www.ibm.com/software/data/**soliddb**

IBM solidDB **Connector Guide**

Version 6.1 | June 2008

solidDB™

IBM solidDB Connector Guide

Copyright © Solid Information Technology Ltd. 1993, 2008
Document number: CON-6.1
Product version: 06.10.0014
Date: 2008-06-13

All rights reserved. No portion of this product may be used in any way except as expressly authorized in writing by Solid Information Technology Ltd. or International Business Machines Corporation.

"IBM", the IBM logo, "DB2", "Informix", "Solid" and "solidDB" are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both.

All other products, services, companies and publications are trademarks or registered trademarks of their respective owners.

This product is protected by U.S. patents 6144941, 7136912, 6970876, 7139775, 6978396, and 7266702.

This product contains lexical analyzer Flex. Copyright (c) 1990 The Regents of the University of California. All rights reserved.

> This code is derived from software contributed to Berkeley by Vern Paxson. The United States Government has rights in this work pursuant to contract no. DE-AC03-76SF00098 between the United States Department of Energy and the University of California. Redistribution and use in source and binary forms are permitted provided that: (1) source distributions retain this entire copyright notice and comment, and (2) distributions including binaries display the following acknowledgement: "This product includes software developed by the University of California, Berkeley and its contributors'" in the documentation or other materials provided with the distribution and in all advertising materials mentioning features or use of this software. Neither the name of the University nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission. THIS SOFTWARE IS PROVIDED "AS IS" AND WITHOUT ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANT-ABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

This product contains zlib general purpose compression library version 1.1.4, March 11th, 2002. Copyright (C) 1995-2002 Jean-loup Gailly and Mark Adler.

> This software is provided "as-is", without any express or implied warranty. In no event will the authors be held liable for any damages arising from the use of this software. Permission is granted to anyone to use this software for any purpose, including commercial applications, and to alter it and redistribute it freely, subject to the following restrictions: 1. The origin of this software must not be misrepresented; you must not claim that you wrote the original software. If you use this software in a product, an acknowledgment in the product documentation would be appreciated but is not required. 2. Altered source versions must be plainly marked as such, and must not be misrepresented as being the original software. 3. This notice may not be removed or altered from any source distribution.

This product contains the Qsort routine in the external sorter, Copyright (c) 1980, 1983, 1990 The Regents of the University of California. All rights reserved.

> Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:
>
> 1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
>
> 2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

3. All advertising materials mentioning features or use of this software must display the following acknowledgement: This product includes software developed by the University of California, Berkeley and its contributors.

4. Neither the name of the University nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE REGENTS AND CONTRIBUTORS ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

This product contains the DES cipher (in ECB mode), parts of this code are Copyright (C) 1996 Geoffrey Keating. All rights reserved.

Its use is FREE FOR COMMERCIAL AND NON-COMMERCIAL USE as long as the following conditions are adhered to.

Copyright remains Geoffrey Keating's, and as such any Copyright notices in the code are not to be removed. If this code is used in a product, Geoffrey Keating should be given attribution as the author of the parts used. This can be in the form of a textual message at program startup or in documentation (online or textual) provided with the package.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the copyright notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

3. All advertising materials mentioning features or use of this software must display the following acknowledgement: This product includes software developed by Eric Young (eay@mincom.oz.au)

THIS SOFTWARE IS PROVIDED BY GEOFFREY KEATING ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Parts of this code (in particular, the string representing SPtrans below) are Copyright (C) 1995 Eric Young (eay@mincom.oz.au). All rights reserved.

Its use is FREE FOR COMMERCIAL AND NON-COMMERCIAL USE as long as the following conditions are adhered to.

Copyright remains Eric Young's, and as such any Copyright notices in the code are not to be removed. If this code is used in a product, Eric Young should be given attribution as the author of the parts used. This can be in the form of a textual message at program startup or in documentation (online or textual) provided with the package.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the copyright notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

3. All advertising materials mentioning features or use of this software must display the following acknowledgement: This product includes software developed by Eric Young (eay@mincom.oz.au)

THIS SOFTWARE IS PROVIDED BY ERIC YOUNG ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

This product is assigned the U.S. Export Control Classification Number ECCN=5D992b.

Table of Contents

List of Figures

List of Tables

Chapter 1. Welcome

IBM solidDB Cache 6.1 provides a high-performance, low-latency database front-end solution for IBM Data Servers. IBM solidDB Cache solution uses a number of in-memory front-end databases to cache traffic from the applications. Typically, each front-end contains writable partition replica(s) of the back-end database and read-only partition replica(s) of the back-end database. The replication takes place asynchronously. In this way, the read and write load can be distributed in the system extremely efficiently to achieve extreme speed.

IBM solidDB Cache uses connector instances to handle traffic between the front-ends and the back-end.

1.1 About This Guide

This guide is intended for Database Administrators (DBA) who administer the connector solution.

For general administration and maintenance information on IBM solidDB databases, see *IBM solidDB Administration Guide*.

1.1.1 Organization

This guide includes the following information:

- Chapter 2, *Introduction to IBM solidDB Connector* describes the concepts, components, and physical configuration options of connector.
- Chapter 3, *Using IBM solidDB Connector* describes the installation and configuration of connector.
- Chapter 4, *Failure Scenarios* describes the different connector failure scenarios and the required recovery procedures, if any.
- Appendix A, *Replication Model Schema* describes the replication model schema and the replication model tables.

1.1.2 Audience

This guide assumes the reader has general DBMS knowledge, and familiarity with SQL and IBM solidDB.

1.2 Conventions

1.2.1 Typographic Conventions

This manual uses the following typographic conventions:

Table 1.1. Typographic Conventions

Format	Used for
Database table	This font is used for all ordinary text.
NOT NULL	Uppercase letters on this font indicate SQL keywords and macro names.
`solid.ini`	These fonts indicate file names and path expressions.
`SET SYNC MASTER YES;` `COMMIT WORK;`	This font is used for program code and program output. Example SQL statements also use this font.
run.sh	This font is used for sample command lines.
`TRIG_COUNT()`	This font is used for function names.
`java.sql.Connection`	This font is used for interface names.
`LockHashSize`	This font is used for parameter names, function arguments, and Windows registry entries.
`argument`	Words emphasised like this indicate information that the user or the application must provide.
IBM solidDB Administration Guide	This style is used for references to other documents, or chapters in the same document. New terms and emphasised issues are also written like this.
File path presentation	File paths are presented in the Unix format. The slash (/) character represents the installation root directory.
Operating systems	If documentation contains differences between operating systems, the Unix format is mentioned first. The Microsoft Windows format is mentioned in parentheses after the Unix format. Other operating systems are separately mentioned.

1.2.2 Syntax Notation

This manual uses the following syntax notation conventions:

Table 1.2. Syntax Notation Conventions

Format	Used for
`INSERT INTO` *`table_name`*	Syntax descriptions are on this font. Replaceable sections are on *`this`* font.
`solid.ini`	This font indicates file names and path expressions.
[ ]	Square 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.
{ }	Curly brackets 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.

