Release Notes

Product	VoltDB
Version	8.2
Release Date	July 12, 2018

This document provides information about known issues and limitations to the current release of VoltDB. If you encounter any problems not listed below, please be sure to report them to support@voltdb.com. Thank you.

Upgrading From Older Versions

The process for upgrading from the recent versions of VoltDB is as follows:

1. Shutdown the database, creating a final snapshot (using **voltadmin shutdown --save**).
2. Upgrade the VoltDB software.
3. Restart the database (using **voltdb start**).

For DR clusters, see the section on "Upgrading VoltDB Software" in the *VoltDB Administrator's Guide* for special considerations related to DR upgrades. If you are upgrading from versions before V6.8, see the section on "Upgrading Older Versions of VoltDB Manually" in the same manual.

For customers upgrading from V7.x or earlier releases of VoltDB, please see the *V7.0 Upgrade Notes*.

For customers upgrading from V6.x or earlier releases of VoltDB, please see the *V6.0 Upgrade Notes*.

For customers upgrading from V5.x or earlier releases of VoltDB, please see the *V5.0 Upgrade Notes*.

For customers upgrading from V4.x or earlier releases of VoltDB, please see the *V4.0 Upgrade Notes*.

Changes Since the Last Release

Users of previous versions of VoltDB should take note of the following changes that might impact their existing applications.

1. Release V8.2 (July 12, 2018)

1.1. New TTL feature automates deleting old data

A new feature, "time to live" (TTL), allows you to define an expiration timestamp for individual tables. Once the TTL value is exceeded, the records from that table are automatically deleted from the database. This makes the processing of streaming data easier by automating the deletion of old data.

You define the expiration timestamp with the new USING TTL {value} ON COLUMN {column-name} clause in the CREATE TABLE statement. You can also monitor the performance of TTL processing using the new

TTL selector for the @Statistics system procedure. See the documentation of CREATE TABLE and @Statistics in the *Using VoltDB* manual for details.

1.2. Support for reading the username and password from a file for the VoltDB command line utilities

When using scripts to manage a secure database, the command line utilities (such as **sqlcmd** and **voltadmin**) require a username and password. Previously, there was no easy way to do this without either using Kerberos or hardcoding the information into the script itself. Now you can save the username and password into a properties file — accessible only to the user running the script — and then reference that file in the script using the new `--credentials` argument. See the description of the command line utilities in the *Using VoltDB* manual for details.

1.3. New option to create cluster-wide unique file names on file export

The file export connector writes export data to files on each server in a cluster. By default, the files are unique per server, but not necessarily across the cluster as a whole. You can now set the property `uniquenames` to *true* in the export configuration to ensure that all files are unique cluster wide. See the description of the file export connector in the *Using VoltDB* manual for details.

1.4. Improved performance when restoring snapshots

Restoring the database from a snapshot can take time, particularly for large databases with many views. This release improves the performance of snapshots by storing the contents of certain views as part of the snapshot, eliminating the need to rebuild the views on the fly when restoring the snapshot. Note that not all views can be saved in the snapshot; views containing partitioned tables but no partitioning column in the GROUP BY clause must still be rebuilt. Also, snapshots created on earlier versions of VoltDB will still have their views rebuilt as part of the restoration process. But for new snapshots with views of replicated tables or partitioned tables with at least one partitioning column in the GROUP BY column, restoring the snapshots should be noticeably faster.

1.5. New sqlcmd directive describes the columns in a table

The **sqlcmd** utility now supports the DESCRIBE *table-name* directive. DESCRIBE lists the columns of the specified table, stream, or view and related information, such as datatype and size. See the description of **sqlcmd** in the *Using VoltDB* manual for details.

1.6. Security Notice

The following change has been made to improve security and eliminate potential threats:

- Ability to "hide" the username and password for command line utilities in a separate credentials file. (See description above.)

- When enabling SSL with a user-generated certificate, you need to specify both a keystore and truststore. When using a commercial certificate a local truststore should not be needed. However, previously VoltDB still required one. Specifying a truststore is no longer required when using a commercial certificate.

