SYBASE®
An SAP® Company

Administration: In-Memory Row-Level Versioning

SAP Sybase IQ 16.0

DOCUMENT ID: DC01840-01-1600-01
LAST REVISED: February 2013

Sybase, Inc., One Sybase Drive, Dublin, CA 94568.

Contents

About In-Memory Row-Level Versioning

With in-memory row-level versioning (RLV) for SAP® Sybase® IQ, more than one user can modify the same table concurrently, users can wait for transaction locks instead of having to retry, and a hybrid storage model optimizes data write-access, without sacrificing read-access performance.

- **Concurrent Table Writes –** in earlier versions of SAP Sybase IQ, incoming stream data had to be batched, and run serially, which caused conversion overhead, and latency in data availability. With in-memory row-level versioning, the IQ server allows concurrent, low-latency modifications to tables. This means that multiple connections can modify the same table, as long as they are adding or modifying different rows
- **Blocking and Locking –** with earlier versions of SAP Sybase IQ, a transaction would lock a table, blocking all other connections from writing to the table while the transaction was open. These other connections would have to implement retry logic through a form of looping, which affected performance. In-memory row-level versioning supports multi-version concurrency control (MVCC), for version management at the row level. Connections can wait for locks (on either the table or a single row), eliminating the need for retry.
- **Hybrid Storage –** in-memory row-level versioning introduces the row-level versioning (RLV) store to SAP Sybase IQ. The new RLV store combines with the existing on-disk IQ main store to provide a hybrid storage mechanism that combines the extreme performance and low-latency of the in-memory store with the robust high-performance and scalability of on-disk storage. Immediate data modifications (load table / insert / update / delete) occur within the write-optimized RLV store. The RLV store is periodically merged into the read-optimized IQ main store through asynchronous data transfer. Thus, most data in an IQ table can be accessed via indexes, and provides expected IQ query performance.

In-Memory Row-Level Versioning Terminology

The definitions of specific terms are helpful when describing in-memory row-level versioning.

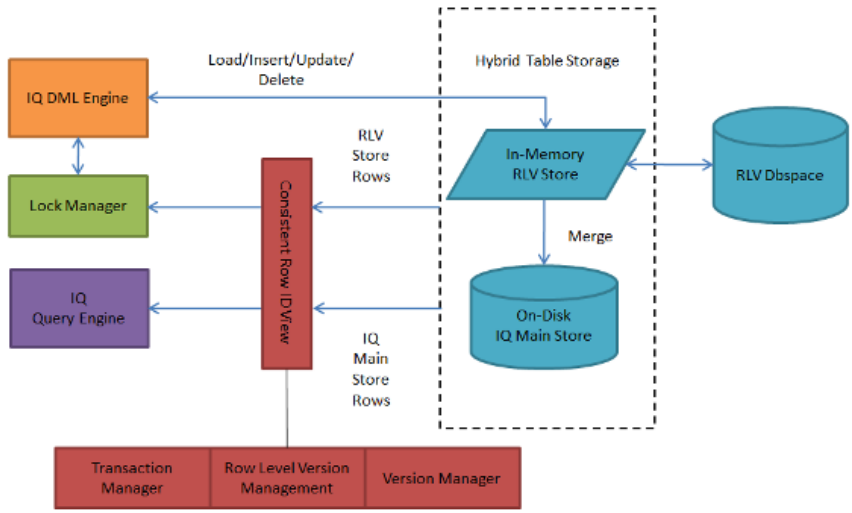
- **Data Definition Language (DDL) –** refers to SQL commands which create or modify the schema of a table, for example **CREATE TABLE**, **ALTER TABLE**, **DROP TABLE**.
- **Data Modification Language (DML) –** refers to SQL commands which create or modify the data in a table, for example **INSERT**, **LOAD**, **UPDATE**, **DELETE** and **TRUNCATE**.
- **Multi-Version Concurrency Control (MVCC) –** a concurrency control mechanism providing stable read-only versions so that writers do not block readers of the same table.
- **Row-Level Versioning (RLV) –** an MVCC versioning technique which logically versions table rows for a write transaction, and allows concurrent writes to different rows of the same table. Each time a writer commits a transaction, the server creates a new version of

the updated row, resulting in a different row-level version being created. An RLV-enabled table is one in which row-level versioning is permitted.

- **RLV Store –** the write-optimized, in-memory store that works with the existing, read-optimized, on-disk IQ main-store to make in-memory row-level versioning possible.
- **RLV Store Merge –** the periodic asynchronous transfer of per-table in-memory data from the RLV store to the IQ main store. The merge occurs automatically, but can also be triggered manually. Only committed transactions are merged.
- **Snapshot –** the technique of establishing a stable version of an object, determined at the start of a transaction.
- **Table-Level Versioning (TLV) –** an MVCC versioning technique which logically versions the entire table for a write transaction, and does not allow concurrent writers of the same table. Each time a writer commits a transaction, the server creates a new version of the entire table, resulting in a different table-level version being created. A TLV table is one which row-level versioning is not enabled.

In-Memory Row-Level Versioning Architecture

The RLV and IQ main stores together provide hybrid table storage which enables row-level snapshot isolation for tables with concurrent transactions. The server tracks the data location when querying and manipulating the data, but logically, the data is in one (hybrid) store.



- **Version Manager –** The version manager works with the table-level version to ensure that there is a consistent row ID view.

- **Transaction Manager –** Only RLV transactions are written to the RLV store. A single TLV write transaction on a table will block all RLV write attempts to that table until the TLV transaction is terminated.
- **Transaction Log –** The RLV store transaction log (persistence log) tracks and makes durable all new and modified data stored in-memory.

In-Memory Row-Level Versioning Restrictions

The in-memory row-level versioning feature has some restrictions including aggregation preference, table type, and data type.

Feature	Restriction
Server Types	Only single-server configurations are supported: tables cannot be enabled for row-level versioning in a multiplex configuration.
Table Types	Only SAP Sybase IQ base tables are supported in the RLV store: catalog, temporary and global temporary tables are not supported.
Data Types	LONG BINARY (LOB) and LONG VARCHAR (CLOB) data types are not supported on RLV-enabled tables. • You cannot add a LOB column to an RLV-enabled table. • You cannot enable RLV storage on a TLV table with a LOB column.
Index Types	TEXT indexes and WORD indexes are not supported on RLV-enabled tables.
Constraint Types	Foreign key constraints are not supported on RLV-enabled tables or across a combination of RLV-enabled and TLV tables. Referential integrity is not supported. • You cannot add a foreign key constraint to an RLV-enabled table. • You cannot enable RLV storage on a TLV table that has a foreign key constraint defined. • You cannot enable RLV storage on a TLV table if another table has a foreign key constraint referencing this table. • When altering a TLV table, you cannot add a foreign key reference to an RLV table.
SQL Statements	• The **LOCK TABLE** statement is not supported on RLV-enabled tables. • The **BEGIN PARALLEL IQ...END PARALLEL IQ** statement results in an error if one of its **CREATE INDEX** statements specifies an RLV-enabled table. • The **TRUNCATE** statement will result in an error if the PARTITION or SUBPARTITION clause specifies an RLV-enabled table.
Database Options	`REVERT_TO_V15_OPTIMIZER='ON'` results in an error if your query specifies an RLV-enabled table.

See also

- *Row ID Stability* on page 15

The Row-Level Versioning (RLV) Store

Row-level versioning allows multiple transactions to modify different rows of the same table concurrently. The in-memory, write-optimized RLV store supplements the SAP Sybase IQ main store in providing concurrent read- and write-access to IQ base tables.

The IQ server provides the ability to configure, on a connection basis, single- or multiple-writer concurrent access to IQ base tables. Multiple connections can modify the same table, as long as they are adding or modifying different rows. Only tables which have RLV storage enabled are eligible for multiple-writer concurrent access.

The RLV store provides:

- Low latency writes with minimal index and compression overhead, resulting in efficient in-memory read/write operations
- Efficient concurrent updates with row versioning and minimal lock contention
- High performance column scans which feature in-memory read operations

The RLV store records incremental DML from an IQ table and automatically merges these changes into the IQ main store. The combined RLV and main stores are write-optimized for efficient DML and low latency data access, yet still have read-optimized on-disk query performance across the majority of data.

Whereas traditional, table-level versioned (TLV) tables reside only in the IQ main column store, row-level versioning (RLV) enabled tables reside in both the IQ main and RLV stores.

The RLV Store Merge

The RLV store is an autonomous self-managing, in-memory store that merges into the IQ main store automatically (either periodically or when thresholds are reached).

The RLV merge process moves rows from committed row-level transactions stored in the RLV store to the IQ main store on a per table basis. The merged data becomes part of a new table level version of the same table, in the IQ main store.

A blocking merge waits for all write-transactions on the table to complete, then blocks all write operations until the merge is complete. A non-blocking merge allows existing and new write-operations to continue on the table while the merge is active.

The merge process can be triggered in one of three ways:

1. Manually, by using the sp_iqmergerlvtable stored procedure. For example

   ```
   call sp_iqmergerlvstore('BLOCKING', 'my_table', 'DBA')
   ```

   which runs a blocking merge on the RLV-enabled table "DBA.my_table".

2. Automatically, when a DDL or table-level DML statement is executed on an RLV-enabled table. For example

   ```
   alter table DBA.my_table add c2 int
   ```

   which forces a blocking merge on the RLV-enabled table "DBA.my_table", and then adds the new column "c2" to the table.
3. Automatically, when a resource threshold is reached, such as the maximum number of rows in the RLV store for a given table, or the maximum amount of memory allowed for the RLV in memory store.

RLV merge failures are not typical. The only legitimate RLV merge failures would be due to the IQ main store running out of dbspace or experiencing storage errors (hardware or permission issues). If the RLV merge were to fail, the transaction would roll back. All changes to the IQ main store TLV version would be undone. This is the same process used when the server executes DML through a transaction.

See also

- *Merge RLV Store into IQ Main Store* on page 31

RLV Merge Phases

The RLV merge executes in five phases.

- **Begin –** the server performs an auto-commit, begins a new transaction, and prepares for the merge.
- **Delete –** the server applies all delete transactions recorded in the IQ main store. Rows deleted from the RLV store portion of the table are not merged since they are deleted in-memory. Any committed IQ main store rows deleted prior to RLV enablement are simply recorded as deleted by the RLV store. The merge will apply the deletes against the new IQ main store table level version being created by the merge.
- **Update –** the server applies all update transactions recorded the IQ main store. Rows updated from the IQ main store table will store the new values in the RLV store. The updated new values will be applied to the IQ main store during the merge.
- **Insert –** the server transfers the new RLV store inserted rows into the IQ main store table
- **End –** the server completes the merge and commits a new TLV version (or rolls back).

The merge is logged in the database **IQMSG** file and the **SYSIQMERGERLVHISTORY** system view.

RLV Store Partitioning

All IQ base tables, whether unpartitioned, range-partitioned, hash-partitioned or range-hash partitioned, can be enabled for row-level versioning, however the in-memory RLV store portion is not partitioned.

The actual partitioning occurs during the merge from the RLV store to the (partitioned) IQ main store. At this time, table rows are placed in the appropriate range-partitioned dbspace.

Every partitioned table has a corresponding pseudo-column in the RLV store which contains partition information. Query engine uses these pseudo-columns to support partitioning semantics.

RLV Store Persistence and Durability

Transactions in the RLV store are fully durable. Committed transactions are guaranteed to be recovered in the event of a system crash.

Data in the RLV store is not persisted to disk, but remains in-memory until it is merged into the IQ main store.

The performance of the I/O system supporting the RLV dbspace is a major factor in the ingestion and commit performance of the RLV store. Sustained write throughput is important for high volume data ingestion.Write latency is important for commit performance. Random write performance is important when simultaneously ingesting data into multiple RLV enabled tables.

A commit request from an application is not acknowledged until the entire transaction, including the commit state, has been persisted to stable storage.

Each RLV enabled table has its own logical persistence log. The space for these logs comes exclusively from the RLV dbspace. Log space is consumed during transactions and is freed by merge. At the end of a merge, the log for a table is truncated back to the oldest open transaction at the beginning of the merge. A long-running open transaction that spans multiple merges has the potential to prevent log space from being truncated. You can monitor the amount of RLV log space used for a table with the **sp_iqtablesize** stored procedure.

Because it is not possible for uncommitted data to be in the database at the start of recovery, recovery is a logical, transaction-oriented process rather than a physical, page-oriented process. RLV recovery is concerned with inserting committed data instead of physically modifying pages.

See also

- *Merge RLV Store into IQ Main Store* on page 31
- *Manage Memory for the RLV Store* on page 57

RLV Store Backup/Restore

Backing up and restoring the RLV store is part of the normal backup process.

The RLV dbspace is a persistence dbspace. The internal structure is the same as for other supported dbspaces. As a result, it is fully integrated into the SAP Sybase IQ dbspace management system.

Various system procedures are able to report meta data from the RLV store persistence dbspace. In addition, it is fully integrated into the existing Sybase IQ backup and restore architecture. In backup restore, the dbspace is part of the normal backup, and is restored. In

normal recovery, the catalog store and IQ main store are recovered first, and then the server recovers the in-memory portion (the RLV-enabled tables). RLV recovery takes place against any RLV enabled table that had committed data in memory.

The RLV Dbspace

The RLV store requires an RLV dbspace for persistence logging.

The RLV dbspace houses persistence logs for RLV-enabled tables.

RLV storage cannot be enabled on any table unless the RLV dbspace is configured.

See also

- *Configure the RLV Dbspace* on page 20

RLV Dbspace Restrictions

There is only one RLV dbspace per database, but it can have multiple dbfiles.

The RLV dbspace files may not be dropped or made read-only while there are RLV-enabled tables in the database. To drop RLV dbspace files, all RLV enabled-tables must be dropped or RLV-disabled first.

Unlike other dbspaces, the RLV dbspace does not allow striping.

RLV Store Persistence Log

The asynchronous write-ahead persistence log for the RLV store is separate from the database persistence log. It tracks and makes durable all new and modified data stored the RLV store

The RLV persistence log contains a disk-based copy of the contents of the RLV store. It is stored in a compressed format to balance disk utilization and runtime recovery performance. The log is organized per-table and is stored exclusively in the RLV dbspace. It uses efficient, asynchronous I/O to minimize table modification overhead, and efficient, parallel processing for fast recovery on restart. The log is used to restore the in-memory RLV store on server restart after clean or abnormal shutdown.

Log Space Usage

RLV log space usage is reported by the **sp_iqtablesize** stored procedure.

In particular, two columns of **sp_iqtablesize** relate to the RLV log.

Column Name	Description
RlvLogPages	Number of IQ pages being used to store RLV logs for this table
RlvLogKBytes	Number of kilobytes being used to store RLV logs for this table

See also

- *sp_iqtablesize Procedure* on page 132

Data and Transaction Management

All tables in SAP Sybase IQ are stored in the IQ main store. This on-disk storage is table-level versioned (TLV). When a table is enabled for row-level versioning (RLV), the on-disk version remains fixed, and the in-memory RLV store is activated. As table transactions are performed, the server manages many row-level versions which are committed and stored in-memory. Over time, this data from the RLV store is merged with the IQ main store.

Data Modification Language (DML)

The disk I/O per row in the RLV store is substantially less than in the IQ main store, thus enabling smaller-granularity DML.

Row-level snapshot isolation allows concurrent DML, whereas table-level snapshot isolation does not.

With an insert transaction (including bulk loads), new rows are added to the RLV store. Data is only inserted into the IQ main store by the merge operation. There is never a lock on an insert because the RLV store is an append-only store. Insert has the advantage of leveraging the fully bulk parallel load engine.

With a delete transaction, there is a lock. When a row is deleted from an RLV-enabled table, the data is not removed from the RLV store during commit. Instead, all deleted rows that are committed remain in-memory until a merge, but are invisible to query operations.

An update transaction is implemented as a delete plus an insert.

Merging the RLV store with the IQ main store does not free up memory used by open transactions, this memory is held until the transaction is closed.

See also

- *Table-Level DML Locking Considerations* on page 14

DML Best Practices

Best practices will improve performance with batch loads and large updates.

For batch loads:

- Perform small- or medium-sized concurrent loads through the RLV store.
- Do not perform excessively large loads through the RLV store, because loads exceeding maximum RLV memory may fail.

Even if a very large load does not exceed the maximum RLV memory (and does not fail), merges of other committed RLV data may be triggered during the load. This may impact load performance.

For large updates:

- Perform large updates on RLV-enabled tables with a table-level versioned connection.
- Do not perform very large updates through the RLV store because updates copy on-disk rows to the RLV in-memory store for modification.

Constraints

To maintain consistent behavior between the IQ main and RLV stores, constraints are checked during load, insert and update statements.

Primary key and unique indexes are supported, but require additional memory and processing during the modification of RLV-enabled tables. To optimize performance while modifying RLV-enabled table, keep these index types to a minimum. Referential integrity and foreign key constraints are not supported on RLV-enabled tables.

Detected constraint violations cause DML commands to fail. The load command provides an IGNORE CONSTRAINT clause which allows a specified number of constraint violations to be reported and ignored before the load command is aborted.

Table Constraints

RLV store loads use the same table constraint evaluation mechanism as IQ main store loads, and have identical performance characteristics with regards to table constraint evaluation.

Table constraints allow expressions to be evaluated against each inserted row. For example

```
col1 >= col2, col1 NOT NULL
```

These expressions may only involve columns within the row being loaded.

See also

- *CREATE TABLE Statement* on page 164
- *ALTER TABLE Statement* on page 146

Unique and Primary Key Constraints

Primary key and unique constraints are supported for RLV store loads, inserts and updates, in a specific way.

Because the RLV store does not maintain secondary indexes, enforcement requires a separate unique value checking process. Furthermore, no exclusive table lock is taken for RLV table writes, so new row values can be inserted, updated or deleted and subsequently committed by other connections while the current connection performs its operations.

Although RLV modifications committed after a transaction snapshot are not visible to the current connection, before any new values can be committed, any values added or removed due to those modifications are considered during unique and primary key checking, to avoid constraint violation.

Unique and primary key value checking has two phases.

1. The new values are checked against values in the RLV store for uniqueness. As values are inserted into the RLV store, each one is checked for uniqueness against committed and

uncommitted values in the RLV store (a "first updater wins" methodology). Uncommitted deletes are not considered for unique checking unless they are part of the current transaction.

2. The new values are checked against IQ main store values. Before inserting into the RLV store, each value is checked against the IQ main store index for uniqueness (excluding any deleted and updated IQ main store rows visible to the transaction in the RLV store).

Referential Integrity Constraints

Referential integrity constraints are not supported in the RLV store.

In the IQ main store, referential constraint enforcement checks that no foreign tables involved in a referential integrity constraint are currently being modified, by taking a write-intent lock on the foreign table for the duration of the transaction. This referential concurrency checking is handled by the global IQ in-memory catalog. However, taking a global write-intent lock for referential-integrity enforcement is not supported for RLV store tables.

Referential integrity relationships on RLV-enabled tables would also require global table locks from RLV transactions that accessed table-level versioned foreign tables.

For these reasons RI constraints are not supported.

Transaction Management

The transaction manager includes support for transaction access to the RLV store. The transaction manager works the version manager to provide table- and row-level versioning support.

The transaction manager also works with the RLV merge to enable blocking and non-blocking merges. Blocking merges do not allow transactions to access an RLV-enabled table until the merge commits or rollback. Non-blocking merges allow transactions to exist beyond the start and end of a merge, so that these transactions still see a consistent state of the database.

After RLV-enabled tables are committed, the data resides in the RLV dbspace until a merge. (The RLV dbspace is an on-disk representation of what is in memory). The merge won't free up memory from open transactions (that is, those which have not yet ended).

When a TLV connection accesses an RLV enabled table, table-level read-write access will force a merge of the RLV store. The TLV transaction will hold a table-level write-lock, which will block RLV and other TLV connections from writing to the table for the duration of the transaction. Reads from other connections are not affected. An example of this scenario is a nightly load. The application performs a large load in the night, when the system is not being accessed. A bulk load into the IQ main portion is more efficient than loading into the RLV store and then manually triggering a merge.

See also

- *Merge RLV Store into IQ Main Store* on page 31

Transactions

A transaction accesses the rows visible in its snapshot. .