1.3 IBM solidDB Documentation

Below is a complete list of documents available for IBM solidDB. IBM solidDB documentation is distributed in an electronic format, usually PDF files and web pages.

- *Release Notes*. This file contains installation instructions and the most up-to-date information about the specific product version. This file (`releasenotes.txt`) is copied onto your system when you install the software.
- *IBM solidDB Getting Started Guide*. This manual gives you an introduction to IBM solidDB.
- *IBM solidDB SQL Guide*. This manual describes the SQL commands that IBM solidDB supports. This manual also describes some of the system tables, system views, system stored procedures, etc. that the engine makes available to you. This manual contains some basic tutorial material on SQL for those readers who are not already familiar with SQL. Note that some specialized material is covered in other manuals. For example, the IBM solidDB "administrative commands" related to the High Availability (HotStandby) component are described in the *IBM solidDB High Availability User Guide*, not the *IBM solidDB SQL Guide*.

- *IBM solidDB Administration Guide*. This guide describes administrative procedures for IBM solidDB servers. This manual includes configuration information. Note that some administrative commands use an SQL-like syntax and are documented in the *IBM solidDB SQL Guide*.

- *IBM solidDB Programmer Guide*. This guide explains in detail how to use features such as IBM solidDB Stored Procedure Language, triggers, events, and sequences. It also describes the interfaces (APIs and drivers) available for accessing IBM solidDB and how to use them with a IBM solidDB database.

- *IBM solidDB In-Memory Database User Guide*. This manual describes how to use the IBM solidDB in-memory database and main memory engine (MME).

- *IBM solidDB Advanced Replication Guide*. This guide describes how to use the IBM solidDB advanced replication technology to synchronize data across multiple database servers.

- *IBM solidDB Linked Library Access User Guide*. Linking the client application directly to the server improves performance by eliminating network communication overhead. This guide describes how to use the linked library access, a database engine library that can be linked directly to the client application.

  This manual also explains how to use two proprietary Application Programming Interfaces (APIs). The first API is the IBM solidDB SA interface, a low-level C-language interface that allows you to perform simple single-table operations (such as inserting a row in a table) quickly. The second API is SSC API, which allows your C-language program can control the behavior of the embedded (linked) database server

  This manual also explains how to set up a IBM solidDB to run without a disk drive.

- *IBM solidDB High Availability User Guide*. IBM solidDB HotStandby allows your system to maintain an identical copy of the database in a backup server or "secondary server". This secondary database server can continue working if the primary database server fails.

- *IBM solidDB Connector Guide*. This guide explains in detail how to use the IBM solidDB Cache solution. IBM solidDB Cache provides a high-performance, low-latency database front-end solution for IBM Data Servers, namely DB2™ and Informix™. IBM solidDB Cache solution uses a number of in-memory front-end databases to handle high-volume traffic from the applications. The connectors are applications that manage data between the back-end and the front-ends in IBM solidDB Cache.

Chapter 2. Introduction to IBM solidDB Connector

IBM solidDB Cache is a front-end database solution for IBM Data Servers. With IBM solidDB Cache, a number of in-memory front-end databases is used to cache traffic from the applications.

A typical IBM solidDB Cache deployment comprises of a single back-end database (DB2 or IDS) and a number of IBM solidDB front-end databases as shown in Figure 2.1, “Example Replication Model”. Typically, the front-end databases run on dedicated computer nodes. The front-ends can be standalone or HotStandby IBM solidDB configurations.

Because the data between the front-ends and the back-end is passed asynchronously with respect to database transactions, high performance and low latency are attainable while operating in front-end nodes. If required, IBM solidDB Cache system maintains full persistence (durability) of data and high availability of database service. The data transmission is based on a replication model whereby logical cache partitions of the back-end database are replicated to front-ends as both writable and read-only partition replicas.

IBM solidDB connector is a component that handles traffic between the front-ends and the back-end. In the normal cache operation, the connector receives log records from front-ends and generates SQL statements (such as `INSERT`, `DELETE` or `UPDATE`) from them. These SQL statements are written to the back-end database.

There is a single instance of IBM solidDB connector for each partition replica

The connector can automatically recover from the most common failures. For more information, refer to Chapter 4, *Failure Scenarios*.

Note

IBM solidDB Cache should not be confused with the IBM solidDB's internal cache (buffer pool). The internal cache is used to facilitate efficient I/O operations in the IBM solidDB database server.

2.1 IBM solidDB Cache Components

The main concepts of a IBM solidDB Cache system are introduced below:

- *Applications*. The applications run on front-ends or they are connected to front-ends. IBM solidDB linked library access library can also be used. Applications operate on replicated partition(s) of the back-end data.

- *Cache partitions*. A partition is a logical part of a database.
- *Partition table*. Partition tables are tables that are part of the partition.
- A *replica* is either a writable replica or a read-only replica. Replicas are located in the front-ends.
  - Writable replicas. The applications can write to these replicas. There can only be one writable replica, of a given partition, in the system.
  - Read-only replicas. The applications can only read from these replicas. There can be several read-only replicas, per partition, in the system.
- *Front-ends*. The in-memory databases run on the front-ends. These databases contain the partition replicas that the applications use.

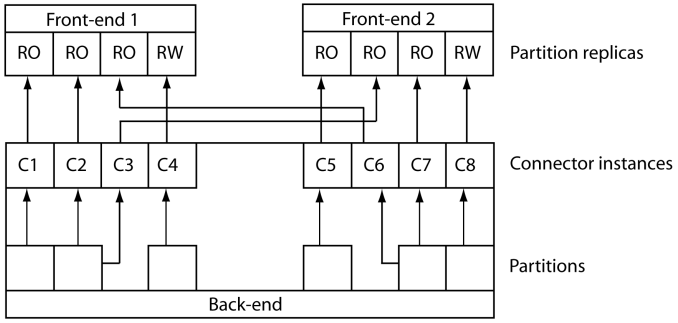
  Front-ends themselves can also contain applications.
- *Back-end*. The back-end contains the actual data server, a DB2 or IDS database. When IBM solidDB Cache system is installed, the back-end manages the initial data population and it also owns the replication model. For more information on the replication model, see Section 2.4, “Replication Model Restrictions”.
- The *connector*. The connector itself is a Java program that loads data from the back-end to the front-ends (the loader role) or propagates data from the front-ends to the back-end (the propagator role). There is a specific command line option that defines whether the connector is started in the loader or propagator role.

  When the connector system is initiated and the initial data must be loaded from the back-end to the front-ends, the connector must be started in the loader role. On the other hand, when the connector system is operating normally, the connector is in the propagator role. Propagator role is the default role and the connector is started in it unless the loader role is specified. The connector in the propagator role is often called "the propagator" for brevity.

  There is a dedicated connector instance for each partition replica. For example in the figure below, connector instances C2 and C3 manage the same back-end partition for different front-ends.

See below for an example replication model. In the figure, *RO* stands for "read-only" and *WR* for "writable".

Figure 2.1. Example Replication Model



2.2 Principle of Operation

The sections below explain the operation principles of IBM solidDB Cache system.

