Release Notes

Product	VoltDB
Version	9.3.2
Release Date	July 7, 2020

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 V8.x or earlier releases of VoltDB, please see the *V8.0 Upgrade Notes*.

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*.

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 V9.3.2 (July 7 2020)

1.1. Support for RHEL and CentOS version 8

VoltDB has completed testing and qualification of the Red Hat Enterprise and CentOS version 8 operating systems as supported platforms for running VoltDB in development and production.

1.2. Recent improvements

The following limitations in previous versions have been resolved:

Save and Restore:

- There was a problem where using the VoltDB Management Center to restore a snapshot would not work. Using the **Restore** button on the web interface would actually start two restore operations, which would then cause constraint violations. This issue has been resolved.
- Using VoltDB V6.5, taking a snapshot of a cluster with database replication (DR) enabled could create a snapshot that cannot be restored in later versions of VoltDB (V7, V8 or V9). The restore operation would fail with an error. This issue has been resolved and the current version of VoltDB can now restore the problematic snapshots.
- There was an issue where restoring a snapshot, either manually or automatically on startup, could fail if the directory contained multiple snapshots and the unique IDs for the snapshots were similar. (That is, one ID matched the starting characters of another ID, such as *test* and *testme*.) When this happened, the restore would fail with an error stating that a table had an "inconsistent transaction ID." This issue has been resolved. VoltDB now performs an exact match on the selected snapshot's unique identifier.

Monitoring:

- The New Relic latency graph data has been adjusted to improve accuracy.
- The New Relic node count report was accurate while the cluster was running, but did not "zero out" for periods while the cluster was down. This report now reports zero running nodes for those intervals when the cluster was stopped.

Export:

- There was an issue where a schema or configuration change on a running database could stall if there was an active export queue connected to a JDBC target. Closing the export connection took too long and could potentially cause a deadlock with the requested change. This issue has been resolved.
- VoltDB stores data for export queues on disk in the export overflow directory. Normally, these files are deleted shortly after the data is received by the export target. However, recent releases did not always remove queue files after they were completed. These extraneous files did not impact database performance, but did occupy unnecessary disk space. This issue has now been resolved.

Other:

- There was a memory leak, caused by changes in low-level memory management in Java 11, where clusters running on Java 11 and utilizing export and/or database replication would slowly accrue unassigned yet unreleased memory, until ultimately they could run out of memory altogether. This issue has been resolved.
- There was a rare case where, if a replica cluster in passive DR is promoted and then a node fails, other nodes could subsequently fail when the cluster tries to reassign partition leadership. The symptom for this particular failure scenario was that the failing nodes reported an error processing an invocation dispatcher request. This issue has now been resolved.
- Under certain circumstances, the **voltadmin show license** command would generate a Python stack trace when evaluating the expiration date. This issue has been resolved.

2. Release V9.3.1 (May 1, 2020)

The current release finalizes two beta features introduced in 9.2, scheduled tasks and configuration of flush intervals. In addition, this release includes the following new features and improvements.

2.1. Export Improvements

The export subsystem has been rewritten to provide significant improvements in both performance and reliability, as well as accommodate planned future enhancements. The new subsystem is available to all customers using the VoltDB Enterprise Edition. The key advantages of the new export subsystem are:

- **Better throughput** — Initial performance tests demonstrate significantly better throughput on export queues using the new subsystem over previous versions of VoltDB.
- **Adjustable thread pools** — The new subsystem let's you set the thread pool size for export as a whole or to define thread pools for individual connectors.
- **Fewer duplicate rows** — When cluster nodes fail and rejoin the cluster, the export subsystem resubmits certain rows to ensure they are delivered. The new subsystem keeps better track of the acknowledged rows and does not need to send as many duplicates to maintain the same level of durability.

2.2. Custom tasks

VoltDB 9.2 introduced tasks, which let you schedule stored procedures for execution on a repeating schedule. Tasks are now complete and ready for production use. In addition, this release extends support to include *custom tasks*, where you can dynamically adjust the procedure called, the parameters to that procedure, the interval between calls, or any combination of the three based on the results of the previous invocation. You write custom tasks as Java classes that set the attributes of the next task run and identify a callback method to invoke once the procedure completes. See the chapter on custom tasks in the *VoltDB Guide to Performance and Customization* for details. There is also a sample custom task available in the VoltDB github repository.

2.3. Thread pools

The VoltDB Enterprise Edition now lets you control the thread pools used to execute the export subsystem. A thread pool defines the number of threads used to run processes concurrently. The more threads available, the more concurrent processes and therefore the more throughput. However, more threads means more system resources are consumed. Thread pools let you tune the export subsystem to balance throughput and resource utilization against your application requirements. You specify the thread pools in the database configuration file.