1.7. Additional improvements

In addition to the new features and capabilities described above, the following limitations in previous versions have been resolved:

- There was an issue with database replication (DR) where a large multi-partition transaction could produce binary logs from each partition on the producer that fell under the 50MB limit on inter-cluster communication. But when aggregated on the consumer for replay, the transaction exceeds the limit. The symptom was that nodes on the cluster would report a "bad message length" exception, causing nodes to

be expelled from the cluster until the cluster itself failed. The possibility of an excessively large transaction still exists, but now the producer cluster rejects the transaction and replication and consistency between the clusters is maintained.

- There was an issue using the VoltDB Management Center (VMC) if the VoltDB http port was set to port 80. Port 80 is the default port for web browsers and if the browser did not send a port number VMC would incorrectly assume a default of 8080 and not operate properly. This issue has been resolved.

- VoltDB reserves the maximum negative value of numeric datatypes as null. (For example, -128 for a TINY integer.) So users should not be allowed to use these reserved values in the context of a given datatype. However, previously the compiler silently accepted such constants and interpreted them as null. This issue has been resolved. The compiler now throws an error when evaluating numeric values equal to a datatype's maximum negative value.

- Certain SQL functions (such as NOW and PI) that take no argument can be entered with or without parentheses. However, these functions were not interpreted consistently in the selection list of a SELECT statement. If the parentheses were left off, NOW was interpreted as the function and PI was interpreted as a column reference. These functions are now both interpreted as functions, whether with or without parentheses.

  Note: this is a slight change of behavior. If you use a column with the name PI, you will now have to fully qualify the name to have it be interpreted as a column rather than a function. For example, in the following statement, the first item in the section list is interpreted as the function PI and the second as the column PI:

  ```
  SELECT pi, a.pi FROM a WHERE...
  ```

- There was a rare condition involving database replication (DR), where replication could break if a producer cluster suffered a network partition. If the production cluster split into two segments due to network issues, a race condition could result in the consumer cluster querying the smaller segment of the cluster for topology information after the separation but before the smaller segment was shutdown by VoltDB's network partition detection. If this occurred, the consumer cluster would wait for the smaller segment and fail to poll the larger, surviving segment. This issue has been resolved.

2. Release V8.1.2 (June 14, 2018)

2.1. Recent improvement

The following limitation in V8.1 has been resolved:

- VoltDB 8.1 introduced a performance improvement to the data loading utilities and bulkloader API. Unfortunately, this feature also introduced a potential error condition where, if the loader encounters a runtime error, such as an input value exceeding the maximum width of a VARCHAR column, rather than rolling back the transaction it could crash the database cluster. This issue has now been resolved. Because this bug can cause the database to stop, we strongly recommend all customers using 8.1 or 8.1.1 upgrade to 8.1.2 at they earliest convenience.

3. Release V8.1.1 (June 7, 2018)

3.1. Recent improvement

The following limitation in V8.1 has been resolved:

- VoltDB 8.1 introduced an issue that could interfere with the resilience of a K-safe cluster. If a node failed while processing a multi-partition transaction, it was possible for the remaining nodes in the cluster to suffer a deadlock. When this happened, the warning "possible multipartition transaction deadlock detected" was reported and all subsequent multi-partition transactions would hang, along with certain system operations

such as snapshots and command log truncation. This issue has now been resolved. Customers using V8.1 are strongly encouraged to update to V8.1.1 at their earliest convenience.

4. Release V8.1 (May 26, 2018)

4.1. Improved performance of export during schema changes

VoltDB now does a better job of managing the interaction of export and ongoing schema changes. Previously, export associated with the original schema had to drain before export using the new schema could begin. Now export from before and after a schema change are managed independently and in parallel.

4.2. Better memory management for replicated tables

The storage of replicated tables has been reorganized in this release. Previously, each partition retained a copy of the replicated tables. Now, all of the partitions on a server share a single copy of the tables. Applications with sizeable replicated tables or a high sites-per-host count should notice a significant reduction in the amount of memory required by VoltDB after the upgrade.

