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.

What's New in VoltDB V9.0

VoltDB 9.0 is a major release incorporating features from recent point releases plus new capabilities. The major new features in V9.0 include:

• **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.

• **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. This automated process not only automates the archiving of old data it ensures that the data stays in the VoltDB database until it is confirmed as received by the export target. 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.

Two important aspects of the new export infrastructure are the effect on the overall export process:

• 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. This meant data written to a stream might be dropped rather than queued for export if the connector was not configured correctly. 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.
This change makes the queuing of export data much more reliable, easier to understand, and easier to control. Data is queued for export as soon as the stream is defined and purged as soon as the stream is dropped. The new ALTER STREAM statement further lets you modify the stream definition without having to clear any existing export queues.

- 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. (See Special Considerations for more information about upgrading community databases that use export.)

- "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.

- 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.

  The 1.0 API is useful if you do not know what data is returned and want to deconstruct the results in detail. The 2.0 API is more useful for rapidly fetching and using known results. Both versions of the API accept the same parameters, as described in the section on using the JSON API in the Using VoltDB manual. So the following calls return the same data except at different levels of detail.

  
  http://myserver:8080/api/1.0/?Procedure=@Statistics&Parameters=["TABLE",0]
  http://myserver:8080/api/2.0/?Procedure=@Statistics&Parameters=["TABLE",0]

- Support for Java 11 — VoltDB now supports both Java 8 and Java 11.

Special Considerations for Existing Customers

Many of the new features and capabilities in VoltDB V9.0 do not impact existing applications. However, there are a few changes that may require action for users upgrading from earlier versions. Existing customers should take note of the following changes:

- Change to how stream data is queued for export

  Previously, if you defined a stream with the EXPORT TO TARGET clause but no matching target was configured, any data inserted into the stream was dropped. No data was queued until the specified target was both configured and successfully connected. With VoltDB 9.0 export queuing has been simplified: data is queued as soon as the stream is declared and the queue is deleted as soon as the stream is dropped.

  This means that if you declare a stream and it is being written to, but you do not configure the associated target, the data inserted into the stream will be queued in the export_overflow directory, consuming disk space that would not have been used in earlier versions of the product.

- Change to the reporting of streams in @Statistics
In VoltDB 8.4, a new @Statistics selector, EXPORT, was introduced to provide improved visibility and more detail concerning the export lifecycle. At that time, export streams continued to be reported under the TABLE selector, so as not to disrupt existing scripts or procedures users might have that rely on that information. With VoltDB 9.0, export streams have been removed from the TABLE selector results. Now the TABLE selector only reports on tables and streams that are not associated with an export target. Information on streams declared with the EXPORT TO TARGET clause is provided under the EXPORT selector.

• Managing export queues during outages

In most cases, VoltDB manages the export queues even in unusual cases where nodes go down in a K-safe cluster. If at any time the node managing an export partition finds a gap in the export queue (due to the current server having been down when that data was written to the stream), the system queries the other servers to find one with the missing data and export continues. In the rare case where VoltDB cannot find the missing records in any of the current server export queues, the export connector for that queue will stop, waiting for a server that has the data.

Even if servers stop and rejoin frequently, the data will eventually be found on a rejoining node. However, in the unusual case that, say, failed nodes are replaced by new, initialized servers, it is possible that the gap in the queue cannot be resolved. Previously, VoltDB would eventually (once the cluster was at a full complement of servers) skip the missing data and restart the connector at the next available export record. Starting with VoltDB 9.0, export will not skip over gaps automatically. The queue will stop and warnings will be logged to the console and reported via SNMP. You must issue the `voltadmin export release` command to resume processing of the export connector at the next available record.

• Limits on streams in Community Edition

Export is now an Enterprise Edition feature and the Community Edition is limited to two streams per database. This means if you try to restore a database from a previous version with more than two streams using the VoltDB 9.0 Community Edition, the restore will fail. If this happens when upgrading, you can initialize a new database root, load the schema without the additional streams, then manually restore the data.

• Support for CentOS and RHEL 6 removed

The officially supported platforms for VoltDB have been updated. CentOS and RHEL version 6 are no longer officially supported. The current list of supported platforms include CentOS and RHEL version 7.0 or later, Ubuntu versions 14.04, 16.04, and 18.04, and Macintosh OS X 10.9 or later.

• Support for export and import to Kafka 0.8.2 removed

The Kafka import and export connectors now require Kafka version 0.10.2 or later.

• Logging name change from JOIN to ELASTIC

The log4J logger reporting on elastic changes to VoltDB clusters has been renamed from JOIN to ELASTIC.

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.
Release 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

In addition to the features outlined in "What's New", the following updates have been made to VoltDB since the last release.

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

1.1. 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.

1.2. 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.

1.3. 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).

1.4. 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.

1.5. 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.

1.6. 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

1.7. 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.

### 1.8. Additional improvements

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

- There was an 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 @Statistics 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. 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.

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.