Transactions can be simple, multi-table or composite:

- Simple transactions involve a single RLV-enabled table and the RLV store only. The log records exist entirely in a single log stream.
- Multi-table transactions involve multiple RLV-enabled tables, and contain records in multiple log streams.
- Composite transactions involve traditional IQ tables (with table-level versioning), SAP® Sybase SQL Anywhere tables (in the IQ catalog store), and RLV-enabled tables coexisting in the same transaction. The server commits the TLV tables, followed by the SQL Anywhere tables and finally, the RLV-enabled tables.

Lock Management

Row locks, and their prerequisite write-intent locks, ensure consistency between concurrent transactions at a row level. For example, a transaction can lock a particular row to prevent another transaction from changing it. Transactions place write-intent locks on the table itself, and on the table rows they intend to modify, to prevent conflicts from both competing row-level and table-level snapshot-versioned transactions.

An RLV-enabled table uses row locks for updated and deleted rows; and write-intent locks when accessed by read-write transactions with row-level snapshot versioning. The IQ write table lock is used when accessed by read-write transactions with table-level snapshot versioning.

DDL changes to an RLV-enabled table require an exclusive table-level lock; the writing connection has an exclusive lock on the table. DML changes to an RLV-enabled table first take out a write-intent lock to block table-level versioned transactions from locking the table, and then take out a row-level lock to prevent other row-level versioned transactions from writing to those rows.

- **Row lock –** a table row write lock allowing the holder to write to any column of a locked row. Only one holder of this lock can exist at a time. A write-intent lock is a prerequisite; you must hold a write-intent lock before the lock manager grants you a row lock.
- **Write-intent lock –** a table write-intent lock indicates you intend to write to a table row in the future. A write-intent lock can be held by multiple requesters.

  The write-intent lock conflicts with table write locks and table exclusive locks. This conflict prevents a table-level snapshot-versioned transaction from writing to the table or performing a DDL operation until the lock manager releases all write-intent locks on the table.

See also

- *Schema Locks* on page 12
- *Row Locks* on page 12

- *Write-Intent Locks* on page 13
- *Row-Level DDL Locking Considerations* on page 14
- *Table-Level DML Locking Considerations* on page 14
- *Monitor Locks and Deadlocks* on page 39

Schema Locks

Whereas a table-lock in the IQ main store places a lock on all the rows in the table, a schema lock places a lock on the table's schema.

Schema locks ensure that transactions using a table are not affected by schema changes initiated by other connections. For example, a schema lock prevents an **ALTER TABLE** statement from dropping a column from a table when that table is being read by an open cursor on another connection. Readers and writers both take schema locks.

A schema lock can be modified for exclusive access. The exclusive access can only be granted when there are no other schema locks present. This means that there are no other readers or writers to the table. All DDL statements will take an exclusive schema lock prior to being allowed to execute. Only one connection can acquire an exclusive schema lock on a table at any time—all other attempts to lock the table's schema will either block or fail with an error.

See also

- *Row Locks* on page 12
- *Write-Intent Locks* on page 13
- *Table-Level DML Locking Considerations* on page 14
- *Row-Level DDL Locking Considerations* on page 14

Row Locks

A row lock is a table-row write lock that allows the holding transaction to write to any column of a locked row. Only one holder of this lock can exist at a time. A *write-intent* lock is a prerequisite; the transaction must hold a write-intent lock before the lock manager grants it a row lock.

A table-row write lock allows the holding transaction to write to any column of locked row. This lock cannot be granted without the requesting transaction first holding the write intent lock. Row write locks are exclusive locks; only one transaction can hold a write lock on a row at any time. Once a transaction acquires a write lock, requests to lock the row by other transactions are denied.

Row locks exist only during row deletions. The RLV store is an append-only store, meaning that every write action results in a new row appended to the store. **INSERT** statements append a new row to the store, as do **UPDATE** statements. The RLV store considers an **UPDATE** to be a **DELETE** followed by an **INSERT**. Before a row is deleted, either in the context of a **DELETE** or **UPDATE** statement, the database takes out a row-level lock.

See also

- *Schema Locks* on page 12
- *Write-Intent Locks* on page 13
- *Table-Level DML Locking Considerations* on page 14
- *Row-Level DDL Locking Considerations* on page 14
- *Tutorial: Monitoring Row-Level Locks* on page 42

Write-Intent Locks

A write-intent lock is a table write lock that grants the transaction permission to write to a table row in the future. A write-intent lock can be held by multiple requesting connections.

A write intent lock always exists when the RLV-enabled portion of the table exists in memory. You can view details of the write intent lock using the **sp_iqlocks** stored procedure.

The write-intent lock conflicts with table write locks and table exclusive locks. This conflict prevents a table-level snapshot-versioned transaction from writing to the table or performing a DDL operation until the lock manager releases all write-intent locks on the table. In a situation where both table-level snapshot-versioned transactions and row-level snapshot-versioned transaction connections write to a table, write-intent locks provide synchronization. Consider this scenario:

Connection	Action
Row-level snapshot-versioned transaction A	• Executes query writing to multiple rows of table_1. • Lock manager creates a write-intent lock for table_1. • Lock manager creates multiple local write-intent locks for row-level DML updates. Lock manager creates row-level locks.
Table-level snapshot-versioned transaction B	Attempts to write to table_1. Transaction B blocked by write intent lock.
Row-level snapshot-versioned transaction A	Commits transaction A. Table changes are merged from the RLV store to the IQ main store. Write-intent locks released.
Table-level snapshot-versioned transaction B	Proceeds with write to table_1.

See also

- *Schema Locks* on page 12
- *Row Locks* on page 12
- *Table-Level DML Locking Considerations* on page 14
- *Row-Level DDL Locking Considerations* on page 14
- *Tutorial: Monitoring Write-Intent Locks* on page 39

Table-Level DML Locking Considerations

If a table is enabled for RLV storage, you can still issue table-level snapshot versioning DML statements against it. The DML engine recognizes table-level and row-level versioned transactions, and manages the locks accordingly.

When a transaction issues a table-level snapshot versioning DML statement against an RLV-enabled table:

- In-memory data is merged into the IQ main store portion of the table
- The write-intent lock releases
- The table-level DML statement proceeds

Once the transaction completes, the next connection issuing a DML statement in a row-level snapshot-versioned transaction causes the RLV portion of the table to be recreated in memory. Until the current transaction issuing the table-level snapshot versioning DML statement completes, row-level snapshot versioned transactions either block, or fail.

Note: If a table is enabled for RLV storage, the **LOCK TABLE** statement cannot be used.

See also

- *Schema Locks* on page 12
- *Row Locks* on page 12
- *Write-Intent Locks* on page 13
- *Row-Level DDL Locking Considerations* on page 14
- *LOCK TABLE Statement* on page 211

Row-Level DDL Locking Considerations

Data Definition Language (DDL) changes (for example, **CREATE INDEX**, **DROP INDEX**, and **ALTER TABLE** ADD, ALTER, or DROP) to an RLV-enabled table require an exclusive table-level lock. For DDL events, the locking behavior for an RLV-enabled table is the same as for an IQ main store table: the writing connection has an exclusive lock on the table. When **BLOCKING** is set to ON, all competing DML and DDL transactions against the table are blocked until the DDL changes are committed. When **BLOCKING** is set to OFF, the competing transaction will immediately fail the lock request.

See also

- *Schema Locks* on page 12
- *Row Locks* on page 12
- *Write-Intent Locks* on page 13
- *Table-Level DML Locking Considerations* on page 14

Version Management

Transactions involving RLV-enabled tables create row-level transaction snapshot versions. A row-level snapshot allows the transaction to commit a version of the table row, rather than a version of the entire table.

Row-level versioning permits concurrent DML changes to the table. (An update to one row may not block another connection's update to another row).

- A row-level operation on an RLV-enabled table creates a row-level version.
- A table-level operation (such as DDL and table-level DML) on an RLV-enabled table creates a table-level version.
- A transaction not involving an RLV-enabled table creates a table-level version.

Note: You must enable row-level transaction snapshot versioning before you can write data to the RLV store.

Open transactions hold versions which are accessible to that transaction. If a transaction is long-running, the server will hold the memory and disk space associated with the RLV and TLV versions until that transaction terminates. Even after a merge, memory for RLV versions will not be freed until the transactions with snapshots referencing these versions are terminated.

See also

- *Configure Snapshot Versioning* on page 27
- *Specifying Snapshot Versioning* on page 28
- *Restricting Snapshot Versioning* on page 29

Row ID Stability

The row ID of rows in an RLV-enabled table may change.

In TLV tables which reside completely on the IQ main store, when a row is inserted, its row ID is stable for the lifespan of that row. Once the row is deleted, the row ID is available to be assigned to a newly inserted row.

In RLV-enabled tables, an inserted row is assigned a temporary RLV row ID.

The temporary row ID assigned to a row in the RLV store is guaranteed to be stable only for the duration of each transaction to which it is visible. Its row ID becomes permanent only after the row has been both committed and merged into the IQ main store

Query and the RLV Store

Queries on RLV-enabled tables scan data from both the on-disk IQ main and in-memory RLV stores.

Because of lack of specialized indexes, some query optimizations are not applied for queries on RLV store:

- IQ indexes (HG, LF) do not cover the RLV store.
- There is limited parallelism for predicate evaluation.

In general, this has a small performance impact, and relates to the proportion of data in the RLV and IQ main stores.

The RLV portion of the query relies on fast in-memory column scans. In some partitioned hash-join cases, in-memory indexes are created on-demand to enhance query performance.

See also

- *The Row-Level Versioning (RLV) Store* on page 4
- *Row ID Stability* on page 15

Impact of Row-Level Versioning on Queries

Specific situations will have performance impacts, including queries with multiple invariant predicates, indexes, and row IDs.

- **Order of Predicate Execution –** when a query has multiple invariant predicates, the order in which the predicates are executed on the data in the RLV store depends on the usefulness of the predicates. The order of predicate execution in the RLV store could differ from the order of execution in the IQ main store.
- **Lack of Indexes –** unlike the IQ main store, the RLV store does not have the capability to execute predicates using the best possible index when present. Therefore, there could be a difference in the query plans, depending on whether the query is executed on data that is in the IQ main store only, or on the same data in the RLV store. Furthermore, some predicates that require special support from specific indexes may result in an error when run on RLV-enabled tables. For example, contains predicate requires support from TEXT or WORD indexes, neither of which are supported in RLV-enabled tables. In order to avoid the performance degradation due to lack of indexes on the RLV side, SAP Sybase IQ may create an ad hoc hash index when these queries are detected. The query plan will indicate that the store is using the hash index.
- **No DQP support –** querying RLV-enabled tables is supported in simplex configurations only.

Note: Do not make use of a specific row ID when querying. If you make use of a specific row ID (for doing a join) and select a row ID from another table, the row ID may not remain consistent.

See also

- *QP Output Details for RLV Tables* on page 17

QP Output Details for RLV Tables

The Optimization Note, Condition 1 RLV Index and Output 1 RLV Index fields of query plan output details give insight as to how querying works with row-level versioning.

In the query plan output details for the leaf node:

- **Optimization Note –** indicates if an on-demand hash index was created for the data in the RLV store.
- **Condition 1 RLV Index –** describes which index was used for predicate execution.
- **Output 1 RLV Index –** lists the RLV indexes present on a column.

Query that Uses Flat FP Index for Execution

```
SELECT * from test_char
WHERE c1>1;
```

If this query were run, in the query plan output details, the Condition 1 RLV Index field would display `FP`, indicating that the query used a Flat FP index for predicate execution. The Output 1 RLV Index field would also display `FP`, indicating that only an FP index was present on c1.

Query that Creates Hash Index During Execution

```
SELECT * from R1KD100
WHERE R1KD100.c1 in (SELECT R100D100.c1 FROM R100D100 WHERE
R1KD100.c1)
      = R100D100.c1)
```

If this query were run, in the query plan output details, the Optimization Note field would indicate that a hash index was created for RLV data. The Output 1 RLV Index field would display `FP, Hash`, indicating that there were two indexes present on c1.

Query that Uses Previously-Created Hash Index for Execution

```
SELECT * from R100D100
        WHERE c1 = 1;
```

If this query were run after the previous query, and the hash index still existed on R100D100.c1, in the query plan output details, the Condition 1 RLV Index field would display `Hash`. (Whenever a hash index exists on a column, it is always preferred over a Flat FP Index for predicate execution on the that column). The Output 1 RLV Index field would display `FP, Hash`, indicating that there were two indexes present on c1.

Query that Creates Partitioned Hash Index During Execution

```
SELECT * from hash1, hash2
WHERE hash1.c1 = hash2.c1;
// hash1 and hash2 are hash partitioned on c1
```

Suppose tables hash1 and hash2 are equi-partitioned tables (both hash partitioned on column c1), and the join condition is on column c1. If this query were run, the IQ query optimizer would create a partitioned index on both the tables. In the query plan output details for each leaf, the Optimization Note field would indicate that a partitioned-index was created for the RLV store data.

See also

- *Impact of Row-Level Versioning on Queries* on page 16

Configure In-Memory Row-Level Versioning

Row-level versioning allows multiple transactions to modify different rows of the same table concurrently. In order to utilize the RLV store, several setup steps must first be configured..

1. *Configuration Prerequisites*

   Before configuring in-memory row level versioning, in addition to the base configuration for SAP Sybase IQ you will need RAM for the RLV store, and a high-performance disk for the RLV dbspace.
2. *Configure RLV Memory*

   RLV memory is configured in addition to existing IQ cache memory. The host machine must have enough memory for both the RLV memory and the IQ main cache, which are independent memory pools.
3. *Configure the RLV Dbspace*

   The RLV dbspace houses persistence logs of tables enabled for in-memory RLV storage.
4. *Configure RLV Storage on Tables*

   A table enabled for RLV storage has two parts: one residing on the IQ main store, and the other residing on the dedicated in-memory RLV Column Store.
5. *Configure Snapshot Versioning*

   Snapshot versioning describes the type of versioning access that the database server uses for tables: table-level snapshot versioning, or row-level snapshot versioning. Row-level versioning allows concurrent writer access and row-level locking for RLV-enabled tables.

Configuration Prerequisites

Before configuring in-memory row level versioning, in addition to the base configuration for SAP Sybase IQ you will need RAM for the RLV store, and a high-performance disk for the RLV dbspace.

Balance the memory size with the merge frequency. Although a smaller in-memory allocation may be sufficient if you anticipate frequent transactions, consider that the smaller in-memory size may affect the frequency of RLV store merges, thus impacting overall performance.

- For batch loads use at least twice the maximum single-transaction load data size, per table.
- For continuous/OLTP loads, the size depends on incoming data and transaction rate.

For the RLV Log dbspace, disk requirements include:

- A minimum of two times the RLV in-memory size.
- High random access write performance, enterprise SSD and HBA attached disk array.

Configure RLV Memory

RLV memory is configured in addition to existing IQ cache memory. The host machine must have enough memory for both the RLV memory and the IQ main cache, which are independent memory pools.

The maximum size of the RLV store, for all tables, is specified with a server option, which can be set with:

- The server startup switch **-iqrlvmem** <max MB>
- The run time **sa_server_option** system procedure option 'rlv_memory_mb', <max MB>

A size of 2 GB or greater is recommended. If the server switch is not specified, the value defaults to 2 GB.

See also

- *Manage Memory for the RLV Store* on page 57
- *-iqrlvmem start_iq Server Option* on page 141
- *sa_server_option System Procedure* on page 84

Configure the RLV Dbspace

The RLV dbspace houses persistence logs of tables enabled for in-memory RLV storage.

A portion of the RLV dbspace must be reserved for memory used by data structures during critical operations. The size of this portion can be set using the **RV_RESERVED_DBSPACE_MB** database option.

See also

- *Configure RLV Storage on Tables* on page 25
- *The RLV Dbspace* on page 7
- *RV_RESERVED_DBSPACE_MB Option* on page 77

Creating the RLV Dbspace

In order to use RLV storage with tables, you first need to create the RLV dbspace with a minimum of one dbfile added to it.

Prerequisites

- SAP Sybase IQ server has a simplex database.

Task

Use the statement **CREATE DBSPACE** *<dbspace name>* **IQ RLV STORE**

```
CREATE DBSPACE d1  using file f1  '/dev/raw/raw1/f1.iq' size 1000 IQ
RLV STORE
```

Disk striping is always OFF for the RLV dbspace. Creating the RLV dbspace with STRIPING ON | OFF or using the STRIPESIZEKB option is not supported.

See also

- *CREATE DBSPACE Statement* on page 161

Permitted ALTER DBSPACE Syntax for RLV Store

You can use the **ALTER DBSPACE** statement to configure the RLV dbspace. The statement usage may differ from other dbspaces.

RLV Dbspace State	ALTER Type	Permitted for RLV Dbspace
Online	ALTER DBSPACE OFFLINE	Yes, if the dbspace is read-only
	ALTER DBSPACE ONLINE	No
	ALTER DBSPACE READONLY	Yes, if the dbspace is read-write and no RLV-enabled objects exist
	ALTER DBSPACE READWRITE	Yes, if the dbspace is read-only
	ALTER STRIPING or STRIPESIZEKB	No
	RENAME DBSPACE	Yes
	ADD FILE	Yes
	DROP FILE	Yes, if the file is not in use
	ALTER FILE READONLY	No
	ALTER FILE READWRITE	No
	ALTER FILE SIZE	Yes, if the dbspace is read-write
	ALTER FILE RENAME LOGICAL NAME	Yes
	ALTER FILE RENAME PATH	No
Offline	ALTER DBSPACE OFFLINE	No
	ALTER DBSPACE ONLINE	Yes
	ALTER DBSPACE READONLY	No
	ALTER DBSPACE READWRITE	No

RLV Dbspace State	ALTER Type	Permitted for RLV Dbspace
	ALTER STRIPING or STRIPESIZEKB	No
	RENAME DBSPACE	Yes
	ADD FILE	No
	DROP FILE	Yes. File is, by definition, empty
	ALTER FILE READONLY	No
	ALTER FILE READWRITE	No
	ALTER FILE SIZE	No
	ALTER FILE RENAME LOGICAL NAME	Yes
	ALTER FILE RENAME PATH	Yes
Read-only	ALTER DBSPACE OFFLINE	Yes
	ALTER DBSPACE ONLINE	Yes
	ALTER DBSPACE READONLY	No
	ALTER DBSPACE READWRITE	Yes
	ALTER STRIPING or STRIPESIZEKB	No
	RENAME DBSPACE	Yes
	ADD FILE	Yes
	DROP FILE	Yes
	ALTER FILE READONLY	No
	ALTER FILE READWRITE	No
	ALTER FILE SIZE	No
	ALTER FILE RENAME LOGICAL NAME	Yes
	ALTER FILE RENAME PATH	No
Read-write	ALTER DBSPACE OFFLINE	No
	ALTER DBSPACE ONLINE	No
	ALTER DBSPACE READONLY	Yes, if no RLV-enabled objects exist
	ALTER DBSPACE READWRITE	No

RLV Dbspace State	ALTER Type	Permitted for RLV Dbspace
	ALTER STRIPING or STRIPESIZEKB	No
	RENAME DBSPACE	Yes, if no RLV-enabled objects exist
	ADD FILE	Yes, if no RLV-enabled objects exist
	DROP FILE	Yes, if the file is not in use
	ALTER FILE READONLY	No
	ALTER FILE READWRITE	No
	ALTER FILE SIZE	Yes, but when decreasing file size, truncated space must be empty
	ALTER FILE RENAME LOGICAL NAME	Yes
	ALTER FILE RENAME PATH	No

See also

- *ALTER DBSPACE Statement* on page 142

Altering the RLV Store Dbspace to Read-only

You can use the **ALTER DBSPACE** statement to set the RLV store to read-only. However, the RLV store is, by definition, a read-write store. Therefore only make the dbspace read-only if necessary (as in the case of dropping the dbspace).

Prerequisites

- SAP Sybase IQ server has a simplex database.
- A single RLV store dbspace exists on the database.

Task

Alter the dbspace to be read-only, using the statement **ALTER DBSPACE** *<dbspacename>* **READONLY**.

```
ALTER DBSPACE d1 READONLY
```

See also

- *Permitted ALTER DBSPACE Syntax for RLV Store* on page 21
- *ALTER DBSPACE Statement* on page 142
- *Dropping the RLV Dbspace* on page 24

Dropping the RLV Dbspace

The RLV dbspace cannot be dropped unless it is empty. Dropping the RLV dbspace will mean that you can no longer create RLV-enabled tables, or alter existing tables to enable RLV storage.

Prerequisites

- SAP Sybase IQ server has a simplex database.
- A single RLV dbspace exists on the database.
- The RLV dbspace is read-only.