4.3. Improved bulk loading of replicated tables

The default process for bulk loading replicated tables — either through the loader utilities such as csvloader or through the bulkloader API — has been improved. When bulk loading data into a replicated table using the default load procedure and performing inserts (not upserts), the load process can be as much as three times faster, according to testing.

4.4. New optimization for limit/offset query performance

There is a limit (50 megabytes) to the amount of data any query can return. When reading large volumes of data from a VoltDB database, use of the LIMIT and OFFSET clauses to "page" though the data is recommended. However, as the OFFSET value increases significantly, each query can take incrementally longer to execute. This release introduces a new feature where, if there is an index on the appropriate columns of the table, the query is optimized eliminating the penalty associated with large offsets.

4.5. Improved ad hoc query performance

Ad hoc queries that perform read-only operations on replicated (non-partitioned) tables have traditionally been executed by a single partition within the database, assuming such queries are a small percentage of the overall workload. However, when they are *not* a small percentage, that one partition can get over extended, resulting in increased latency. This release changes the execution model to "round robin" read-only queries of replicated tables to more evenly distribute the workload.

4.6. New system procedure @Ping

A new procedure has been added to the list of supported system procedures for VoltDB, @Ping. It returns a value of zero (0) if the database is up and operational. The @Ping system procedure is a lightweight procedure and does not require any interaction between cluster nodes, which makes it a better choice than other system procedures (such as @Statistics) if all you need to do is check if the database is running. See the description of @Ping in the *Using VoltDB* manual for details.

4.7. Additional improvements

In addition to the new features and capabilities described above, the following limitations in previous versions have been resolved:

- There was an issue where the **voltadmin stop node** command would not properly authenticate with the server when using Kerberos security. This issue has been resolved.

- There was an issue where, if a database was upgraded to a new version of VoltDB using **voltadmin shutdown --save**, restarted, but then crashes unexpectedly (for example, using **kill -9**), the database could not restart a second time. This issue has been resolved.

- Since VoltDB V7.7, it was possible for the response from a multi-partition read transaction to be "lost" during a node failure. This could only happen on K-safe clusters, where the node that failed was the multi-partition initiator (that is, the node responsible for coordinating multi-partition transactions), the node failed before the transaction completed, and the procedure call was invoked on a *different* node of the cluster. Under these conditions, the calling application might not receive a response from the invocation. Note that this issue only occurred on K-safe clusters, for read-only multi-partition queries, and for non-topology-aware clients. This issue has now been resolved.

- There was a sporadic problem with authentication of the VoltDB Management Center (VMC) on slow network connections. Starting with VoltDB 7.9, the JSON interface uses stay-alive connections with a timeout period of 10 seconds. VMC sends requests every 5 seconds. However, on slow networks the VMC calls could be delayed beyond the timeout period, forcing the user to re-authenticate manually. The timeout period has been extended to alleviate the unexpected timeouts.

- There was an issue with Kafka import where, if the database cluster was paused and then resumed, it was possible for certain Kafka records that were being processed when the database was paused to be lost. This issue has been resolved.

- There was a longstanding and somewhat obscure bug involving partitioned views joined to a derived table from a sub SELECT statement. If a partitioned view did not include the table's partition column and was joined to another table derived from a sub selection (that is a SELECT statement in parentheses in the FROM clause of the main SELECT) the query could result in unexpected behavior at runtime, including possibly crashing the database. This issue has been resolved.

5. Release V8.0 (February 6, 2018)

5.1. TLS/SSL encryption for intra-cluster communication

VoltDB now supports encrypting communication on the internal port, the port used for communication between nodes in the cluster, using TLS/SSL encryption. Note that encrypting the internal port automatically adds latency to any operations that require inter-node communication, such as K-safety and multi-partition procedures. The actual impact depends on the configuration and application workload. It is strongly recommended you benchmark your application before enabling internal TLS/SSL on production systems. See the chapter on "Security" in the *Using VoltDB* manual for details.

5.2. New behavior for placement groups

Placement groups, or rack-aware provisioning, was introduced in VoltDB 5.5. Placement groups let you specify where each node is located, so in a virtualized K-safe environment multiple copies of a partition are distributed onto distinct hardware, racks, etc. However, changes in VoltDB 7.0 to optimize K-safe partitioning in all cases ended up superceding placement groups and invalidating the rack-aware positioning.