You can define a default thread pool for export connectors, using the `defaultpoolsize` attribute to the `<export>` element. You can also assign specific pool sizes to individual export connectors by defining a named thread pool in the `<threadpools>` element and assigning it to a connector using the `threadpool` attribute of the `<configuration>` element. For example:

```
<export defaultpoolsize="2">
    <configuration target="log" type="file" threadpool="logpool">
        <property name="type">csv</property>
        <property name="nonce">Myapp</property>
    </configuration>
</export>

<threadpools>
    <pool name="logpool" size="5"/>
</threadpools>
```

2.4. Improved handling of non-deterministic procedures

When using K-safety, it is important that stored procedures produce deterministic results so multiple copies of a partition can run transactions concurrently with predictable results. Without consistent results in every partition, the data could diverge. To avoid this, traditionally VoltDB would stop the database if a non-deterministic result was detected (referred to as a "hash mismatch" in the log file and error messages).

Writing deterministic stored procedures is still important. But VoltDB now takes less disruptive action when divergence is detected. Rather than stopping the database every time a mismatch is detected, in most cases VoltDB now shuts down the extra copies of the partitions and runs in a single-copy reduced mode, eliminating the possibility of divergence. Of course, in the reduced mode, the cluster is no longer K-safe, which should be corrected as soon as possible. But until the offending procedure can be replaced and the cluster restarted, the database will continue to run and continue to process transactions.

2.5. Large schema

VoltDB easily handles databases with large numbers of tables. However, the schema also contains any stored procedures and auxiliary class JAR files loaded into the database. For databases using extremely large JAR files — most noticeably databases including machine learning (ML) models — it was possible to exceed the database's 50MB limit for the schema.

VoltDB has now been rewritten to accommodate schema of arbitrary size. Note, however, that limitations on the size of individual tables (that is, no more than 1,024 columns per table and not to exceed 2MB for each table) are still in effect.

2.6. Java 11 performance testing

In addition to the reliability tests that are run on an ongoing basis, Java 11 was run through a series of performance tests to assess its impact on VoltDB, with a particular focus on the different garbage collection schemes. It turns out that all of the garbage collectors available in Java 11 perform as well or better than previous versions, in several tests notably reducing the size and frequency of GC pauses.

2.7. Improved performance for schema changes

Schema changes (and configuration updates) must be treated as multi-partition transactions in VoltDB. As a result, frequent schema changes can impact ongoing throughput and latency. This is particularly noticeable for large schema. Version 9.3 provides performance improvements to reduce the time required to apply schema changes and, as a result, reduce the impact on application latency.

2.8. New Relic enhancements

The current release improves the content and structure of VoltDB performance and management data provided to the New Relic monitoring suite.

2.9. SNMP improvements

All SNMP traps are now recorded in the log file. Previously, SNMP traps were sent, but not recorded. Log entries provide additional confirmation that SNMP events are triggered.

2.10. LIMIT PARTITION ROWS deprecated

The LIMIT PARTITION ROWS clause of the CREATE TABLE statement is being deprecated. LIMIT PARTITION ROWS was designed to avoid data overflowing the available space in the database. However, it was an all-or-nothing setting: once you exceeded the limit the database was automatically paused. Transactions could not continue until you manually reduced the row count. The feature itself did nothing to resolve the situation.

LIMIT PARTITION ROWS is being deprecated and replaced by two significantly better capabilities now available in VoltDB. USING TTL lets you automatically and incrementally purge old data from tables and scheduled tasks let you build complex responsive algorithms for monitoring, managing, and resolving data volumes.

Although still available in V9.3, customers should replace any use of the LIMIT PARTITION ROWS clause at their earliest convenience, because the syntax will be removed from the product in a future major release.

2.11. Old export API deprecated

In VoltDB V8.0, the interface for writing custom export clients was updated and replaced with a new API. To accommodate old clients, the export system uses the method signatures to distinguish between the old and new interface. However, the old interface is now being deprecated and support will be removed in the next major release.

Any custom clients that use the old signature, where the onBlockStart() andonBlockCompletion() accept no arguments and processRow() expects two, should be updated to use the new interface. See the chapter on custom export and import in the *VoltDB Guide to Performance and Customization* for information on the new interface.

2.12. RabbitMQ support is deprecated

The export connector for RabbitMQ is now deprecated and will be removed in a future release.

2.13. Security Notice

The following libraries used by VoltDB have been updated to ensure the latest security and performance patches are applied:

- Commons-compress 1.19
- Dom4j 2.1.1
- Jackson 2.9.10
- Jetty 3.27
- Kafka client 0.10.2.2
- Netty 4.1.43
- Openssl 1.0.2t
- Tomcat 7.0.96

2.14. Additional improvements

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

- Previously, there was a rare case where an ad hoc query using parameters in a complex combination of CAST() and SUM() operations would give different results than the same query using constant values. For example, the following fragment with the arguments `"A"` and `1` gave different results than expected if the placeholders were replaced by the SQL constants `'A'` and `1`:

  ```
  SUM( DECODE(column1, CAST(? AS VARCHAR), CAST(? AS INTEGER), NULL))
  ```

  This issue has been resolved.

- Under certain rare conditions, if a node failed on a K-safe cluster while a node was being added, a snapshot was in progress, or the @SwapTables system procedure was executing, the command logs for that cluster could be rendered incomplete. If, after this event, the cluster stopped and restarted before the next command log snapshot was taken, the command logs could not be replayed beyond the point of the node failure. This issue has now been resolved.

