triggers. As the triggers are executed, the trigger procedure logic is cached in the trigger cache and is resumed when the trigger is executed again.

The cache size is set in the `TriggerCache` parameter in the SQL section of the SOLID.INI configuration file:

`[SQL] TriggerCache=`*n*

where *n* is the number of triggers that is reserved for the cache.

Checking for Errors

At times, it is possible to receive an error in executing a trigger. The error may be due to execution of SQL statements or business logic. If a trigger returns an error, it causes its invoking DML command to fail. To automatically return errors during the execution of an DML statement, you must use the WHENEVER SQLERROR ABORT statement in the trigger body. Otherwise, errors must be checked explicitly within the trigger body after each procedure call or SQL statement.

For any errors in the user written business logic as part of the trigger body, users can receive errors in a procedure variable using the SQL statement:

RETURN SQLERROR *error_string*

or

RETURN SQLERROR *char_variable*

The error is returned in the following format:

```
User error: <error_string>
```

If a user does not specify the RETURN SQLERROR statement in the trigger body, then all trapped SQL errors are raised with a default *error_string* determined by the system. For details, see *"Error Codes"* on page A-1 in the **SOLID Administrator Guide**.

Note

Triggered SQL statements are a part of the invoking transaction. If the invoking DML statement fails due to either the trigger or another error that is generated outside the trigger, all SQL statements within the trigger are rolled back along with the failed invoking DML command.

Triggers Stack Functions

The following functions may be used to analyze the current contents of the trigger stack:

TRIG_COUNT() returns the number of triggers in the trigger stack. This includes the current trigger. The return value is an integer.

TRIG_NAME(n) returns the nth trigger name in the trigger stack. The first trigger position or offset is zero.

TRIG_SCHEMA(n) returns the nth trigger schema name in the trigger stack. The first trigger position or offset is zero. the return value is a string.

Example

```
"CREATE TRIGGER TRIGGER_BI ON TRIGGER_TEST
        BEFORE INSERT
        REFERENCING NEW BI AS NEW_BI
BEGIN
        EXEC SQL PREPARE BI INSERT INTO TRIGGER_OUTPUT VALUES (
        'BI', TRIG_NAME(0), TRIG_SCHEMA(0));
        EXEC SQL EXECUTE BI;
        SET NEW_BI = 'TRIGGER_BI';
END";
```

CREATE USER

CREATE USER *user_name* IDENTIFIED BY *password*

Usage

Creates a new user with a given password.

Example

```
CREATE USER HOBBES IDENTIFIED BY CALVIN;
```

CREATE VIEW

CREATE VIEW *viewed_table_name* [(*column_identifier* [, *column_identifier*]...)]

AS *query-specification*

Usage

A view can be viewed as a virtual table; that is, a table that does not physically exist, but rather is formed by a query specification against one or more tables.

Example

```
CREATE VIEW TEST_VIEW
   (VIEW_I, VIEW_C, VIEW_ID)
   AS SELECT I, C, ID FROM TEST;
```

DELETE

DELETE FROM *table_name* [WHERE *search_ondition*]

Usage

Depending on your search condition the specified row(s) will be deleted from a given table.

Example

```
DELETE FROM TEST WHERE ID = 5;
DELETE FROM TEST;
```

DELETE (positioned)

DELETE FROM *table_name* WHERE CURRENT OF *cursor_name*

Usage

The positioned DELETE statement deletes the current row of the cursor.

Example

```
DELETE FROM TEST WHERE CURRENT OF MY_CURSOR;
```

DROP CATALOG

DROP CATALOG *catalog_name*

Usage

The DROP CATALOG statement drops the specified catalog from the database. All the objects associated with the specified *catalog_name* must be dropped prior to using this statement; the DROP CATALOG statement is not a cascaded operation.

Example

```
DROP CATALOG C1;
```

DROP EVENT

DROP EVENT *event_name*

Usage

The DROP EVENT statement removes the specified event from the database.

Example

```
DROP EVENT EVENT_TEST;
```

DROP INDEX

DROP INDEX *index_name*

Usage

The `DROP INDEX` statement removes the specified index from the database.

Example

```
DROP INDEX UX_TEST;
```

DROP PROCEDURE

DROP PROCEDURE *procedure_name*

Usage

The `DROP PROCEDURE` statement removes the specified procedure from the database.

Example

```
DROP PROCEDURE PROCTEST;
```

DROP ROLE

DROP ROLE *role_name*

Usage

The DROP ROLE statement removes the specified role from the database.

Example

```
DROP ROLE GUEST_USERS;
```

DROP SCHEMA

DROP SCHEMA *schema_name*

Usage

The DROP SCHEMA statement drops the specified schema from the database. All the objects associated with the specified *schema_name* must be dropped prior to using this statement; the DROP SCHEMA statement is not a cascaded operation.

Example

```
DROP SCHEMA FINANCE;
```

DROP SEQUENCE

DROP SEQUENCE *sequence_name*

Usage

The DROP SEQUENCE statement removes the specified sequence from the database.

Example

```
DROP SEQUENCE SEQ1;
```

DROP TABLE

DROP TABLE *base_table_name*

Note

Objects are always dropped with drop behavior RESTRICT.

Usage

The DROP TABLE statement removes the specified table from the database.

Example

```
DROP TABLE TEST;
```

DROP TRIGGER

DROP TRIGGER [*catalog_name*[*schema_name*]]*trigger_name*

DROP TRIGGER *trigger_name*
DROP TRIGGER *schema_name.trigger_name*
DROP TRIGGER c*atalog_name.schema_name.trigger_name*

Usage

Drops (or deletes) a trigger defined on a table from the system catalog.

You must be the owner of a table, or a user with DBA authority to delete a trigger from the table.

Example

```
DROP TRIGGER TRIGA;
```

DROP USER

DROP USER *user_name*

Usage

The `DROP USER` statement removes the specified user from the database. All the objects associated with the specified user_name must be dropped prior to using this statement; the DROP USER statement is not a cascaded operation.

Example

```
DROP USER HOBBES;
```

DROP VIEW

DROP VIEW *viewed_table_name*

Usage

The `DROP VIEW` statement removes the specified view from the database.

Note

Objects are always dropped with drop behavior RESTRICT.

Example

```
DROP VIEW TEST_VIEW;
```

EXPLAIN PLAN FOR

EXPLAIN PLAN FOR *sql_statement*

Usage

The `EXPLAIN PLAN FOR` statement shows the selected search plan for the specified SQL statement.

Example

```
EXPLAIN PLAN FOR select * from tables;
```

GRANT

GRANT {ALL | *grant_privilege* [, *grant_privilege*]...}

ON *table_name*

TO {PUBLIC | *user_name* [, *user_name*]... |

role_name [, *role_name*]... }

[WITH GRANT OPTION]

GRANT *role_name* TO *user_name*

grant_privilege ::= DELETE | INSERT | SELECT |

UPDATE [(*column_identifier* [, *column_identifier*]...)] |

REFERENCES [(*column_identifier* [, *column_identifier*]...)]

GRANT EXECUTE ON *procedure_name*

TO {PUBLIC | *user_name* [, *user_name*]... | *role_name* [, *role_name*]... }

GRANT {SELECT | INSERT} ON *event_name*

TO {PUBLIC | *user_name* [, *user_name*]... | *role_name* [, *role_name*]... }

GRANT {SELECT | UPDATE} ON *sequence_name*

TO {PUBLIC | *user_name* [, *user_name*]... | *role_name* [, *role_name*]... }

Usage

The `GRANT` statement is

1. used to grant privileges to the specified user or role.
2. used to grant privileges to the specified user by giving

the user the privileges of the specified role.

If you do use the optional `WITH GRANT OPTION`, you give permission for the user(s) to whom you are granting the privilege to pass it on to other users.

Example

```
GRANT GUEST_USERS TO CALVIN;
GRANT INSERT, DELETE ON TEST TO GUEST_USERS;
```

HINT

--(* vendor (SOLID), product (Engine), option(hint)

--*hint* ***)--**

hint:=

[MERGE JOIN |

LOOP JOIN |

JOIN ORDER FIXED |

INTERNAL SORT |

EXTERNAL SORT |

INDEX [REVERSE] *table_name.index_name* |

PRIMARY KEY [REVERSE] *table_name*

FULL SCAN *table_name* |

[NO] SORT BEFORE GROUP BY]

Following is a description of the keywords and clauses used in the syntax:

Pseudo comment identifier

The pseudo comment prefix is followed by identifying information. You must specify the vendor as SOLID, product as Engine, and the option, which is the pseudo comment class name, as hint.

Hint

Hints always follow the SELECT, UPDATE, or DELETE keyword that applies to it.

Note

Hints are not allowed after the INSERT keyword.

Each subselect requires its own hint; for example, the following are valid uses of hints syntax:

INSERT INTO ... SELECT *hint* FROM ...

UPDATE *hint* TABLE ... WHERE *column* = (SELECT *hint* ... FROM ...)

DELETE *hint* TABLE ... WHERE *column* = (SELECT hint ... FROM ...)