This unintentional side effect has been corrected and placement groups once again provide rack-aware provisioning. However, the algorithm for interpreting placement groups has changed. Where before you could use a hierarchical list of names separated by periods (such as rack1.switch3.server5) the new algorithm focuses on the first name only and subnames are largely ignored.

Use of simple (non-hierarchical) placement names is recommended. In addition, the following rules apply to the top-level names:

- There must be more than one top-level group specified for the cluster

- The same number of nodes must be included in each group
- The number of partition copies (that is, K+1) must be a multiple of the number of top-level groups

5.3. Kafka 0.10.2 is now the default for Kafka import and kafkaloader

The default for the Kafka import connector and the kafkaloader command line utility has changed to support Kafka 0.10.2 and later, including the recent 1.0.0 release. Earlier versions of Kafka (0.8.2) are still supported through configuration options and an alternate kafkaloader8 utility.

5.4. Support for common table expressions

VoltDB now supports common table expressions, including recursive common table expressions. See the description of the SELECT statement in the *Using VoltDB* manual for details.

5.5. Deprecated features removed from the product

The following features, that had previously been deprecated, have now been removed from the product as of VoltDB 8.0:

- "Fast" read consistency (`<consistency>` element)
- Old, non-elastic partitioning (`<cluster>` `elastic` attribute)
- @ProcInfo Java annotation for specifying procedure partitioning
- Old shell commands (**voltdb add**, **create**, **recover**, and **rejoin**)

We are also deprecating the VoltDB Deployment Manager.

5.6. VoltDB Deployment Manager is deprecated

The VoltDB Deployment Manager was designed as a console for deploying VoltDB clusters. However, it has not met its goals for ease-of-use and flexibility. Therefore, we are deprecating it and it will be removed from the product in a future release. In its place we recommend using one of the existing frameworks for managing distributed systems such as (but not limited to) Chef, Puppet, Docker, and Kubernetes.

5.7. Security Notice

The following changes have been made to improve security and eliminate potential threats:

- Ability to enable TLS/SSL encryption on the internal port. (See description above.)

5.8. Additional improvements

In addition to the new features and capabilities described above, the following limitations in previous versions have been resolved:

- There was an issue using the SQL Query tab of the web-based VoltDB Management Console (VMC) to insert or filter records if any text fields in the query contained multiple consecutive spaces. (For example, two or more leading spaces or multiple spaces between two words.) Some of the spaces were interpreted as the UTF-8 character for a non-breaking space (\u00A0) rather than ASCII code 32, causing incorrect data insertion or filtering. This issue has been resolved.
- In the past it was possible for queries containing both a FULL or RIGHT OUTER JOIN and a GROUP BY operation on a floating point (FLOAT) column to produce incorrect results. This issue has been resolved.

- Previously, certain combinations of UNION, ORDER BY, and LIMIT clauses in a single query could produce incorrect results. This issue has been resolved.

- There was an issue with the Nagios script for monitoring replica clusters when using database replication (DR). The script could generate a series of false alarms if the replica database was idle for more than two minutes. The alarms incorrectly reported that replication was falling behind. This issue has been resolved.

Known Limitations

The following are known limitations to the current release of VoltDB. Workarounds are suggested where applicable. However, it is important to note that these limitations are considered temporary and are likely to be corrected in future releases of the product.

1. Command Logging

1.1. Do not use the subfolder name "segments" for the command log snapshot directory.

VoltDB reserves the subfolder "segments" under the command log directory for storing the actual command log files. Do not add, remove, or modify any files in this directory. In particular, do not set the command log snapshot directory to a subfolder "segments" of the command log directory, or else the server will hang on startup.

2. Database Replication

2.1. Some DR data may not be delivered if master database nodes fail and rejoin in rapid succession.

Because DR data is buffered on the master database and then delivered asynchronously to the replica, there is always the danger that data does not reach the replica if a master node stops. This situation is mitigated in a K-safe environment by all copies of a partition buffering on the master cluster. Then if a sending node goes down, another node on the master database can take over sending logs to the replica. However, if multiple nodes go down and rejoin in rapid succession, it is possible that some buffered DR data — from transactions when one or more nodes were down — could be lost when another node with the last copy of that buffer also goes down.