Task

Unlike an IQ main store dbspace, you cannot relocate objects (transaction logs) resident on the RLV dbspace in order to empty it. Instead, in order to empty the RLV dbspace, you must ensure that there are no RLV-enabled table in the store.

Note: Dropping the RLV dbspace will mean that you can no longer create RLV-enabled tables, or modify existing tables to be RLV-enabled.

1. Check for RLV-enabled tables (for example, **SYSIQTAB**).
   a) If RLV-enabled tables exist, disable RLV storage, or drop the tables.

```
SELECT table_id, table_name
FROM SYSIQTAB
WHERE table_id IN (SELECT table_id FROM SYSIQTAB WHERE is_rlv='T')
```

2. Alter the dbspace to be read-only, using the statement **ALTER DBSPACE** *<dbspacename>* **READONLY**.

```
ALTER DBSPACE d1 READONLY
```

3. Drop the dbspace using the command **DROP DBSPACE** *<dbspacename>*.

```
DROP DBSPACE d1
```

See also

- *Permitted ALTER DBSPACE Syntax for RLV Store* on page 21

Adding a File to the RLV Dbspace

You may wish to add a file to the RLV dbspace for extra capacity in storing RLV transaction logs.

Prerequisites

- SAP Sybase IQ server has a simplex database.
- A single RLV dbspace exists on the database, and is online.

- If the dbspace is read-write, no RLV-enabled objects exist.

Task

Use the statement **ALTER DBSPACE** *<dbspace name>* **ADD FILE** *<filename>*

```
ALTER DBSPACE d1 ADD FILE 'rlv2.iq'
```

Because of the nature of in-memory RLV storage, you cannot specify files as being READONLY.

See also

- *Permitted ALTER DBSPACE Syntax for RLV Store* on page 21

Dropping a File from the RLV Dbspace

You can remove a file from the RLV dbspace, provided that it is not the only file, and it is not in use.

Prerequisites

- SAP Sybase IQ server has a simplex database.
- A single RLV dbspace exists on the database.
- The RLV dbspace is read-only, or the file is not in use if the dbspace is read-write.

Task

Use the statement **ALTER DBSPACE** *<dbspace name>* **DROP FILE** *<dbspace filename>*

```
ALTER DBSPACE d1 DROP FILE rlv2
```

See also

- *Permitted ALTER DBSPACE Syntax for RLV Store* on page 21

Configure RLV Storage on Tables

A table enabled for RLV storage has two parts: one residing on the IQ main store, and the other residing on the dedicated in-memory RLV Column Store.

You can have tables enabled for row-level versioning storage coexisting on the same database with other tables having table-level versioning storage.

Row-level versioning of global and local temporary tables is not supported.

See also

- *Configure the RLV Dbspace* on page 20
- *Configure Snapshot Versioning* on page 27

Creating a New Table with RLV Storage Settings

When creating a new base table, you can specifically enable or disable RLV storage.

Prerequisites

- SAP Sybase IQ server has a simplex database.
- The RLV dbspace exists with at least one dbfile.
- You must have RESOURCE authority.
- You must have CREATE permission on the RLV store dbspace.

Task

Use the statement **CREATE TABLE** *<table-name>* **{ENABLE | DISABLE} RLV STORE**

- **<table-name> –** the name of the table for which the RLV storage is to be enabled.
- **CREATE TABLE [owner] table-name IN dbspace-name –** not allowed, unless you specifically use the exact name of the RLV store dbspace.

Note: If the ENABLE | DISABLE RLV STORE clause is omitted, RLV storage settings for the table will default to the value of the global database option **BASE_TABLES_IN_RLV_STORE**.

See also

- *BASE_TABLES_IN_RLV_STORE Option* on page 71

Enabling or Disabling RLV Settings for an Existing Table

You can alter an existing base table so that it is enabled or disabled for RLV storage.

Prerequisites

- SAP Sybase IQ server has a simplex database.
- The RLV dbspace exists with at least one dbfile.
- You must have RESOURCE authority.
- You must be the table owner or have ALTER permission on the table.
- You must have CREATE permission on the RLV store dbspace.

Task

Use the statement **ALTER TABLE** *<table-name>* **{ ENABLE | DISABLE} RLV STORE**

- **<table-name> –** the name of table for which the RLV storage is to be enabled or disabled.
- **{ ALTER column-name MOVE { PARTITION (partition-name TO new-dbspace-name) | TO new-dbspace-name } } | MOVE PARTITION partition-name TO new-**

dbspace-name | MOVE TO new-dbspace-name | MOVE METADATA TO new-dbspace-name – not supported syntax when altering an RLV-enabled table.

Configuring Default Storage for Tables

When the {ENABLE | DISABLE} RLV STORE clause of the **CREATE TABLE** statement is not present, the **BASE_TABLES_IN_RLV_STORE** option setting is used to determine RLV storage. Thus, you can allow existing **CREATE TABLE** statements to run without modifying scripts to enable RLV storage.

Prerequisites

- SAP Sybase IQ server has a simplex database.
- The RLV dbspace exists with at least one dbfile.

Task

Enable the option at the database level.

```
Set option PUBLIC.BASE_TABLES_IN_RLV_STORE = 'ON'
```

Note: The settings on an individual table made by using the **CREATE TABLE** statement override this option. The default setting is OFF. If set to ON, any **CREATE TABLE** statement without the ENABLE | DISABLE RLV clause results in the creation of an RLV-enabled table.

See also

- *BASE_TABLES_IN_RLV_STORE Option* on page 71

Configure Snapshot Versioning

Snapshot versioning describes the type of versioning access that the database server uses for tables: table-level snapshot versioning, or row-level snapshot versioning. Row-level versioning allows concurrent writer access and row-level locking for RLV-enabled tables.

Table-level versioning is the default, and provides versioning based on the entire table.

Row-level versioning provides versioning at the row level against a fixed table version. Versioning occurs at transaction start time. Once the snapshot version has been defined for a started transaction, you cannot change it until the transaction is complete.

See also

- *Configure RLV Storage on Tables* on page 25

Row-Level Snapshot Versioning

Row-level snapshot versioning applies only to tables enabled for in-memory RLV storage. Row-level snapshot versioning allows multiple writers to make concurrent DML changes to a table, but never to the same rows at the same time.

Row-level snapshot versioning locks the table at the row level using row locks. A row lock provides a write lock for a table row, meaning the transaction gets blocked, or fails, depending on the **BLOCKING** and **BLOCKING_TIMEOUT** option settings. If **BLOCKING** is ON, the transaction blocks. If **BLOCKING** is OFF, the transaction fails immediately with an `ALREADY LOCKED` SQL exception.

Transaction blocking enables row-level snapshot versioning to write to different rows of the same table simultaneously. Depending on the **BLOCKING** and **BLOCKING_TIMEOUT** option settings, row-lock contention results either in an error, or a retry to obtain the lock if it is released within the specified timeout period. When a transaction configured for table-level versioning attempts to write to a table with a row locked by a row-level versioned transaction, the table-level transaction either fails with an error, or blocks and retries if the lock is released within the specified timeout period.

DDL changes to a table (**CREATE**, **DROP**, and **ALTER**), however, lock the table at the table level.

See also

- *Manage Blocking in the RLV Store* on page 51
- *BLOCKING Option* on page 71
- *BLOCKING_TIMEOUT Option* on page 72

Specifying Snapshot Versioning

Use the **SNAPSHOT_VERSIONING** option to set the snapshot versioning type to either `Row-level` or `Table-Level`. You can set the option at the database (PUBLIC) level, connection level (TEMPORARY) or user level. To use the in-memory RLV store, enable row-level snapshot versioning for your transactions. For simultaneous updates to different rows of the same table, each transaction or connection must also enable row-level snapshot versioning.

Prerequisites

- If setting to Row-level, the RLV store dbspace exists with at least one dbfile.
- If setting to Row-level, the table is RLV-enabled.
- Requires the SET ANY PUBLIC OPTION system privilege to set this option for PUBLIC or for other user or role.

Task

Once the snapshot versioning property has been set for a transaction, it remains the same until the transaction commits.

1. Determine the scope of the **SET OPTION** command to set the option as a database-wide option, connection-level option, or user-level option:
   - **SET OPTION public.SNAPSHOT_VERSIONING...**
   - **SET TEMPORARY OPTION SNAPSHOT_VERSIONING...**
   - **SET OPTION** *username*.**SNAPSHOT_VERSIONING...**
2. Specify the snapshot versioning type.

Level	Option
Row-level	Row-level snapshot versioning. Required for in-memory RLV storage. Row-level snapshot versioning allows multiple writers to make concurrent DML changes to a table, but never to the same rows at the same time.
Table-level	Classic (backward-compatible) SAP Sybase IQ versioning behavior. Takes snapshots at the table-level. Multiple writers cannot make concurrent DML changes to a table.

```
SET TEMPORARY OPTION Snapshot_Versioning = 'Row-level';

CREATE TABLE rv_locks(c1 int, c2 int, c3 int);

INSERT INTO rv_locks VALUES (1,1,1);
INSERT INTO rv_locks VALUES (2,2,2);
INSERT INTO rv_locks VALUES (3,3,3);
INSERT INTO rv_locks VALUES (4,4,4);
COMMIT;
```

See also

- *SNAPSHOT_ VERSIONING Option* on page 77

Restricting Snapshot Versioning

Use the **ALLOW_SNAPSHOT_VERSIONING** database option to restrict the versioning allowed in the database to `Table-level`, `Row-level`, or `any` (no restriction). **ALLOW_SNAPSHOT_VERSIONING** can be set at the database (PUBLIC) level only. You cannot set it at the connection level, or user level.

Prerequisites

- If setting to Row-level, the SAP Sybase IQ server has a simplex database.
- Requires the SET ANY SYSTEM OPTION system privilege.

Task

1. Use the **SET OPTION** command to set the option at the PUBLIC level. **SET TEMPORARY OPTION...** and **SET OPTION** *<username>*... are not allowed.
2. Restrict the type of versioning allowed in the database to either Table-level only, or Row-level only.

Restriction	Option
Table-level only	**set PUBLIC option ALLOW_SNAPSHOT_VERSIONING = 'Table-level'**
Row-level only	**set PUBLIC option ALLOW_SNAPSHOT_VERSIONING = 'Row-level'**

Setting the option to 'Table-level' prevents RLV access to any tables in the database, effectively turning off the RLV store.

To remove the versioning restriction from the database, set the option to 'any.'

Restriction	Option
No restriction	**set PUBLIC option ALLOW_SNAPSHOT_VERSIONING = 'any'**

See also

- *SNAPSHOT_ VERSIONING Option* on page 77

Merge RLV Store into IQ Main Store

Over time, or when thresholds are triggered, the data committed in-memory is merged to the IQ main store, through an asynchronous data transfer process, the RLV store merge.

The merge of RLV-enabled table data into the IQ main store uses one of two approaches:

1. Non-blocking (background) merge: Transactions normally still read and write to the RLV store while a non-blocking merge is in progress. There is a possible impact on runtime operations due to the merge's use of system resources. The non-blocking merge briefly locks write-access to the table being merged. This may result in a brief pause, but will not cause transactions to fail.
   - The non-blocking merge is initiated by the server as needed. The server merge evaluator executes a merge periodically, at a configurable interval. The merge can also be triggered by automated merge thresholds.
   - It is also possible to execute a non-blocking merge manually. However, this is not recommended.
2. Blocking (foreground) merge: The table containing the data to be merged into the IQ main store is locked while the merge takes place. The RLV merge operates as a transaction which creates a new version of the table. The visibility of this table version follows normal table versioning rules.
   - Certain events trigger a blocking merge to execute automatically.
   - In the event that you need to execute a manual merge, on most occasions you would run a blocking merge.

See also

- *RLV Store Persistence and Durability* on page 6
- *Manage Memory for the RLV Store* on page 57
- *The RLV Store Merge* on page 4

Automated Foreground Merge

The IQ server will automatically perform a blocking (foreground) merge when necessary.

Certain actions require an automatic blocking merge before the action can commence:

- Accessing a table at table-level snapshot isolation (rather than row-level snapshot isolation)
- RLV DML approaching RLV memory limit
- Using DDL commands such as **CREATE** or **ALTER**.

Commands which require an immediate automatic blocking merge may experience a pause while the merge is executed, before the command proceeds.

See also

- *Setting Merge Trigger Thresholds* on page 32
- *Running a Manual Merge* on page 33
- *Viewing Merge History* on page 34
- *Logged Merge Phases in IQMSG File* on page 35
- *Post-Merge Table Fragments* on page 36

Setting Merge Trigger Thresholds

The IQ server periodically evaluates the adjustable set of merge thresholds for each RLV-enabled table, and automatically performs background (non-blocking) merges as needed. The threshold settings can be changed, however care should be exercised before doing so because of possible impacts on performance.

Prerequisites

- SAP Sybase IQ server has a simplex database.
- RLV storage is configured.

Task

When a merge occurs, there may be a performance impact in the server because of the resources used by the merge.

1. (Optional) Change the interval between IQ server merge evaluations.

   The merge evaluator examines the merge parameters of each row-level versioning (RLV) enabled table against configured threshold values to determine whether a non-blocking (background) merge of the RLV table to IQ main stores should occur. You can change the interval between activation times of the merge evaluator. If the merge evaluator is already active or if a merge is already running when the interval ends, the merge evaluator waits for the next interval to re-initiate.

Database Option	Description	Default
RV_AUTO_MERGE_EVAL_INTERVAL	Length of time between merge evaluations	15 minutes

2. (Optional) Change the table thresholds.

   When a table threshold is exceeded, a merge will be triggered for that specific table.

Table Threshold	Description	Default
RV_MERGE_TABLE_NUMROWS	Number of committed RLV rows	10 million
RV_MERGE_TABLE_MEMPERCENT	Percentage of total RLV memory size	100/ number of RLV tables

3. (Optional) Change the node threshold.

   When a node threshold is exceeded, the merge condition evaluator will determine which table(s) to merge. If multiple tables must be merged to satisfy the node threshold, parallel merges will be triggered for each table to be merged.

Node Threshold	Description	Default
RV_MERGE_NODE_MEMSIZE	Total RLV memory size	75% of configured size

See also

- *Automated Foreground Merge* on page 31
- *Running a Manual Merge* on page 33
- *Viewing Merge History* on page 34
- *Logged Merge Phases in IQMSG File* on page 35
- *Post-Merge Table Fragments* on page 36
- *Manage Memory for the RLV Store* on page 57
- *RV_AUTO_MERGE_EVAL_INTERVAL Option* on page 74
- *RV_MERGE_TABLE_NUMROWS Option* on page 76
- *RV_MERGE_TABLE_MEMPERCENT Option* on page 75
- *RV_MERGE_NODE_MEMSIZE Option* on page 75

Running a Manual Merge

The RLV store is self-managing and performs automatic merges, as needed. However, in a few cases, you may wish to trigger a merge manually.

Prerequisites

- SAP Sybase IQ server has a simplex database.
- RLV storage is configured.

Task

Some reasons you might consider a manual merge include

- When you are preparing to perform a table-level load (to ensure that the DDL or load command performance is not impacted by an automatic merge). You would run a blocking merge in this instance.
- To free memory before a DML operation, such as a bulk load, on a table which is known to affect a large data volume (to ensure that an automatic merge does not run concurrently with the DML command). You would run a blocking merge in this instance.
- Prior to shutdown, in order to reduce startup time (otherwise, RLV recovery will be performed, which may be time-consuming).
- If the automated merge period is set to a large time, and system resources are approaching threshold limits. You would run a non-blocking merge in this instance, but would also modify the times and thresholds so that you would not need to monitor as closely in the future.

To manually run an RLV merge, use the SQL stored procedure **sp_iqmergerlvstore [[*merge_type*], [*table_name*], [*table_owner*]]**.

- If a table name is not specified, all the active data (from all RLV-enabled tables) in the RLV store will be merged into the IQ main store.
- Merge-type can be BLOCKING | NON-BLOCKING .
- After performing the merge, the stored procedure will automatically commit the merge transaction.

See also

- *Automated Foreground Merge* on page 31
- *Setting Merge Trigger Thresholds* on page 32
- *Viewing Merge History* on page 34
- *Logged Merge Phases in IQMSG File* on page 35
- *Post-Merge Table Fragments* on page 36
- *sp_iqmergerlvstore Procedure* on page 112

Viewing Merge History

View a list of merges which took place on a specific table, including information on merge date, merge type, and merge statistics.

Prerequisites

- SAP Sybase IQ server has a simplex database.
- RLV storage is configured.

Task

1. Use the SYSIQMERGERLVHISTORY view to see when the merge occurred, what data was merged, and why.
2. Look at the IQMSG file to see logged merge failures.

See also

- *Automated Foreground Merge* on page 31
- *Setting Merge Trigger Thresholds* on page 32
- *Running a Manual Merge* on page 33
- *Logged Merge Phases in IQMSG File* on page 35
- *Post-Merge Table Fragments* on page 36
- *SYSIQRLVMERGEHISTORY System View* on page 219

Logged Merge Phases in IQMSG File

The server logs RLV merge activity in the database IQMSG file.

The five phases of the merge are logged with a line suffixed by **Mrg**.

Log Suffix	Merge Phase
Mrg B <table_id> <merge_type> [merge_host]	Begin
Mrg D	Delete
Mrg U	Update
Mrg I	Insert
Mrg E	End

For the merge Begin phase, <merge_type> is one of:

- **D –** automatic DDL blocking merge
- **M –** automatic DML blocking merge
- **N –** non-blocking merge

[merge_host] is an optional connection handle for an internal connection used to run the merge. This occurs if you execute a DDL merge. The connection running the DDL will use another internal server connection to run the merge. When the merge completes, the connection calling the DDL connection resumes.

An example IQMSG entry for a merge is:

```
I. 01/20 17:25:27. 0000000022 Txn 179 0 179
I. 01/20 17:25:27. 0000000021 Mrg  B 775 D 0000000017
```

```
I. 01/20 17:26:28. 0000000021 Mrg D
I. 01/20 17:26:29.  0000000021 Mrg U
I. 01/20 17:26:34. 0000000021 Mrg I
I. 01/20 17:26:39.  0000000021 Mrg E
I. 01/20 17:26:39. 0000000021 Cmt 188
```

The prefix to each line is a timestamp and the connection ID logging the request. In this case, connection 21 is an internal connection running the merge. In the first line, `Txn 179 0 179` is the standard message denoting a begin transaction with ID 179. The third line shows that a merge has begun against table ID 775, that it is a DDL blocking merge and that the connection that launched the merge is 17. The last line shows that connection 21 commits, and the commit ID is 188.

See also

- *Automated Foreground Merge* on page 31
- *Setting Merge Trigger Thresholds* on page 32
- *Running a Manual Merge* on page 33
- *Viewing Merge History* on page 34
- *Post-Merge Table Fragments* on page 36

Post-Merge Table Fragments

In a NON-BLOCKING merge, the most recently committed data from the RLV store is written to the IQ main store to create a new table-level version of the RLV-enabled table. This new table-level version combines the previous table-level version plus the changes from the RLV store (the in-memory changes from committed transactions). Uncommitted transactions may reference snapshot versions on the pre-merged RLV store. These fragments are held in-memory until the transactions terminate.

The merge operation itself has an impact on the RLV store:

- When the merge starts, a new RLV store instance is created.
- From then on, all data changes go to the new instance.
- The committed changes of the original instance of the RLV store are merged into the IQ main store
- Then the merge ends

An active merge operation uses two RLV store disjoint instances. The original RLV store instance contains all committed changes done before the beginning of the merge; the new RLV store contains all changes done after the beginning of the merge. Because of any open transactions residing in the original instance (transactions begun, but not committed before the merge), the original instance is preserved until all transactions have been committed.

For BLOCKING merges the scenario is much simpler. There are no uncommitted transactions referencing snapshot versions on the pre-merged RLV store, nor are there any data changes

happening while the merge is running. Hence, when a BLOCKING merge completes, there is only ever a single, empty RLV table fragment.

See also

- *Automated Foreground Merge* on page 31
- *Setting Merge Trigger Thresholds* on page 32
- *Running a Manual Merge* on page 33
- *Viewing Merge History* on page 34
- *Logged Merge Phases in IQMSG File* on page 35

Monitor Locks and Deadlocks

Use the **sp_iqlocks** stored procedure to display details about row-locks, write-intent locks, and deadlocks in the database.