Example 1

```
SELECT
--(* vendor(SOLID), product(Engine), option(hint)
--MERGE JOIN
--JOIN ORDER FIXED *)--
*
FROM TAB1 A, TAB2 B;
```

```
WHERE A.INTF = B.INTF;
```

Example 2

```
SELECT
--(* vendor(SOLID), product(Engine), option(hint)
--INDEX TAB1.INDEX1
--INDEX TAB1.INDEX1 FULL SCAN TAB2 *)--
*
FROM TAB1, TAB2
WHERE TAB1.INTF = TAB2.INTF;
```

Hint is a specific semantic, corresponding to a specific behavior. Following is a list of possible hints:

Hint	Definition
MERGE JOIN	Directs the Optimizer to choose the merge join access plan in a select query for all tables listed in the FROM clause. Use this hint when the data is sorted by a join key and the nested loop join performance is not adequate. The MERGE JOIN option selects the merge join only where there is an equal predicate between tables. Otherwise, the Optimizer selects LOOP JOIN even if the MERGE JOIN hint is specified. Note that when data is not sorted before performing the merge operation, the SOLID query executor sorts the data. When considering the usage of this hint, keep in mind that the merge join with a sort is more resource intensive than the merge join without the sort.
LOOP JOIN	Directs the Optimizer to pick the nested loop join in a select query for all tables listed in the FROM clause. By default, the Optimizer does not pick the nested loop join. Using the loop join when tables are small and fit in memory may offer greater efficiency than using other complex join algorithms.
JOIN ORDER FIXED	Specifies that the Optimizer use tables in a join in the order listed in the FROM clause of the query. This means that the Optimizer does not attempt to rearrange any join order and does not try to find alternate access paths to complete the join. Before using this hint, be sure to run the EXPLAIN PLAN to view the associated plan. This gives you an idea on the access plan used for executing the query with this join order.

Hint	Definition
INTERNAL SORT	Specifies that the query executor use the internal sort. Use this hint if the expected result set is small (100s of rows as opposed to 1000s of rows); for example, if you are performing some aggregates, ORDER BY with small result sets, or GROUP BY with small result sets, etc. This hint avoids the use of the more expensive external sort.
EXTERNAL SORT	Specifies that the query executor use the external sort. Use this hint when the expected result set is large and does not fit in memory; for example, if the expected result set has 1000s of rows. In addition, specify the SORT working directory in the `solid.ini` before using the external sort hint. If a working directory is not specified, you will receive a run-time error.
INDEX {REVERSE] *table_name.index_name*	Forces a given index scan for a given table. In this case, the Optimizer does not proceed to evaluate if there are any other indexes that can be used to build the access plan or whether a table scan is better for the given query. Before using this hint, it is recommended that you run the EXPLAIN PLAN output to ensure that the plan generated is optimal for the given query. The optional keyword **REVERSE** returns the rows in the reverse order. In this case, the query executor begins with the last page of the index and starts returning the rows in the descending (reverse) key order of the index. Note that in *tablename.indexname*, the *tablename* is a fully qualified table name which includes the *catalogname* and *schemaname*.
PRIMARY KEY [REVERSE] *tablename*	Forces a primary key scan for a given table. The optional keyword **REVERSE** returns the rows in the reverse order. If the primary KEY is not available for the given table, then you will receive a run-time error.

Hint	Definition
FULL SCAN *table_name*	Forces a table scan for a given table. In this case, the optimizer does not proceed to evaluate if there are any other indexes that can be used to build the access plan or whether a table scan is better for the given query. Before using this hint, it is recommended that you run the EXPLAIN PLAN output to ensure that the plan generated is optimal for the given query. In this FULL SCAN, the query executor tries to use the PRIMARY KEY, if one is available. If not, then it uses the SYSTEM KEY.
[NO] SORT BEFORE GROUP BY	Indicates whether the SORT operation occurs before the result set is grouped by the GROUP BY columns. If the grouped items are few (100s of rows) then use NO SORT BEFORE. On the other hand, if the grouped items are large (1000s of rows), then use SORT BEFORE.

Usage

Due to various conditions with the data, user query, and database, the SQL Optimizer is not always able to choose the best possible execution plan. For more efficiency, you may want to force a merge join because you know, unlike the Optimizer, that your data is already sorted.

Or sometimes specific predicates in queries cause performance problems that the Optimizer cannot eliminate. The Optimizer may be using an index that you know is not optimal. In this case, you may want to force the Optimizer to use one that produces faster results.

Optimizer hints is a way to have better control over response times to meet your performance needs. Within a query, you can specify directives or *hints* to the Optimizer, which it then uses to determine its query execution plan. Hints are detected through a pseudo comment syntax from SQL2.

You can place a hint(s) in a SQL statement as a static string, just after a SELECT, INSERT, UPDATE, or DELETE keyword. The hint always follows the SQL statement that applies to it.

Table name resolution in optimizer hints is the same as in any table name in a SQL statement. When there is an error in a hint specification, then the whole SQL statement fails with an error message.

Hints are enabled and disabled using the following configuration parameter in the SOLID.INI.

```
[Hints]
EnableHints = YES | NO
```

The default is **YES**.

Example

```
SELECT
--(* vendor(SOLID), product(Engine), option(hint)
-- INDEX TAB1.IDX1 *)--
* FROM TAB1 WHERE I > 100

SELECT
--(* vendor(SOLID), product(Engine), option(hint)
-- INDEX MyCatalog.mySchema.TAB1.IDX1 *)--
* FROM TAB1 WHERE I > 100

SELECT
--(* vendor(SOLID), product(Engine), option(hint)
-- JOIN ORDER FIXED *)--
* FROM TAB1, TAB2 WHERE TAB1.I >= TAB2.I

SELECT
--(* vendor(SOLID), product(Engine), option(hint)
-- LOOP JOIN *)--
* FROM TAB1, TAB2 WHERE TAB1.I >= TAB2.I

SELECT
--(* vendor(SOLID), product(Engine), option(hint)
-- INDEX REVERSE MyCatalog.mySchema.TAB1.IDX1 *)--
* FROM TAB1 WHERE I > 100

SELECT
--(* vendor(SOLID), product(Engine), option(hint)
-- SORT BEFORE GROUP BY *)--
```

```
AVG(I) FROM TAB1 WHERE I > 10 GROUP BY I2

SELECT
--(* vendor(SOLID), product(Engine), option(hint)
-- INTERNAL SORT *)--
* FROM TAB1 WHERE I > 10 ORDER BY I2
```

INSERT

INSERT INTO *table_name* [(*column_identifier* [, *column_identifier*]...)]

VALUES (*insert_value*[, *insert_value*]...)

Usage

There are several variations of the `INSERT` statement. In the simplest instance, a value is provided for each column of the new row in the order specified at the time the table was defined (or altered). In the preferable form of the `INSERT` statement the columns are specified as part of the statement and they needn't to be in any specific order as long as the orders of the column and value lists match with one another.

Example

```
INSERT INTO TEST (C, ID) VALUES (0.22, 5);
INSERT INTO TEST VALUES (0.35, 9);
```

INSERT (Using Query)

INSERT INTO *table_name* [(*column_identifier* [, *column_identifier*]...)]

query_specification

Usage

The query specification creates a virtual table. Using the `INSERT` statement the rows of created virtual table are inserted into the specified table (the degree and data types of the virtual table and inserted columns must match).

Example

```
INSERT INTO TEST (C, ID) SELECT A, B FROM INPUT_TO_TEST;
```

REVOKE (Role from User)

REVOKE {*role_name* [, *role_name*]... }

FROM {PUBLIC | *user_name* [, *user_name*]... }

Usage

The `REVOKE` statement is used to take a role away from users.

Example

```
REVOKE GUEST_USERS FROM HOBBES;
```

REVOKE (Privilege from Role or User)

REVOKE

{ALL | *revoke_privilege* [, *revoke_privilege*]... } ON *table-name*

FROM {PUBLIC | *user_name* [, *user_name*]... | *role_name* [, *role_name*]... }

revoke-privilege ::= DELETE | INSERT | SELECT |

UPDATE [(*column_identifier* [, *column_identifier*]...)] |

REFERENCES

REVOKE EXECUTE ON *procedure_name*

FROM {PUBLIC | *user_name* [, *user_name*]... | *role_name* [, *role_name*]... }

REVOKE {SELECT | INSERT} ON *event_name* FROM

{PUBLIC | *user_name* [, *user_name*]... | *role_name* [, *role_name*]... }

REVOKE {SELECT | INSERT} ON *sequence_name*

FROM {PUBLIC | *user_name* [, *user_name*]... | *role_name* [, *role_name*]... }

Note

Keywords CASCADE and RESTRICT are not supported in the SQL grammar of SOLID *Embedded Engine*.

Usage

The REVOKE statement is used to take privileges away from users and roles.

Example

```
REVOKE INSERT ON TEST FROM GUEST_USERS;
```

ROLLBACK

ROLLBACK WORK

Usage

The changes made in the database are discarded by ROLLBACK statement. It terminates the transaction.

Example

```
ROLLBACK WORK;
```

SELECT

SELECT [ALL | DISTINCT] *select-list*

FROM *table_reference_list*

[WHERE *search_condition*]

[GROUP BY *column_name* [, *column_name*]...]