- Export data optionally contains six columns of metadata, including a timestamp identifying when the row was exported. Previously, this timestamp was mistakenly set 12 years prior to the actual date. This issue has been resolved.

- There was an issue where multiple left joins, under certain circumstances and in a particular order, could result in an error where VoltDB reports that it is "unable to resolve a column index for join TVE." This issue has been resolved.

- Previously, if a GROUP BY query included an arithmetic expression combining an aggregate function and a parameter cast to a specific datatype with the CAST() function, the query failed to compile. For example:

```
SELECT a, CAST(? AS INTEGER) + COUNT(b)
   FROM TABLE c GROUP BY a;
```

  This issue has been resolved for the CAST() function. However, combining other parameterized functions with an aggregate may also cause a compilation error. The workaround, until the more general case is fixed, is to put the parameterized function in a subquery.

- Internal stress testing uncovered an extremely rare edge case where, in a K-safe cluster configured for export, if a node stops and rejoins, then during the rejoin while it is being reassigned mastership of a partition the node fails again, the cluster could crash with an error indicating a "duplicate counter collision." This condition has never been reported in the field but has now been resolved.

- There was an issue where SNMP traps would fail if the server had been running for around 24 days. After 24 days, whenever an SNMP trap was triggered, VoltDB would report that the trap failed with an illegal argument exception error. This issue has been resolved.

3. Release V9.2.2 (December 12, 2019)

3.1. Maven improvements

The packaging of VoltDB in the Maven Central Repository has been adjusted to ensure the correct artifacts are provided.

4. Release V9.2.1 (December 8, 2019)

4.1. Improved handling of SSL/TLS connections in the JDBC interface

The handling of secure export connections (using SSL/TLS) through the JDBC interface has been improved. Specifically, the requirement for a truststore when using a commercial certificate has been removed.

5. Release V9.2 (October 28, 2019)

This section describes new features in VoltDB V9.2 and known issues that have been fixed. Several new features are identified as beta software. Beta software means that the features are fully functional but have not received sufficient real-world usage or integration testing to ensure production readiness. We do not recommend using Beta features in production. However, we encourage you to try them and provide feedback on their usefulness to your business needs. Thank you.

5.1. Change to Kafka export default behavior

The Kafka `acks` property determines whether VoltDB waits for acknowledgement of receipt from the Kafka brokers. Previously, the default was set to "1" (one), but the recommendation was to set it to "all" to protect against the loss of records if the Kafka brokers fail. The default has changed to match the recommended setting.

Existing customers who use Kafka export but do not explicitly set the `acks` property may notice a slight change in export latency. The new default of "all" is the recommended setting. However, if you are willing to accept less durability on the part of the Kafka brokers, you can explicitly set the property back to "1" to replicate previous behavior.

5.2. Export from tables ready for general use

VoltDB 9.1 introduced two new beta features: the ability to export data directly from tables using the CREATE TABLE... EXPORT TO TARGET statement and the MIGRATE statement to simplify the export and deletion of records. These features are now fully supported for production use.

5.3. Scheduling stored procedures as repetitive tasks (BETA)

VoltDB 9.2 introduces a new feature, scheduled tasks. Tasks let you schedule the repeated execution of stored procedures at a set interval or using a cron-style declaration. You schedule tasks using the CREATE TASK statement. There is also a corresponding @Statistics selector, TASK, and ALTER TASK and DROP TASK statements. See the section on scheduling tasks or the reference pages describing each statement in the *Using VoltDB* manual for details.

5.4. Directed procedures for distributing transactions to every partition

One aspect of scheduling tasks is defining how they run; that is, as a single multi-partition transaction or as separate transactions on each partition. To support the latter execution model, a new type of stored procedure is being introduced, the *directed procedure*. Directed procedures are partitioned procedures in that each instance of the procedure runs on a separate partition. However, directed procedures do not have a specific partitioning value. You use the CREATE TASK statement or the Java `callAllPartitionProcedure` method to have a separate instance of the procedure run on every partition of the database. See the section on directed procedures in the *Using VoltDB* manual for details.

5.5. New Export tab and statistics in VMC

The VoltDB Management Center (VMC) has a new tab, Export, which provides enhanced statistics and graphs that help analyze the performance of export connectors.

5.6. User-defined aggregate functions (BETA)

The CREATE FUNCTION statement lets you declare a user-defined scalar function. Starting with VoltDB 9.2, you can develop and declare user-defined aggregate functions as well using the CREATE AGGREGATE FUNCTION statement. Aggregate functions process data from multiple rows and return a single aggregated value. See the chapter on user-defined functions in the *VoltDB Guide to Performance and Customization* for details.