See also

- *sp_iqlocks Procedure* on page 109

Tutorial: Monitoring Write-Intent Locks

In this tutorial, create RLV-enabled tables, execute a transaction, and use the **sp_iqlocks** stored procedure to report on schema-level locks and write-intent locks in the database. Then use the **sp_iqconnection** and **sa_conn_info** stored procedures to view the internal connection controlling the write-intent lock.

Prerequisites

- SAP Sybase IQ server has a simplex database.
- RLV storage is configured.

Task

Tip: You can monitor locks using Sybase Control Center. For more information, see the Sybase Control Center for SAP Sybase IQ online help in SCC or at *http://sybooks.sybase.com/sybooks/sybooks.xhtml?prodID=10680*.

1. Create RLV-enabled tables `rv_locks` and `rv_locks2`, and configure table-level snapshot versioning.

```
SET TEMPORARY OPTION SNAPSHOT_VERSIONING = 'Table-level';

CREATE TABLE rv_locks(c1 INT, c2 INT, c3 INT);

INSERT INTO rv_locks VALUES (1,1,1);
INSERT INTO rv_locks VALUES (2,2,2);
INSERT INTO rv_locks VALUES (3,3,3);
INSERT INTO rv_locks VALUES (4,4,4);
COMMIT;

CREATE TABLE rv_locks2(c1 int, c2 int, c3 int);

INSERT INTO rv_locks2 VALUES (1,1,1);
INSERT INTO rv_locks2 VALUES (2,2,2);
INSERT INTO rv_locks2 VALUES (3,3,3);
INSERT INTO rv_locks2 VALUES (4,4,4);
COMMIT;
```

```
ALTER TABLE rv_locks ENABLE RLV STORE;
ALTER TABLE rv_locks2 ENABLE RLV STORE;
```

2. Enable connection blocking and set the blocking timeout threshold:

```
SET TEMPORARY OPTION BLOCKING = 'ON';
SET TEMPORARY OPTION BLOCKING_TIMEOUT = '0';
```

3. Use the **sp_iqlocks** stored procedure to view the current set of database locks. At this point, no locks are returned.

```
sp_iqlocks
```

   The absence of a write-intent lock for the RLV-enabled table indicates that the in-memory RLV portion of the table has yet to be created.

4. Set the snapshot versioning property of the transaction to row-level.

```
SET TEMPORARY OPTION SNAPSHOT_VERSIONING = 'row-level';
```

5. Write to the table.

```
INSERT INTO rv_locks VALUES (5,5,5);
```

   Writing to, or querying, an RLV-enabled table creates the RLV-enabled portion of the table in memory, on demand.

6. Re-execute **sp_iqlocks**.

```
sp_iqlocks
```

   This time, the procedure returns a write-intent lock.

```
conn_name,conn_id,user_id,table_type,creator,table_name,index_id,
lock_class,lock_duration,lock_type,row_identifier,row_range
'SQL_DBC_13cd6038',
3,'DBA','BASE','DBA','rv_locks',,'Schema','Transaction','Shared',
,
'RLV_CONN_T775',
1000000407,'','BASE','DBA','rv_locks',,'Table','Transaction','Int
ent',,
```

   Connection ID 100000407 has a write-intent lock on the `rv_locks` table. The lock type is set to `Intent`, which indicates a write intent lock.

   Note: The connection ID number (100000407) is large because it represents an internal connection within the server itself. This internal connection is used to manage locks on the RLV-enabled table.

   ConnectionID 3 has a schema lock on the table. The lock type is set to `Shared`, which indicates a shared schema lock

7. Return to the uncommitted transaction that performed the insert, and commit it:

```
Commit
```

   During the commit, the database releases the locks held by the transaction. For the tutorial, this releases only the shared schema lock. The RLV-enabled table now exists in memory,

with committed data. Therefore, the only lock present at this point is the write-intent lock held by the RLV-enabled portion of the table.

8. Re-execute **sp_iqlocks**.

```
sp_iqlocks
```

The schema lock is gone, but the write-intent lock remains:

```
conn_name,conn_id,user_id,table_type,creator,table_name,index_id,
lock_class,lock_duration,lock_type,row_identifier,row_range
'RVL_CONN_T775',
1000000407,'','BASE','DBA','rv_locks2',,'Table','Transaction','In
tent',,
```

Note: The row for conn_id 100000407 has not changed since the last time you executed **sp_iqlocks**.

9. Execute **sp_iqconnection** to view connection details

```
sp_iqconnection
```

You see:

```
ConnHandle,Name,Userid,LastReqTime,ReqType,IQCmdType,LastIQCmdTim
e,IQCursors,LowestIQCursorState,IQthreads,TxnID,ConnCreateTime,Te
mpTableSpaceKB,TempWorkSpaceKB,IQconnID,satoiq_count,iqtosa_count
,CommLink,NodeAddr,LastIdle,MPXServerName,LSName,INCConnName,INCC
onnSuspended
1,'SQL_DBC_13de5fd8','DBA','2012-08-08
08:49:25.629','PREFETCH','NONE',2012-08-08 08:49:25.0,0,'NONE',
0,0,2012-08-08 08:49:24.0,0,0,70,40,2,'local','',0,,,'','N'
3,'SQL_DBC_13cd6038','DBA','2012-08-08
09:25:32.920','OPEN','IQUTILITYOPENCURSOR',2012-08-08
09:25:32.0,0,'NONE',0,1008,2012-08-08
08:50:04.0,0,0,92,187,413,'local','',8789,,,'','N'
1000000407,'INT: RLVLockConn','','','unknown (0)','NONE',
0001-01-01 00:00:00.0,0,'NONE',0,0,2012-08-08
09:00:40.0,0,0,410,2,0,'NA','NA',0,,,'','N'
```

The third row (ConnHandle 1000000407) provides information on the internal connection (RLVLockConn) used by the RLV-enabled table to control the write-intent lock.

Note: ConnHandle 1000000407 matches conn_id 100000407 in **sp_iqlocks** output. It also matches ConnHandle 1000000407 in **sp_iqtransaction** output.

10. Execute **sa_conn_info** to view additional connection details. **sa_conn_info** is similar to **sp_iqconnection**.

```
sa_conn_info
```

You see:

```
Number,Name,Userid,DBNumber,LastReqTime,ReqType,CommLink,NodeAddr
,ClientPort,ServerPort,BlockedOn,LockRowID,LockIndexID,LockTable,
UncommitOps,ParentConnection
1000000407,sa_'INT: RLVLockConn','',0,'','unknown (0)','NA','NA',
0,0,0,0,,'',0,
3,'SQL_DBC_13cd6038','DBA',0,'2012-08-08
```

```
09:30:43.799','FETCH','local','',0,0,0,0,,'',0,
1,'SQL_DBC_13de5fd8','DBA',0,'2012-08-08
08:49:25.629','PREFETCH','local','',0,0,0,0,,'',0,
```

Note: In the first row, Number 1000000407 matches ConnHandle 1000000407 in the **sp_iqconnection** output, and conn_id 100000407 in the **sp_iqlocks** output.

Userid "INT: RLVLockConn" indicates an internal connection. This connection is used by the RLV-enabled table to control the write-intent lock.

See also

- *Manage Blocking in the RLV Store* on page 51
- *Row-Level Snapshot Versioning* on page 28
- *sa_conn_info system procedure* on page 78
- *sp_iqconnection Procedure* on page 99
- *sp_iqlocks Procedure* on page 109

Tutorial: Monitoring Row-Level Locks

In this tutorial, create RLV-enabled tables, commit a transaction, and delete the committed row to show row locking, and row-range locking. The **sp_iqlocks** stored procedure reports on the row-level locks.

Prerequisites

- SAP Sybase IQ server has a simplex database.

Task

Tip: You can monitor locks using Sybase Control Center. See the Sybase Control Center online help.

1. Create RLV-enabled tables `rv_locks` and `rv_locks2`, and configure table-level snapshot versioning:

```
SET TEMPORARY OPTION SNAPSHOT_VERSIONING = 'Table-level';

CREATE TABLE rv_locks(c1 INT, c2 INT, c3 INT);

INSERT INTO rv_locks VALUES (1,1,1);
INSERT INTO rv_locks VALUES (2,2,2);
INSERT INTO rv_locks VALUES (3,3,3);
INSERT INTO rv_locks VALUES (4,4,4);
COMMIT;

CREATE TABLE rv_locks2(c1 int, c2 int, c3 int);

INSERT INTO rv_locks2 VALUES (1,1,1);
```

```
INSERT INTO rv_locks2 VALUES (2,2,2);
INSERT INTO rv_locks2 VALUES (3,3,3);
INSERT INTO rv_locks2 VALUES (4,4,4);
COMMIT;

ALTER TABLE rv_locks ENABLE RLV STORE;
ALTER TABLE rv_locks2 ENABLE RLV STORE;
```

2. Enable connection blocking and set the blocking timeout threshold.

```
set temporary option blocking = 'On';
set temporary option blocking_timeout = '0';
```

3. Write to the table.

```
insert into rv_locks values (5,5,5);
```

Writing to, or querying, an RLV-enabled table creates the RLV-enabled portion of the table in memory, on demand.

4. Execute **sp_iqlocks**.

```
sp_iqlocks
```

A write-intent lock displays.

```
conn_name,conn_id,user_id,table_type,creator,table_name,index_id,
lock_class,lock_duration,lock_type,row_identifier,row_range
'SQL_DBC_13cd6038',
3,'DBA','BASE','DBA','rv_locks',,'Schema','Transaction','Shared',
,
'RLV_CONN_T775',
1000000407,'','BASE','DBA','rv_locks',,'Table','Transaction','Int
ent',,
```

Connection ID 100000407 has a write intent lock on the rv_locks table. The lock type is set to `Intent`, which indicates a write intent lock.

Note: The connection ID number (100000407) is large because it represents an internal connection within the server itself. This internal connection is used to manage locks on the RLV-enabled table.

ConnectionID 3 has a schema lock on the table. The lock type is set to `Shared`, which indicates a shared schema lock. Shared schema locks prevent other transactions from performing DML actions against the table.

5. Commit the transaction.

```
Commit
```

During the commit, the database releases the locks held by the transaction. In this example, this releases only the shared schema lock. The RLV-enabled table now exists in memory, with committed data. Therefore, the only lock present at this point is the write-intent lock held by the RLV-enabled portion of the table.

6. Delete the row that was previously committed.

```
delete from rv_locks where c1 = 5;
```

Before deleting the row, the database takes-out a row-level lock.

7. Execute **sp_iqlocks** again.

```
sp_iqlocks
```

Three locks display: a shared lock, a row lock, and a write-intent lock.

```
conn_name,conn_id,user_id,table_type,creator,table_name,index_id,
lock_class,lock_duration,lock_type,row_identifier,row_range
'SQL_DBC_13cd6038',
3,'DBA','BASE','DBA','rv_locks2',,'Schema','Transaction','Shared'
,,
'SQL_DBC_13cd6038',
3,'DBA','BASE','DBA','rv_locks2',,'Row','Transaction','Row',
281474976710656,1
'RVL_CONN_T775',
1000000407,'','BASE','DBA','rv_locks2',,'Table','Transaction','In
tent',,
```

Row 1 shows a lock type of `Shared`, indicating a shared schema lock. This lock is held by the DML for the **DELETE** statement. The shared schema lock prevents other transactions from performing DDL actions against the table.

Row 2 shows a lock type of `Row`, indicating a row-level lock:

- **row_identifier –** 281474976710656 is the row identifier of the row the lock starts on.
- **row_range –** 1 indicates that a single row was locked.

8. Delete a range of rows to illustrate row-range locking.
   a) Roll back the current transaction which is performing the delete, where c1=5.

```
Rollback
```

   b) Delete all rows where c1>0.

```
Delete from rv_locks2 where c1 > 0;
```

9. Execute **sp_iqlocks** again.

```
sp_iqlocks
```

Four locks display: a shared lock, two row locks (one with a row_range value), and a write-intent lock:

```
conn_name,conn_id,user_id,table_type,creator,table_name,index_id,
lock_class,lock_duration,lock_type,row_identifier,row_range
'SQL_DBC_13cd6038',
3,'DBA','BASE','DBA','rv_locks2',,'Schema','Transaction','Shared'
,,
'SQL_DBC_13cd6038',
3,'DBA','BASE','DBA','rv_locks2',,'Row','Transaction','Row',1,4
'SQL_DBC_13cd6038',
3,'DBA','BASE','DBA','rv_locks2',,'Row','Transaction','Row',
281474976710656,1
'RVL_CONN_T775',
1000000407,'','BASE','DBA','rv_locks2',,'Table','Transaction','In
tent',,
```

Note: The second output row represents rows locked from the table-level version, whereas the third output row represents the rows locked from the row-level version.

See also

- *Manage Blocking in the RLV Store* on page 51
- *Row-Level Snapshot Versioning* on page 28
- *Row Locks* on page 12
- *sp_iqlocks Procedure* on page 109

Tutorial: Monitoring Deadlocks

In this tutorial, add deadlocks to the RLV store, log the deadlocks for reporting purposes, and report deadlock information using **sa_report_deadlocks**.

Prerequisites

- SAP Sybase IQ server has a simplex database.
- RLV storage is configured.

Task

This tutorial creates a cycle between two transactions in order to create the deadlock:

1. Transaction A will have lock A, and will then request lock B. Transaction B will have lock B and then attempt to request lock A.
2. Transaction A will block on the request for lock B, which will not be released until transaction B releases it.
3. At the same time, transaction B will request lock A, which will not be released until transaction A releases it.

These releases will never happen, since each transaction is waiting on a resource currently held by the other transaction. This is a classic deadlock scenario. The database server prevents user statements from inducing a deadlock scenario and automatically rolls back the transaction for the statement that introduced the deadlock.

Tip: You can monitor locks using Sybase Control Center. See the Sybase Control Center online help.

1. Create RLV-enabled tables `rv_locks` and `rv_locks2`, and configure table-level snapshot versioning.

```
SET TEMPORARY OPTION SNAPSHOT_VERSIONING = 'Table-level';

CREATE TABLE rv_locks(c1 INT, c2 INT, c3 INT);

INSERT INTO rv_locks VALUES (1,1,1);
```

```
INSERT INTO rv_locks VALUES (2,2,2);
INSERT INTO rv_locks VALUES (3,3,3);
INSERT INTO rv_locks VALUES (4,4,4);
COMMIT;

CREATE TABLE rv_locks2(c1 int, c2 int, c3 int);

INSERT INTO rv_locks2 VALUES (1,1,1);
INSERT INTO rv_locks2 VALUES (2,2,2);
INSERT INTO rv_locks2 VALUES (3,3,3);
INSERT INTO rv_locks2 VALUES (4,4,4);
COMMIT;

ALTER TABLE rv_locks ENABLE RLV STORE;
ALTER TABLE rv_locks2 ENABLE RLV STORE;
```

2. Use **sp_iqlocks** to ensure no row locks exist on the rv_locks and rv_locks2 tables.
3. Enable connection blocking and set the blocking timeout threshold.

```
set temporary option blocking = 'On';
set temporary option blocking_timeout = '0';
```

4. Each connection deletes a row.
   - Connection A:

     ```
     delete from rv_locks where c1 = 1
     ```
   - Connection B:

     ```
     delete from rv_locks2 where c1 = 1
     ```

   These two DML actions begin the two separate transactions.
5. Execute **sp_iqlocks**.

```
sp_iqlocks
```

6. View the locks held by the two transactions.

```
Conn_name,conn_id,user_id,table_type,creator,table_name,index_id,
lock_class,lock_duration,lock_type,row_identifier,row_range
'SQL_DBC_13de5fd8',
1,'DBA','BASE','DBA','rv_locks2',,'Schema','Transaction','Shared'
,,
'SQL_DBC_13de5fd8',
1,'DBA','BASE','DBA','rv_locks2',,'Row','Transaction','Row',1,1
'SQL_DBC_13cd6038',
3,'DBA','BASE','DBA','rv_locks',,'Schema','Transaction','Shared',
,
'SQL_DBC_13cd6038',
3,'DBA','BASE','DBA','rv_locks',,'Row','Transaction','Row',1,1
'RVL_CONN_T775',
1000000407,'','BASE','DBA','rv_locks2',,'Table','Transaction','In
tent',,
'RVL_CONN_T774',
1000006141,'','BASE','DBA','rv_locks',,'Table','Transaction','Int
ent',,
```

In the output rows, note the conn_id's 1, and 3.

7. Connection A deletes the same row that Connection B already locked:
   ```
   delete from rv_locks2 where c1 = 1
   ```
   This connection blocks because Connection B already has the lock on that row.
8. Connection B tries to delete the same row that Connection A already locked:
   ```
   delete from rv_locks where c1 = 1
   ```
   The connection deadlocks. The database server recognizes the deadlock, and does not allow it to continue. The database server cancels this delete statement, rolls back the transaction, releases its locks, and issues this error message to Connection B's application: `SQL error, state = 40001 "Deadlock Detected".`
9. Roll back the Connection A delete action.
   - Connection A:
     ```
     rollback
     ```
10. Enable deadlock logging.
    - Connection B:
      ```
      set option public.log_deadlocks = 'on';
      ```
    Set logging for Connection B, since that is the connection that will induce the deadlock. Setting the **log_deadlocks** option for Connection A will not record any deadlocks.
11. Repeat the deadlock scenario.
    - Connection A:
      ```
      delete from rv_locks where c1 = 1
      ```
    - Connection A:
      ```
      delete from rv_locks2 where c1 = 1
      ```
    - Connection B:
      ```
      delete from rv_locks2 where c1 = 1
      ```
    - Connection B:
      ```
      delete from rv_locks where c1 = 1
      ```
    Connection B receives a SQL error and its transaction is rolled back, as before. However, because deadlock logging is enabled, the system logged the deadlock event.
12. Execute **sa_report_deadlocks**.
    ```
    sa_report_deadlocks
    ```
13. View the logged deadlock event.
    ```
    snapshotId,snapshotAt,waiter,who,what,object_id,record_id,owner,i
    s_victim,rollback_operation_count,iq_rid,iq_txn_id
    1,2012-08-08 12:24:04.339,3,'DBA',delete from rv_locks2 where c1 =
    1,1,775,1,false,0,1,13184
    1,2012-08-08 12:24:04.339,1,'DBA',delete from rv_locks where c1 =
    1,0,774,3,true,0,1,13160
    ```

The is_victim column indicates which transaction was selected as the rollback candidate.

Creating a Deadlock Reporting Event in Interactive SQL

Create a table and a system event for obtaining information about deadlocks.

Prerequisites

SAP Sybase IQ server has a simplex database.

Task

1. Create a table to store the data returned from the **sa_report_deadlocks** system procedure.

```
CREATE TABLE DeadlockDetails(
  deadlockId INT PRIMARY KEY DEFAULT AUTOINCREMENT,
  snapshotId BIGINT,
  snapshotAt TIMESTAMP,
  waiter INTEGER,
  who VARCHAR(128),
  what LONG VARCHAR,
  object_id UNSIGNED BIGINT,
  record_id BIGINT,
  owner INTEGER,
  is_victim BIT,
  rollback_operation_count UNSIGNED INTEGER );
```

2. Create an event that sends an e-mail notification when a deadlock occurs.

```
CREATE EVENT DeadlockNotification
TYPE Deadlock
HANDLER
BEGIN
 INSERT INTO DeadlockDetails WITH AUTO NAME
 SELECT snapshotId, snapshotAt, waiter, who, what, object_id,
record_id,
        owner, is_victim, rollback_operation_count
    FROM sa_report_deadlocks ();
 COMMIT;
 CALL xp_startmail ( mail_user ='John Smith',
                     mail_password ='mypwd' );
 CALL xp_sendmail( recipient='DBAdmin',
                   subject='Deadlock details added to the
DeadlockDetails table.' );
 CALL xp_stopmail ( );
END;
```

   This event copies the results of the **sa_report_deadlocks** system procedure into a table and notifies the administrator about the deadlock.

3. Set the **log_deadlocks** option on.

```
SET OPTION PUBLIC.log_deadlocks = 'On';
```

4. Enable logging of the most-recently executed statement.

```
CALL sa_server_option( 'RememberLastStatement', 'YES' );
```

Manage Blocking in the RLV Store

The RLV store uses the same transaction blocking mechanism as the IQ main store. Suppose you set the **BLOCKING** option to ON. If write lock A held be transaction 1 conflicts with write lock B which transaction 2 is attempting to obtain, then transaction 2 must wait until write lock A is released, or until the **BLOCKING_TIMEOUT** threshold is reached. By default, **BLOCKING** is OFF.