If this occurs and the replica recognizes that some binary logs are missing, DR stops and must be restarted.

To avoid this situation, especially when cycling through nodes for maintenance purposes, the key is to ensure that all buffered DR data is transmitted before stopping the next node in the cycle. You can do this using the @Statistics system procedure to make sure the last ACKed timestamp (using @Statititcs DR on the master cluster) is later than the timestamp when the previous node completed its rejoin operation.

2.2. Avoid bulk data operations within a single transaction when using database replication

Bulk operations, such as large deletes, inserts, or updates are possible within a single stored procedure. However, if the binary logs generated for DR are larger than 45MB, the operation will fail. To avoid this situation, it is best to break up large bulk operations into multiple, smaller transactions. A general rule of thumb is to multiply the size of the table schema by the number of affected rows. For deletes and inserts, this value should be under 45MB to avoid exceeding the DR binary log size limit. For updates, this number should be under 22.5MB (because the binary log contains both the starting and ending row values for updates).

2.3. Database replication ignores resource limits

There are a number of VoltDB features that help manage the database by constraining memory size and resource utilization. These features are extremely useful in avoiding crashes as a result of unexpected or unconstrained growth. However, these features could interfere with the normal operation of DR when passing data from one

cluster to another, especially if the two clusters are different sizes. Therefore, as a general rule of thumb, DR overrides these features in favor of maintaining synchronization between the two clusters.

Specifically, DR ignores any resource monitor limits defined in the deployment file when applying binary logs on the consumer cluster. DR also ignores any partition row limits defined in the database schema when applying binary logs. This means, for example, if the replica database in passive DR has less memory or fewer unique partitions than the master, it is possible that applying binary logs of transactions that succeeded on the master could cause the replica to run out of memory. Note that these resource monitor and tables row limits *are* applied on any original transactions local to the cluster (for example, transactions on the master database in passive DR).

2.4. Different cluster sizes can require additional Java heap

Database Replication (DR) now supports replication across clusters of different sizes. However, if the replica cluster is smaller than the master cluster, it may require a significantly larger Java heap setting. Specifically, if the replica has fewer unique partitions than the master, each partition on the replica must manage the incoming binary logs from more partitions on the master, which places additional pressure on the Java heap.

A simple rule of thumb is that the worst case scenario could require an additional P * R * 20MB space in the Java heap , where P is the number of sites per host on the replica server and R is the ratio of unique partitions on the master to partitions on the replica. For example, if the master cluster is 5 nodes with 10 sites per host and a K factor of 1 (i.e. 25 unique partitions) and the replica cluster is 3 nodes with 8 sites per host and a K factor of 1 (12 unique partitions), the Java heap on the replica cluster may require approximately 320MB of additional space in the heap:

Sites-per-host * master/replace ratio * 20MB
8 * 25/12 * 20 = ~ 320MB

An alternative is to reduce the size of the DR buffers on the master cluster by setting the DR_MEM_LIMIT Java property. For example, you can reduce the DR buffer size from the default 10MB to 5MB using the VOLTDB_OPTS environment variable before starting the master cluster.

```
$ export VOLTDB_OPTS="-DDR_MEM_LIMIT=5"

$ voltdb start
```

Changing the DR buffer limit on the master from 10MB to 5MB proportionally reduces the additional heap size needed. So in the previous example, the additional heap on the replica is reduced from 320MB to 160MB.

3. Cross Datacenter Replication (XDCR)

3.1. Avoid replicating tables without a unique index.

Part of the replication process for XDCR is to verify that the record's starting and ending states match on both clusters, otherwise known as *conflict resolution*. To do that, XDCR must find the record first. Finding uniquely indexed records is efficient; finding non-unique records is not and can impact overall database performance.

To make you aware of possible performance impact, VoltDB issues a warning if you declare a table as a DR table and it does not have a unique index.