5.7. Ability to query, filter and merge statistics using SQL (BETA)

You can now query statistics from the VoltDB @Statistics system procedure as if the results were SQL tables. You can use the querystats directive in **sqlcmd** or the new @QueryStats system procedure specifying the column names from the @Statistics results in the selection expression. For example, the following **sqlcmd** command aggregates the row count from all of the partitions for each table using the TABLE selector:

```
$ sqlcmd
1> querystats select table_name, sum(tuple_count) from statistics(table,0) group by t
```

Note that not all SQL syntax is supported as input to the querystats parser and for the initial release any syntax errors are reported in the server log but not reported to the user's console. See the descriptions of sqlcmd or the @QueryStats system procedure in the *Using VoltDB* manual for further details.

5.8. Configurable flush intervals for DR and export buffers (BETA)

You can now control how frequently the buffers for database replication (DR) and export are flushed. Normally, DR and export data is buffered until a certain amount of data is ready and then the data is sent as a batch. To avoid small amounts of data lingering in the buffer, there is a time limit after which, the data is sent even if the full batch size has not been reached. This time limit is called the "flush interval". You can now control these settings in the configuration file by setting a system-wide minimum and separate flush intervals for DR and export. See the *VoltDB Administrator's Guide* for details on configuring flush intervals.

5.9. Undocumented feature deprecated

In conjunction with the new configuration options for flushing buffers, an older undocumented attribute for setting the DR flush interval is being deprecated. The flushInterval attribute of the <dr> element is deprecated and will be removed in a future major release. In the meantime, this attribute will continue to operate and will supersede the new settings and defaults, so as not to change existing behavior for any customers who may have used this feature. However, those users are strongly encouraged to switch to the new, supported feature at their earliest convenience to avoid problems when the deprecated attribute is removed from the product in the future.

5.10. 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 an attempt to reduce the size of a running cluster using the **voltadmin resize** command would fail with a plan fragment error if the cluster had an enterprise license but without support for database replication (DR). DR is *not* required for cluster resizing to work and this issue has been resolved.

- There was issue where, in rare cases, if a schema update or configuration change failed, subsequent attempts to update the schema or configuration would also fail. This could only happen if the original update failed with an unhandled exception (such as a reference to a missing Java method), at which point subsequent update attempts reported that another update was still in progress. This issue has been resolved.

- Previously, there was the rare possibility that a node failing in a K-safe cluster could cause a multi-partition deadlock forcing the cluster to stall. This issue has been resolved.

- There was an issue with the loopback export connector. If a table or stream was declared as exporting to a target associated with the loopback connector, any attempt to alter the table definition would result in the schema change failing due to a timeout. This issue has been resolved.

- Beginning with V9.0, VoltDB supported the use of Java 11 for running servers. However, changes in Java 11 could cause VoltDB to incrementally leak memory until all available resources are exhausted. This issue has now been resolved. We strongly recommend that customers using VoltDB with Java 11 upgrade to V9.2.

6. Release V9.1.1 (September 3, 2019)

6.1. Licensing change for the VoltDB client in Maven.

The license for the VoltDB JAR file in Maven has been changed from AGPL to an MIT license.

7. Release V9.1 (August 8, 2019)

7.1. Reduce the size of a running cluster.

Previously, you could expand a running cluster to increase capacity by starting the new server with the **voltdb start --add** command. However, until now there was no way to reduce the cluster size without stopping and reconfiguring the database. With the introduction of the **voltadmin resize** command, you can now elastically shrink a running cluster as well. The **voltadmin resize** command tests the cluster to make sure it can be reduced in size, tells you which nodes will be removed, and then starts the resize process. See the section on "Removing Nodes with Elastic Scaling" in the *Using VoltDB* manual for details on reducing the size of a running VoltDB cluster.

7.2. MIGRATE TO TARGET finishes beta testing and is ready for production use.

VoltDB V9.0 introduced migration as a beta feature, making it possible to automate the data lifecycle by migrating data to an external resource before deleting it from the VoltDB database. With V9.1, the migration feature is now fully integrated, tested, and extended to round out its capabilities.

Warning

The syntax for the CREATE TABLE statement has changed from the original beta release. For consistency and to allow use without TTL, the MIGRATE TO TARGET clause now appears after the table name and before the column definitions, rather than after the USING TTL clause. If you used the MIGRATE TO TARGET clause during the beta test period, you will need to modify your schema and reload your database after installing V9.1.

See the description of the CREATE TABLE statement in the *Using VoltDB* manual for details on automating the migration of data to external targets.

7.3. New MIGRATE statement.

Using the MIGRATE TO TARGET clause with USING TTL automates the export of data to an external target before the data is deleted. In addition, you can now use the MIGRATE TO TARGET clause by itself — without the USING TTL clause — for situations where you want to manually control when the data is migrated. To do this, you can use the new MIGRATE statement to manually initiate the migration during a transaction. Note that you can also use the MIGRATE to statement for tables defined with both MIGRATE TO TARGET and USING TTL, in which case an explicit MIGRATE statement can preemptively migrate the data before the TTL value is reached. See the description of the MIGRATE statement in the *Using VoltDB* manual for details.