2.2.1 Connector Run Life Cycle

When IBM solidDB Cache system is initiated, the connector is started in the loader role. The loader reads the replication schema and loads the relevant front-end tables with the back-end contents.

The loader loads a single partition replica. If there are several partition replicas to be loaded on one or more front-ends, several connector instances can be initiated in parallel to perform the loads.

Once the loader has finished loading data to the front-ends, the connector is to be started in the propagator role. The propagator reads input from the front-ends and converts it into SQL transactions and feeds them to the back-end.

Note

The connector cannot be run simultaneously in the loader and propagator role for the same partition replica. If the connector is started in the propagator role while the loader is active, the propagator is automatically shut down.

2.2.2 Loader Internals

When the connector is started as a loader, it reads the contents of the replication model and replaces the contents of the relevant front-end tables with the back-end contents. During the load phase, the use of the partition replica tables by the applications is disabled (the tables are locked with an exclusive lock). Any effort to access those pages will results blocking the request until the load is fully performed.

The loader can be also used to empty the partition replica tables, or to unlock the partition replicas if they are erroneously locked. For all connector options, see Section 3.2.1, “IBM solidDB Connector Startup Options”.

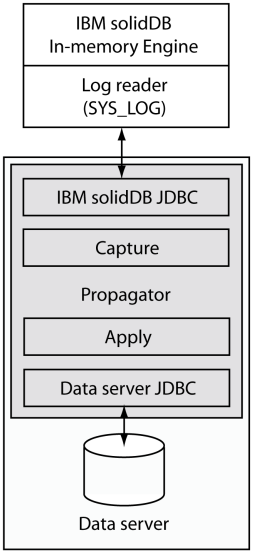
The loader loads a single partition replica. If there are several partition replicas to be loaded (on one or many front-ends), several connector instances can be initiated in parallel to perform the load.

2.2.3 Propagator Internals

Once the loader has finished loading data, the connector is started as a propagator. The propagator reads the log stream from the log reader and converts it into SQL transactions fed to the back-end.

The propagator role is depicted in the figure below:

Figure 2.2. Propagator



The capture part of the propagator uses the log reader implemented in the front-end server. The log reader is a virtual table (`SYS_LOG`) that is accessed by using an "active query" over standard ODBC/JDBC drivers. Active query means that consecutive fetches will bring new log records as they are created in the front-end. Only log records of committed transactions are passed by the log reader. The `SYS_LOG` log position is maintained durably in the back-end database to facilitate recovery of the propagation state and catchup.

Occasionally, the log reader log position may be a certain distance away from from the actual log head. This may happen when the propagator cannot keep up with the pace of incoming transactions, or when the

propagator is temporarily stopped. There is a maximum distance between the the log reader log position and the log head, that is maintained by the system. This distance is called the log reader max log size, and there is a corresponding log reader configuration parameter for it, `LogReader.MaxLogSize`. If the propagator is not running, and the log position reaches the distance of max log size from the log head, the replica tables become read-only, and no log-generation operations are allowed for those tables. The server also protects the log files against removing so that they are retained to maintain at least the space specified with the max log size parameter.

However, the log reader max log size setting does not limit the total size of the log stored in the node. In a normal operation, the log files are accumulated without any limit, until a backup is performed. The backup removes the log files that would not be needed for checkpoint-based recovery. If, for some reason, backups are not performed, a setting is available (`General.CheckPointDeleteLog=Yes`) whereby log files are automatically removed at every checkpoint. In either case, the latest portion of the log, up to the log reader's current log position, is protected against removal.

The protection of the log position up to the log reader max log size is enacted only if the propagator is not running, for example, because of a failure or a maintenance break. During the normal propagator operation, another smaller, in-memory log reader buffer is maintained to even out load peaks. Its size is determined by a parameter called log reader max space (`LogReader.MaxSpace`). When this buffer is filled up, the log reader starts to throttle incoming transactions, i.e. to slow them down by blocking, to maintain the log position within the in-memory log reader buffer.

In the apply part, the committed transactions are fed into the back-end database. To increase throughput of update propagation, the propagator uses two optimization techniques

- First, the propagator batches transactions. The batches are committed to the backend as single transactions. The copy consistency in the backend is preserved by way of committing batches in the same order as they were read from the log reader.

- Second, the propagator uses parallel threads and connections to the back-end. The maximum number of threads is configurable and by default is 64. The actual number of threads used depends on the number of concurrently processed transaction batches.

The operational model of the propagator maintains the following consistency criteria:

- *Eventual consistency*: In a quiesced system state, the backend partitions represent a consistent snapshot of the front-end replicas.

- *Committed transactions*: At any time, only the effects committed front-end transactions are visible in the backend.

- *Strong snapshot consistency*: Under continuous load, the transactions are committed in the backed in the same order as in the front-end.

2.3 On Data Ownership

Data ownership refers to a principle that a component owns a data item if it has rights to modify it. There are different data items in a connector system, for a replication model schema and user data. All data items are owned by either the back-end on the front-end.

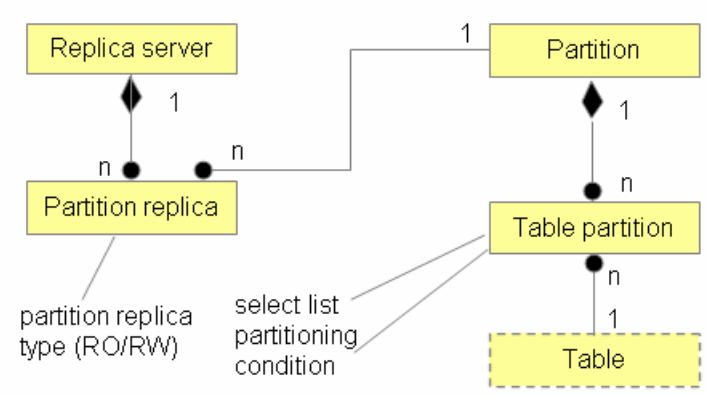
In the connector, data is owned as follows:

- The replication model schema is created at the backend. The back-end owns the replication model schema at all times.
- The replication model is populated at the back-end and it can only be changed at the back-end. The back-end owns the replication model.
- The contents of all the partitions on front-ends are initially populated at the back-end. The back-end owns the initial data population.
- When the partition data has been loaded to the front-ends, the data ownership is as follows:
    - The back-end owns the data in the read-only partitions
    - The front-end owns the data in the writable partitions

2.4 Replication Model Restrictions

The replication model specifies the division of data between the front-ends and the back-end. The replication model is implemented in the replication model tables on the back-end. It defines partitions, replicated tables for the partitions, and replica servers where the partitions are replicated to. The user must populate the model. See below for a conceptual presentation of the replication model:

Figure 2.3. Conceptual Replication Model



When you populate the model, follow the rules given in Appendix A, *Replication Model Schema*. Except for the normal referential integrity check, no consistency checks are done on the populated data.

Limitations of the replication model are explained below. If, in the following limitations, a rule or a limitation is unenforced, the system does not prevent the violation from happening. On the other hand, if a limitation is enforced, an effort to violate the rule will result in an error. The same applies for limitations explained in other sections of this manual.