Although transaction blocking minimizes lock contention, transaction blocking can lead to deadlock.

See also

- *sp_iqconnection Procedure* on page 99
- *sp_iqtransaction Procedure* on page 134
- *sa_conn_info system procedure* on page 78

Enabling Connection Blocking

Enable connection blocking to force any transaction attempting to obtain a lock that conflicts with another transaction's existing lock to wait: either until every conflicting lock is released, or until the **BLOCKING_TIMEOUT** threshold is reached.

Prerequisites

- SAP Sybase IQ server has a simplex database.

Task

Set the **BLOCKING** database option to ON.

```
set temporary option blocking = 'On';
```

Note: The blocking option can be set either at the connection or PUBLIC level.

See also

- *Disabling Connection Blocking* on page 52
- *Setting the Blocking Timeout Threshold* on page 52
- *Transaction Blocking Deadlocks* on page 53
- *Tutorial: Monitoring Blocking* on page 54
- *BLOCKING Option* on page 71

Disabling Connection Blocking

Disable connection blocking to force any transaction attempting to obtain a lock that conflicts with another transaction's existing lock to roll back the transaction and display an error.

Prerequisites

- SAP Sybase IQ server has a simplex database.

Task

Set the **BLOCKING** database option to OFF.

```
set temporary option blocking = 'Off';
```

Note: The blocking option can be set either at the connection or PUBLIC level.

See also

- *Enabling Connection Blocking* on page 51
- *Setting the Blocking Timeout Threshold* on page 52
- *Transaction Blocking Deadlocks* on page 53
- *Tutorial: Monitoring Blocking* on page 54
- *BLOCKING Option* on page 71
- *BLOCKING_TIMEOUT Option* on page 72

Setting the Blocking Timeout Threshold

Use the threshold to set the length of time, in milliseconds, a transaction waits to obtain a lock. If the transaction attempting to obtain a lock conflicts with another transaction's existing lock, it waits until the **BLOCKING_TIMEOUT** option threshold is reached. If the conflict still exists, the transaction rolls back and you see an error.

Prerequisites

- SAP Sybase IQ server has a simplex database.

Task

Note: The default value, 0, indicates that a blocked transaction must wait indefinitely until all conflicting transactions release their locks.

Set the **BLOCKING_TIMEOUT** database option value to the number of milliseconds you want the transaction to wait for conflicting transactions to release their locks.

```
set temporary option blocking_timeout = '400';
```

Note: The blocking option can be set either at the connection or PUBLIC level.

See also

- *Enabling Connection Blocking* on page 51
- *Disabling Connection Blocking* on page 52
- *Transaction Blocking Deadlocks* on page 53
- *Tutorial: Monitoring Blocking* on page 54

Transaction Blocking Deadlocks

Transaction blocking can lead to a deadlock situation, in which a set of transactions arrive at a state where none of them can proceed.

A deadlock can arise for two reasons:

- **Cyclical blocking conflict –** transaction A is blocked on transaction B, and transaction B is blocked on transaction A. Additional time cannot solve the problem, and one of the transactions must be canceled, allowing the other to proceed. The same situation can arise with more than two transactions blocked in a cycle.

  To eliminate a transactional deadlock, the database server selects a connection from those involved in the deadlock, rolls back the changes for the transaction that is active on that connection and returns an error. The database server selects the connection to roll back by using an internal heuristic that prefers the connection with the smallest blocking wait time left as determined by the **BLOCKING_TIMEOUT** option. If all connections are set to wait forever, then the connection that caused the server to detect a deadlock is selected as the victim connection.
- **All workers are blocked –** when a transaction becomes blocked, its worker is not relinquished. For example, a database server is configured with three workers. Transactions A, B, and C are blocked on transaction D, which is not currently executing a request. A deadlock situation arises because there are no available workers. This situation is called thread deadlock.

  Suppose that the database server has *n* workers. Thread deadlock occurs when *n*-1 workers are blocked, and the last worker is about to block. The database server's kernel cannot permit this last worker to block, since doing so results in all workers being blocked, and the database server stops responding. Instead, the database server ends the task that is about to block the last worker, rolls back the changes for the transaction active on that connection, and returns an error.

  Database servers with tens or hundreds of connections may experience thread deadlocks in long-running requests, either because of the size of the database or because of blocking. In this case, you may want to increase the value of the **-gn** server option of the **start_iq** utility.

To view locks and deadlocks in Sybase Control Center, see the Sybase Control Center online help.

See also

- *Enabling Connection Blocking* on page 51
- *Disabling Connection Blocking* on page 52
- *Setting the Blocking Timeout Threshold* on page 52
- *Tutorial: Monitoring Blocking* on page 54
- *LOG_DEADLOCKS Option* on page 73
- *sa_report_deadlocks System Procedure* on page 82
- *sa_conn_info system procedure* on page 78
- *sp_iqconnection Procedure* on page 99
- *sp_iqtransaction Procedure* on page 134

Tutorial: Monitoring Blocking

In this tutorial, create RLV-enabled tables, execute a transaction, and use the **sp_iqtransaction** stored procedure to report on connection blocking and blocking timeout information for all transactions in the database.

Prerequisites

- SAP Sybase IQ server has a simplex database.
- RLV storage is configured.

Task

1. Create RLV-enabled tables `rv_locks` and `rv_locks2`, and configure table-level snapshot versioning:

```
SET TEMPORARY OPTION SNAPSHOT_VERSIONING = 'Table-level';

CREATE TABLE rv_locks(c1 INT, c2 INT, c3 INT);

INSERT INTO rv_locks VALUES (1,1,1);
INSERT INTO rv_locks VALUES (2,2,2);
INSERT INTO rv_locks VALUES (3,3,3);
INSERT INTO rv_locks VALUES (4,4,4);
COMMIT;

CREATE TABLE rv_locks2(c1 int, c2 int, c3 int);

INSERT INTO rv_locks2 VALUES (1,1,1);
INSERT INTO rv_locks2 VALUES (2,2,2);
INSERT INTO rv_locks2 VALUES (3,3,3);
INSERT INTO rv_locks2 VALUES (4,4,4);
```

```
COMMIT;

ALTER TABLE rv_locks ENABLE RLV STORE;
ALTER TABLE rv_locks2 ENABLE RLV STORE;
```

2. Set the snapshot versioning property of the transaction to row-level.

```
set temporary option Snapshot_Versioning = 'Row-level';
```

3. Enable connection blocking and set the blocking timeout threshold.

```
set temporary option blocking = 'On';
set temporary option blocking_timeout = '0';
```

4. Write to the table.

```
insert into rv_locks values (5,5,5);
```

   Writing to, or querying, an RLV-enabled table creates the RLV-enabled portion of the table in-memory, on-demand.

5. Execute **sp_iqtransaction** to view information for all transactions in the database.

```
sp_iqtransaction
```

   Transaction information displays, with each row representing a different transaction:

```
Name,Userid,TxnID,CmtID,VersionID,State,ConnHandle,IQConnID,MainT
ableKBCr,MainTableKBDr,TempTableKBCr,TempTableKBDr,TempWorkSpaceK
B,TxnCreateTime,CursorCount,SpCount,SpNumber,MPXServerName,Global
TxnID,VersioningType,Blocking,BlockingTimeout
'SQL_DBC_13cd6038','DBA',1008,0,0,'ACTIVE',
3,92,0,0,0,0,0,'2012-08-08 09:00:39.511',0,4,36,,0,'Row-
level','True',0
```

   The `Blocking` value is `True`, meaning that connection blocking is enabled using the **BLOCKING** database option. Connection blocking means that when lock contention is detected, the transaction waits (or blocks) for the conflicting lock to release before requesting the lock again.

   The **Blocking_Timeout** value is 0 (the default value), meaning the transaction will wait indefinitely for the conflicting lock to release.

See also

- *Enabling Connection Blocking* on page 51
- *Disabling Connection Blocking* on page 52
- *Setting the Blocking Timeout Threshold* on page 52
- *Transaction Blocking Deadlocks* on page 53

Manage Memory for the RLV Store

You can configure the amount of memory to use for the RLV store. The amount of memory relates directly to the amount of data that the RLV store can hold. When memory consumption reaches the set threshold, the RLV store merges automatically with the IQ main store.

When the RLV store memory size approaches its limit, the automated merge moves committed rows from the RLV store to the IQ main store, thus freeing memory for new RLV store rows.

If the merge operation does not free enough memory, and there are uncommitted rows pending in the RLV store from other transactions, current operations are blocked until other operations commit, and an automated merge occurs. If no further memory can be freed by merging, the current operation is rolled back.

See also

- *Merge RLV Store into IQ Main Store* on page 31
- *RLV Store Persistence and Durability* on page 6

Configuring RLV Store Memory Size

You can configure, on a per-server basis, the maximum memory size of the RLV store.

Prerequisites

- SAP Sybase IQ server has a simplex database.

Task

The size of the RLV store should be carefully chosen to avoid exceeding the physical memory on the host. Therefore, consider:

- Physical memory available to the host
- Size of IQ main store and temporary buffer caches
- Size of IQ large memory pool
- Memory requirements of other applications running on the host

Choosing a value which is too small will result in extra merges. This may potentially cause DMLs to fail if the frequency of the automated merge is too high.

1. Use the **-iqrlvmem** boot parameter, to specify the maximum size of the RLV store in Mb.

   ```
   -iqrlvmem 8192
   ```

   This specifies an RLV size of 8192 Mb.

Note: A DML which causes the size of the RLV store to exceed the configured memory limit will immediately trigger an automatic merge. However, the RV_MERGE_NODE_MEMSIZE node threshold will usually trigger the automated merge before this limit is reached.

2. (Optional) At runtime, change the size of the RLV store using the **sa_server_option**.

```
sa_server_option 'RLV_memory_mb', 16384
```

This reconfigures the RLV store size to 16384 Mb.

Note: This size is a "soft" limit. It is possible for the RLV store to temporarily exceed the configured memory. RLV store memory is normally freed by merge. However merge itself requires memory. Therefore during merge the memory limit may be temporarily exceeded. Furthermore, open transactions remain in the RLV store (and thus use memory), even after the merge.

See also

- *RV_MERGE_NODE_MEMSIZE Option* on page 75
- *-iqrlvmem start_iq Server Option* on page 141
- *sa_server_option System Procedure* on page 84
- *Setting Merge Trigger Thresholds* on page 32
- *Monitoring RLV Memory Usage* on page 58

Monitoring RLV Memory Usage

Monitor system-wide RLV memory use and/or per table memory use.

Prerequisites

- SAP Sybase IQ server has a simplex database.
- RLV storage is configured.

Task

1. Monitor system-wide RLV memory use with the **sp_iqstatus** stored procedure.

Row Name	Description
RLV memory limit	The memory limit as specified by **sp_iqrlvmemory** stored procedure or **sa_server_option RLV_memory_mb**

Row Name	Description
RLV memory used	Amount of RLV store memory used. **Note:** Memory used may legally exceed the memory limit during a merge of the RLV and main stores.

2. Monitor per-table RLV memory use with the **sp_iqrlvmemory** stored procedure for a specified owner and table name.

   The procedure outputs one row per table consuming RLV memory.

Output Column	Description
table_id	ID of the table this row represents
fragments	number of store fragments for this table
total	total RLV memory used by this table
data	RLV memory used to store the table data
dictionary	RLV memory used to store the dictionaries for this table
bitmap	RLV memory used to store table level bitmaps

   Note: Version-specific data, such as version bitmaps and on-demand indexes, do not count against the RLV memory limit and are not reported in **sp_iqrlvmemory**. **sp_iqrlvmemory** accepts optional parameters for owner name and table name which limit the output to a single row.

```
sp_iqrlvmemory 'RLV_table','DBA'
```

See also

- *sp_iqrlvmemory Procedure* on page 112
- *Configuring RLV Store Memory Size* on page 57

Appendix: Troubleshoot the RLV Store

The troubleshooting appendix includes a collection of symptoms, with information to further diagnose or solve the problem.

RLV Store Out of Memory

Problem: You receive the error message "`RLV Store has run out of memory`".

Explanation A
Long running active transaction(s) may be holding RLV store fragments in memory. To correct, try:

- Running **sp_iqrlvmemory** stored procedure to determine whether old RLV store fragments exist.
- Using **sp_iqtransaction** stored procedure to locate old active transactions, and terminating them.

Explanation B
Row-level versioned transactions adding or modifying too much data on an RLV-enabled table will eventually run out of store memory. To correct, try:

- Reducing the update size.
- Reducing the load size.
- Increasing the frequency of commits (to reduce the amount of data modified per transaction).

Explanation C
Automated merge is not keeping pace with number of RLV transactions. To correct, try:

- Reviewing the merge history table, SYSIQRLVMERGEHISTORY.
- Adjusting the automated merge period, **RV_AUTO_MERGE_EVAL_INTERVAL** option.
- Adjusting the automated merge thresholds, **RV_MERGE_TABLE_NUMROWS**, **RV_MERGE_TABLE_MEMPERCENT**, and **RV_MERGE_NODE_MEMSIZE** options.

Explanation D
Maximum RLV memory configured value is too low. To correct, try:

- Increasing the maximum RLV memory, using server startup option **-iqrlvmem**.

Note: Increasing the maximum RLV memory may require adjusting IQ caches, or adding RAM. It also requires a server restart, as the parameter is not dynamic.

See also

- *sp_iqrlvmemory Procedure* on page 112
- *sp_iqtransaction Procedure* on page 134
- *SYSIQRLVMERGEHISTORY System View* on page 219
- *RV_AUTO_MERGE_EVAL_INTERVAL Option* on page 74
- *RV_MERGE_NODE_MEMSIZE Option* on page 75
- *RV_MERGE_TABLE_MEMPERCENT Option* on page 75
- *RV_MERGE_TABLE_NUMROWS Option* on page 76
- *-iqrlvmem start_iq Server Option* on page 141

Cannot Convert to Multiplex

Problem: You receive the error message "`Cannot convert database to multiplex. An RLV dbspace exists`".

Explanation

A row-level versioning (RLV) dbspace exists on the simplex database you are trying to convert to multiplex. Row-level versioning is not supported in multiplex in the current version of SAP Sybase IQ.

Cannot Create RLV Dbspace in Multiplex

Problem: You receive the error message "`Create RLV dbspace not allowed in multiplex`".

Explanation

Row-level versioning (RLV) is not supported in multiplex in the current version of SAP Sybase IQ.

RLV Dbspace Already Exists

Problem: You receive the error message "`RLV dbspace already exists. Cannot create more than one RLV dbspace`".

Explanation A

A row-level versioning (RLV) dbspace exists on the database. SAP Sybase IQ only supports one RLV dbspace per simplex database.

Explanation B

You could encounter this error while trying to increase the size of the RLV dbspace. If so, instead of trying to create another RLV dbspace, add a new dbfile to the existing RLV dbspace.

Cannot Make RLV Dbspace Read-Only

Problem: You receive the error message "`The RLV dbspace N cannot be made read only because it contains RLV enabled tables`".

Explanation
A row-level versioning (RLV) dbspace is intended to be read-write, because of the role it plays in real-time transactions. However, it is possible to make the dbspace read-only as long as there are no RLV-enabled tables.

- Drop or disable the RLV-enabled tables before making the RLV dbspace read-only.

See also
- *Configure the RLV Dbspace* on page 20

Cannot Create Table in RLV Dbspace

Problem: You receive the error message "`Cannot create table N in an RLV dbspace`".

Explanation
Each RLV-enabled table automatically makes use of the RLV dbspace during transactions. However, SAP Sybase IQ does not support the creation of tables in the RLV dbspace.

Cannot Enable Table for RLV Storage

Problem: You receive an error message indicating that the table cannot be enabled for row-level versioning (RLV).

Explanation A
RLV storage cannot be enabled on tables unless the RLV store dbspace is configured.

Explanation B
RLV storage cannot be enabled on any table that has:

- LONG BINARY (LOB) or LONG VARCHAR (CLOB) data types.
- WORD (WD) or TEXT indexes.
- Foreign key constraints.

Explanation C
RLV storage cannot be enabled on temporary or global temporary tables.

Cannot Use Foreign Keys in RLV Enabled Table

Problem: You receive the error message "`The use of foreign keys in a RLV table is not supported`".

Explanation

The use of foreign keys within a row-level versioning (RLV) enabled table is not supported in the current version of SAP Sybase IQ. You cannot create foreign keys on a table enabled for row-level versioning. Furthermore, you can not enable row-level versioning on a table which has foreign keys.

Cannot Use Index Type in RLV Enabled Table

Problem: You either receive the error message "`The index type 'TEXT' cannot be used with an RLV enabled table`", or "`The index type 'WD' cannot be used with an RLV enabled table`".

Explanation

The use of text indexes and word indexes within a row-level versioning (RLV) enabled table are not supported in the current version of SAP Sybase IQ. You cannot create these indexes on a table enabled for row-level versioning. Furthermore, you can not enable row-level versioning on a table which has a text or word index.

Merge Required Before Table Level Modification

Problem: You receive the error message "`Table N requires an RLV store merge before table-level modification`".

Explanation A

A table-level (TLV) read-write operation is attempting to modify a row-level versioning (RLV) enabled table with the **BLOCKING** option set to OFF. This TLV transaction requires a merge of the RLV store, but because **BLOCKING** is set to OFF, it cannot wait for the automated merge to complete. To correct, try:

- Setting the **BLOCKING** option ON, and ensuring that the first statement in the transaction is a table-level read-write.
- Performing a manual merge before the table level DML transaction, using the **sp_iqmergerlvstore**stored procedure.

Explanation B
A table-level (TLV) read-write operation is attempting to modify a row-level versioning (RLV) enabled table. This TLV transaction requires a merge of the RLV store. However, this merge cannot occur automatically because the TLV transaction has already established a snapshot version for this table. To correct, try:

- Ensuring that the first statement in the transaction is a table-level read-write.
- Performing a manual merge before the table level DML transaction, using the **sp_iqmergerlvstore** stored procedure.

Explanation C
A TLV transaction is attempting to modify multiple RLV-enabled tables. To correct, try:

- Dividing the transaction so that individual transactions each modify one RLV table.

See also
- *BLOCKING Option* on page 71
- *sp_iqmergerlvstore Procedure* on page 112

Cannot Perform Merge of RLV Store

Problem: You receive an error message indicating that the RLV store cannot be merged with the IQ main store.

Explanation A
An RLV store merge was attempted on a table which was not RLV-enabled. Only tables which are RLV-enabled may be merged.

Explanation B
There was an error during the merge. Consult the database IQMSG file for possible exceptions or errors codes, which may describe a problem and allow you to perform a DML operation to address the item causing the issue.

For example, if there was an error inserting a row from the RLV store into the IQ main store, and the IQMSG file contained an exception on row X or value Y, it might be possible to:

- Identify the source row
- Save the row
- Delete the row.

In this case, the merge should then succeed. The exported row could then be dealt with separately.

See also

- *sp_iqtransaction Procedure* on page 134

RLV Store Merge Already in Progress

Problem: You receive the error message "`RLV store merge already in progress for table N`".

Explanation

There is already an RLV store merge in progress for this table. Only one RLV merge is allowed at a time for an RLV table.

Cannot Open the Requested Object for Write in the Current Transaction

Problem: You receive an error message indicating that the system cannot open the requested object for write in the current transaction, and that another user has the row locked.

Explanation

Multiple connections are trying to update the same row of a table when you are running a manual merge. A table-level versioning transaction exists that owns a write lock on the table.

Transaction Seems to Hang

Problem: Transaction appears to hang.

Explanation

Multiple connections may be waiting on a table-level write-lock. If the transactions are in progress when a foreground (blocking) merge of the RLV store and IQ main store is triggered, the transactions will appear to hang. To troubleshoot further, run the appropriate stored procedures from another connection to see if there are locks being held or a merge in progress.

Failed RLV Recovery

You encounter a recovery problem such as Checksum error reading a page from disk, mismatched sequence number on head / tail of page, or OS exception reading a page from disk.

Recover occurs in four high level phases:

1. Initialization (SYSIQRVLOG table is scanned and log identity block is loaded. Tables are added to the recovery list if log pages exist).
2. Commit log analysis
3. Table log analysis
4. Operations which belong to committed transactions are redone.

Recovery errors in phases 1, 3, or 4 will result in an IQ server shutdown. An error in phase 2 is handled by doing an extended phase 3.