7.4. EXPORT TO TARGET for exporting data directly from tables (Beta Feature).

VoltDB 9.1 introduces a new feature that makes it possible to connect tables (not just streams) to export targets. If a table is declared with the EXPORT TO TARGET clause, just like a stream, any data inserted into the table is passed to the export connector. This simplifies applications that want to stream incoming data to external systems, where previously there had to be both a table and a matching stream.

Warning

CREATE TABLE... EXPORT TO TARGET is a beta feature. All functionality is believed to be complete as described. However, it is possible individual aspects of the feature may change before it is deemed production ready. For that reason export directly from tables is *not* recommended for production use at this time. However, we encourage you to try it in development and welcome feedback on its usefulness.

For example, the following table declaration allows any data inserted into the *Alerts* table to also be exported to the *Messagelog* target:

```
CREATE TABLE Alerts
  EXPORT TO TARGET Messagelog 15519
   ( eventtime TIMESTAMP,
     sender VARCHAR(16),
     message VARCHAR(128)
   );
```

You can also customize which events trigger export. Triggering events include inserts, updates (both before and after the update) and deletes. By default, only inserts trigger export; you can specify a different list of triggers with the ON clause:

```
CREATE TABLE user
  EXPORT TO TARGET userarchive
    ON INSERT, UPDATE, DELETE
   ( username VARCHAR(128),
```

```
    id BIGINT,
    lastlogin TIMESTAMP
  );
```

See the description of the CREATE TABLE statement in the *Using VoltDB* manual for more information about associating tables with export targets.

7.5. Improved handling of multi-partition transactions with large intermediate result sets

VoltDB limits each transaction to 50 MB of results. In fact, each transaction fragment is limited to 50MB. However, for multi-partition transactions, this means each partition can return up to 50MB of data to the coordinator. For large clusters with many unique partitions, it is possible for all this data to exceed the allocated Java heap for the coordinator, causing that node to fail with an out of memory error.

To avoid this situation, VoltDB now limits the amount of data each fragment can return, based on the current maximum heap size and number of partitions. By default, each partition in a multi-partition transaction is only allowed to return the lesser of 65% of the maximum heap size divided by the number of unique partitions (sitesperhost * number of nodes / k+1) or 50MB. The limit is further reduced for read-only multi-partition transactions to accommodate for the fact that multiple read-only transactions can be run simultaneously. If, at runtime, a fragment exceeds the limit, it throws an exception and the transaction rolls back gracefully.

You can adjust this per-partition multi-partition response limit by setting the environment variable MP_MAX_TOTAL_RESP_SIZE. You can either set it as the percentage of max heap to use in the calculation (by using the percent sign) or as a specific number of bytes (by using an integer value with no suffix). For example, to allow a maximum of only 50% of the allowable heap size, you can set the following environment variable before starting the server:

```
$ export MP_MAX_TOTAL_RESP_SIZE="50%"
$ voltdb start -D ~/mydb ...
```

You can set MP_MAX_TOTAL_RESP_SIZE as an environment variable or as a Java system property through VOLTDB_OPTS:

```
$ export VOLTDB_OPTS="-DMP_MAX_TOTAL_RESP_SIZE=50%"
$ voltdb start -D ~/mydb ...
```

7.6. New command to show license information.

There is a new command, **voltadmin show license**, that lists information about the current license in use by a running VoltDB cluster. You can get similar information programmatically using the @SystemInformation system procedure with the LICENSE selector.

7.7. Additional improvements

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

- There was a rare edge case where a query could return incomplete results. If an index on a table included two columns and the WHERE clause of a query on the table included both an IN clause applied to one column and a less than or equal to (<=) evaluation of the second column, fewer than expected rows were selected. This issue has been resolved.

- Previously, it was not possible to use a table alias in a DELETE statement. This issue has been resolved.

- The V9.0 Enterprise kit accidentally left out the scripts necessary for running VoltDB in a Kubernetes environment. This issue has been resolved.

- Previously, the ALTER TABLE statement did not accept the ALTER keyword before USING TTL when altering a TTL definition. This issue has been resolved and the ALTER TABLE now requires ALTER before USING TTL. For example:

  ```
  ALTER TABLE users ALTER USING TTL 5 DAYS ON COLUMN last_login;
  ```

- There was an issue related to JDBC export when using the `ignoregenerations=false` property. When generations are *not* ignored, VoltDB is supposed to create a new table name each time the schema changes. However, starting with VoltDB V8.4, JDBC export could create new tables when nodes failed or the database restarted, even if the schema did not change. This issue has been resolved.

- In a related JDBC export issue, if there is a schema change to an export stream associated with a JDBC target while the target is disabled, the export connector could fail, stopping export, the next time a record was inserted into the stream. This issue has been resolved.

- There was an issue with file export when using TSV (tab-separated value) format and exporting data containing quotations marks or backslashes. The export connector incorrectly attempted to "escape" the output, although TSV format does not support quoting or escaping. The result was incorrect output in the export file. This issue has been resolved. The export connector no longer attempts to escape special characters in the output.

- Using the VoltDB V9.0 Java client JAR with Java version 11 could result in a run-time error when deallocating a direct byte buffer. This issue did not affect the full server JAR file and has now been resolved.