- If a partition has multiple replicas, at most one of them can be a writable replica. Other replicas are read-only replicas. The front-ends will enforce the read-only replicas.
- Partition replicas cannot overlap (at the table level) in a front-end. This restriction is enforced by the loader. The loader will fail if there are intersecting replicas.
- For any two partitions having writable replicas, no column value may participate in both partitions at the same time. In other words, partitions that have writable replicas cannot intersect at the level of data items. This restriction is unenforced.

For more information on the replication model schema, see Appendix A, *Replication Model Schema*.

Note

This note concerns case sensitivity of SQL names. In the replication model, SQL names are used for tables, columns, schemas, user names, and so on.

As concerns names applying to the front-end databases, the names are, by default, case-insensitive. They are converted to upper case before transferred to the front-end. This happens even if the names are specified with the "quoted notation". However, if needed, the front-end database may be configured to accept mixed-case names in the quoted notation.

As concerns the back-end names (if they are specified), they are treated according to the settings of the back-end database.

2.5 Database Restrictions

These restrictions apply to user data tables that are part of the partition. Most of the data manipulation restrictions apply to replica tables residing at front-ends.

- Data types
  - The commonly supported data types of IBM solidDB, DB2, and IDS are supported.
  - Additionally, there is a restriction on BLOB types (BLOB, CLOB, VARCHAR, and so on). If the BLOB size exceeds a certain limit, an error is returned in the front-end. If the connector in the loader role encounters a size limit while loading the data, the connector fails. Technically, the restriction applies to the row size. It cannot exceed the currently used database block size. The factory block size is 8 Kb and the maximum size is 64 Kb.
- Only persistent M (in-memory) and D (disk) tables can be used in partition replicas. In other words, Transient and Temporary tables cannot be used. However, this restriction is unenforced.
- Primary key constraints
  - Primary keys are mandatory, both in the front-end and back-end partition tables. This restriction is enforced. Primary keys must be specified by using the `PRIMARY KEY` SQL syntax. If a primary key value is updated, this is to be done with a singleton statement (single row per statement). This restriction is unenforced.
- Referential integrity constraints
  - With the normal operational mode of multithreaded apply, referential integrity constraints in the back-end database are not allowed. This restriction is enforced (the propagator will terminate, with an error, if foreign keys are detected in the back-end database, and the size of the thread pool is greater than 1).

Note

Because it is the front-end that owns and modifies the data of writable partition replicas, foreign keys are not needed in the back-end database. As the partition schemas are to be defined separately for the back-end and front-end, there is a possibility to exclude the foreign key definitions from the back-end schema.

If there is a definitive need to maintain referential integrity constraints in the back-end database, the propagator should be run in a single-threaded apply mode (the connector parameter BackendDB.ThreadPoolSize is to be set to 1)

- Whenever used, all the foreign key constraints in a partition must be confined to that partition. This restriction is unenforced.

  A partial referential integrity tree may be allowed in a partition, if the front-end application semantics allow for it. For example, some referencing tables may be excluded from a partition if the corresponding primary keys in the referenced tables are never removed, or changed, by the front-end application.

- No cascading referential actions are allowed within partitions. This restriction is unenforced.

- No cyclic referential integrity relationships are allowed among the tables. In other words, the inter-referenced tables have to form a tree. This restriction is enforced by the loader. The loader will fail when the restriction is violated.

- `UNIQUE` constraints. Unique constraints are supported in the front-end replica tables. In the back-end, if the multithreaded apply mode is used, no `UNIQUE` constraints are allowed, other than in primary keys.

- As far as replica tables are concerned, multiple NULLs are not allowed in columns defined as UNIQUE. This restriction is enforced on the front-ends.

- Triggers are allowed in the front-end. Triggers in the back-end are not allowed if they inflict changes to the contents of partitions. This restriction is enforced.

- There is no automatic migration of the data table schemas to the front-ends. The front-end partition replica table schemas (table definitions) must be created manually before the replica data is loaded.

- SQL behavior. The SQL Data Manipulation Language of IBM solidDB is supported, with the following exceptions:

  - The `TRUNCATE` statement is illegal. This restriction is enforced (an SQL error is returned).

  - Functions can be freely used, both at the front-ends and the back-end.

    - You can only update one primary key value or one UNIQUE column value in a single SQL statement. This restriction is enforced (and SQL error is returned).

- The front-ends support IBM solidDB procedures.

- Transactions. As a rule, all transactional characteristics are maintained in the presence of propagation. However, the assumptions below apply:

    - The commit mechanism in IBM solidDB supports atomicity. Only committed transactions are propagated to the back-end, and the connector in the propagator role guarantees that they are transferred in their totality. Atomicity is also preserved in most failures cases.

    - Isolation levels can be used in front-ends. The read committed isolation level (known as Cursor Stability in DB2) should be used for read-only transactions in the back-end to avoid blocking propagation. This restriction is unenforced. To preserve global isolation, no back-end update activity is allowed on the data being part of a writable partition.

    - The durability of transactions is maintained mainly at the front-tier, because propagation is asynchronous. There are two ways of maintaining durability:

        1. by using synchronous (write-thru) logging to disk (disk-based durability), either in the Primary or Secondary, or both, or

        2. by using a 2-Safe HSB replication protocol (network-based durability).

    - Lock granularity. Because the propagator uses parallel threads to propagate write transactions, the row-level lock granularity is required in the backend. If it is not set by default, it should be set (for example, in IDS). However, take special care to avoid lock escalation to the table level. Should the lock escalation happen, a deadlock may occur in the back-end.

Chapter 3. Using IBM solidDB Connector

This chapter provides step-by-step instructions for setting up a connector system.

The connector is a network-enabled component and can be run on any node in the system. For performance reasons, it is assumed that the connector instances are run in the back-end computer.

This chapter assumes you have already installed and started the back-end and the front-ends. Be sure to follow the installation instructions that came with the product.

Note

Your software package also includes a connector sample called *solconnector*.

3.1 IBM solidDB Connector Deployment and Preparation

To deploy a connector system, follow the instructions below. For startup instructions, see Section 3.2.1, “IBM solidDB Connector Startup Options”.

1. On the back-end:

   Load the replication model schema (that is, the table definitions).

2. On the front-end:

   Define the tables of the partition replicas. No applications using those tables can run at the table definition time.

3. On the back-end:

   Populate the replication model tables with the relevant data representing the total replication setup.

While feeding the front-end table names into the replication model, the parts of a fully qualified IBM solidDB table name have to be given. A IBM solidDB qualified table name is in the form:

<catalog-name>.<schema-name>.<table-name>.

You must specify a catalog name when you create a database (there is no factory value). You can also create a schema. If you do not create one, the default schema name is the user ID (user name).

3.2 IBM solidDB Connector Operation

The connector operation is explained in the sections below.

3.2.1 IBM solidDB Connector Startup Options

The connector startup command syntax is presented below:

```
java solconnector [options] <replica-server-id> <partition-id>
```

The parameters are:

- *replica-server-id*

  This parameter refers to the descriptive name of the replica server specified in the replication model.