[HAVING *search_condition*]

[[UNION | INTERSECT | EXCEPT] [ALL] *select_statement*]...

[ORDER BY {*unsigned integer* | *column_name*}

[ASC | DESC]]

Usage

The SELECT statement is used to retrieve information.

Important

SOLID provides a consistent view of data within one transaction; that is, it sees the database as it was at the moment it was started. This is implemented by the multiversion SOLID Bonsai Tree that stores the active data, that is, data that has been written to the database since the beginning of the oldest active transaction in central memory. Also a SELECT begins a new transaction and if not committed or rolled back, it remains active thus causing the Bonsai

Tree to grow.

New data is merged to the main storage tree as soon as no transaction needs to see the old versions of the rows. To ensure the efficient operation of the Bonsai Tree, also commit read-only transactions as soon as all rows are retrieved. This releases the read level and allows the merge process to keep the Bonsai Tree smaller.
Using AUTOCOMMIT does not help. This is because SOLID cannot immediately commit SELECTs since the rows need to be retrieved by the client application first. In AUTOCOMMIT mode, the next SQL statement processing triggers the commit for previous SELECT statement. But if that next statement never comes, the transaction is left open until the connection timeout expires.

Example

```
SELECT ID FROM TEST;

SELECT DISTINCT ID, C FROM TEST WHERE ID = 5;

SELECT DISTINCT ID FROM TEST ORDER BY ID ASC;

SELECT NAME, ADDRESS FROM CUSTOMERS UNION SELECT NAME, DEP FROM
PERSONNEL;
```

SET

SET CATALOG '*catalog_name*'

SET SQL INFO {ON | OFF} [FILE {*file_name* | "*file_name*" | '*file_name*'}]
[LEVEL *info_level*]

SET SQL SORTARRAYSIZE {*array-size* | DEFAULT}

SET SQL JOINPATHSPAN {*path-span* | DEFAULT}

SET SQL CONVERTORSTOUNIONS
{YES [COUNT *value*] | NO | DEFAULT}

SET OPTIMISTIC LOCK TIMEOUT *seconds*

SET LOCK TIMEOUT *timeout-in-seconds*

SET SCHEMA '*schema_name*' | '*user_name*'

SET SCHEMA USER

SET STATEMENT MAXTIME *minutes*

SET TRANSACTION READ ONLY

SET TRANSACTION READ WRITE

SET TRANSACTION ISOLATION LEVEL READ COMMITTED

SET TRANSACTION ISOLATION LEVEL
REPEATABLE READ

SET TRANSACTION ISOLATION LEVEL SERIALIZABLE

Usage

All the settings are read per user session unlike the settings in the `solid.ini` file which are automatically read each time SOLID *Embedded Engine* is started.

In `SQL INFO` the default file is a global `soltrace.out` shared by all users. If the file name is given, all future `INFO ON` settings will use that file unless a new file is set. It is recommended that the file name is given in single quotes, because otherwise the file name is converted to uppercase. The info output is appended to the file and the file is never truncated, so after the info file is not needed anymore, the user must manually delete the file. If the file open fails, the info output is silently discarded.

The default SQL INFO LEVEL is 4. A good way to generate useful info output is to set info on with a new file name and then execute the SQL statement using EXPLAIN PLAN FOR syntax. This method gives all necessary estimator information but does not generate output from the fetches which may generate a huge output file.

The sort array is used for in memory sorts in the SQL interpreter. The minimum value for SORTARRAYSIZE is 100. If a smaller value is given, minimum value 100 will be used. If large sorts are needed, it is recommended that the external sorter facility is used (in Sorter section in solid.ini) instead on using very large SORTARRAYSIZE.

The COUNT parameter in SQL CONVERTORSTOUNIONS tells how many ors are converted to unions. The default is 10 which should be enough in most cases.

SET CATALOG sets the catalog name context in a program.

SET OPTIMISTIC LOCK TIMEOUT sets lock time out separately for optimistic tables per connection. Note that when using SELECT FOR UPDATE, the selected rows are locked also for tables with optimistic concurrency control.

SET SCHEMA sets the schema name context when implicitly qualifying a database object name in a session. To remove the schema context, use the SET SCHEMA USER command. For details, read *"SET SCHEMA"* on page D-51.

SET STATEMENT MAXTIME sets connection specific maximum execution time in minutes. Setting is effective until a new maximum time if set. Zero time means no maximum time, which is also the default.

The SET TRANSACTION settings are borrowed from ANSI SQL. It sets the transaction isolation level. To detect conflicts between transactions, define the transaction with a Repeatable Read or Serializable isolation level using the SET TRANSACTION ISOLATION LEVEL command.

Example

```
SET SQL INFO ON FILE 'sqlinfo.txt' LEVEL 5
```

SET SCHEMA

SET SCHEMA {'*schema_name*' | USER | '*user_name*'}

Usage

SOLID *Embedded Engine* supports SQL89 style schemas for database entity name qualifying. All created database entities belong to a schema, and different schemas may contain entities with same name. In keeping with the ANSI SQL2 standard, the *user_name* or *schema_name* may be enclosed in single quotes.

The default schema can be changed with the SET SCHEMA statement. Schema can be changed to the current user name by using the SET SCHEMA USER statement. Alternatively schema can be set to '*user_name*' which must be a valid user name in the database.

The algorithm to resolve entity names [*schema_name*.]*table_identifier* is the following:

1. If *schema_name* is given then *table_identifier* is searched only from that schema.
2. If *schema_name* is not given, then
   a. First *table_identifier* is searched from default schema. Default schema is initially the same as user name, but can be changed with SET SCHEMA statement
   b. Then *table_identifier* is searched from all schemas in the database. If more than one entity with same *table_identifier* and type (table, procedure, ...) is found, a new error code 13110 (Ambiguous entity name *table_identifier*) is returned.

   The SET SCHEMA statement effects only to default entity name resolution and it does not change any access rights to database entities. It sets the default schema name for unqualified names in statements that are prepared in the current session by an execute immediate statement or a prepare statement.

Example

```
SET SCHEMA 'CUSTOMERS'
```

UPDATE (Positioned)

UPDATE *table_name*

SET [*table_name.*]*column_identifier* = {*expression* | NULL}

[, [*table_name.*]*column_identifier* = {*expression* | NULL}]...

WHERE CURRENT OF *cursor_name*

Usage

The positioned UPDATE statement updates the current row of the cursor. The name of the cursor is defined using ODBC API function named SQLSetCursorName.

Example

```
UPDATE TEST SET C = 0.33
WHERE CURRENT OF MYCURSOR
```

UPDATE (Searched)

UPDATE *table-name*

SET [*table_name.*]*column_identifier* = {*expression* | NULL}

[, [*table_name.*]*column_identifier* = {*expression* | NULL}]...

[WHERE *search_condition*]

Usage

The UPDATE statement is used to modify the values of one or more columns in one or more rows, according the search conditions.

Example

```
UPDATE TEST SET C = 0.44 WHERE ID = 5
```

Table_reference

Table_reference	
table_reference_list	::= *table_reference* [, *table-reference* …]
table_reference	::= *table_name* [[AS] *correlation_name*] | *derived_table* [[AS] *correlation_name* [(*derived_column_list*)]] | *joined_table*
table_name	::= *table_identifier* | *schema_name.table_identifier*
derived_table	::= *subquery*
derived_column_list	::= *column_name_list*
joined_table	::= *cross_join* | *qualified_join* | (*joined_table*)
cross_join	::= *table_reference* CROSS JOIN *table_reference*
qualified_join	::= *table_reference* [NATURAL] [*join_type*] JOIN *table_reference* [*join_specification*]
join_type	::= INNER | *outer_join_type* [OUTER] | UNION
outer_join_type	::= LEFT | RIGHT | FULL
join_specification	::= *join_condition* | *named_columns_join*
join_condition	::= ON *search_condition*
named_columns_join	::= USING (*column_name_list*)
column_name_list	::= *column_identifier* [{ , *column_identifier*} …]

Query_specification

Query_specification	
query_specification	::= SELECT [DISTINCT | ALL] *select_list* *table_expression*
select_list	::= * | *select_sublist* [{, *select_sublist*} ...]
select_sublist	::= *derived_column* | [*table_name* | *table_identifier*].*
derived_column	::= *expression* [[AS] *column_alias*]]
table_expression	::= FROM *table_reference_list* [WHERE *search_condition*] [GROUP BY *column_name_list* [[UNION | INTERSECT | EXCEPT] [ALL] [CORRE- SPONDING [BY (*column_name_list*)]] *query_specification*] [HAVING *search_condition*]

Search_condition

Search_condition	
search_condition	::= *search_item* | *search_item* { AND | OR } *search_item*
search_item	::= [NOT] { *search_test* | (*search_condition*) }
search_test	::= *comparison_test* | *between_test* | *like_test* | *null_test* | *set_test* | *quantified_test* | *existence_test*
comparison_test	::= *expression* { = | <> | < | <= | > | >= } { *expression* | *subquery* }
between_test	::= *column_identifier* [NOT] BETWEEN *expression* AND *expression*
like_test	::= *column_identifier* [NOT] LIKE *value* [ESCAPE *value*]