- Previously, if there was an index on a table where the index uses a function that could fail (due to an invalid input value, for example), in some instances inserting a row into the table could succeed even if the update to the index failed. This issue has been resolved and now a failed index update causes the table insert to fail as well.

- VoltDB 8.2 introduced an issue that could cause a cluster to hang or crash when attempting to add nodes using elastic scaling. Under certain conditions, where a database has views, it was possible for the cluster to hang or to crash reporting a deserialization error while attempting to pass copies of the current database contents to the joining nodes. This issue has been resolved.

- There was an issue where changing the Kafka export configuration on a running database could cause the update to hang. The issue was triggered by a Kafka export configuration specifying an invalid Kafka broker. Attempting to update the configuration to change the invalid broker specification would cause the update process to hang. This issue has been resolved.

- There was an issue introduced in VoltDB 9.0 that affects certain views on streams. If the view definition included a function, it was possible for the CREATE VIEW statement to return an error stating that VoltDB could not get the row count from native storage. This issue has been resolved.

- There was an edge case related to indexes that could cause a VoltDB database to crash. If the index was defined with STARTS WITH or LIKE in the WHERE clause and a record was inserted into the table resulting in a value of a datatype other than VARCHAR being evaluated, the operation would fail, bringing down the database. This issue has been resolved and use of these expressions in indexes are now protected against illegal datatype casts.

- In a related issue, if the WHERE clause of a partial index definition includes a column reference as an argument to a function inside an expression (such as LIKE), the index could fail when the table is updated, crashing the database. This issue has been resolved.

- When fetching column values from a VoltTable using Java, you can either use a column index or the column name. Previously, using the column name for the lookup was significantly slower than using the column

index. This code has been optimized to significantly improve lookups by name, minimizing the difference between column index and name lookups.

- VoltDB V9.0 introduced several new features associated with export. In addition to further extending export functionality, a number of edge cases associated with the durability of export queues during unusual system failure scenarios were identified and resolved in the current release.

- There was an issue in recent versions of VoltDB where after restoring a database from a snapshot, the export statistics reported by the @Statistics system procedure could be inaccurate, because the export sequence number was not correctly reset. This did not affect the actual database contents being exported. However, the incorrect sequence number could also appear in the export metadata columns. This issue has been resolved.

- Under certain conditions, recent versions of VoltDB could fill the logs with repeated warnings that it "received export message *x* for partition *y*... which does not exist on this node" after a rejoin, recovery, or elastically adding a node. These messages did not indicate any real problem with the database or export and have now been rate limited and demoted to informational messages.

- There was an edge case where attempting to update the schema while a snapshot is being saved could hang the database. This issue has been resolved. Attempts to change the schema or configuration during a snapshot are no longer allowed and must wait until the snapshot is complete.

- There was an issue with elastic expansion of a cluster. If, while adding nodes "on the fly", ongoing transactions within one of the partitions being rebalanced generates a constraint violation, it could result in the rebalance operation reporting a fatal "failed to delete tuple" error, causing the server process to exit. This issue has now been resolved.

- Under certain rare error conditions, an attempt to elastically add nodes to a cluster could fail, crashing the cluster and reporting an illegal argument exception. This issue has been resolved.

- There was an issue where, if database replication (DR) encountered a corrupt file in the overflow directory, it reported an error "retrieving invocation buffer from disk." Unfortunately, it did not resolve the issue and as a result logged this error repeatedly, flooding the log file. VoltDB now identifies and addresses bad overflow files as part of its startup behavior.

- Previously, the bulkloader interface, which is used by VoltDB data utilities such as csvloader and is available through the Java API, did not correctly account for the additional data structures required by cross-data center replication (XDCR) or TTL with migrate. As a result, attempting to bulk load data into an XDCR cluster or a table with MIGRATE TO TARGET and USING TTL could cause the cluster to crash. This issue has been resolved.

8. Release V9.0 (April 11, 2019)

8.1. Updated operating system and software requirements

The requirements on the underlying operating system and software environment have been updated for VoltDB V9.0. The older CentOS and RHEL version 6.6 is no longer supported and Java 11 support has been added. In addition, support for kafka 0.8.2 has been dropped and Kafka import and export now requires version 0.10.2 or later. See the *VoltDB Administrator's Guide* for details on the platform requirements for running VoltDB clusters.

8.2. Support for Java 11

VoltDB now supports both Java 8 and Java 11.

8.3. Automated Deletion of Old Data

VoltDB 8.4 introduced a new feature, USING TTL ("time to live"), that lets you define when records expire and can be deleted. This feature simplifies application design by automatically removing old data from the database based on settings you define in the table schema. With VoltDB 9.0, this feature is extended to include the migration of deleted data to other systems for archival purposes, as described next.

8.4. New Export Capabilities

The code that supports export of data to external systems has been rewritten to provide flexibility, improve reliability, reduce system resource utilization, and support new and future product features. The new export system reinforces the durability of data queued to the export connectors across unexpected system and network failures and allows export to be extended to add new capabilities.