- *partition-id*

  This parameter refers to the descriptive partition name specified in the replication model.

The options are:

- `-i`

  The option enables to specify the name and path of the connector configuration file (by default it is `solconnector.ini`).

- `-load`

  If this option is specified, the connector is started in the loader role. Once the load has been performed, the program terminates.

- `-clean`

This option empties the partition model tables of a given partition in the front-end.

- `-release`

  This option releases partition replica locks and terminates the program. This option can be used, for example, after a failed load if there is a need to access the partition replica tables.

- `-verbose`

  This option enables verbose output mode.

- `-debug`

  This option enables the most verbose output mode for debugging purposes.

- `-help`

  This option provides help for using the startup command.

If none of the options `load`, `clean`, or `release` is used, the connector is started in the propagator role. Other options can be used at any time. The `clean` and `load` options cannot be used at the same time.

Important

All required JDBC drivers (and licenses) as well as the `solconnector.jar` must be specified in the `CLASSPATH` environment variable.

Tip

You can also use an alternative syntax, where the `CLASSPATH` settings do not apply. In this case, the JDBC drivers must be located in the `lib` directory, in the connector's working directory. The predefined driver jar file names are:

- `./lib/SolidDriver2.0.jar`
- `./lib/db2jcc.jar`
- `./lib/ifxjdbc.jar`

The alternative syntax is presented below:

```
java -jar solconnector.jar [options] <replica-server-id> <partition-id>
```

The parameters and options are the same for both syntax formats.

3.2.2 Loading Data

Loading data is necessary if there is no data on the front-ends or a reload is necessary.

Loading data updates the front-end schema and tables in line with the back-end schema and tables. The loader uses the `LOADORDER` column, in the replication model, to infer the table loading order in such a way that no referential integrity constraints are violated during the load. The tables are loaded in the ascending order of `LOADORDER` values.

When you want to load data from the back-end to the front-ends, you must start the connector in the loader role. In the loader role, the connector reads the replication schema and loads the relevant front-end tables with the back-end contents.

Then, start the connector in the loader role for each partition replica separately.

Load may fail for a variety of reasons. There may be incorrect table definitions in the front-end, incorrect load order in the replication schema, and so on. If the load operation fails, the partition replica becomes locked. This is to protect possibly inconsistent user tables against inadvertent use.

If your load operation fails, you can use the `release` option when starting the connector. For more information on startup options, see Section 3.2.1, “IBM solidDB Connector Startup Options”.

A reload operation replaces replication model data in the front-end(s) with new replication model data from the back-end. If the new replication model data contains new tables, these tables must exist before the load is started. Tables belonging to the old replication model data but not to new replication model data are emptied (but not dropped).

3.2.3 Propagating Data

When the connector propagates data, it reads input from the front-end transaction log, converts it into SQL transactions, and feeds them to the back-end database. This can be considered a "normal operation" of IBM solidDB Cache system. During the operation, the connector maintains its log position and stores it in the back-end database for later recovery from any breaks in operation that may happen.

To start the connector in the propagator role, proceed as follows:

1. On the back-end:

   Once the loader has finished its job, start the propagator. Start one propagator instance for each partition replica.

2. On the front-end:

   Start the front-end applications.

The connector is designed to survive temporal load peaks and short-lived failures.

If the connector in the propagator role ("propagator") cannot process the transactions at the rate they are processed at the front-end, the propagator's log position moves more and more away from the transaction log head in the server's transaction buffer. When the buffering limit specified with the parameter *LogReader.MaxSpace* (expressed as a number of log records) is reached, the server starts to throttle the user transactions by blocking them until space is available in the buffer again. The throttling takes place only when the propagator is active.

If the propagator fails, the front-end continues to process transactions until another limit is reached. The *LogReader.MaxLogSize* parameter specifies (in bytes) the maximum distance between the propagator's log position and the log head, in the server's log. When this distance is reached, the front-end server returns an error, to the application, on every log-generating SQL statement, such as `UPDATE` or `INSERT`. This is called log overflow.

Read-only transactions will succeed. The purpose of this solution is to make sure that recovery is possible, that is, the propagator can perform a catchup when it resumes processing.

The value of *LogReader.MaxLogSize* parameter may be set to any value, up to the theoretical 2PB. For example, with the factory value of 10 GB, and the log writing rate of 1MB/s, the maximum propagator downtime of about 180 minutes will be sustained without any major disturbance of the front-end processing. The described behavior applies to all failure cases resulting in the propagator failure. Such are, for example, back-end server failure or a system failure of the back end server node.

Note

The parameter *LogReader.MaxLogSize* does not affect the real size of the physical log stored in the front-end. The log files are accumulated indefinitely, until a backup operation is performed, or the parameter *General.CheckPointDeleteLog* is set to "yes". In the latter case, the log files not needed for the next possible checkpoint-based recovery are removed, at each checkpoint, with the exception of the files protected by the *LogReader.MaxLogSize* parameter.

3.2.4 Shutting Down IBM solidDB Connector

When the connector runs in the propagator role, it can be terminated by using the admin command **solconnector propagator shutdown** to the front-end server. If the connector runs in the loader mode, the command has not effect. The **solconnector propagator shutdown** command syntax is presented below:

```
admin command 'solconnector propagator shutdown [all|<partition-id>]'
```

where <*partition-id*> is the partition name stored in the replication model.

For example:

```
admin command 'solconnector propagator shutdown all';
```

When you issue this command, either one partition replica-specific connector instance or all of them are shut down. Before shutting down, each connector writes and commits all the transactions that have been retrieved in the back-end database.

3.2.5 Removing Partition Replicas from Front-ends

If you want to permanently remove partition replicas from front-ends, you must start the connector with the `clean` startup option. This option removes both the replica tables and the replica model description from the front-end.

For more information on startup options, see Section 3.2.1, “IBM solidDB Connector Startup Options”.

3.2.6 Replica Model Evolution

Because the back-end owns the partition schema, the following procedure is required to change the schemas of the replicated data:

1. Suspend the active applications at all front-ends.
2. Redefine or alter the partition tables in the back-end.
3. Redefine or alter the partition replica tables at front-ends.
4. Update the replication model if necessary.
5. Reload the partitions.
6. Resume application processing at front-ends.

3.2.7 Connector Errors

Given the architecture of IBM solidDB Cache, the applications running on the front-end or back-end are not likely to receive any errors from the connector. This is because (1) the applications are not directly connected to the connector, and (2) the connector operation is asynchronous with respect to the SQL-level requests and transactions executed by the applications.

Any problem a connector encounters is reported in the connector message log. The connector errors may be divided into three categories:

1. SQL or server errors originated from the front-end, resulting from the connector's request executed at the front-end. IBM solidDB documentation can be consulted for more information about those errors.

2. SQL or server errors originated from the back-end, resulting from the connector's request executed at the back-end. The documentation of the respective database product can be used, in that case.

3. Connector errors. These are errors detected by the connector's logic. Examples are typical operational errors like incorrect connector configuration or missing database instances. There may be also errors resulting from incorrect model table population, or some lack of consistency at the user data level. The connector errors messages are meant to be self-explanatory.