3.2. When starting XDCR for the first time, only one database can contain data.

You cannot start XDCR if both databases already have data in the DR tables. Only one of the two participating databases can have preexisting data when DR starts for the first time.

3.3. During the initial synchronization of existing data, the receiving database is paused.

When starting XDCR for the first time, where one database already contains data, a snapshot of that data is sent to the other database. While receiving and processing that snapshot, the receiving database is paused. That is, it is in read-only mode. Once the snapshot is completed and the two database are synchronized, the receiving database is automatically unpaused, resuming normal read/write operations.

3.4. A large number of multi-partition write transactions may interfere with the ability to restart XDCR after a cluster stops and recovers.

Normally, XDCR will automatically restart where it left off after one of the clusters stops and recovers from its command logs (using the **voltdb recover** command). However, if the workload is predominantly multi-partition write transactions, a failed cluster may not be able to restart XDCR after it recovers. In this case, XDCR must be restarted from scratch, using the content from one of the clusters as the source for synchronizing and recreating the other cluster (using the **voltdb create --force** command) without any content in the DR tables.

3.5. A TRUNCATE TABLE transaction will be reported as a conflict with any other write operation to the same table.

When using XDCR, if the binary log from one cluster includes a TRUNCATE TABLE statement and the other cluster performs any write operation to the same table before the binary log is processed, the TRUNCATE TABLE operation will be reported as a conflict. Note that currently DELETE operations always supercede other actions, so the TRUNCATE TABLE will be executed on both clusters.

3.6. Exceeding a LIMIT PARTITION ROWS constraint can generate multiple conflicts

It is possible to place a limit on the number of rows that any partition can hold for a specific table using the LIMIT PARTITION ROWS clause of the CREATE TABLE statement. When close to the limit, transactions on either or both clusters can exceed the limit simultaneously, resulting in a potentially large number of delete operations that then generate conflicts when the the associated binary log reaches the other cluster.

3.7. Use of the VoltProcedure.getUniqueId method is unique to a cluster, not across clusters.

VoltDB provides a way to generate a deterministically unique ID within a stored procedure using the getUniqueId method. This method guarantees uniqueness *within the current cluster*. However, the method could generate the same ID on two distinct database clusters. Consequently, when using XDCR, you should combine the return values of VoltProcedure.getUniqueId with VoltProcedure.getClusterId, which returns the current cluster's unique DR ID, to generate IDs that are unique across all clusters in your environment.

3.8. XDCR cannot be used with deprecated export syntax.

You cannot use cross datacenter replication (XDCR) with the deprecated export syntax, that is the EXPORT TABLE statement. To use XDCR with export, you must use the current CREATE STREAM syntax for declaring the source streams for export targets.

4. TTL

4.1. Use of TTL (time to live) with replicated tables and Database Replication (DR) can result in increased DR activity.

TTL, or time to live, is a feature that automatically deletes old records based on a timestamp or integer column. For replicated tables, the process of checking whether records need to be deleted is performed as a write transaction — even if no rows are deleted. As a consequence, any replicated DR table with TTL defined will generate frequent DR log entries, whether there are any changes or not, significantly increasing DR traffic.

Because of the possible performance impact this behavior can have on the database, use of TTL with replicated tables and DR is not recommended at this time.

5. Export

5.1. Synchronous export in Kafka can use up all available file descriptors and crash the database.

A bug in the Apache Kafka client can result in file descriptors being allocated but not released if the producer.type attribute is set to "sync" (which is the default). The consequence is that the system eventually runs out of file descriptors and the VoltDB server process will crash.

Until this bug is fixed, use of synchronous Kafka export is not recommended. The workaround is to set the Kafka producer.type attribute to "async" using the VoltDB export properties.

6. Import

6.1. Data may be lost if a Kafka broker stops during import.

If, while Kafka import is enabled, the Kafka broker that VoltDB is connected to stops (for example, if the server crashes or is taken down for maintenance), some messages may be lost between Kafka and VoltDB. To ensure no data is lost, we recommend you disable VoltDB import before taking down the associated Kafka broker. You can then re-enable import after the Kafka broker comes back online.