The first two new capabilities are:

- **ALTER STREAM** — The ability to modify an existing stream. You can use the new ALTER STREAM statement to modify the schema of the stream or the target for export without interrupting any already queued export data. See the description of ALTER STREAM in the *Using VoltDB* manual for details.

- **Automated Data Migration** — You can now automate the export of data from VoltDB database tables to other systems as part of the data aging process. For tables declared with the USING TTL clause you can now add a MIGRATE TO TARGET clause. With MIGRATE TO TARGET, data that exceeds its "time to live" is queued to the specified export connector. Once the data is exported and acknowledged by the external system, it is then deleted from the VoltDB database. See the description of the CREATE TABLE statement in the *Using VoltDB* manual for more information about the USING TTL and MIGRATE TO TARGET options.

Export now starts when the stream is defined, not when the target is defined. Previously, stream data was not queued for export until a valid export connector was configured and connected. Starting with VoltDB 9.0, data written to streams declared with the EXPORT TO TARGET clause are queued for export whether the target is configured or not. Similarly, the queued data is removed as soon as the stream itself is removed with the DROP STREAM statement.

Also, export is now an enterprise feature. The VoltDB Community Edition provides access to two streams per database, so users have access to basic export functionality. But for unlimited access to export and migration features, the Enterprise Edition is required.

8.5. "Live" Schema Updates with Database Replication

Previously, database replication (DR) required the schema of the cooperating databases to match for all DR tables. So updating the schema required a pause while all of the affected databases were updated. Starting with 9.0, this limitation has been loosened. DR continues even if the schema are different. So it is possible to update the schema without interrupting ongoing transactions.

Of course, it is not possible for VoltDB to resolve individual transactions if the schema differ. So if a DR consumer (either a replica in passive DR or an XDCR cluster in active replication) receives a binary log where the schema of the affected table(s) does not match, DR will stall and wait for the schema to be updated to match the incoming data. Therefore, care must be taken when updating the schema to ensure that no transactions that are affected by the schema change are processed during the interval when the clusters' schema do not match. See the sections on updating DR schema for passive and active DR for more information.

8.6. Simplified JSON interface

A new version of the VoltDB JSON API, 2.0, is now available. The original JSON interface provides complete information about the schema for the data being returned, including separate entries for the data, the column names, and datatypes. The 2.0 API returns a much more compact result set with each row represented by an associative array with each element consisting of the column name and value.

8.7. New @Statistics selector IDLETIME

There was an undocumented feature of the @Statistics system procedure that reported on how busy the execution queues for the individual partitions are. This data is now supported as the IDLETIME selector. See the description of the @Statistics system procedure in the *Using VoltDB* manual for details.

8.8. JVM stats automatically disabled

The I/O activity that JVM stats generates can interface with performance for applications like VoltDB, to the point where it can cause cluster nodes to disconnect. VoltDB now automatically disables JVM stats if /tmp is not defined as tmpfs (temporary in-memory storage).

8.9. Changes to how export streams are reported

Export streams (that is streams defined with the EXPORT TO TARGET clause) are no longer reported under the TABLE statistics of the @Statistics system procedure. Export streams are now reported under the EXPORT selector and tables and streams without export are reported under TABLE.

8.10. Support for multiple schema and classes files when initializing the database root

The **voltdb init** command now allows you to specify multiple files as arguments to the **--schema** and **--classes** flags. Separate multiple files with commas. You can also use the asterisk (*) as a wildcard character. For example, the following command initializes a root directory with two schema files, plus all the JAR files from one folder and another JAR file from the current working directory:

```
$ voltdb init -D ~/mydb \
     --schema=customers.sql,companies.sql \
     --classes=storedprocs/*.jar,myfunctions.jar
```

It is also possible to specify multiple schema and classes files when configuring VoltDB for use in Kubernetes. See in the readme file in the `tools/kubernetes/` subfolder where VoltDB is installed for details.

8.11. Log4J logger JOIN has been renamed to ELASTIC

The Log4J logger for elastic operations such as adding new cluster nodes on the fly has been renamed from JOIN to ELASTIC

8.12. Security Notice

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

- The Kafka import and export connectors now support the use of Kerberos authentication.
- Previously, you could not enable SSL/TLS and Kerberos authentication at the same time. This limitation has been removed.

8.13. Additional improvements

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

- There was an obscure issue where if a database cluster was restored from snapshots multiple times, the fourth restore command could stall and eventually timeout. For this unusual situation to occur the database schema must contain views, must be K safe, and nodes must have stopped and rejoined between each snapshot restore operation. This issue has been resolved.
- When rejoining a node to a running cluster , the system clock on the rejoining node must be within the limits for clock skew on the cluster, just like when starting the cluster for the first time. If not, the rejoin operation

will fail. Previously, there was an issue where if a rejoin failed due to clock skew, subsequent attempts to rejoin nodes would fail even if the clock skew had been corrected. This issue has been resolved.