   Most connector errors will force the connector to terminate. In such a case, find out the reason for the termination, rectify the cause, and restart the connector.

3.3 IBM solidDB Connector Configuration File

The connector configuration is stored in the `solconnector.ini` configuration file. The configuration parameters are explained in the tables below.

The `solconnector.ini` is divided into different sections. The section names are also mentioned in the table.

The `solconnector.ini` file must be located in the connector's working directory.

3.3.1 Configuration File Parameters

Table 3.1. General Parameters

[general]	Description	Factory Value
Verbose	This parameter enables the verbose operation of the connector. Verbose operation is useful for troubleshooting purposes.	NO
TraceSizeKb	The connector maintains at most two trace files at a time. When the currently written file (for example `solconnector.out`) reaches the *TraceSizeKb* value, it is renamed (to `solconnector.bak`) and a new current trace file is created.	100
TraceFilePrefix	This parameter defines the connector trace file name prefix.	solconnector

Table 3.2. Back-end Database Parameters

[backenddb]	Description	Factory Value
BatchSize	With this parameter, the user can control how log records read from the front-end are batched into bigger transactions to be executed in the back-end. In the case the propagator reaches the head of the log, it commits the batch immediately (there is no waiting). This parameter specifies the number of front-end committed log operations batched into a single back-end transaction. If *BatchSize* is 1, single transactions are propagated without batching. If *BatchSize* is bigger than 1, committed transactions are propagated with the given batch size (transactions are grouped). Unit: log operations.	300
ThreadPoolSize	This parameter specifies the maximum number of apply threads.	8
Driver	This parameter specifies the back-end database driver. It can be, for example, `com.ibm.db2.jcc.DB2Driver`.	com.ibm.db2.jcc.DB2Driver

[backenddb]	Description	Factory Value
Url	This parameter specifies the back-end database Universal Resource Locator (URL). It can be, for example, `jdbc:db2://localhost:50000/db2dbe`.	jdbc:db2://
User	This parameter specifies the back-end database user name.	N/A
Password	This parameter specifies the back-end database password.	N/A
UseQuotedNames	This parameter specifies whether to use double quotes around schema, table or column names.	No
UseSchemaName	If this parameter is set to yes, the back-end data tables are addressed by the propagator using a qualified name (prefixing the table name with the back-end schema name). In that case, the schema name has to be entered in the replication model (it cannot be NULL). If this parameter is set to no, the connector's session default schema name is used (typically, the username).	No

Table 3.3. Front-end Parameters

[frontenddb]	Description	Factory Value
Driver	This parameter specifies the front-end database driver. It can be, for example, `solid.jdbc.SolidDriver`.	soliddb.jdbc.SoliddbDriver

Table 3.4. Loader Parameters

[loader]	Description	Factory Value
CommitBlock	This parameter specifies how often the connector commits data when operating in the loader role.	1500
PropagatorShutdownTimeout	The time in seconds that the loader waits for the propagator to terminate when it is requested to do so by the loader. When the timeout expires, the loader shuts down the propagator forcefully. In that case, the propagator generates an error. Otherwise than for the error report, both ways to shut down the propagator are correct.	10

3.3.2 Example solconnector.ini File

See below for an example `solconnector.ini` configuration file:

```
;Example solconnector.ini
;------------------------
;IBM solidDB Connector
;2008-06-11

[general]
; enable verbose output; factory value is NO
;Verbose=yes

; trace file split size; factory value is 100
TraceSizeKb=500
;
; connector trace file name prefix; factory value is solconnector
;TraceFilePrefix=connector1

[backenddb]
ThreadPoolSize=16 ;max number of apply threads; factory value is 8
;
; driver class name
Driver=com.ibm.db2.jcc.DB2Driver

; JDBC URL
Url=jdbc:db2://localhost:50000/db2dbe
;
; back-end username
User=master

; back-end password
Password=jshdfj55

;use schema.tablename (factory value is No)
UseSchemaName=no

;use double quotes around SQL object names (factory value is No)
UseQuotedNames=no
```

```
[frontenddb]
driver=soliddb.jdbc.SoliddbDriver

[loader]
; how often loader commits; factory value is 1500
;CommitBlock=1500
```

3.3.3 Using IBM solidDB Connector with IBM solidDB HSB

IBM solidDB connector can be used with IBM solidDB HotStandby (HSB). In that configuration, the connector is capable of automatic failover to the standby node, in the case of server failovers and switchovers.

In order to enable automatic failover in the connector, the JDBC URL has to include the TF (transparent failover) directive. The following is an example front-end JDBC URL entered in `solconnector.ini`:

```
jdbc:solid://host1:1324,host2:1424/dba/dba?solid_tf_level=CONNECTION
```

Note

The TF level `CONNECTION` is mandatory. No other TF level can be used with the connector.

3.4 IBM solidDB Connector Configuration Parameters on the Front-end

IBM solidDB connector configuration parameters are in the `[LogReader]` section of the `solid.ini` configuration file on the front-end server(s). The parameters are described in the table below:

Table 3.5. LogReader Parameters

[LogReader]	Description	Factory Value	Access Mode
LogReaderEnabled	By using this parameter, you can enable or disable logreader capability, that is, the connector access.	no	RO
MaxLogSize	This parameter defines the size of the protected portion of the disk-based transaction log. When the log files are removed, for example, in conjunction with a backup, at least the specified amount of the log data is retained. The	10240	RW

[LogReader]	Description	Factory Value	Access Mode
	protected portion of the log facilitates a possible catchup after a failure case when the propagator has not been active for some time. The actual log size may exceed the *MaxLogSize* value, if the log files are not removed. Catchup is possible as long as the propagator log position is within the existing log. The minimum value is 5 (5 MB). If you attempt to define a smaller log size, it is automatically changed to 5 MB. The maximum possible log size is practically unlimited. Unit: megabytes.		
MaxSpace	This parameter defines the maximum number of log operations buffered before slowdown. The log operations that are buffered in an in-memory logreader buffer. When the buffer fills up, throughput throttling is applied in the front-end: the operations are blocked until there is room in the logreader buffer. The throttling only takes place when the propagator is active. If the propagator fails or is not started, the front-end processing continues until the *MaxLogSize* limit is reached (see above). Later propagation recovery is based on the catchup.	100000	RW

3.5 Using the IBM Universal JDBC Driver with IDS

If the back-end server is Informix Dynamic Server (IDS), the connector can use any of the two JDBC drivers: the native one (`ifxjdbc.jar`) or the IBM universal JDBC driver (`db2jcc.jar`). If the latter is used, the following guidelines have to be taken in to account:

- The back-end user tables have to created by using `DELIMIDENT=y` (for example in Linux bash shell **export DELIMIDENT=y**) and by using uppercase double quoted names on each table name and column name.