Search_condition	
null_test	::= *column_identifier* IS [NOT] NULL
set_test	::= *expression* [NOT] IN ({ *value* [,*value*]... | *subquery* })
quantified_test	::= *expression* { = | <> | < | <= | > | >= } [ALL | ANY | SOME] *subquery*
existence_test	::= EXISTS *subquery*

Check_condition

Check_condition	
check_condition	::= *check_item* | *check_item* { AND | OR } *check_item*
check_item	::= [NOT] { *check_test* | (*check_condition*) }
check_test	::= *comparison_test* | *between_test* | l*ike_test* | *null_test* | *list_test*
comparison_test	::= *expression* { = | <> | < | <= | > | >= } { *expression* | *subquery* }
between_test	::= *column_identifier* [NOT] BETWEEN *expression* AND *expression*
like_test	::= *column_identifier* [NOT] LIKE *value* [ESCAPE *value*]
null_test	::= *column_identifier* IS [NOT] NULL
l*ist_test*	::= *expression* [NOT] IN ({ *value* [,*value*]...})

Expression

Expression	
expression	::= *expression_item* | *expression_item* { + | _ | * | / } *expression_item*
expression_item	::= [+ | _] { *value* | *column_identifier* | *function* | *case_expression* | *cast_expression* | (*expression*) }
value	::= *literal* | USER | *variable*
function	::= *set_function* | *null_function* | *string_function* | *numeric_function* | *datetime_function* | *system_function* | *datatypeconversion_function*
set_function	::= COUNT (*) | { AVG | MAX | MIN | SUM | COUNT } ({ ALL | DISTINCT } *expression*)
null_function	::= { NULLVAL_CHAR( ) | NULLVAL_INT( ) }
datatypeconversion_function	::= CONVERT_CHAR(*value_exp*) | CONVERT_DATE(*value_exp*) | CONVERT_DECIMAL(*value_exp*) | CONVERT_DOUBLE(*value_exp*) | CONVERT_FLOAT(*value_exp*) | CONVERT_INTEGER(*value_exp*) | CONVERT_LONGVARCHAR(*value_exp*) | CONVERT_NUMERIC(*value_exp*) | CONVERT_REAL(*value_exp*) | CONVERT_SMALLINT(*value_exp*) | CONVERT_TIME(*value_exp*) | CONVERT_TIMESTAMP(*value_exp*) | CONVERT_TINYINT(*value_exp*) | CONVERT_VARCHAR(*value_exp*)
case_expression	::= *case_abbreviation* | *case_specification*
case_abbreviation	::= NULLIF(*value_exp*, *value_exp*) | COALESCE(*value_exp* {, *value_exp*}…)

Expression	
case_specification	::= CASE [*value_exp*] WHEN *value_exp* THEN {*value_exp*} [WHEN *value_exp* THEN {*value_exp*} …] [ELSE {*value_exp*}] END
cast_expression	::= CAST (*value-exp* AS *-data-type*)

String Function

Function	Purpose
ASCII(*str*)	Returns the integer equivalent of string *str*
CHAR(*code*)	Returns the character equivalent of code
CONCAT(*str1*, *str2*)	Concatenates *str2* to *str1*
str1 { + | || } *str2*	Concatenates *str2* to *str1*
INSERT(*str1*, *start*, *length*, *str2*)	Merges strings by deleting length characters from *str1* and inserting *str2*
LCASE(*str*)	Converts string *str* to lowercase
LEFT(*str, count*)	Returns leftmost count characters of string *str*
LENGTH(*str*)	Returns the number of characters in *str*
LOCATE(*str1*, *str2* [, *start*])	Returns starting position of *str1* within *str2*
LTRIM(*str*)	Removes leading spaces of *str*
POSITION (*str1* IN *str2*)	Returns starting position of *str1* within *str2*
REPEAT(*str, count*)	Returns characters of *str* repeated count times
REPLACE(*str1*, *str2*, *str3*)	Replaces occurrences of *str2* in *str1* with *str3*
RIGHT(*str, count*)	Returns the rightmost count characters of string *str*
RTRIM(*str*)	Removes trailing spaces in *str*
SPACE(*count*)	Returns a string of count spaces
SUBSTRING(*str, start, length*)	Derives substring from *str* beginning at *start*

Function	Purpose
UCASE(*str*)	Converts *str* to uppercase

Numeric Function

Function	Purpose
ABS(*numeric*)	Absolute value of *numeric*
ACOS(*float*)	Arccosine of *float*
ASIN(*float*)	Arcsine of *float*
ATAN(*float*)	Arctangent of *float*
ATAN2(*float1*, *float2*)	Arctangent of the *x* and *y* coordinates, specified by *float1* and *float2*, respectively, as an angle, expressed in radians
CEILING(*numeric*)	Smallest integer greater than or equal to *numeric*
COS(*float*)	Cosine of *float*
COT(*float*)	Cotangent of *float*
DEGREES(*numeric*)	Converts *numeric* radians to degrees
EXP(*float*)	Exponential value of *float*
FLOOR(*numeric*)	Largest integer less than or equal to *numeric*
LOG(*float*)	Natural logarithm of *float*
LOG10(*float*)	Base 10 log of *float*
MOD(*integer1*, *integer2*)	Modulus of *integer1* divided by *integer2*
PI()	Pi as a floating point number
POWER(*numeric*, *integer*)	Value of *numeric* raised to the power of *integer*
RADIANS(*numeric*)	Number of radians converted from *numeric*
ROUND(*numeric*, *integer*)	*Numeric* rounded to *integer*
SIGN(*numeric*)	Sign of *numeric*
SQRT(*float*)	Square root of *float*
TAN(*float*)	Tangent of *float*
TRUNCATE(*numeric*, *integer*)	*Numeric* truncated to *integer*

Date Time Function

Function	Purpose
CURDATE()	Returns the current date
CURTIME()	Returns the current time
DAYNAME(*date*)	Returns a string with the day of the week
DAYOFMONTH(*date*)	Returns the day of the month as an integer between 1 and 31
DAYOFWEEK(*date*)	Returns the day of the week as an integer between 1 and 7, where 1 represents Sunday
DAYOFYEAR(*date*)	Returns the day of the year as an integer between 1 and 366
EXTRACT (*date field* FROM *date_exp*)	Isolates a single field of a datetime or a interval and converts it to a number.
HOUR(*time_exp*)	Returns the hour as an integer between 0 and 23
MINUTE(*time_exp*)	Returns the minute as an integer between 0 and 59
MONTH(*date*)	Returns the month as an integer between 1 and 12
MONTHNAME(*date*)	Returns the month name as a string
NOW()	Returns the current date and time as a timestamp
QUARTER(*date*)	Returns the quarter as an integer between 1 and 4
SECOND(*time_exp*)	Returns the second as an integer between 0 and 59
TIMESTAMPADD(interval, *integer_exp*, timestamp_exp)	Calculates a timetamp by adding *integer_exp* intervals of type interval to *timestamp_exp* Keywords used to express valid TIMESTAMPADD interval values are: SQL_TSI_FRAC_SECOND SQL_TSI_SECOND SQL_TSI_MINUTE SQL_TSI_HOUR SQL_TSI_DAY SQL_TSI_WEEK SQL_TSI_MONTH SQL_TSI_QUARTER SQL_TSI_YEAR

Function	Purpose
TIMESTAMPDIFF(interval, *timestamp-exp1*, *timestamp-exp2*)	Returns the integer number of intervals by which *timestamp-exp2* is greater than *timestamp-exp1* Keywords used to express valid TIMESTAMPDIFF interval values are: SQL_TSI_FRAC_SECOND SQL_TSI_SECOND SQL_TSI_MINUTE SQL_TSI_HOUR SQL_TSI_DAY SQL_TSI_WEEK SQL_TSI_MONTH SQL_TSI_QUARTER SQL_TSI_YEAR
WEEK(*date*)	Returns the week of the year as an integer between 1 and 52
YEAR(*date*)	Returns the year as an integer

System Function

Function	Purpose
IFNULL(*exp*, *value*)	If *exp* is null, returns value; if not, returns exp
USER()	Returns the user authorization name
UIC()	Returns the connection id associated with the connection

Data_type

Data_type	
data_type	::= {BINARY | CHAR [*length*] | DATE | DECIMAL [(*precision* [, *scale*])] | DOUBLE PRECISION | FLOAT [(*precision*)] | INTEGER | LONG VARBINARY | LONG VARCHAR | LONG WVARCHAR | NUMERIC [(*precision* [, *scale*])] | REAL | SMALLINT | TIME | TIMESTAMP [(*timestamp precision*)] | TINYINT | VARBINARY | VARCHAR [(*length*)] } | WCHAR | WVARCHAR [*length*]

Date and Time Literals

Date/time literal	
date_literal	´YYYY-MM-DD´
time_literal	´HH:MM:SS´
timestamp_literal	´YYYY-MM-DD HH:MM:SS´

Pseudo Columns

The following pseudo columns may also be used in the select-list of a SELECT statement:

Pseudo column	Type	Explanation
ROWVER	VARBINARY(254)	Version of the row in a table.
ROWID	VARBINARY(10)	Persistent id for a row in a table.
ROWNUM	DECIMAL(16,2)	Row number indicates the sequence in which a row was selected from a table or set of joined rows. The first row selected has a ROWNUM of 1, the second row has 2, etc. ROWNUM is chiefly useful for limiting the number of rows returned by a query for example, WHERE ROWNUM < 10).