Recommendations

1. Use two server startup switches to restrict access:
   - Use **-gd DBA** so that only users with the SERVER OPERATOR system privilege can start and stop databases on a running server.
   - Use **-gm 1** to allow a single connection plus one DBA connection above the limit so that a DBA can connect and drop others in an emergency.
2. Set **-iqrvrec_bypass = 1** to bypass all RLV recovery. This option is intended to be for emergency repairs, such as dropping a problematic RLV table. As currently implemented this disables further logging, but there are no other checks in the code that will prevent general RLV operations. As such, this mode is likely unstable if non-DBA users are allowed general access.
3. Manually establish / correct the consistency of the database.
4. Truncate the RLV portion of a table. This may leave the database inconsistent, but will allow a subsequent recovery.

   Note: All data in the RLV portion of the table will be lost.
5. Reboot with normal recovery.

Appendix: SQL Reference

Reference material for SQL statements, database options, functions, system procedures, system tables, and views mentioned in this document.

Database Options

Database options customize and modify database behavior.

AGGREGATION_PREFERENCE Option

Controls the choice of algorithms for processing an aggregate.

Allowed Values
-6 to 6

Default
0

Scope
Option can be set at the database (PUBLIC) or user level. When set at the database level, the value becomes the default for any new user, but has no impact on existing users. When set at the user level, overrides the PUBLIC value for that user only. No system privilege is required to set option for self. System privilege is required to set at database level or at user level for any user other than self.

Requires the SET ANY PUBLIC OPTION system privilege to set this option. Can be set temporary for an individual connection or for the PUBLIC role. Takes effect immediately.

Description
For aggregation (**GROUP BY**, **DISTINCT**, **SET** functions) within a query, the SAP Sybase IQ optimizer has a choice of several algorithms for processing the aggregate. `AGGREGATION_PREFERENCE` lets you override the costing decision of the optimizer when choosing the algorithm. the option does not override internal rules that determine whether an algorithm is legal within the query engine.

This option is normally used for internal testing and for manually tuning queries that the optimizer does not handle well. Only experienced DBAs should use it. Inform SAP Sybase Technical Support, if you need to set `AGGREGATION_PREFERENCE`, as setting this option might mean that a change to the optimizer may be appropriate.

Value	Action
0	Let the optimizer choose
1	Prefer aggregation with a sort
2	Prefer aggregation using IQ indexes
3	Prefer aggregation with a hash
4	Prefer aggregation with a distinct/grouping sort
5	Prefer aggregation with a sort if grouping columns include all the partitioning keys of a hash partitioned table.
6	Prefer aggregation with a hash if grouping columns include all the partitioning keys of a hash partitioned table.
-1	Avoid aggregation with a sort
-2	Avoid aggregation using IQ indexes
-3	Avoid aggregation with a hash
-4n	Avoid aggregation with a distinct/grouping sort
-5	Avoid aggregation with a sort if grouping columns include all the partitioning keys of a hash partitioned table.
-6	Avoid aggregation with a hash if grouping columns include all the partitioning keys of a hash partitioned table.

ALLOW_SNAPSHOT_VERSIONING Option

Applies to all base tables in the database (as opposed to RLV-enabled tables only). Restricts table versioning for all base tables to either table-level or row-level snapshot versioning. This option does not apply to the IQ catalog store.

Allowed Values
any, table-level, row-level

Default
any

Scope
Option can be set at the database (PUBLIC) level only.

Requires the SET ANY SYSTEM OPTION system privilege to set this option. Takes effect immediately.

Description

Value	Action
any	No restrictions on snapshot versioning.
row-level	Allows only row-level snapshot versioning. Any transactions attempting to use table-level versioning to modify a table will fail with an `Illegal snapshot isolation` error.
table-level	Allows only table-level snapshot versioning. Any transactions attempting to use row-level versioning to modify a table will fail with an `Illegal snapshot isolation` error.

BASE_TABLES_IN_RLV_STORE Option

Registers tables in the RLV store, enabling row-level versioning. RLV-enabled tables are eligible for multiple writer concurrent access. You can override this setting at the table level using the **CREATE_TABLE** statement.

Allowed Values
ON, OFF

Default
OFF

Scope
Option can be set at the database (PUBLIC) level only.

Requires the SET ANY SYSTEM OPTION system privilege to set this option. Takes effect immediately.

Description
This option turns row-level versioning on and off. When set to ON, tables are registered in the RLV store. RLV-enabled tables are optimized for real-time updates.

The **{ ENABLE | DISABLE } RLV STORE** clause of the **CREATE_TABLE** statement always overrides the **BASE_TABLES_IN_RLV_STORE** option.

BLOCKING Option

Controls the behavior in response to locking conflicts.

Allowed Values
ON, OFF

Default
OFF

Scope
Option can be set at the database (PUBLIC) or user level. When set at the database level, the value becomes the default for any new user, but has no impact on existing users. When set at the user level, overrides the PUBLIC value for that user only. No system privilege is required to set option for self. System privilege is required to set at database level or at user level for any user other than self.

Requires the SET ANY PUBLIC OPTION system privilege to set this option. Can be set temporary for an individual connection or for the PUBLIC role. Takes effect immediately.

Description
When `BLOCKING` is off, a transaction receives an error when it attempts a write operation and is blocked by the read lock of another transaction.

When `BLOCKING` is on, any transaction attempting to obtain a lock that conflicts with an existing lock held by another transaction waits until every conflicting lock is released or until the blocking_timeout is reached. If the lock is not released within blocking_timeout milliseconds, then an error is returned for the waiting transaction.

BLOCKING_TIMEOUT Option

Controls the length of time a transaction waits to obtain a lock.

Allowed Values
Integer, in milliseconds.

Default
0

Scope
Option can be set at the database (PUBLIC) or user level. When set at the database level, the value becomes the default for any new user, but has no impact on existing users. When set at the user level, overrides the PUBLIC value for that user only. No system privilege is required to set option for self. System privilege is required to set at database level or at user level for any user other than self.

Requires the SET ANY PUBLIC OPTION system privilege to set this option. Can be set temporary for an individual connection or for the PUBLIC role. Takes effect immediately.

Description
When the blocking option is on, any transaction attempting to obtain a lock that conflicts with an existing lock waits for the indicated number of milliseconds for the conflicting lock to be

released. If the lock is not released within blocking_timeout milliseconds, an error is returned for the waiting transaction.

Set the option to 0 to force all transactions attempting to obtain a lock to wait until all conflicting transactions release their locks.

ENABLE_ASYNC_IO Option

Allows a DBA to enable or disable the asynchronous IO used by the RLV persistence log.

Allowed Values
TRUE, FALSE

A change in value requires a database close and re-open, or a server restart.

Default
TRUE

Scope
Option can be set at the database (PUBLIC) or user level. When set at the database level, the value becomes the default for any new user, but has no impact on existing users. When set at the user level, overrides the PUBLIC value for that user only. No system privilege is required to set option for self. System privilege is required to set at database level or at user level for any user other than self.

Requires the SET ANY PUBLIC OPTION system privilege to set this option. Can be set temporary for an individual connection or for the PUBLIC role. If permitted, can be set for an arbitrary other user or role, or for all users via the role. Takes effect immediately.

LOG_DEADLOCKS Option

Controls whether deadlock reporting is turned on or off.

Allowed values
On, Off

Default
Off

Scope
Option can be set at the database (PUBLIC) level only.

Requires the SET ANY SYSTEM OPTION system privilege to set this option. Takes effect immediately.

Remarks
When this option is set to On, the database server logs information about deadlocks in an internal buffer. The size of the buffer is fixed at 10000 bytes. You can view the deadlock information using the sa_report_deadlocks stored procedure. The contents of the buffer are retained when this option is set to Off.

When deadlock occurs, information is reported for only those connections involved in the deadlock. The order in which connections are reported is based on which connection is waiting for which row. For thread deadlocks, information is reported about all connections.

When you have deadlock reporting turned on, you can also use the Deadlock system event to take action when a deadlock occurs

RV_AUTO_MERGE_EVAL_INTERVAL Option

This option configures the evaluation period used to determine when an automated merge of the row-level versioned (RLV) and IQ main stores should occur.

Allowed Values
1 – MAX_UINT (minutes)

Default
15 (minutes)

Scope
Option can be set at the database (PUBLIC) level only.

Requires the SET ANY SYSTEM OPTION system privilege to set this option. Takes effect immediately.

Description
This option is be used to configure the period of wait time, in minutes, between activations of the merge evaluator. The merge evaluator examines the merge parameters of each row-level versioning (RLV) enabled table against configured threshold values to determine whether a non-blocking (background) merge of the RLV table to IQ main stores should occur.

If the interval ends while the evaluator is active, or when a merge is already in progress, the interval re-sets.

Any new value for the interval is used when the merge evaluator is next activated.

RV_MERGE_NODE_MEMSIZE Option

An automated merge of the row-level versioned (RLV) store and IQ main stores occurs based on the merge thresholds, including RV_MERGE_NODE_MEMSIZE. When this node threshold is exceeded, a merge will be triggered.

Allowed Values
0 - 100 (percent)

Default
75 (percent)

Scope
Option can be set at the database (PUBLIC) level only.

Requires the SET ANY SYSTEM OPTION system privilege to set this option. Takes effect immediately.

Description
This option sets the percentage of total RLV memory size as a merge threshold for the node. If the total RLV memory size surpasses the threshold, the merge condition evaluator will determine which table(s) to merge. If multiple tables must be merged to satisfy the node threshold, parallel merges will be triggered for each table to be merged.

RV_MERGE_TABLE_MEMPERCENT Option

An automated merge of the row-level versioned (RLV) store and IQ main stores occurs based on the merge thresholds, including RV_MERGE_TABLE_MEMPERCENT. If this table threshold is exceeded, a merge will be triggered for the specific table.

Allowed Values
0 - 100 (percent)

Default
0 (percent)

Note: When RV_MERGE_TABLE_MEMPERCENT = 0, then the system uses a (per-table) threshold of 100% / N, where N is the number of RLV-enabled tables that have been loaded.

Scope
Option can be set at the database (PUBLIC) level only.

Requires the SET ANY SYSTEM OPTION system privilege to set this option. Takes effect immediately.

Description
This option sets the percentage of memory used as a merge threshold for an RLV-enabled table. If the memory used surpasses the threshold, a merge will occur.

The system evaluates whether to merge the RLV and IQ main stores on a per-table basis. It enumerates through all loaded RLV tables, and for each one decides whether a merge is warranted. A merge for a single table is deemed warranted if:

1. The table violates either the memory threshold (RV_MERGE_TABLE_MEMPERCENT) or the row threshold (RV_MERGE_TABLE_NUMROWS), and
2. The system does not determine that a large percentage of the RLV rows are uncommitted, and are therefore unable to be merged.

RV_MERGE_TABLE_NUMROWS Option

An automated merge of the row-level versioned (RLV) store and IQ main stores occurs based on the merge thresholds, including RV_MERGE_TABLE_NUMROWS. If this table threshold is exceeded, a merge will be triggered for the specific table.

Allowed Values
1000 - 100000000

Default
10000000

Scope
Option can be set at the database (PUBLIC) level only.

Requires the SET ANY SYSTEM OPTION system privilege to set this option. Takes effect immediately.

Description
This option sets the number of rows used as a merge threshold for an RLV-enabled table. If the number of rows used surpasses the threshold, a merge will occur.

The system evaluates whether to merge the RLV and IQ main stores on a per-table basis. It enumerates through all loaded RLV tables, and for each one decides whether a merge is warranted. A merge for a single table is deemed warranted if:

1. The table violates either the memory threshold (RV_MERGE_TABLE_MEMPERCENT) or the row threshold (RV_MERGE_TABLE_NUMROWS), and
2. The system does not determine that a large percentage of the RLV rows are uncommitted, and are therefore unable to be merged.

RV_RESERVED_DBSPACE_MB Option

A portion of the RLV store must be reserved for memory used by data structures during critical operations.

Allowed Values
Integer greater than or equal to 50 (megabytes)

Default
The minimum of 50 Mb or half the size of the RLV dbspace

Scope
Option can be set at the database (PUBLIC) level only.

Requires the SET ANY SYSTEM OPTION system privilege to set this option. Takes effect immediately. The server does not need to be restarted in order to change reserved space size.

Description
This option allows you to control the amount of space set aside in the RLV store for small but critical data structures used during release savepoint, commit, and rollback operations.

SNAPSHOT_VERSIONING Option

Applies to RLV-enabled tables only (as opposed to all base tables in the database). Controls whether RLV-enabled tables are accessed using single-writer table-level versioning, or multiple writer row-level versioning. This option does not apply to the IQ catalog store.

Allowed Values

Value	Action
row-level	Enables concurrent writer access and row-level versioning for RLV-enabled tables. The first transaction to modify a table row establishes a row write lock that persists until the end of the transaction. Subsequent transactions attempting to modify a locked row either fail with a lock/future version error, or block until the lock is released based on the value of the BLOCKING option.

Value	Action
table-level	Enables single-writer access and table-level versioning. The first transaction to access the table establishes a table write lock which persists until the end of the transaction. Subsequent transactions attempting to write to a locked table either fail with a lock/future version error, or block until the lock is released based on the value of the BLOCKING option.

Default
table-level

Scope
Requires the SET ANY PUBLIC OPTION system privilege to set this option for PUBLIC or for other user or role.

Procedures

Use the system-supplied stored procedures in SAP Sybase IQ databases to retrieve system information.

sa_conn_info system procedure

Reports connection property information.

Syntax

```
sa_conn_info( [ connidparm ] )
```

Arguments

- ***connidparm* –** This optional INTEGER parameter specifies the connection ID number. The default is NULL.

Result set

Column name	Data type	Description
Number	INTEGER	Returns the connection ID (a number) for the current connection.

Column name	Data type	Description
Name	VARCHAR(255)	Returns the connection ID (a number) for the current connection. Temporary connection names have `INT:` prepended to the connection name.
Userid	VARCHAR(255)	Returns the user ID for the connection.
DBNumber	INTEGER	Returns the ID number of the database.
LastReqTime	VARCHAR(255)	Returns the time at which the last request for the specified connection started. This property can return an empty string for internal connections, such as events.
ReqType	VARCHAR(255)	Returns the type of the last request. If a connection has been cached by connection pooling, its ReqType value is CONNECT_POOL_CACHE.
CommLink	VARCHAR(255)	Returns the communication link for the connection. This is one of the network protocols supported by SAP Sybase IQ, or local for a same-computer connection.
NodeAddr	VARCHAR(255)	Returns the address of the client in a client/server connection.
ClientPort	INTEGER	Returns the client's TCP/IP port number or 0 if the connection isn't a TCP/IP connection.
ServerPort	INTEGER	Returns the database server's TCP/IP port number or 0.

Column name	Data type	Description
BlockedOn	INTEGER	Returns zero if the current connection isn't blocked, or if it is blocked, the connection number on which the connection is blocked because of a locking conflict.
LockRowID	UNSIGNED BIGINT	Returns the identifier of the locked row. LockRowID is NULL if the connection is not waiting on a lock associated with a row (that is, it is not waiting on a lock, or it is waiting on a lock that has no associated row).
LockIndexID	INTEGER	Returns the identifier of the locked index. LockIndexID is -1 if the lock is associated with all indexes on the table in LockTable. LockIndexID is NULL if the connection is not waiting on a lock associated with an index (that is, it is not waiting on a lock, or it is waiting on a lock that has no associated index).
LockTable	VARCHAR(255)	Returns the name of the table associated with a lock if the connection is currently waiting for a lock. Otherwise, LockTable returns an empty string.
UncommitOps	INTEGER	Returns the number of uncommitted operations.

Column name	Data type	Description
ParentConnection	INTEGER	Returns the connection ID of the connection that created a temporary connection to perform a database operation (such as performing a backup or creating a database). For other types of connections, this property returns NULL.

Remarks

If *connidparm* is less than zero, then a result set consisting of connection properties for the current connection is returned. If *connidparm* is not supplied or is NULL, then connection properties are returned for all connections to all databases running on the database server.

In a block situation, the BlockedOn value returned by this procedure allows you to check which users are blocked, and who they are blocked on. The sa_locks system procedure can be used to display the locks held by the blocking connection.

For more information based on any of these properties, you can execute something similar to the following:

```
SELECT *, DB_NAME( DBNumber ),
   CONNECTION_PROPERTY( 'LastStatement', Number )
   FROM sa_conn_info( );
```

The value of LockRowID can be used to look up a lock in the output of the sa_locks procedure.

The value in LockIndexID can be used to look up a lock in the output of the sa_locks procedure. Also, the value in LockIndexID corresponds to the primary key of the ISYSIDX system table, which can be viewed using the SYSIDX system view.

Every lock has an associated table, so the value of LockTable can be used to unambiguously determine whether a connection is waiting on a lock.

Privileges

No privileges are required to execute this system procedure for the current connection ID. To execute this system procedure for other connections, you must have either the SERVER OPERATOR, MONITOR, or DROP CONNECTION system privilege.

Side effects

None

Examples

The following example uses the sa_conn_info system procedure to return a result set summarizing connection properties for all connections to the server.

```
CALL sa_conn_info( );
```

Number	Name	Userid	DBNumber	...
79	SQL_DBC_10dcf810	DBA	0	...
46	setup	User1	0	...
...	...	...	...	...

The following example uses the sa_conn_info system procedure to return a result set showing which connection created a temporary connection.

```
SELECT Number, Name, ParentConnection FROM sa_conn_info();
```

Connection 8 created the temporary connection that executed a CREATE DATABASE statement.

```
Number      Name                ParentConnection
-------------------------------------------------
1000000048  INT: CreateDB       8
9           SQL_DBC_14675af8    (NULL)
8           SQL_DBA_152d5ac0    (NULL)
```

sa_report_deadlocks System Procedure

Retrieves information about deadlocks from an internal buffer created by the database server.

Syntax

sa_report_deadlocks( )

Result set

Column Name	Data Type	Description
snapshotId	BIGINT	The deadlock instance (all rows pertaining to a particular deadlock have the same ID).
snapshotAt	TIMESTAMP	The time when the deadlock occurred.
waiter	INT	The connection handle of the waiting connection.
who	VARCHAR(128)	The user ID associated with the connection that is waiting.

Column Name	Data Type	Description
what	LONG VARCHAR	The command being executed by the waiting connection. This information is only available if you have turned on capturing of the most recently-prepared SQL statement by specifying the **-zl** option on the database server command line.
object_id	UNSIGNED BIGINT	The object ID of the table containing the row.
record_id	BIGINT	The row ID for system tables
owner	INT	The connection handle of the connection owning the lock being waited on.
is_victim	BIT	Identifies the rolled back transaction.
rollback_operation_count	UNSIGNED INT	The number of uncommitted operations that may be lost if the transaction rolls back.
iq_rid	UNSIGNED BIGINT	The row ID for IQ RLV enabled tables.
iq_txn_id	UNSIGNED BIGINT	The transaction id of the associated row

Remarks
When the log_deadlocks option is set to On, the database server logs information about deadlocks in an internal buffer. You can view the information in the log using the sa_report_deadlocks system procedure.

Privileges
You must have the MONITOR system privilege.

Side effects
None.

sa_server_option System Procedure

Overrides a server option while the server is running..

Syntax

sa_server_option(opt , val)

Arguments

- **opt –** Use this CHAR(128) parameter to specify a server option name.
- **val –** Use this CHAR(128) parameter to specify the new value for the server option.

Remarks

Option Name	Values	Additional information
AutoMultiProgrammingLevel	YES, NO	Default is YES. When set to YES, the database server automatically adjusts its multiprogramming level, which controls the maximum number of tasks that can be active at a time. If you choose to control the multiprogramming level manually by setting this option to NO, you can still set the initial, minimum, and maximum values for the multiprogramming level.
AutoMultiProgrammingLevelStatistics	YES, NO	Default is NO. When set to YES, statistics for automatic multiprogramming level adjustments appear in the database server message log.
CacheSizingStatistics	YES, NO	Default is NO. When set to YES, display cache information in the database server messages window whenever the cache size changes.
CollectStatistics	YES, NO	Default is YES. When set to YES, the database server collects Performance Monitor statistics.
ConnsDisabled	YES, NO	Default is NO. When set to YES, no other connections are allowed to any databases on the database server.
ConnsDisabledForDB	YES, NO	Default is NO. When set to YES, no other connections are allowed to the current database.