  For example:

```
CREATE TABLE "TEST_TAB" ("I" INTEGER NOT NULL PRIMARY KEY,
```

```
"C" CHARACTER(10), "U" INTEGER);
```

- Configure *UseQuotedNames=yes* in the `solconnector.ini`. This parameter is located in the `[backenddb]` section.
- The connection string to the IDS back-end (configured in the `solconnector.ini`) configuration file, must follow the example below:

  ```
  jdbc:ids://server:9088/db_name:DELIMIDENT=TRUE;
  ```

  This causes the IDS to use quoted names with the connection.
- If the select list (the column `FEDT_TABLE_PARTITION.SELECT_LIST`)) in the replication model is something else than `*` (or `NULL`), the column names have to be quoted and written in capital letters. For example, if the above example table `TEST_TAB` is defined in the front-end with only the columns `"I"` and `"C"`, the `select_list` column in the replication model has to contain the following string to work properly:

  ```
  "I", "C"
  ```

Chapter 4. Failure Scenarios

The following sections provide an overview of different failure scenarios and the required recovery procedures, if any.

Important

If the recovery instructions contain manual tasks, they can often be alternatively automated by using scripts.

4.1 Standalone Front-End Server Fails

If the standalone front-end server fails, the connector terminates. To recover, proceed as follows:

1. Restart the front-end server and recover the database
2. Restart the connector

As a result, the connector resynchronizes the databases and continues operation normally.

4.2 A Server in The solidDB HA Mode (HotStandby) fails

4.2.1 Primary Front-End Fails

The primary front-end failure can only happen in a HotStandby configuration. In this case, no manual intervention is needed to continue with the connector operation.

If the primary front-end fails, a high availability manager, like the High-Availability Controller (HAC), in the front-end tier performs a fail-over to the secondary front-end as a standard procedure. If the 2-Safe protocol is used, the database and log states are fully preserved. The applications perceive a failover time of less than one second. [1]

The IBM solidDB JDBC driver in the connector fails over to the secondary connection by using the Transparent Failover feature, which is a standard behavior of the Transparent Connectivity driver in IBM solidDB.

The connector preserves its state and continues normally.

[1] For more information on the High-Availability Controller (HAC), see *IBM solidDB High Availability User Guide*.

4.2.2 Secondary Front-End Fails

The secondary front-end failure can only happen in a HotStandby configuration. In this case, no manual intervention is needed to continue with the connector operation.

If the secondary front-end fails, the secondary front-end node is recovered in a normal way that is specific to the installation (e.g. automatically rebooted). HAC automatically performs the rest of the recovery. The failure is not visible to applications or to the connector.

4.3 IBM solidDB Connector Fails

If the connector in the propagator role fails, it should be restarted.

If the propagator fails, the front-end continues to process transactions until it reaches the limit specified by the *LogReader.MaxLogSize* parameter. This parameter specifies (in Mbytes) the maximum distance between the propagator's log position and the log head, in the server's log. When this distance is reached, the server returns an error on every log-generating SQL statement, such as `UPDATE` or `INSERT`. Read-only transactions will succeed. The purpose of this solution is to make sure that recovery is possible, that is, the propagator can perform a catchup when it resumes processing.

4.4 Communication Link between IBM solidDB Connector and Back-end Fails

If the communication link between the connector and a back-end fails, the connector fails and it must be restarted. This failure is unlikely to happen. However, if it happens, the front-end continues to process transactions until it reaches the limit specified by the *LogReader.MaxLogSize* parameter.

4.5 Back-end Fails, HADR/HDR Used

In any case of a back-end server failure, the connector terminates at the next propagation event. This means is has to be restarted (in any location) once the back-end recovery is performed. During the back-end recovery, the front-end continues to process transactions until the *LogReader.MaxLogSize* limit is reached.

If, on the back-end, either High Availability Disaster Recovery (HADR) or Host Data Replicator (HDR) is used, the back-end primary fails-over to the back-end secondary normally. Once the failover is done, the connector should be restarted, for example, at the back-end secondary (the new primary). The connector will resume processing from the log position stored in the back-end database.

In the case in question, the failover time is usually short (tens of seconds, at the maximum), and in most cases the buffered transactions never reach the limit specified by the *LogReader.MaxLogSize* parameter, and the catchup is automatically performed before transactions begin to fail.

4.6 Back-end Fails, HADR/HDR not Used

If the back-end fails, and HADR/HDR is not used, necessary actions should be performed first to restart and recover the database. When the back-end database has recovered, the connector should be restarted. It resumes from the log position stored in the back-end.

In the mean time, the front-end continues to process transactions until it reaches the limit specified by the *LogReader.MaxLogSize* parameter. When the limit is reached, the replica tables become read-only.

If a back-end configuration without HADR/HDR is used, make sure the value of *LogReader.MaxLogSize* in the front-end configuration file (solid.ini) is set to a value allowing to sustain a typical break in the back-end operation.

Appendix A. Replication Model Schema

The replication model specifies the division of data between the front-ends and the back-end. It is based on a model schema, which is defined on the back-end. To be more specific:

- The replication model data on the back-end server defines the partitions, replicated tables and all the replica servers. The user must populate the model.
- On the front-end(s), the data derived from the back-end replication model defines the user tables that are part of the partition and that are located on the node. The data on these user tables can be modified and the modifications are transmitted to the back-end through the log transmitter.

  Additionally, the data derived from the back-end replication model defines the tables that can only be used for read-only queries (that is, `SELECT`s).

On the front-ends, the tables corresponding to the back-end replication model are automatically predefined as system tables. The tables names are the same except that the front-end system tables use the `SYS_` prefix. The front-end model tables are automatically maintained by the system.

The user must define the back-end replication model by populating the tables described below and by following the related rules.

In all the table descriptions below, the term *SQL name* refers to a name that must conform to the SQL naming standard for SQL objects. The standard defines, for example, that you cannot use spaces in the name. SQL names are restricted to 30 characters.

In all the table descriptions below, the columns not marked as optional or `NULL` must be populated before a load may be performed. If the column is marked as `NULL`, it is managed by the front-end or by the connector.

IBM solidDB can be used as a back-end database, too. In that case, note that the replication model tables are already build-in as system tables. They are named in such a way that the `SYS_` prefix is added before the table name shown below. For example the `FEDT_DB_PARTITION` table becomes `SYS_FEDT_DB_PARTITION` in IBM solidDB. This should be taken into account when populating the model tables in IBM solidDB.

The tables are introduced in the sections below.

Note

The primary keys are in **`bold`** in the tables.

A.1 FEDT_DB_PARTITION

The FEDT_DB_PARTITION table describes the partitions. The whole replication model is based on these partitions.

The rules laid out below must be applied when the replication model is populated in the back-end. The contents will be automatically transferred to front-ends upon load.

Table A.1. FEDT_DB_PARTITION Table

Column Name	Data Type	Description and Population Rules
PARTITION_ID	CHAR(30)	This column contains a descriptive partition name. The name is an SQL name.
DESCRIPTION	VARCHAR(255)	This column contains an optional free-form partition description.
LOADACTIVE	INTEGER	This value must be set to NULL on the back-end. The LOADACTIVE attribute is only used on the front-end databases. It denotes that tables belonging to this partition are in the process of loading data from the back-end server.