Note

Since ROWID and ROWVER refer to a single row, they may only be used with queries that return rows from a single table.

E

System Views and System Tables

System Views

SOLID *Embedded Engine* supports views as specified in the X/Open SQL Standard.

COLUMNS

The COLUMNS system view identifies the columns which are accessible to the current user.

Column name	Data type	Description
TABLE_CATALOG	WVARCHAR	the name of the catalog containing TABLE_NAME
TABLE_SCHEMA	WVARCHAR	the name of the schema containing TABLE_NAME
TABLE_NAME	WVARCHAR	the name of the table or view
COLUMN_NAME	WVARCHAR	the name of the column of the specified table or view
DATA_TYPE	WVARCHAR	the data type of the column
SQL_DATA_TYPE_NUM	SMALLINT	ODBC compliant data type number
CHAR_MAX_LENGTH	INTEGER	maximum length for a character data type column; for others NULL

Column name	Data type	Description
NUMERIC_PRECISION	INTEGER	the number of digits of mantissa precision of the column, if DATA_TYPE is approximate numeric data type, NUMERIC_PREC_RADIX indicates the units of measurement; for other numeric types contains the total number of decimal digits allowed in the column; for character data types NULL
NUMERIC_PREC_RADIX	SMALLINT	the radix of numeric precision if DATA_TYPE is one of the approximate numeric data types; otherwise NULL
NUMERIC_SCALE	SMALLINT	total number of significant digits to the right of the decimal point; for INTEGER and SMALLINT 0; for others NULL
NULLABLE	CHAR	if column is known to be not nullable 'NO'; otherwise 'YES'
NULLABLE_ODBC	SMALLINT	ODBC, if column is known to be not nullable '0'; otherwise '1'
REMARKS	LONG WVARCHAR	reserved for future use

SERVER_INFO

The SERVER_INFO system view provides attributes of the current database system or server.

Column name	Data type	Description
SERVER_ATTRIBUTE	WVARCHAR	identifies an attribute of the server
ATTRIBUTE_VALUE	WVARCHAR	the value of the attribute

TABLES

The TABLES system view identifies the tables accessible to the current user.

Column name	Data type	Description
TABLE_CATALOG	WVARCHAR	the name of the catalog containing TABLE_NAME
TABLE_SCHEMA	WVARCHAR	the name of the schema containing TABLE_NAME
TABLE_NAME	WVARCHAR	the name of the table or view
TABLE_TYPE	WVARCHAR	the type of the table
REMARKS	LONG WVARCHAR	reserved for future use

USERS

The USERS system view identifies users and roles.

Column name	Data type	Description
ID	INTEGER	User or role id
NAME	WVARCHAR	User or role name
TYPE	WVARCHAR	User type, either USER or ROLE
PRIV	INTEGER	Privilege information
PRIORITY	INTEGER	Reserved for future use
PRIVATE	INTEGER	Is user private or public (used in SOLID *SynchroNet* only)

System Tables

SQL_LANGUAGES

The SQL_LANGUAGES system table lists the SQL standards and SQL dialects which are supported.

Column name	Data type	Description
SOURCE	WVARCHAR	the organization that defined this specific SQL version
SOURCE_YEAR	WVARCHAR	the year the relevant standard was approved
CONFORMANCE	WVARCHAR	the conformance level at which conformance to the relevant standard
INTEGRITY	WVARCHAR	indicates whether the Integrity Enhancement Feature is supported
IMPLEMENTATION	WVARCHAR	identifies uniquely the vendor's SQL language; NULL if SOURCE is 'ISO'
BINDING_STYLE	WVARCHAR	the binding style 'DIRECT', *EMBED' or 'MODULE'
PROGRAMMING_LANG	WVARCHAR	the host language used

SYS_ATTAUTH

Column name	Data type	Description
REL_ID	INTEGER	table id
UR_ID	INTEGER	user or role id
ATTR_ID	INTEGER	column id
PRIV	INTEGER	privilege info
GRANT_ID	INTEGER	grantor id
GRANT_TIM	TIMESTAMP	grant time

SYS_CARDINAL

Column name	Data type	Description

REL_ID	INTEGER	the relation id as in SYS_TABLES
CARDIN	INTEGER	the number of rows in the table
SIZE	INTEGER	the size of the data in the table
LAST_UPD	TIMESTAMP	the timestamp of the last update in the table

SYS_CATALOGS

The SYS_CATALOGS lists available catalogs.

Column name	Data type	Description
ID	INTEGER	catalog id
NAME	WVARCHAR	catalog name
CREATIME	TIMESTAMP	create date and time
CREATOR	WVARCHAR	creator name

SYS_COLUMNS

Column name	Data type	Description
ID	INTEGER	unique column identifier
REL_ID	INTEGER	the relation id as in SYS_TABLES
COLUMN_NAME	WVARCHAR	the name of the column
COLUMN_NUMBER	INTEGER	the number of the column in the table (in creation order)
DATA_TYPE	WVARCHAR	the data type of the column
SQL_DATA_TYPE_NUM	SMALLINT	ODBC compliant data type number
DATA_TYPE_NUMBER	INTEGER	internal data type number
CHAR_MAX_LENGTH	INTEGER	maximum length for a CHAR field
NUMERIC_PRECISION	INTEGER	numeric precision
NUMERIC_PREC_RADIX	SMALLINT	numeric precision radix
NUMERIC_SCALE	SMALLINT	numeric scale

Column name	Data type	Description
NULLABLE	CHAR	are NULL values allowed (Yes, No)
NULLABLE_ODBC	SMALLINT	ODBC, are NULL values allowed (1,0)
FORMAT	WVARCHAR	reserved for future use
DEFAULT_VAL	VARBINARY	reserved for future use
ATTR_TYPE	INTEGER	user defined (0) or internal (>0)
REMARKS	LONG WVARCHAR	reserved for future use

SYS_EVENTS

Column name	Data type	Description
ID	INTEGER	unique event identifier
EVENT_NAME	WVARCHAR	the name of the event
EVENT_PARAMCOUNT	INTEGER	number of parameters
EVENT_PARAMTYPES	LONG VARBINARY	types of parameters
EVENT_TEXT	WVARCHAR	the body of the event
EVENT_SCHEMA	WVARCHAR	the owner of the event
EVENT_CATALOG	WVARCHAR	the owner of the event
CREATIME	TIMESTAMP	creation time
TYPE	INTEGER	reserved for future use

SYS_FORKEYPARTS

Column name	Data type	Description
KEY_CATALOG	INTEGER	creator name or the owner of the key
ID	INTEGER	foreign key identifier
KEYP_NO	INTEGER	keypart number
ATTR_NO	INTEGER	column number

ATTR_ID	INTEGER	column identifier
ATTR_TYPE	INTEGER	column type
CONST_VALUE	VARBINARY	possible internal constant value; otherwise NULL

SYS_FORKEYS

Column name	Data type	Description
ID	INTEGER	foreign key identifier
REF_REL_ID	INTEGER	referenced table identifier
CREATE_REL_ID	INTEGER	creator table identifier
REF_KEY_ID	INTEGER	referenced key identifier
REF_TYPE	INTEGER	reference type
KEY_SCHEMA	WVARCHAR	creator name
KEY_CATALOG	WVARCHAR	creator name or the owner of the key
KEY_NREF	INTEGER	number of referenced key parts

SYS_INFO

Column name	Data type	Description
PROPERTY	WVARCHAR	the name of the property
VALUE_STR	WVARCHAR	value as a string
VALUE_INT	INTEGER	value as an integer

SYS_KEYPARTS

Column name	Data type	Description
ID	INTEGER	unique key identifier

REL_ID	INTEGER	the relation id as in SYS_TABLES
KEYP_NO	INTEGER	keypart identifier
ATTR_ID	INTEGER	column identifier
ATTR_NO	INTEGER	the number of the column in the table (in creation order)
ATTR_TYPE	INTEGER	the type of the column
CONST_VALUE	VARBINARY	constant value or NULL
ASCENDING	CHAR	is the key ascending (Yes) or descending (No)

SYS_KEYS

Column name	Data type	Description
ID	INTEGER	unique key identifier
REL_ID	INTEGER	the relation id as in SYS_TABLES
KEY_NAME	WVARCHAR	the name of the key
KEY_UNIQUE	CHAR	is the key unique (Yes, No)
KEY_NONUNIQUE_ODBC	SMALLINT	ODBC, is the key NOT unique (1, 0)
KEY_CLUSTERING	CHAR	is the key a clustering key (Yes, No)
KEY_PRIMARY	CHAR	is the key a primary key (Yes, No)
KEY_PREJOINED	CHAR	reserved for future use
KEY_SCHEMA	WVARCHAR	the owner of the key
KEY_NREF	INTEGER	internal system specific information

SYS_PROCEDURES

Column name	Data type	Description

ID	INTEGER	unique procedure identifier
PROCEDURE_NAME	WVARCHAR	procedure name
PROCEDURE_TEXT	LONG WVARCHAR	procedure body
PROCEDURE_BIN	LONG VARBINARY	compiled form of the procedure
PROCEDURE_SCHEMA	WVARCHAR	the name of the schema containing PROCEDURE_NAME
PROCEDURE_CATALOG	WVARCHAR	the name of the catalog containing PROCEDURE_NAME
CREATIME	TIMESTAMP	creation time
TYPE	INTEGER	reserved for future use

SYS_PROCEDURE_COLUMNS

The SYS_PROCEDURE_COLUMNS defines input parameters and result set columns.