Option Name	Values	Additional information
ConsoleLogFile	filename	The name of the file used to record database server message log information. Specifying an empty string stops logging to the file. Double any backslash characters in the path because this value is a SQL string.
ConsoleLog-MaxSize	file-size (bytes)	The maximum size, in bytes, of the file used to record database server message log information. When the database server message log file reaches the size specified by either this property or the **-on** server option, the file is renamed with the extension **.old** appended (replacing an existing file with the same name if one exists). The database server message log file is then restarted.
CurrentMulti-ProgrammingLevel	integer	Default is 20. Sets the multiprogramming level of the database server.
DatabaseCleaner	ON, OFF	Default is ON. Do not change the setting of this option except on the recommendation of Technical Support.
DeadlockLogging	ON, OFF, RESET, CLEAR	Default is OFF. Controls deadlock logging. The value deadlock_logging is also supported. The following values are supported: • **ON –** Enables deadlock logging. • **OFF –** Disables deadlock logging and leaves the deadlock data available for viewing. • **RESET –** Clears the logged deadlock data, if any exists, and then enables deadlock logging. • **CLEAR –** Clears the logged deadlock data, if any exists, and then disables deadlock logging. Once deadlock logging is enabled, you can use the sa_report_deadlocks system procedure to retrieve deadlock information from the database.
DebuggingInformation	YES, NO	Default is NO. Displays diagnostic messages and other messages for troubleshooting purposes. The messages appear in the database server messages window.

Option Name	Values	Additional information
DiskSandbox	ON, OFF	Default is OFF. Sets the default disk sandbox settings for all databases started on the database server that do not have explicit disk sandbox settings. Changing the disk sandbox settings by using the sa_server_option system procedure does not affect databases already running on the database server. To use the sa_server_option system procedure to change disk sandbox settings, you must provide the secure feature key for the manage_disk_sandbox secure feature.
DropBadStatistics	YES, NO	Default is YES. Allows automatic statistics management to drop statistics that return bad estimates from the database.
DropUnusedStatistics	YES, NO	Default is YES. Allows automatic statistics management to drop statistics that have not been used for 90 consecutive days from the database.
IdleTimeout	Integer (minutes)	Default is 240. Disconnects TCP/IP connections that have not submitted a request for the specified number of minutes. This prevents inactive connections from holding locks indefinitely
IPAddressMonitorPeriod	Integer (seconds)	The minimum value is 10 and the default is 0. For portable devices, the default value is 120. Sets the time to check for new IP addresses in seconds.
LivenessTimeout	Integer (seconds)	Default is 120. A liveness packet is sent periodically across a client/server TCP/IP network to confirm that a connection is intact. If the network server runs for a LivenessTimeout period without detecting a liveness packet, the communication is severed.
MaxMultiProgrammingLevel	Integer	Default is four times the value for CurrentMultiProgrammingLevel. Sets the maximum database server multiprogramming level.
MessageCategoryLimit	Integer	Default is 400. Sets the minimum number of messages of each severity and category that can be retrieved using the sa_server_messages system procedure.
MinMultiProgrammingLevel	Integer	Default is the minimum of the value of the -gtc server option and the number of logical CPUs on the computer.

Option Name	Values	Additional information
OptionWatchAction	MESSAGE, ERROR	Default is MESSAGE. Specifies the action that the database server takes when an attempt is made to set an option in the list. When OptionWatchAction is set to MESSAGE, and an option specified by OptionWatchList is set, a message appears in the database server messages window indicating that the option being set is on the options watch list.When OptionWatchAction is set to ERROR, an error is returned indicating that the option cannot be set because it is on the options watch list. You can view the current setting for this property by executing `SELECT DB_PROPERTY( 'OptionWatchAction' );`

OptionWatchList	Comma-separated list of database options	Specifies a comma-separated list of database options that you want to be notified about, or have the database server return an error for, when they are set. The string length is limited to 128 bytes. By default, it is an empty string. For example, the following command adds the automatic_timestamp, float_as_double, and tsql_hex_constant option to the list of options being watched:
		`CALL sa_server_option( 'OptionWatchList','automatic_timestamp, float_as_double,tsql_hex_constant' );`
		You can view the current setting for this property by executing:
		`SELECT DB_PROPERTY( 'OptionWatchList' );`

ProcedureProfiling	YES, NO, RESET, CLEAR	Default is NO.
ProfileFilterConn	connection-id	Instructs the database server to capture profiling information for a specific connection ID, without preventing other connections from using the database. When connection filtering is enabled, the value returned for **SELECT PROPERTY('ProfileFilterConn')** is the connection ID of the connection being monitored. If no ID has been specified, or if connection filtering is disabled, the value returned is -1.

Option Name	Values	Additional information
ProcessorAffinity	Comma-delimited list of processor numbers and/or ranges. The default is that all processors are used or the setting of the -gta option.	Instructs the database server which logical processors to use on Windows or Linux. Specify a comma-delimited list of processor numbers and/or ranges. If the lower endpoint of a range is omitted, then it is assumed to be zero. If the upper endpoint of a range is omitted, then it is assumed to be the highest CPU known to the operating system. The in_use column returned by the sa_cpu_topology system procedure contains the current processor affinity of the database server, and the in_use column indicates whether the database server is using a processor. Alternatively, you can query the value of the ProcessorAffinity database server property. The database server might not use all of the specified logical processors in the following cases: • If one or more of the specified logical processors does not exist, or is offline. • If the license does not allow it. If you specify an invalid processor ID, sa_server_option returns an error.
ProfileFilterUser	user-id	Instructs the database server to capture profiling information for a specific user ID.
QuittingTime	Valid date and time	Instructs the database server to shut down at the specified time.
RememberLastPlan	YES, NO	Default is NO. Instructs the database server to capture the long text plan of the last query executed on the connection. This setting is also controlled by the -zp server option.When RememberLastPlan is turned on, obtain the textual representation of the plan of the last query executed on the connection by querying the value of the LastPlanText connection property: `SELECT CONNECTION_PROPERTY( 'LastPlanText' );`

Option Name	Values	Additional information

RememberLast-Statement	YES, NO	Default is NO.
		
		Instructs the database server to capture the most recently prepared SQL statement for each database running on the server. For stored procedure calls, only the outermost procedure call appears, not the statements within the procedure.When RememberLastStatement is turned on, you can obtain the current value of the LastStatement for a connection by querying the value of the LastStatement connection property:
		`SELECT CONNECTION_PROPERTY( 'LastState-ment' );`
		
		When client statement caching is enabled, and a cached statement is reused, this property returns an empty string. When RememberLast-Statement is turned on, the following statement returns the most recently-prepared statement for the specified connection:
		`SELECT CONNECTION_PROPERTY( 'LastStatement', connection-id );`
		
		The sa_conn_activity system procedure returns this same informa-tion for all connections.
		
		Note: When -zl is specified, or when the RememberLastStatement server setting is turned on, any user can call the sa_conn_activity system procedure or obtain the value of the LastStatement connection property to find out the most recently-prepared SQL statement for any other user. Use this option with caution and turn it off when it is not required.

Option Name	Values	Additional information

RequestFilterConn	connection-id, -1	Filter the request logging information so that only information for a particular connection is logged. This filtering can reduce the size of the request log file when monitoring a database server with many active connections or multiple databases. You can obtain the connection ID by executing the following:
		`CALL sa_conn_info( );`
		To log a specific connection once you have obtained the connection ID, execute the following statement:
		`CALL sa_server_option( 'RequestFilterConn', connection-id );`
		Filtering remains in effect until it is explicitly reset, or until the database server is shut down. To reset filtering, use the following statement:
		`CALL sa_server_option( 'RequestFilterConn', -1 );`

RequestFilterDB	database-id, -1	Filter the request logging information so that only information for a particular database is logged. This can help reduce the size of the request log file when monitoring a server with multiple databases. You can obtain the database ID by executing the following statement when you are connected to the desired database:
		`SELECT CONNECTION_PROPERTY( 'DBNumber' );`
		To log only information for a particular database, execute the following statement:
		`CALL sa_server_option( 'RequestFilterDB', database-id );`
		Filtering remains in effect until it is explicitly reset, or until the database server is shut down. To reset filtering, use the following statement:
		`CALL sa_server_option( 'RequestFilterDB', -1 );`

Option Name	Values	Additional information
RequestLogFile	filename	The name of the file used to record request information. Specifying an empty string stops logging to the request log file. If request logging is enabled, but the request log file was not specified or has been set to an empty string, the server logs requests to the database server messages window. Double any backslash characters in the path because this value is a SQL string. When client statement caching is enabled, set the **max_client_statements_cached** option to 0 to disable client statement caching while the request log is captured, if the log will be analyzed using the **tracetime.pl** Perl script.

Option Name	Values	Additional information
RequestLogging	SQL, HOST-VARS, PLAN, PROCE-DURES, TRIG-GERS, OTHER, BLOCK S, RE-PLACE, ALL, YES, NONE, NO	Default is NONE. This call turns on logging of individual SQL statements sent to the database server for use in troubleshooting with the database server -zr and -zo options. Values can be combinations of the following, separated by either a plus sign (+), or a comma: • **PLAN –** enables logging of execution plans (short form). If logging of procedures (PROCEDURES) is enabled, execution plans for procedures are also recorded. • **HOSTVARS –** enables logging of host variable values. If you specify HOSTVARS, the information listed for SQL is also logged. • **PROCEDURES –** enables logging of statements executed from within procedures. • **TRIGGERS –** enables logging of statements executed from within triggers. • **OTHER –** enables logging of additional request types not included by SQL, such as FETCH and PREFETCH. However, if you specify OTHER but do not specify SQL, it is the equivalent of specifying SQL+OTHER. Including OTHER can cause the log file to grow rapidly and could negatively impact server performance. • **BLOCKS –** enables logging of details showing when a connection is blocked and unblocked on another connection. • **REPLACE –** at the start of logging, the existing request log is replaced with a new (empty) one of the same name. Otherwise, the existing request log is opened and new entries are appended to the end of the file. • **ALL –** logs all supported information. This value is equivalent to specifying SQL+PLAN+HOSTVARS+PROCEDURES+TRIGGERS+OTHER+BLOCKS. This setting can cause the log file to grow rapidly and could negatively impact server performance. • **NO or NONE –** turns off logging to the request log. You can view the current setting for this property by executing: `SELECT PROPERTY( 'RequestLogging' );`

Option Name	Values	Additional information
RequestLogMax-Size	file-size (bytes)	The maximum size of the file used to record request logging information, in bytes. If you specify 0, then there is no maximum size for the request logging file, and the file is never renamed. This value is the default. When the request log file reaches the size specified by either the sa_server_option system procedure or the -zs server option, the file is renamed with the extension **.old** appended (replacing an existing file with the same name if one exists). The request log file is then restarted.
RequestLog-NumFiles	Integer	The number of request log file copies to retain.If request logging is enabled over a long period, the request log file can become large. The **-zn** option allows you to specify the number of request log file copies to retain
RequestTiming	YES, NO	Default is NO. Instructs the database server to maintain timing information for each new connection. This feature is turned off by default. When it is turned on, the database server maintains cumulative timers for all new connections that indicate how much time the connection spent in the server in each of several states. The change is only effective for new connections, and lasts for the duration each connection.You can use the sa_performance_diagnostics system procedure to obtain a summary of this timing information, or you can retrieve individual values by inspecting the following connection properties: • **ReqCountUnscheduled** • **ReqTimeUnscheduled** • **ReqCountActive** • **ReqTimeActive** • **ReqCountBlockIO** • **ReqTimeBlockIO** • **ReqCountBlockLock** • **ReqTimeBlockLock** • **ReqCountBlockContention** • **ReqTimeBlockContention** When the RequestTiming server property is on, there is a small overhead for each request to maintain the additional counters.

Option Name	Values	Additional information
rlv_auto_merge	ON, OFF	The default is ON. Enables or disables automatic merges of the RLV store into the IQ main store for row-level versioning-enabled tables. If rlv_auto_merge is OFF, no automated merges of the RLV and IQ main stores occur. This implies that you assume responsibility to manually merge data so that the RLV store gets synced to the IQ main store before the upper rlv_memory_mb threshold is reached.
rlv_memory_mb	The minimum value is 1 MB. The maximum value is 2048. Any other value will set the amount of memory to 2048 MB.	Specifies the maximum amount of memory (the RLV store), in MB, to reserve for row-level versioning . The default value is 2048 MB.

Option Name	Values	Additional information
SecureFeatures	feature-list	Allows you to manage secure features for a database server that is already running. The feature-list is a comma-separated list of feature names or feature sets. By adding a feature to the list, you limit its availability. To remove items from the list of secure features, specify a minus sign (-) before the secure feature name. To call sa_server_option('SecureFeatures',...), the connection must have the ManageFeatures secure feature enabled on the connection. The -sf key (the system secure feature key) enables ManageFeatures, as well as all of the other features. So if you used the system secure feature key, then changing the set of SecureFeatures will not have any effect on the connection. But if you used another key (for example a key that had been created using the create_secure_feature_key system procedure) then your connection may be immediately affected by the change, depending on what other features are included in the key. Any changes you make to allow or prevent access to features take effect immediately for the database server. The connection that executes the sa_server_option system procedure may or may not be affected, depending on the secure feature key the connection is using and whether or not it allows the connection access to the specified features. For example, to secure two features, use the following syntax: `CALL sa_server_option('SecureFeatures', 'CONSOLE_LOG,WEBCLIENT_LOG' );` After executing this statement, the list of secure features is set according to what has been changed.
StatisticsCleaner	ON, OFF	Default is ON. The statistics cleaner fixes statistics that give bad estimates by performing scans on tables. By default the statistics cleaner runs in the background and has a minimal impact on performance. Turning off the statistics cleaner does not disable the statistic governor, but when the statistics cleaner is turned off, statistics are only created or fixed when a query is run.
WebClientLogFile	filename	The name of the web service client log file. The web service client log file is truncated each time you use the -zoc server option or the WebClientLogFile property to set or reset the file name. Double any backslash characters in the path because this value is a string.

Option Name	Values	Additional information
WebClientLog-ging	ON, OFF	Default is OFF. This option enables and disables logging of web service clients. The information that is logged includes HTTP requests and response data. Specify ON to start logging to the web service client log file, and specify OFF to stop logging to the file.

Privileges
You must have the MANAGE PROFILING system privilege to use the following options, which are related to application profiling or request logging:

- ProcedureProfiling
- ProfileFilterConn
- ProfileFilterUser
- RequestFilterConn
- RequestFilterDB
- RequestLogFile
- RequestLogging
- RequestLogMaxSize
- RequestLogNumFiles

For all other options, your must have the SERVER OPERATOR system privilege.

Side effects
None.

Example
The following statement causes cache information to be displayed in the database server messages window whenever the cache size changes:

```
CALL sa_server_option( 'CacheSizingStatistics', 'YES' );
```

The following statement disallows new connections to the current database:

```
CALL sa_server_option( 'ConnsDisabledForDB', 'YES' );
```

The following statement enables logging of all SQL statements, procedure calls, plans, blocking and unblocking events, and starts a new request log:

```
CALL sa_server_option( 'RequestLogging', 'SQL+PROCEDURES+BLOCKS+PLAN
+REPLACE' );
```

sp_iqcolumn Procedure

Displays information about columns in a database.

Syntax1

```
sp_iqcolumn ( [ table_name ],[ table_owner ], [table_loc] )
```

Syntax2

```
sp_iqcolumn [ table_name='table_name' ],[ table_owner='tableowner' ],
[table_loc='table_loc']
```

Usage

Syntax	Description
Syntax1	If you specify *table_owner* without specifying *table_name*, you must substitute NULL for *table_name*. For example, `sp_iqcolumn NULL,DBA`.
Syntax2	The parameters can be specified in any order. Enclose *'table_name'* and *'table_owner'* in single quotes.

Privileges

No specific system privileges are required to run this stored procedure.

Description

Displays information about columns in a database. Specifying the *table_name* parameter returns the columns only from tables with that name. Specifying the table_owner parameter returns only tables owned by that user. Specifying both *table_name* and table_owner parameters chooses the columns from a unique table, if that table exists. Specifying table_loc returns only tables that are defined in that segment type. Specifying no parameters returns all columns for all tables in a database. **sp_iqcolumn** does not return column information for system tables.

Column name	Description
table_name	The name of the table
table_owner	The owner of the table
column_name	The name of the column
domain_name	The data type

Column name	Description
width	The precision of numeric data types that have precision and scale or the storage width of numeric data types without scale; the width of character data types
scale	The scale of numeric data types
nulls	'Y' if the column can contain NULLS, 'N' if the column cannot contain NULLS
default	'Identity/Autoincrement' if the column is an identity/autoincrement column, null if not.
cardinality	The distinct count, if known, by indexes.
location	TEMP = IQ temporary store, MAIN = IQ main store, SYSTEM = catalog store
isPartitioned	'Y' if the column belongs to a partitioned table and has one or more partitions whose dbspace is different from the table partition's dbspace, 'N' if the column's table is not partitioned or each partition of the column resides in the same dbspace as the table partition.
remarks	User comments added with the **COMMENT** statement
check	the check constraint expression

sp_iqcolumn Procedure Example

Use the example as a reference for **sp_iqcolumn** usage.

The following variations in syntax both return all of the columns in the table `Departments`:

```
sp_iqcolumn Departments
```

```
call sp_iqcolumn (table_name='Departments')
```

```
table_name   table_owner  column_name     domain_name  width  scale
  nulls  default
Departments  GROUPO       DepartmentID    integer       4        0
  N      (NULL)
Departments  GROUPO       DepartmentName  char         40        0
  N      (NULL)
Departments  GROUPO       DepartmentHead  integer       4        0
  Y      (NULL)

cardinality  location  isPartitioned  remarks  check
5            Main       N             (NULL)   (NULL)
0            Main       N             (NULL)   (NULL)
0            Main       N             (NULL)   (NULL)
```

The following variation in syntax returns all of the columns in all of the tables owned by table owner DBA.

```
sp_iqcolumn table_owner='DBA'
```

sp_iqconnection Procedure

Shows information about connections and versions, including which users are using temporary dbspace, which users are keeping versions alive, what the connections are doing inside SAP Sybase IQ, connection status, database version status, and so on.

Syntax

```
sp_iqconnection [ connhandle ]
```

Applies to

Simplex and multiplex.

Usage

connhandle is equal to the Number connection property and is the ID number of the connection. The **connection_property** system function returns the connection ID:

```
SELECT connection_property ( 'Number' )
```

When called with an input parameter of a valid *connhandle*, **sp_iqconnection** returns the one row for that connection only.

Privileges

Requires the DROP CONNECTION, MONITOR or SERVER OPERATOR system privilege. Users without one of these system privileges must be granted EXECUTE permission to run the stored procedure.

Description

sp_iqconnection returns a row for each active connection. The columns ConnHandle, Name, Userid, LastReqTime, ReqType, CommLink, NodeAddr, and LastIdle are the connection properties Number, Name, Userid, LastReqTime, ReqType, CommLink, NodeAddr, and LastIdle respectively, and return the same values as the system function **sa_conn_info**. The additional columns return connection data from the SAP Sybase IQ side of the SAP Sybase IQ engine. Rows are ordered by ConnCreateTime.

The column MPXServerName stores information related to internode communication (INC), as shown:

Server Where Run	MPXServerName Column Content
Simplex server	NULL (All connections are local/user connections)

Server Where Run	MPXServerName Column Content
Multiplex coordinator	• NULL for local/user connections • Contains value of secondary node's server name (source of connection) for every INC connection (either on-demand or dedicated heartbeat connection)
Multiplex secondary	• NULL for local/user connections • Contains value of coordinator's server name (source of connection).

In Java applications, specify SAP Sybase IQ-specific connection properties from TDS clients in the RemotePWD field. This example, where **myconnection** becomes the IQ connection name, shows how to specify IQ specific connection parameters:

```
p.put("RemotePWD",",,CON=myconnection");
```

Column Name	Description
ConnHandle	The ID number of the connection.
Name	The name of the server.
Userid	The user ID for the connection.
LastReqTime	The time at which the last request for the specified connection started.
ReqType	A string for the type of the last request.
IQCmdType	The current command executing on the SAP Sybase IQ side, if any. The command type reflects commands defined at the implementation level of the engine. These commands consist of transaction commands, DDL and DML commands for data in the IQ store, internal IQ cursor commands, and special control commands such as **OPEN** and **CLOSE DB**, **BACKUP**, **RESTORE**, and others.
LastIQCmdTime	The time the last IQ command started or completed on the IQ side of the SAP Sybase IQ engine on this connection.
IQCursors	The number of cursors open in the IQ store on this connection.