- Previously, when using database replication, if you dropped a DR table using the DROP TABLE statement and then recreated a table with the same name using CREATE TABLE, the new table was treated as a DR table, even though it had not been declared as such. This issue has been resolved.

- The new export infrastructure corrects a number of issues related to the management of export queues across planned and unplanned cluster operations such as shutdowns, restarts, node failures and rejoins, and configuration changes. Although rare, these issues tended to fall into the category of export data not draining or excessive numbers of duplicate records being exported. These issue are resolved as part of the infrastructure redesign.

- There was an issue where if a graceful shutdown operation was interrupted (for example, by a CTRL-C on the **voltadmin shutdown** command), a subsequent **voltadmin shutdown --force** command would fail. This issue has been resolved.

- It was possible for frequent schema changes to interfere with a node's attempt to rejoin the cluster. When this happened the rejoin operation would time out, reporting that the cluster could not send data to the rejoining node for more than 60 seconds. This issue has been resolved.

- Previously, attempting to restore a snapshot created on a standalone cluster to a cluster configured for cross datacenter replication (XDCR) would fail with a misleading error message (indicating that the configuration could not be updated). This issue has been resolved, the snapshot is restored, and no error is reported.

- Previously, if you created a table with no columns on a cluster with command logging enabled, the cluster would crash when it attempted to truncate the command logs. This issue has been resolved.

- There was an issue introduced in VoltDB 8.2 when the USING TTL clause was added to allow you to automatically delete old records from tables based on the specified column value. Accidentally the USING TTL clause was also allowed on CREATE STREAM statements, although it has no application to streams. This issue has now been resolved.

- Under certain rare conditions, the @Quiesce system procedure could return control to the calling program before all export and DR data is successfully processed. If this occurs it was possible for an orderly shutdown (that is, **voltadmin shutdown** without the **--force** argument) to stop the cluster before all pending DR or export data was made durable. This rare race condition has now been resolved.

- In the unusual case where a subselect statement of a partitioned table did not *need* to be enclosed (that is, the outer SELECT statement did no filtering of the subselect results), the VoltDB parser could produce incorrect results. This issue has been resolved.

- There was an issue with the ALTER TABLE statement when modifying a table with an existing USING TTL clause. Altering the table to add or drop a column would result in the USING TTL clause being dropped by mistake. This issue has been resolved.

- There was an issue where complex queries with many LEFT JOIN subclauses would consume large quantities of heap space during the planning phase, ultimately running out of memory in the worst case. This issue has been resolved. Note however, that such queries may still take a long time to execute once planned and possibly exceed the query timeout limit.

- There was a rare edge case that could impact database replication (DR) and elastic expansion. When adding nodes to a running cluster, DR is stopped and restarted. Due to a race condition, there was a very rare possibility that the DR restart could generate an unexpected error such as "unable to find tuple for deletion." When this happened, DR would stop and the cluster would have to restart from scratch to reestablish replication. This issue has now been resolved.

- Previously, if a GEOGRAPHY column appeared in both an index and a view, the index was not created correctly, potentially leading to a subsequent crash when a transaction attempted to update the index. This issue has been resolved.

- Previously, attempting to start a VoltDB database using Kerberos authentication but an invalid user name would, as expected, fail. However, the resulting error messages did not identify the principal in use. The error messages have been improved to provide more information about the specific cause of the failure.

- There was an issue in the VoltDB Enterprise Edition where, if you attempted to add a node on the fly (known as elastic scaling) but did not have a license for database replication (DR), the operation would cause the cluster to crash and interfere with restarting the cluster from command logs. This issue has been resolved.

- There was an issue associated with partial indexes and the COUNT(*) function. If the index did not cover all of the rows in the table (for example `CREATE INDEX expensive product(cost) WHERE cost > 5000;`) then a query selecting COUNT(*) and using the index could give a wrong answer. 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 @Statistitcs 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.

2.5. The **voltadmin status --dr** command does not work if clusters use different client ports

The **voltadmin status --dr** command provides real-time status on the state of database replication (DR). Normally, this includes the status of the current cluster as well as other clusters in the DR environment. (For example, both the master and replica in passive DR or all clusters in XDCR.) However, if the clusters are configured to use different port numbers for the client port, VoltDB cannot reach the other clusters and the command hangs until it times out waiting for a response from the other clusters.

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. Avoid using TRUNCATE TABLE in XDCR environments.

TRUNCATE TABLE is optimized to delete all data from a table rather than deleting tuples row by row. This means that the binary log does not identify which rows are deleted. As a consequence, a TRUNCATE TABLE statement and a simultaneous write operation to the same table can produce a conflict that the XDCR clusters cannot detect or report in the conflict log.

Therefore, do not use TRUNCATE TABLE with XDCR. Instead, explicitly delete all rows with a DELETE statement and a filter. For example, `DELETE * FROM table WHERE column=column` ensures all deleted rows are identified in the binary log and any conflicts are accurately reported. Note that `DELETE FROM table` without a WHERE clause is *not* sufficient, since its execution plan is optimized to equate to TRUNCATE TABLE.

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.

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.