Column name	Data type	Description
PROCEDURE_ID	INTEGER	procedure id
COLUMN_NAME	WVARCHAR	procedure column name
COLUMN_TYPE	SMALLINT	procedure column type (SQL_PARAM_INPUT or SQL_RESULT_COL)
DATA_TYPE	SMALLINT	column's SQL data type
TYPE_NAME	WVARCHAR	column's SQL data type name
COLUMN_SIZE	INTEGER	size of the procedure column
BUFFER_LENGTH	INTEGER	column size in bytes
DECIMAL_DIGITS	SMALLINT	decimal digits of the procedure column
NUM_PREC_RADIX	SMALLINT	radix for numeric data types (2, 10, or NULL if not applicable)
NULLABLE	SMALLINT	whether the procedure column accepts a NULL value
REMARKS	WVARCHAR	a description of the procedure column
COLUMN_DEF	WVARCHAR	column's default value. Always NULL, that is, no default value is specified.
SQL_DATA_TYPE	SMALLINT	SQL data type
SQL_DATETIME_SUB	SMALLINT	subtype code for datetime. Always NULL.
CHAR_OCTET_LENGTH	INTEGER	maximum length in bytes of a character or binary data type column.
ORDINAL_POSITION	INTEGER	ordinal position of the column
IS_NULLABLE	WVARCHAR	always "YES"

SYS_RELAUTH

Column name	Data type	Description
REL_ID	INTEGER	relation id
UR_ID	INTEGER	user or role id
PRIV	INTEGER	privilege info
GRANT_ID	INTEGER	grantor id
GRANT_TIM	TIMESTAMP	grant time
GRANT_OPT	CHAR	grant option info

SYS_SCHEMAS

The SYS_SCHEMAS lists available schemas.

Column name	Data type	Description
ID	INTEGER	schema id
NAME	WVARCHAR	schema name
OWNER	WVARCHAR	schema owner name
CREATIME	TIMESTAMP	create date and time
SCHEMA_CATALOG	WVARCHAR	schema catalog

SYS_SEQUENCES

Column name	Data type	Description
SEQUENCE_NAME	WVARCHAR	sequence name
ID	INTEGER	unique id
DENSE	CHAR	is the sequence dense or sparse
SEQUENCE_SCHEMA	WVARCHAR	the name of the schema containing SEQUENCE_NAME
SEQUENCE_CATALOG	WVARCHAR	the name of the catalog containing SEQUENCE_NAME
CREATIME	TIMESTAMP	creation time

SYS_SYNONYM

Column name	Data type	Description
TARGET_ID	INTEGER	reserved for future use
SYNON	INTEGER	reserved for future use

SYS_TABLEMODES

Column name	Data type	Description
ID	INTEGER	relation id
MODE	WVARCHAR	special mode info
MODIFY_TIME	TIMESTAMP	last modify time
MODIFY_USER	WVARCHAR	last user that modified

SYS_TABLES

Column name	Data type	Description
ID	INTEGER	unique table identifier
TABLE_NAME	WVARCHAR	the name of the table
TABLE_TYPE	WVARCHAR	the type of the table (BASE TABLE or VIEW)
TABLE_SCHEMA	WVARCHAR	the name of the catalog containing TABLE_NAME
TABLE_CATALOG	WVARCHAR	the name of the catalog containing TABLE_NAME
CREATIME	TIMESTAMP	the creation time of the table
CHECKSTRING	LONG WVARCHAR	possible check option defined for the table
REMARKS	LONG WVARCHAR	reserved for future use

SYS_TRIGGERS

Column name	Data type	Description
ID	INTEGER	unique table identifier
TRIGGER_NAME	WVARCHAR	trigger name
TRIGGER_TEXT	LONG WVARCHAR	trigger body
TRIGGER_BIN	LONG VARBINARY	compiled form of the trigger
TRIGGER_SCHEMA	WVARCHAR	the name of the schema containing TRIGGER_NAME
TRIGGER_CATALOG	WVARCHAR	the name of the catalog containing TRIGGER_NAME
TRIGGER_ENABLED	CHAR	if triggers are enabled "YES"; otherwise "NO."
CREATIME	TIMESTAMP	the creation time of the trigger
TYPE	INTEGER	reserved for future use
REL_ID	INTEGER	the relation id

SYS_TYPES

Column name	Data type	Description
TYPE_NAME	WVARCHAR	the name of the data type
DATA_TYPE	SMALLINT	ODBC, data type number
PRECISION	INTEGER	ODBC, the precision of the data type
LITERAL_PREFIX	WVARCHAR	ODBC, possible prefix for literal values
LITERAL_SUFFIX	WVARCHAR	ODBC, possible suffix for literal values
CREATE_PARAMS	WVARCHAR	ODBC, the parameters needed to create a column of the data type
NULLABLE	SMALLINT	ODBC, can the data type contain NULL values

CASE_SENSITIVE	SMALLINT	ODBC, is the data type case sensitive
SEARCHABLE	SMALLINT	ODBC, the supported search operations
UNSIGNED_ATTRIBUTE	SMALLINT	ODBC, is the data type unsigned
MONEY	SMALLINT	ODBC, whether the data is a money data type
AUTO_INCREMENT	SMALLINT	ODBC, whether the data type is autoincrementing
LOCAL_TYPE_NAME	WVARCHAR	ODBC, has the data type another implementation defined name
MINIMUM_SCALE	SMALLINT	ODBC, the minimum scale of the data type
MAXIMUM_SCALE	SMALLINT	ODBC, the maximum scale of the data type

SYS_UROLE

The SYS_UROLE contains mapping of users to roles.

Column name	Data type	Description
U_ID	INTEGER	User id
R_ID	INTEGER	Role id

SYS_USERS

The SYS_USERS list information about users and roles.

Column name	Data type	Description
ID	INTEGER	User or role id
NAME	WVARCHAR	User or role name
TYPE	WVARCHAR	User type, either USER or ROLE
PRIV	INTEGER	Privilege information
PASSW	VARBINARY	Password inencrypted format
PRIORITY	INTEGER	Reserved for future use

PRIVATE	INTEGER	Is user private or public (used in SOLID *SynchroNet* only)
LOGIN_CATALOG	WVARCHAR	Reserved for future use

SYS_VIEWS

Column name	Data type	Description
V_ID	INTEGER	unique identifier for this view
TEXT	LONG WVARCHAR	view definition
CHECKSTRING	LONG WVARCHAR	possible CHECK OPTION defined for the view
REMARKS	LONG WVARCHAR	reserved for future use

F

Reserved Words

The following words are reserved in several SQL standards: ODBC 3.0, X/Open and SQL Access Group SQL CAE specification, Database Language - SQL: ANSI X3H2 (SQL-92). Some words are used by SOLID SQL. Applications should avoid using any of these keywords for other purposes. The following table contains also potential reserved words; these markings are enclosed in parenthesis.

Reserved word	ODBC	X/Open SQL	ANSI SQL2	SOLID SQL
ABSOLUTE	•		•	
ACTION	•		•	
ADA	•			
ADD	•	•	•	•
ADMIN				•
AFTER			(•)	•
ALIAS			(•)	
ALL	•	•	•	•
ALLOCATE	•	•	•	
ALTER	•	•	•	•
AND	•	•	•	•
ANY	•	•	•	•
APPEND				•
ARE	•		•	
AS	•	•	•	•

Reserved word	ODBC	X/Open SQL	ANSI SQL2	SOLID SQL
ASC	•	•	•	•
ASSERTION	•		•	
ASYNC			(•)	•
AT	•		•	
AUTHORIZATION	•		•	•
AVG	•	•	•	
BEFORE			(•)	•
BEGIN	•	•	•	•
BETWEEN	•	•	•	•
BINARY				•
BIT	•		•	
BIT_LENGTH	•		•	
BOOKMARK				•
BOOLEAN			(•)	
BOTH	•		•	
BREADTH			(•)	
BY	•	•	•	•
CALL			(•)	•
CASCADE	•	•	•	•
CASCADED	•		•	•
CASE	•		•	•
CAST	•		•	•
CATALOG	•		•	•
CHAR	•	•	•	•
CHAR_LENGTH	•		•	
CHARACTER	•	•	•	
CHARACTER_LENGTH	•		•	