A.2 FEDT_TABLE_PARTITION

The FEDT_TABLE_PARTITION table provides a mapping between tables and partitions.

The optional names for back-end-related names allow for name mapping between the back-end and front-end. If the columns are NULL, it is assumed that the back-end and front-end table names match.

Table A.2. FEDT_TABLE_PARTITION Table

Column Name	Data Type	Description and Population Rules
TABLE_CATALOG	VARCHAR(255)	In IBM solidDB, each table has an associated catalog name. A catalog is created in IBM solidDB by a user when the database is created, or afterwards by using a CREATE CATALOG statement. There is no factory value for a catalog name. This column indicates the front-end catalog name. The catalog name is an SQL name.

Column Name	Data Type	Description and Population Rules
TABLE_SCHEMA	VARCHAR(255)	In IBM solidDB, each table has an associated schema name. By default, the schema name is the user ID (username). A schema may be also created by using the CREATE SCHEMA statement. This column indicates the front-end schema name. The schema name is an SQL name.
TABLE_NAME	CHAR(30)	This column indicates the table name. The table name is an SQL name.
PARTITION_ID	CHAR(30)	This column indicates the partition name. The partition name is an SQL name. This column is a foreign key to the FEDT_TABLE_PARTITION table.
BE_TABLE_DATABASE	VARCHAR(255)	The BE_TABLE_DATABASE value denotes the database name in the backend. If IBM solidDB is used as a backend server, the name corresponds to the catalog name in IBM solidDB. If the backend database is DB2, this value should be NULL. The use of this column is enabled with the connector parameter *Backend.UseSchemaName* set to "yes". This is an optional column.
BE_TABLE_SCHEMA	VARCHAR(255)	The BE_TABLE_SCHEMA defines the back-end schema name This is an optional column. However this column is mandatory if BE_TABLE_DATABASE is specified. The use of this column is enabled with the connector parameter *Backend.UseSchemaName* set to "yes".
BE_TABLE_NAME	CHAR(30)	The BE_TABLE_NAME defines the back-end table name. This is an optional column.

Column Name	Data Type	Description and Population Rules
		If not specified, TABLE_NAME is used to denote the backend table name. The value is mandatory if BE_TABLE_SCHEMA is specified.
RANGE_COL_NAME	CHAR(30)	This column indicates the range column name denoting the single column that is used for horizontal partitioning. See also LOWER_LIMIT and UPPER_LIMIT columns below. The range column name is an SQL name. This is an optional column.
LOWER_LIMIT	VARCHAR(255)	If horizontal partitioning is used, LOWER_LIMIT denotes the lower limit to the range of values belonging to the described partition. This is an optional column.
UPPER_LIMIT	VARCHAR(255)	If horizontal partitioning is used, UPPER_LIMIT denotes the upper limit to the range of values belonging to the described partition. This is an optional column.
SELECT_LIST	VARCHAR(255)	SELECT_LIST denotes a list of column names that are projected from the back-end before inserting data to the front-end server. This is an optional column.
LOADORDER	INTEGER	LOADORDER denotes, if given, the order in which tables are copied from the back-end server to the front-end server. If LOADORDER is NULL, the tables can be loaded concurrently or in any order. The tables are loaded in the ascending order of the LOADORDER values. This is an optional column.

A.3 FEDT_REPLICA_SERVER

A replica server refers to a front-end site. FEDT_REPLICA_SERVER contains the network server name and the connect string to that server.

Table A.3. FEDT_REPLICA_SERVER Table

Column Name	Data Type	Description and Population Rules
REPL_SERVER_ID	CHAR(30)	This column contains the replica server identification. The identification is an SQL name.
SERVER_NAME	CHAR(30)	This column contains the server name, that is, the host name of the replica server.
CONNECT_STRING	VARCHAR(255)	This column contains the connect string to the replica server.

A.4 FEDT_PARTITION_REPLICA

FEDT_PARTITION_REPLICA provides a mapping between partitions and replica sites.

Table A.4. FEDT_PARTITION_REPLICA Table

Column Name	Data Type	Description and Population Rules
REPL_SERVER_ID	CHAR(30)	This column contains the replica server identification. This column is a foreign key to the FEDT_REPLICA_SERVER table. The identification is an SQL name.
PARTITION_ID	CHAR(30)	This column contains the partition name. This column is a foreign key to the FEDT_DB_PARTITION table. The partition name is an SQL name.
LOAD_TS	TIMESTAMP	This column contains the load timestamp for the data. (NULL)
READONLY	INTEGER	The optional READONLY attribute defines whether this partition is a read-only or read-write copy of data. The replica is a read-write replica if this column is set to NULL or zero. Any other value denotes that the replica is a read-only replica. If the replica is a read-only replica, the back-end or another front-end server owns the data and no updates or inserts are allowed to this partition on this server.

A.5 FEDT_PARTITION_REPLICA_POS

The FEDT_PARTITION_REPLICA_POS table is used to store the current log position of each propagating thread in the connector. This position is needed in recovery from failures.

Note

This table is fully managed by the connector. Do not populate or update it.

Table A.5. FEDT_PARTITION_REPLICA_POS Table

Column Name	Data Type	Description and Population Rules
REPL_SERVER_ID	CHAR(30)	This column contains the replica server identification. This column is a foreign key to the FEDT_REPLICA_SERVER table. (NULL)
PARTITION_ID	CHAR(30)	This column contains the partition name. This column is a foreign key to the FEDT_DB_PARTITION table. (NULL)
THREAD_ID	INTEGER	This column contains the connector thread ID. (NULL)
POSITION_ID	INTEGER	POSITION_ID is a connector-specific counter value incremented each time a new log position is entered. (NULL)
LOG_POS	CHAR(8)	This column contains the log position. (NULL)

A.6 Example Model Population Script

The following SQL script illustrates a simple replication model. A Cache partition called TESTPARTITION is composed of two tables: TABLE1 and TABLE2. All the rows and columns of the tables are included in the partition. A single front-end is specified, and it is assigned a writable replica of the partition.

```
-- Replication model example

-- define a partition
INSERT INTO fedt_db_partition (PARTITION_ID, DESCRIPTION)
        VALUES ('TESTPARTITION', 'Example partition');

-- asign tables to the partition
INSERT INTO fedt_table_partition (TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME,
```

```
PARTITION_ID)
        VALUES ('DBA', 'DBA', 'TABLE1', 'TESTPARTITION');
INSERT INTO fedt_table_partition (TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME
PARTITION_ID)
        VALUES ('DBA', 'DBA', 'TABLE2', 'TESTPARTITION');

-- specify a front-end
INSERT INTO fedt_replica_server (REPL_SERVER_ID, SERVER_NAME,
CONNECT_STRING)
        VALUES ('FE', 'solidDB-imdb',
'jdbc:solid://localhost:1326/dba/dba');

-- assign a writable partition replica to the front-end
INSERT INTO fedt_partition_replica (REPL_SERVER_ID, PARTITION_ID,
READONLY)
        VALUES ('FE', 'TESTPARTITION', 0);

COMMIT WORK;
```

Index

P

R

S

T

U

V