6.2. Kafka import can lose data if multiple nodes stop in succession.

There is an issue with the Kafka importer where, if multiple nodes in the cluster fail and restart, the importer can lose track of some of the data that was being processed when the nodes failed. Normally, these pending imports are replayed properly on restart. But if multiple nodes fail, it is possible for some in-flight imports to get lost. This issue will be addressed in an upcoming release.

7. SQL and Stored Procedures

7.1. Comments containing unmatched single quotes in multi-line statements can produce unexpected results.

When entering a multi-line statement at the sqlcmd prompt, if a line ends in a comment (indicated by two hyphens) and the comment contains an unmatched single quote character, the following lines of input are not interpreted correctly. Specifically, the comment is incorrectly interpreted as continuing until the next single quote character or a closing semi-colon is read. This is most likely to happen when reading in a schema file containing comments. This issue is specific to the sqlcmd utility.

A fix for this condition is planned for an upcoming point release

7.2. Do not use assertions in VoltDB stored procedures.

VoltDB currently intercepts assertions as part of its handling of stored procedures. Attempts to use assertions in stored procedures for debugging or to find programmatic errors will not work as expected.

7.3. The UPPER() and LOWER() functions currently convert ASCII characters only.

The UPPER() and LOWER() functions return a string converted to all uppercase or all lowercase letters, respectively. However, for the initial release, these functions only operate on characters in the ASCII character set. Other case-sensitive UTF-8 characters in the string are returned unchanged. Support for all case-sensitive UTF-8 characters will be included in a future release.

8. Client Interfaces

8.1. Avoid using decimal datatypes with the C++ client interface on 32-bit platforms.

There is a problem with how the math library used to build the C++ client library handles large decimal values on 32-bit operating systems. As a result, the C++ library cannot serialize and pass Decimal datatypes reliably on these systems.

Note that the C++ client interface *can* send and receive Decimal values properly on 64-bit platforms.

9. SNMP

9.1. Enabling SNMP traps can slow down database startup.

Enabling SNMP can take up to 2 minutes to complete. This delay does not always occur and can vary in length. If SNMP is enabled when the database server starts, the delay occurs after the server logs the message "Initializing SNMP" and before it attempts to connect to the cluster. If you enable SNMP while the database is running, the delay can occur when you issue the **voltadmin update** command or modify the setting in the VoltDB Management Center Admin tab. This issue results from a Java constraint related to secure random numbers used by the SNMP library.

10. VoltDB Management Center

10.1. The VoltDB Management Center currently reports on only one DR connection.

With VoltDB V7.0, cross datacenter replication (XDCR) supports multiple clusters in an XDCR network. However, the VoltDB Management Center currently reports on only one such connection per cluster. In the future, the Management Center will provide monitoring and statistics for all connections to the current cluster.

Implementation Notes

The following notes provide details concerning how certain VoltDB features operate. The behavior is not considered incorrect. However, this information can be important when using specific components of the VoltDB product.

1. VoltDB Management Center

1.1. Schema updates clear the stored procedure data table in the Management Center Monitor section

Any time the database schema or stored procedures are changed, the data table showing stored procedure statistics at the bottom of the Monitor section of the VoltDB Management Center get reset. As soon as new invocations of the stored procedures occur, the statistics table will show new values based on performance after the schema update. Until invocations occur, the procedure table is blank.

2. SQL

2.1. You cannot partition a table on a column defined as ASSUMEUNIQUE.

The ASSUMEUNIQUE attribute is designed for identifying columns in partitioned tables where the column values are known to be unique but the table is not partitioned on that column, so VoltDB cannot verify complete uniqueness across the database. Using interactive DDL, you can create a table with a column marked as ASSUMEUNIQUE, but if you try to partition the table on the ASSUMEUNIQUE column, you receive an error. The solution is to drop and add the column using the UNIQUE attribute instead of ASSUMEUNIQUE.

2.2. Adding or dropping column constraints (UNIQUE or ASSUMEUNIQUE) is not supported by the ALTER TABLE ALTER COLUMN statement.