Reserved word	ODBC	X/Open SQL	ANSI SQL2	SOLID SQL
CHECK	•	•	•	•
CLOSE	•	•	•	•
COALESCE	•		•	•
COLLATE	•		•	
COLLATION	•		•	
COLUMN	•		•	•
COMMIT	•	•	•	•
COMMITBLOCK				•
COMMITTED				•
COMPLETION			(•)	
CONNECT	•	•	•	
CONNECTION	•	•	•	
CONSTRAINT	•		•	•
CONSTRAINTS	•		•	
CONTINUE	•	•	•	
CONVERT	•		•	
CORRESPONDING	•		•	•
COUNT	•	•	•	
CREATE	•	•	•	•
CROSS	•		•	•
CURRENT	•	•		•
CURRENT_DATE	•		•	
CURRENT_TIME	•		•	
CURRENT_TIMESTAMP	•		•	
CURRENT_USER	•		•	
CURSOR	•	•	•	•
CYCLE			(•)	

Reserved word	ODBC	X/Open SQL	ANSI SQL2	SOLID SQL
DATA			(•)	•
DATE	•		•	•
DAY	•		•	
DEALLOCATE	•	•	•	
DEC	•	•	•	•
DECIMAL	•	•	•	•
DECLARE	•	•	•	•
DEFAULT	•	•	•	•
DEFERRABLE	•		•	
DEFERRED	•		•	
DELETE	•	•	•	•
DENSE				•
DEPTH			(•)	
DESC	•	•	•	•
DESCRIBE	•	•	•	
DESCRIPTOR	•	•	•	
DIAGNOSTICS	•	•	•	
DICTIONARY			(•)	
DISCONNECT	•	•	•	
DISTINCT	•	•	•	•
DOMAIN	•		•	•
DOUBLE	•	•	•	•
DROP	•	•	•	•
EACH			(•)	
ELSE	•		•	•
ELSEIF			(•)	•
ENABLE				•

Reserved word	ODBC	X/Open SQL	ANSI SQL2	SOLID SQL
END	•	•	•	•
END-EXEC	•		•	
EQUALS			(•)	
ESCAPE	•		•	•
EVENT				•
EXCEPT	•		•	•
EXCEPTION	•	•	•	
EXEC	•	•	•	•
EXECUTE	•	•	•	•
EXISTS	•	•	•	•
EXPLAIN				•
EXPORT				•
EXTERNAL	•		•	•
EXTRACT	•		•	•
FALSE	•		•	
FETCH	•	•	•	•
FIRST	•		•	
FIXED				•
FLOAT	•	•	•	•
FOR	•	•	•	•
FOREIGN	•	•	•	•
FOREVER				•
FORTRAN	•			
FORWARD				•
FOUND	•	•	•	
FROM	•	•	•	•
FROMFIXED				•
FULL	•		•	•

Reserved word	ODBC	X/Open SQL	ANSI SQL2	SOLID SQL
GENERAL			(•)	
GET	•	•	•	•
GLOBAL	•		•	
GO	•		•	
GOTO	•	•	•	
GRANT	•	•	•	•
GROUP	•	•	•	•
HAVING	•	•	•	•
HINT				•
HOUR	•		•	
IDENTIFIED				•
IDENTITY	•		•	
IF			(•)	•
IGNORE	•		(•)	
IMMEDIATE	•	•	•	
IMPORT				•
IN	•	•	•	•
INCLUDE	•	•		
INDEX	•	•		•
INDICATOR	•		•	
INITIALLY	•		•	
INNER	•		•	•
INPUT	•		•	
INSENSITIVE	•		•	
INSERT	•	•	•	•
INT	•	•	•	•
INTEGER	•	•	•	•
INTERNAL				•

Reserved word	ODBC	X/Open SQL	ANSI SQL2	SOLID SQL
INTERSECT	•		•	•
INTERVAL	•		•	
INTO	•	•	•	•
IS	•	•	•	•
ISOLATION	•		•	•
JAVA				•
JOIN	•		•	•
KEY	•	•	•	•
LANGUAGE	•		•	
LAST	•		•	
LEADING	•		•	
LEAVE			(•)	•
LEFT	•		•	•
LESS			(•)	
LEVEL	•		•	•
LIKE	•	•	•	•
LIMIT			(•)	
LOCAL	•		•	•
LOCK				•
LONG				•
LOOP			(•)	•
LOWER	•		•	
MAINMEMORY				•
MASTER				•
MATCH	•		•	
MAX	•	•	•	
MERGE				•
MESSAGE				•

Reserved word	ODBC	X/Open SQL	ANSI SQL2	SOLID SQL
MIN	•	•	•	
MINUTE	•		•	
MODIFY			(•)	•
MODULE	•		•	
MONTH	•		•	
NAMES	•		•	
NATIONAL	•		•	
NATURAL	•		•	•
NCHAR	•		•	
NEW			(•)	•
NEXT	•		•	•
NO	•		•	•
NONE	•		(•)	
NOT	•	•	•	•
NULL	•	•	•	•
NULLIF	•		•	•
NUMERIC	•	•	•	•
OBJECT			(•)	
OCTET_LENGTH	•		•	
OF	•	•	•	•
OFF	•		(•)	
OID			(•)	
OLD			(•)	•
ON	•	•	•	•
ONLY	•		•	•
OPEN	•	•	•	
OPERATION			(•)	
OPERATORS			(•)	

Reserved word	ODBC	X/Open SQL	ANSI SQL2	SOLID SQL
OPTIMISTIC				•
OPTION	•		•	•
OR	•	•	•	•
ORDER	•	•	•	•
OTHERS			(•)	
OUTER	•		•	•
OUTPUT	•		•	
OVERLAPS	•		•	
PARAMETERS			(•)	
PARTIAL	•		•	
PASCAL	•			
PENDANT			(•)	
PESSIMISTIC				•
PLAN				•
PLI	•			
POSITION	•		•	
POST				•
PRECISION	•	•	•	•
PREORDER			(•)	
PREPARE	•	•	•	•
PRESERVE	•		•	
PRIMARY	•	•	•	•
PRIOR	•		•	
PRIVATE			(•)	
PRIVILEGES	•		•	•
PROCEDURE	•		•	•
PROPAGATE				•

Reserved word	ODBC	X/Open SQL	ANSI SQL2	SOLID SQL
PROTECTED			(•)	
PUBLIC	•	•	•	•
PUBLICATION				•
READ			•	•
REAL		•	•	•
RECURSIVE			(•)	
REF			(•)	
REFERENCES	•	•	•	•
REFERENCING			(•)	•
REGISTER				•
RELATIVE	•		•	
RENAME				•
REPEATABLE				•
REPLACE			(•)	
REPLICA				•
REPLY				•
RESIGNAL			(•)	
RESTART				•
RESTRICT	•	•	•	•
RESULT				•
RETURN			(•)	•
RETURNS			(•)	•
REVERSE				•
REVOKE	•	•	•	•
RIGHT	•		•	•
ROLE			(•)	•
ROLLBACK	•	•	•	•
ROUTINE			(•)	

Reserved word	ODBC	X/Open SQL	ANSI SQL2	SOLID SQL
ROW			(•)	
ROWID				•
ROWNUM				•
ROWSPERMESSAGE				•
ROWVER				•
ROWS	•		•	
SAVEPOINT			(•)	•
SCAN				•
SCHEMA	•		•	•
SCROLL	•		•	
SEARCH			(•)	
SECOND	•		•	
SECTION	•	•	•	
SELECT	•	•	•	•
SENSITIVE			(•)	
SEQUENCE			(•)	•
SERIALIZABLE				•
SESSION	•		•	
SESSION_USER	•		•	
SET	•	•	•	•
SIGNAL			(•)	
SIMILAR			(•)	
SIZE	•		•	
SMALLINT	•	•	•	•
SOME	•		•	•
SORT				•
SPACE	•			
SQL	•	•	•	•

Reserved word	ODBC	X/Open SQL	ANSI SQL2	SOLID SQL
SQLCA	•	•		
SQLCODE	•		•	
SQLERROR	•	•	•	•
SQLEXCEPTION			(•)	
SQLSTATE	•		•	
SQLWARNING	•		(•)	
START				•
STRUCTURE			(•)	
SUBSCRIBE				•
SUBSCRIPTION				•
SUBSTRING	•		•	
SUM	•	•	•	
SYNC_CONFIG				•
SYSTEM	•			
SYSTEM_USER			•	
TABLE	•	•	•	•
TEMPORARY	•		•	
TEST			(•)	
THEN	•		•	•
THERE			(•)	
TIME	•		•	•
TIMEOUT				•
TIMESTAMP	•		•	•
TIMEZONE_HOUR	•		•	
TIMEZONE_MINUTE	•		•	
TINYINT				•
TO	•	•	•	•
TRAILING			•	

Reserved word	ODBC	X/Open SQL	ANSI SQL2	SOLID SQL
TRANSACTION	•		•	•
TRANSACTIONS				•
TRANSLATE	•		•	
TRANSLATION	•		•	
TRIGGER			(•)	•
TRIM	•		•	
TRUE	•		•	
TYPE			(•)	
UNDER			(•)	
UNION	•	•	•	•
UNIQUE	•	•	•	•
UNKNOWN	•		•	
UNREGISTER				•
UPDATE	•	•	•	•
UPPER	•		•	
USAGE	•		•	
USER	•	•	•	•
USING	•	•	•	•
VALUE	•	•	•	•
VALUES	•	•	•	•
VARBINARY				•
VARCHAR	•	•	•	•
VARIABLE			(•)	
VARWCHAR				•
VARYING	•	•	•	
VIEW	•	•	•	•
VIRTUAL			(•)	
VISIBLE			(•)	

Reserved word	ODBC	X/Open SQL	ANSI SQL2	SOLID SQL
WAIT			(•)	•
WCHAR				•
WHEN	•		•	•
WHENEVER	•	•	•	
WHERE	•	•	•	•
WHILE			(•)	•
WITH	•	•	•	•
WITHOUT			(•)	
WORK	•	•	•	•
WRITE			•	•
WCHAR				•
WVARCHAR				•
YEAR	•		•	
ZONE			•	

G

SOLID *Embedded Engine* Command Line Options

General Options

Option	Description	Examples
-c*dir*	Changes working directory.	
-f	Starts server in foreground.	
-h	Displays help.	
-m	Monitors users' messages and SQL statements.	
-*n**name*	Sets server name.	

-s{**start** | **install** | **remove**}, *name*, *fullexepath*, [**autostart**]	The Windows NT version of SOLID *Embedded Engine* is by default an icon exe version. Using the option **-sstart**, SOLID *Embedded Engine* can be started as a service executable and started and stopped from the service manager. If SOLID *Embedded Engine* is started without the **-sstart** option, it starts as an icon exe like the w95, w98, and w2000 versions. The service version of SOLID *Embedded Engine* cannot interact with the display and cannot create a new database. The service version writes warning and error messages also to the NT event log. SOLID *Embedded Engine* can also install and remove services using this command line option.	`SOLID.EXE -s"install,SOLID, D:\SOLID\SOLID.EXE -sstart -cd:\SOLID"`
		
		`SOLID.EXE -s"install,SOLID, D:\SOLID\SOLID.EXE -sstart -cd:\SOLID,autostart"`
		
		`SOLID.EXE -s"remove,SOLID"`

Option	Description	Examples
-U*username*	See option **-x** execute or **-x** exit. If used without the **-x** option, specifies the username for the database being created.	
-P*password*	See option **-x** execute or **-x** exit. If used without the **-x** option, specifies the given password for the database being created.	
-x autoconvert -C*catalogname*	Converts database format to version 3.5 and starts server process. -C*catalogname* is required to specify the default system catalog name for the database.	
-x convert -C*catalogname*	Converts database format to version 3.5 and exits. -Ccatalogname is required to specify the default system catalog name for the database.	
-x execute:i*nput file*	Prompts for the database administrator's user name and password, creates a new database, executes SQL statements from a file, and exits. The options -U and -P can be used to give the database the administrator's user name and password.	`solid.exe -x execute:init.sql` `solid.exe -x execute:init.sql -Udba -Pdba`
-x exit	Prompts for the database administrator's user name and password, creates a new database, and exits. Options -U and -P can be used to give the database administrator's user name and password.	`solid.exe -x exit` `solid.exe -x exit -Udba -Pdba`
-x forcerecovery	Does a forced roll-forward recovery.	
-x hide	Hides server icon.	
-x ignoreerrors	Ignores index errors.	
-x testblocks	Tests database blocks.	
-x testindex	Tests database index.	
-?	Help = Usage.	

Glossary

This glossary gives you a description of the terminology used in SOLID documentation.

Binary Large Object (BLOb)

A BLOb is a large block of binary information such as a picture, video clip, sound excerpt, or a formatted text document. BLObs can be saved to and retrieved from SOLID *Embedded Engine*.

Catalog

A catalog logically partitions a database so that data is organized in ways that meet business or application requirements. A catalog can qualify one or more schemas. A schema is a persistent database object that provides a definition for the entire database; it represents a collection of database objects associated with a specific schema name. The catalog name is used to qualify a database object name, such as tables, views, indexes, stored procedures, triggers, and sequences. They are qualified as: *catalog_name.schema_name.database_object* or *catalog_name.user_id.database_object* in DML statements.

Checkpoint

Checkpoints are used to store a consistent state of the database quickly onto the disk. After a system crash, the database will start recovering transactions from the latest checkpoint. The more frequently checkpoints are made, the fewer transactions need to be recovered from the log file.

Client/server computing

Client/server computing divides a large piece of software into modules that need not all be executed within the same memory space nor on the same processor. The calling module becomes the ‘client’ that requests services, and the called module becomes the ‘server’ that provides services. Client and server processes exchange information by sending messages through a computer network. They may run on different hardware and software platforms as appropriate for their special functions.

Two basic client/server architecture types are called two-tier and three-tier application architectures.

Communication protocol

A communication protocol is a set of rules and conventions used in the communication between servers and clients. The server and client have to use the same communication protocol in order to establish a connection.

Database administrator

The database administrator is a person responsible for tasks such as:

- managing users, tables, and indices
- backing up data
- allocating disk space for the database files

Database management system (DBMS)

A DBMS is a system that stores information in and retrieves information from a database. A DBMS typically consists of a database server, administration utilities, an application interface, and development tools.

Database procedures (Stored procedures)

Database procedures allow programmers to split the application logic between the client and the server. These procedures are stored in the database, and they accept parameters in the activation call from the client application. This arrangement is beneficial especially in the case of heavy updates that first require extensive queries and that can be initiated with a small amount of parameter information. In these cases, the network traffic is significantly reduced, and much better performance can be achieved.

Event Alerts

Events are objects with a name and parameters. Event alerts are used to signal an event in the database. The signal is sent from an application using the `POST EVENT` command. The signal is received by one or more client applications waiting for the event. The use of event alerts removes resource consuming database polling from applications.

Log file (Transaction log)

This file holds a log of all committed operations executed by the database server. If a system crash occurs, the database server uses this log to recover all data inserted or modified after the latest checkpoint.

Network name

The network name of a server consists of a communication protocol and a server name. This combination identifies the server in the network.

SOLID Clients support Logical Data Source Names. These names can be used to give a database a descriptive name. This name is mapped to a network name using either parameter settings in the clients `solid.ini` file or in Windows operating systems' registry settings.

Open Database Connectivity (ODBC)

ODBC is a programming interface standard for SQL database programs. SOLID *Embedded Engine* offers a native ODBC programming interface.

Optimizer Hints

Optimizer hints (which is an extension of SQL) are directives specified through embedded pseudo comments within query statements. The Optimizer detects these directives or *hints* and bases its query execution plan accordingly. Optimizer hints allow applications to be optimized under various conditions to the data, query type, and the database. They not only provide solutions to performance problems occasionally encountered with queries, but shift control of response times from the system to the user.

Relational database management system (RDBMS)

SOLID *Embedded Engine* is an RDBMS, which stores and retrieves information that is organized into two-dimensional tables. This name derives from the relational theory that formalizes the data manipulation requests as set operations and allows mathematical analysis of these sets. RDBMSs typically support the SQL language for data manipulation requests.

Sequence objects

Sequence objects generate number sequences for objects stored in databases. Sequences have an advantage over separate tables. They are specifically fine-tuned for fast execution and result in less overhead than normal update statements.

SQL Access Group's Call Level Interface (SAG CLI)

SAG CLI is a programming interface standard that defines the functions that are used to submit dynamic SQL clauses to a database server for execution. The ODBC interface is also based on SAG CLI. The SOLID *SQL API* conforms to the SAG CLI standard.

Schema

All tables are contained in a higher level construct called schema. It is a place where tables and related objects are gathered together under one qualifying name. For each schema there

are zero or more tables, and for each table, there is exactly one schema to which it belongs. The relationship between a schema and its tables is similar to that of an operating system directory and the files contained within that directory.

SOLID directory

The default directory for storing SOLID DBMS database files. This is the server program's working directory.

Structured Query Language (SQL)

SQL is a standardized query language designed for handling database requests and administration. The SQL syntax used in SOLID *Embedded Engine* is based on the ANSI X3H2-1989 Level 2 standard including important ANSI X3H2-1992 (SQL2) extensions. For a more formal definition of the syntax, refer to *Appendix D SOLID SQL Syntax* of **SOLID Administrator Guide.**

Three-tier client/server architecture model

Compared to the two-tier architecture the three-tier architecture has an additional layer or layers of application servers. This allows splitting the application logic between client processes to a specialized application server process handling the resources management, other I/O, or calculation intensive tasks.

Instead of sending small SQL statements the client application sends whole procedures for the application server to be processed. This reduces the number of messages thus minimizing the network load. The application logic is often more easily managed because several applications use centrally maintained procedures.

Triggers

Triggers are pieces of logic, which a Solid server automatically executes when a user attempts to change the data in a table. When a user modifies data within the table, the trigger that corresponds to the command (such as insert, delete, or update) is activated.

Two-tier client/server architecture model

Generally, the two-tier architecture refers to a client/server system, where a client application containing all the business logic is running on a workstation and a database server is taking care of data management.

Index

D

E

F

G

H

I

L

M

S

T

U

V

W