You cannot add or remove a column constraint such as UNIQUE or ASSUMEUNIQUE using the ALTER TABLE ALTER COLUMN statement. Instead to add or remove such constraints, you must first drop then add the modified column. For example:

```
ALTER TABLE employee DROP COLUMN empID;
ALTER TABLE employee ADD COLUMN empID INTEGER UNIQUE;
```

2.3. Do not use UPDATE to change the value of a partitioning column

For partitioned tables, the value of the column used to partition the table determines what partition the row belongs to. If you use UPDATE to change this value and the new value belongs in a different partition, the UPDATE request will fail and the stored procedure will be rolled back.

Updating the partition column value may or may not cause the record to be repartitioned (depending on the old and new values). However, since you cannot determine if the update will succeed or fail, you should not use UPDATE to change the value of partitioning columns.

The workaround, if you must change the value of the partitioning column, is to use both a DELETE and an INSERT statement to explicitly remove and then re-insert the desired rows.

2.4. Ambiguous column references no longer allowed.

Starting with VoltDB 6.0, ambiguous column references are no longer allowed. For example, if both the *Customer* and *Placedorder* tables have a column named *Address*, the reference to *Address* in the following SELECT statement is ambiguous:

```
SELECT OrderNumber, Address FROM Customer, Placedorder
   . . .
```

Previously, VoltDB would select the column from the leftmost table (Customer, in this case). Ambiguous column references are no longer allowed and you must use table prefixes to disambiguate identical column names. For example, specifying the column in the preceding statement as *Customer.Address*.

A corollary to this change is that a column declared in a USING clause can now be referenced using a prefix. For example, the following statement uses the prefix *Customer.Address* to disambiguate the column selection from a possibly similarly named column belonging to the *Supplier* table:

```
SELECT OrderNumber, Vendor, Customer.Address
   FROM Customer, Placedorder Using (Address), Supplier
    . . .
```

3. Runtime

3.1. File Descriptor Limits

VoltDB opens a file descriptor for every client connection to the database. In normal operation, this use of file descriptors is transparent to the user. However, if there are an inordinate number of concurrent client connections, or clients open and close many connections in rapid succession, it is possible for VoltDB to exceed the process limit on file descriptors. When this happens, new connections may be rejected or other disk-based activities (such as snapshotting) may be disrupted.

In environments where there are likely to be an extremely large number of connections, you should consider increasing the operating system's per-process limit on file descriptors.

3.2. Use of Resources in JAR Files

There are two ways to access additional resources in a VoltDB database. You can place the resources in the `/lib` folder where VoltDB is installed on each server in the cluster or you can include the resource in a subfolder of a JAR file you add using the sqlcmd **LOAD CLASSES** directive. Adding resources via the `/lib` directory is useful for stable resources (such as third-party software libraries) that do not require updating. Including resources (such as XML files) in the JAR file is useful for resources that may need to be updated, as a single transaction, while the database is running.

LOAD CLASSES is used primarily to load classes associated with stored procedures and user-defined functions. However, it will also load any additional resource files included in subfolders of the JAR file. You

can remove classes that are no longer needed using the **REMOVE CLASSES** directive. However, there is no explicit command for removing other resources.

Consequently, if you rename resources or move them to a different location and reload the JAR file, the database will end up having multiple copies. Over time, this could result in more and more unnecessary memory being used by the database. To remove obsolete resources, you must first reinitialize the database root directory, start a fresh database, reload the schema (including the new JAR files with only the needed resources) and then restore the data from a snapshot.

3.3. Servers with Multiple Network Interfaces

If a server has multiple network interfaces (and therefore multiple IP addresses) VoltDB will, by default, open ports on all available interfaces. You can limit the ports to an single interface in two ways:

- Specify which interface to use for internal and external ports, respectively, using the **--internalinterface** and **--externalinterface** arguments when starting the database process with the **voltdb start** command.
- For an individual port, specify the interface and port on the command line. For example **voltdb start --client=32.31.30.29:21212**.

Also, when using an IP address to reference a server with multiple interfaces in command line utilities (such as **voltadmin stop node)**, use the @SystemInformation system procedure to determine which IP address VoltDB has selected to identify the server. Otherwise, if you choose the wrong IP address, the command might fail.