Column Name	Description
LowestIQCursorState	The IQ cursor state, if any. If multiple cursors exist on the connection, the state that appears is the lowest cursor state of all the cursors; that is, the furthest from completion. Cursor state reflects internal SAP Sybase IQ implementation detail and is subject to change in the future. For this version, cursor states are: NONE, INITIALIZED, PARSED, DESCRIBED, COSTED, PREPARED, EXECUTED, FETCHING, END_OF_DATA, CLOSED and COMPLETED. As suggested by the names, cursor state changes at the end of the operation. A state of PREPARED, for example, indicates that the cursor is executing.
IQthreads	The number of SAP Sybase IQ threads currently assigned to the connection. Some threads may be assigned but idle. This column can help you determine which connections are using the most resources.
TxnID	The transaction ID of the current transaction on the connection. This is the same as the transaction ID in the `.iqmsg` file by the BeginTxn, CmtTxn, and PostCmtTxn messages, as well as the Txn ID Seq logged when the database is opened.
ConnCreateTime	The time the connection was created.
TempTableSpaceKB	The number of kilobytes of IQ temporary store space in use by this connection for data stored in IQ temp tables.
TempWorkSpaceKB	The number of kilobytes of IQ temporary store space in use by this connection for working space such as sorts, hashes, and temporary bitmaps. Space used by bitmaps or other objects that are part of indexes on SAP Sybase IQ temporary tables are reflected in TempTableSpaceKB.
IQConnID	The ten-digit connection ID included as part of all messages in the `.iqmsg` file. This is a monotonically increasing integer unique within a server session.
satoiq_count	An internal counter used to display the number of crossings from the SQL Anywhere side to the IQ side of the SAP Sybase IQ engine. This might be occasionally useful in determining connection activity. Result sets are returned in buffers of rows and do not increment satoiq_count or iqtosa_count once per row.
iqtosa_count	An internal counter used to display the number of crossings from the IQ side to the SQL Anywhere side of the SAP Sybase IQ engine. This might be occasionally useful in determining connection activity.
CommLink	The communication link for the connection. This is one of the network protocols supported by SAP Sybase IQ, or is local for a same-machine connection.
NodeAddr	The node for the client in a client/server connection.
LastIdle	The number of ticks between requests.

Column Name	Description
MPXServerName	If an INC connection, the varchar(128) value contains the name of the multiplex server where the INC connection originates. NULL if not an INC connection.
LSName	The logical server name of the connection. NULL if logical server context is unknown or not applicable.
INCConnName	The name of the underlying INC connection for a user connection. The data type for this column is varchar(255). If **sp_iqconnection** shows an INC connection name for a suspended user connection, that user connection has an associated INC connection that is also suspended.
INCConnSuspended	The value "Y" in this column indicates that the underlying INC connection for a user connection is in a suspended state. The value "N" indicates that the connection is not suspended.

Example

sp_iqconnection

```
ConnHandle      Name    Userid                 LastReqTime   ReqType
=== ================== ======  ==========================   =======
 1  'SQL_DBC_100525210' 'DBA'   '2011-03-28  09:29:24.466'  'OPEN'


           IQCmdType    LastIQCmdTime      IQCursors  LowestIQCursorState
====================    =============     =========  ====================
'IQUTILITYOPENCURSOR'  2011-03-28 09:29:24.0       0             'NONE'


IQthreads   TxnID          ConnCreateTime TempTableSpaceKB TempWorkSpaceKB
=========  =======  ====================  =============== ===============
        0  3352568  2011-03-28 09:29:20.0               0             0


IQconnID satoiq_count iqtosa_count CommLink NodeAdd LastIdle MPXServerName
======== ============ ============ ======== ======= ======== ============
      34           43            2  'local'      ''      244  (NULL)

LSName      INCConnName                 INCConnSuspended
==========  ==================          ================
Finance_LS  'IQ_MPX_SERVER_P54'         'Y'
```

sp_iqdbsize Procedure

Displays the size of the current database.

Syntax

```
sp_iqdbsize ( [ main ] )
```

Applies to

Simplex and multiplex.

Privileges
Requires the ALTER DATABASE system privilege. Users without the ALTER DATABASE system privilege must be granted `EXECUTE` permission to run the stored procedure.

Description
Returns the total size of the database. Also returns the number of pages required to hold the database in memory and the number of IQ pages when the database is compressed (on disk).

If run on a multiplex database, the default parameter is **main**, which returns the size of the shared IQ store.

If run when there are no rows in any RLV-enabled tables, the Physical Blocks, the RLVLogBlocks and RLVLogKBytes columns will contain non-zero entries, and the remaining columns contain zeros. This indicate no row-level versioned tables.

Column Name	Description
Database	The path name of the database file.
Physical Blocks	Total database size in blocks. An IQ database consists of one or more dbspaces. Each dbspace has a fixed size, which is originally specified in units of megabytes. This megabyte quantity is converted to blocks using the IQ page size and the corresponding block size for that IQ page size. The Physical Blocks column reflects the cumulative total of each SAP Sybase IQ dbspace size, represented in blocks.
KBytes	The total size of the database in kilobytes. This value is the total size of the database in blocks (Physical Blocks in the previous **sp_iqdbsize** column) multiplied by the block size. The block size depends on the IQ page size.
Pages	The total number of IQ pages necessary to represent in memory all of the data stored in tables and the metadata for these objects. This value is always greater than or equal to the value of Compressed Pages (the next **sp_iqdbsize** column).
Compressed Pages	The total number of IQ pages necessary to store on disk the data in tables and metadata for these objects. This value is always less than or equal to the value of Pages (the previous **sp_iqdbsize** column), because SAP Sybase IQ compresses pages when the IQ page is written from memory to disk. The **sp_iqdbsize** Compressed Pages column represents the number of compressed pages.
NBlocks	The total size in blocks used to store the data in tables. This value is always less than or equal to the **sp_iqdbsize** Physical Blocks value.

Column Name	Description
Catalog Blocks	The total size in blocks used to store the metadata for tables.
RLVLogBlocks	Number of blocks used for log information for the RLV store.
RLVLogKBytes	Total size of the RLV log, in Kb.

Example

Displays size information for the database `iqdemo`:

```
sp_iqdbsize
```

```
Database

PhysicalBlocks KBytes Pages CompressedPages NBlocks CatalogBlocks
RLVLogBlocks RLVLogKBytes
============== ====== ===== =============== ======= =============
============ ============
/system1/sybase/IQ-16_0/demo/iqdemo.db
          1280    522   688             257    1119             18
```

sp_iqdbspace Procedure

Displays detailed information about each IQ dbspace.

Syntax

sp_iqdbspace [*dbspace-name*]

Applies to

Simplex and multiplex.

Privileges

Requires MANAGE ANY DBSPACE system privilege. Users without MANAGE ANY DBSPACE system privilege must be granted `EXECUTE` permission.

Description

Use the information from **sp_iqdbspace** to determine whether data must be moved, and for data that has been moved, whether the old versions have been deallocated.

Column Name	Description
DBSpaceName	Name of the dbspace as specified in the **CREATE DBSPACE** statement. Dbspace names are always case-insensitive, regardless of the **CREATE DATABASE**...**CASE IGNORE** or **CASE RESPECT** specification.
DBSpaceType	Type of the dbspace (`MAIN`, `SHARED_TEMP`, `TEMPORARY`, or `RLV`).
Writable	T (writable) or F (not writable).

Column Name	Description
Online	T (online) or F (offline).
Usage	Percent of dbspace currently in use by all files in the dbspace.
TotalSize	Total size of all files in the dbspace in the units B (bytes), K (kilobytes), M (megabytes), G (gigabytes), T (terabytes), or P (petabytes).
Reserve	Total reserved space that can be added to all files in the dbspace.
NumFiles	Number of files in the dbspace.
NumRWFiles	Number of read/write files in the dbspace.
Stripingon	F (Off).
StripeSize	Always 1, if disk striping is on.
BlkTypes	Space used by both user data and internal system structures.
OkToDrop	"Y" indicates the dbspace can be dropped; otherwise "N".

Values of the BlkTypes block type identifiers:

Identifier	Block Type
A	Active version
B	Backup structures
C	Checkpoint log
D	Database identity
F	Free list
G	Global free list manager
H	Header blocks of the free list
I	Index advice storage
M	Multiplex CM*
O	Old version
R	RLV free list manager
T	Table use
U	Index use
N	Column use

Identifier	Block Type
X	Drop at checkpoint

*The multiplex commit identity block (actually 128 blocks) exists in all IQ databases, even though it is not used by simplex databases.

Example
Displays information about dbspaces:

```
sp_iqdbspace;
```

Note: The following example shows objects in the `iqdemo` database to better illustrate output. `iqdemo` includes a sample user dbspace named `iq_main` that may not be present in your own databases.

DBSpaceName	DBSpaceType	Writable	Online	Usage	Total Size	Reserve	Num Files	Num RWFiles	Stripingon	Stripe Size	Blk Types	Ok To Drop
IQ_MAIN	MAIN	T	T	55	75M	200 M	1	1	T	1K	1H, 5169A, 190	N
IQ__ SYSTEM_ MAIN	MAIN	T	T	21	300 M	50M	1	1	F	8K	1H, 7648F, 32D, 128 M	N
IQ_SYSTEM_ TEMP	TEMPORARY	T	T	1	100 M	50M	1	1	F	8K	1H, 64F, 32A	N

sp_iqfile Procedure

Displays detailed information about each dbfile in a dbspace.

Syntax

```
sp_iqfile [ dbspace-name ]
```

Applies to
Simplex and multiplex.

Privileges

Requires the MANAGE ANY DBSPACE system privilege. Users without the MANAGE ANY DBSPACE system privilege must be granted `EXECUTE` permission.

Description

sp_iqfile displays the usage, properties, and types of data in each dbfile in a dbspace. You can use this information to determine whether data must be moved, and for data that has been moved, whether the old versions have been deallocated.

Column Name	Description
DBSpaceName	Name of the dbspace as specified in the **CREATE DBSPACE** statement. Dbspace names are always case-insensitive, regardless of the **CREATE DATABASE...CASE IGNORE** or **CASE RESPECT** specification.
DBFileName	Logical file name.
Path	Location of the physical file or raw partition.
SegmentType	Type of dbspace (`MAIN`, `TEMPORARY`, or `RLV`).
RWMode	Mode of the dbspace: always read-write (RW).
Online	T (online) or F (offline).
Usage	Percent of dbspace currently in use by this file in the dbspace. When run against a secondary node in a multiplex configuration, this column displays NA.
DBFileSize	Current size of the file or raw partition. For a raw partition, this size value can be less than the physical size.
Reserve	Reserved space that can be added to this file in the dbspace.
StripeSize	Always 1, if disk striping is on.
BlkTypes	Space used by both user data and internal system structures.
FirstBlk	First IQ block number assigned to the file.
LastBlk	Last IQ block number assigned to the file.
OkToDrop	"Y" indicates the file can be dropped; otherwise "N".

Identifier	Block Type
A	Active Version
B	Backup Structures

Identifier	Block Type
C	Checkpoint Log
D	Database Identity
F	Free list
G	Global Free list Manager
H	Header Blocks of the Free List
I	Index Advice Storage
M	Multiplex CM*
O	Old Version
R	RLV Free list manager
T	Table Use
U	Index Use
N	Column Use
X	Drop at Checkpoint

*The multiplex commit identity block (actually 128 blocks) exists in all IQ databases, even though it is not used by simplex databases.

Example

Displays information about the files in the dbspaces:

```
sp_iqfile;
```

```
sp_iqfile;
DBSpaceName,DBFileName,Path,SegmentType,RWMode,Online,
Usage,DBFileSize,Reserve,StripeSize,BlkTypes,FirstBlk,
LastBlk,OkToDrop

'IQ_SYSTEM_MAIN','IQ_SYSTEM_MAIN','/sun1-c1/users/smith/mpx/m/
mpx_db.iq','MAIN','RW','T','21','
2.92G','0B','1K','1H,76768F,32D,19A,185O,128M,34B,32C'
,1,384000,'N'

'mpx_main1','mpx_main1','/sun1-c1/users/smith/mpx/m/
mpx_main1.iq','MAIN','RW','T','1'
,'100M','0B','1K','1H',1045440,1058239,'N'

'IQ_SHARED_TEMP','sharedfile1_bcp','/sun1-c1/users/smith/mpx/m/
f1','SHARED_TEMP','RO','T','0',
'50M','0B','1K','1H',1,6400,'N'
```

```
'IQ_SHARED_TEMP','sharedfile2_bcp','/sun1-c1/users/smith/mpx/m/
f2','SHARED_TEMP','RO','T','0',
'50M','0B','1K','1H',1045440,1051839,'N'

'IQ_SYSTEM_TEMP','IQ_SYSTEM_TEMP','/sun1-c1/users/smithmpx/m/
mpx_db.iqtmp','TEMPORARY','RW',
'T','1','2.92G','0B','1K','1H,64F,33A',1,384000,'N'
```

sp_iqlocks Procedure

Shows information about locks in the database, for both the IQ main store and the IQ catalog store.

Syntax

```
sp_iqlocks ([connection,] [[owner.]table_name,] max_locks,]
[sort_order])
```

Privileges

Requires the MONITOR system privilege. Users without the MONITOR system privilege must be granted `EXECUTE` permission to run the stored procedure.

Usage

Optional **sp_iqlocks** parameters you can specify to restrict results:

Parameter	Data Type	Description
connection	integer	Connection ID. With this option, the procedure returns information about locks for the specified connection only. Default is zero, which returns information about all connections.
owner.table_name	char(128)	Table name. With this option, the procedure returns information about locks for the specified table only. Default is NULL, which returns information about all tables in the database. If you do not specify owner, it is assumed that the caller of the procedure owns the table.
max_locks	integer	Maximum number of locks for which to return information. Default is 0, which returns all lock information.
sort_order	char(1)	Order in which to return information: • C sorts by connection (default) • T sorts by table_name

Description

Displays information about current locks in the database. Depending on the options you specify, you can restrict results to show locks for a single connection, a single table, or a specified number of locks.

sp_iqlocks displays the following information, sorted as specified in the sort_order parameter:

Column	Data type	Description
conn_name	VARCHAR(128)	The name of the current connection.
conn_id	INTEGER	Connection ID that has the lock.
user_id	CHAR(128)	User associated with this connection ID.
table_type	CHAR(6)	The type of table. This type is either BASE for a table, GLBTMP for global temporary table, or MVIEW for a materialized view. Materialized views are only supported for SQL Anywhere tables in the IQ catalog store.
creator	VARCHAR(128)	The owner of the table.
table_name	VARCHAR(128)	Table on which the lock is held.
index_id	INTEGER	The index ID or NULL

Column	Data type	Description
lock_class	CHAR(8)	String of characters indicating the type of lock: • S – share. • SW – share and write. • EW – exclusive and write. • E – exclusive. • P – phantom. • A – antiphantom. • W – write. All locks listed have one of S, E, EW, or SW, and may also have P, A, or both. Phantom and anti-phantom locks also have a qualifier of T or *: • T – the lock is with respect to a sequential scan. • * – the lock is with respect to all scans. • *nnn* – Index number; the lock is with respect to a particular index. SAP Sybase IQ obtains a share lock before a write lock. If a connection has exclusive lock, share lock does not appear. For write locks, if a connection has all-exclusive, share, and write locks, it is EW.
lock_duration	CHAR(11)	The duration of the lock. One of Transaction, Position, or Connection.
lock_type	CHAR(9)	Value identifying the lock (dependent on the lock class)
row_identifier	UNSIGNED BIGINT	The identifier for the row the lock starts on, or NULL.
row_range	BIGINT	The number of contiguous rows that are locked. Row locks in the RLV store can either be a single row, or a range of rows.

If **sp_iqlocks** cannot find the connection ID or user name of the user who has a lock on a table, it displays a 0 (zero) for the connection ID and `User unavailable` for the user name.

Note: Exclusive, phantom, or antiphantom locks can be placed on IQ catalog store tables, but not on SAP Sybase IQ tables in the IQ main store. Unless you have explicitly taken out locks on a table in the catalog store, you never see these types of locks (or their qualifiers T, *, and *nnn*) in a SAP Sybase IQ database.

Examples

The example shows the **sp_iqlocks** procedure call and its output in the SAP Sybase IQ database. The procedure is called with all default options, so that the output shows all locks, sorted by connection.

```
call sp_iqlocks()
```

```
conn_name         conn_id
user_id  table_type  creator  table_name
=========          =======
=======  ==========  =======  ==========
SQL_DBC_13cd6038  3          DBA      BASE        DBA      rv_locks2
SQL_DBC_13cd6038  3          DBA      BASE        DBA      rv_locks2
SQL_DBC_13cd6038  3          DBA      BASE        DBA      rv_locks2
RVL_CONN_T775     1000000407           BASE        DBA      rv_locks2

index_id  lock_class  lock_duration  lock_type  row_identifier
row_range
========  ==========  =============  =========  ==============
=========
          Schema      Transaction    Shared
          Row         Transaction    Row        1                 4
          Row         Transaction    Row        281474976710656   1
          Table       Transaction    Intent
```

sp_iqmergerlvstore Procedure

Triggers a merge of the row-level versioned (RLV) store with the IQ main store.

Syntax

```
sp_iqmergerlvstore (merge_type, table_name, [ table_owner ])
```

Usage

- If a table name is not specified, all the active data (from all RLV-enabled tables) in the RLV store will be merged into the IQ main store.
- Merge-type can be **BLOCKING | NON-BLOCKING** . (Of the rare cases where you may wish to run a manual merge, almost all would be blocking merges.)
- After performing the merge, the stored procedure will automatically commit the merge transaction.

sp_iqrlvmemory Procedure

Monitors RLV store memory usage per table.

Syntax

```
sp_iqrlvmemory (table_name, [ table_owner ])
```

Privileges
Requires the MONITOR system privilege. Users without the MONITOR system privilege must be granted EXECUTE permission to run the stored procedure.

Usage
Version-specific data, such as version bitmaps and on-demand indexes, are not included in RLV memory accounting. They do not count against the RLV memory limit, and are not reported in sp_iqrlvmemory.

The table owner and table name are optional. Specify the table owner and/or table name to limit the scope.

Description
sp_iqrlvmemory outputs one row per table consuming RLV store memory, with the following output columns:

Column Name	Description
table_id	ID of the table this row represents.
fragments	Number of store fragments for this table.
total	Total RLV store memory used by this table.
data	RLV store memory used for the column fragments for this table.
dictionary	RLV store memory used for the dictionaries for this table.
bitmap	RLV store memory used to store table-level bitmaps.

```
sp_iqrlvmemory 'DBA', 'rlv_table1'
```

sp_iqspaceinfo Procedure

Displays the number of blocks used by each object in the current database and the name of the dbspace in which the object is located.

Syntax

```
sp_iqspaceinfo [ 'main
| [table table-name | index index-name] [...] ']
```

Applies to
Simplex and multiplex.

Privileges

Requires the MANAGE ANY DBSPACE system privilege. Users without the MANAGE ANY DBSPACE system privilege must be granted `EXECUTE` permission.

Description

For the current database, displays the object name, number of blocks used by each object, and the name of the dbspace. **sp_iqspaceinfo** requires no parameters.

The information returned by **sp_iqspaceinfo** is helpful in managing dbspaces.

If run on a multiplex database, the default parameter is **main**, which returns the size of the shared IQ store.

If you supply no parameter, you must have at least one user-created object, such as a table, to receive results.

Example

This output is from the **sp_iqspaceinfo** stored procedure run on the `iqdemo` database. Output for some tables and indexes are removed from this example.

```
             Name                                NBlocks    dbspace_name

Contacts                                        19       IQ_SYSTEM_MAIN
SalesOrderItems.DBA.ASIQ_IDX_T205_C5_FP          56      IQ_SYSTEM_MAIN
Contacts.DBA.ASIQ_IDX_T206_C10_FP                55      IQ_SYSTEM_MAIN
Contacts.DBA.ASIQ_IDX_T206_C1_FP                 61      IQ_SYSTEM_MAIN
...
Contacts.DBA.ASIQ_IDX_T206_C9_FP                 55      IQ_SYSTEM_MAIN
Contacts.DBA.ASIQ_IDX_T206_I11_HG                19      IQ_SYSTEM_MAIN
Customers                                       20       IQ_SYSTEM_MAIN
Customers.DBA.ASIQ_IDX_T207_C1_FP                61      IQ_SYSTEM_MAIN
Customers.DBA.ASIQ_IDX_T207_C2_FP                55      IQ_SYSTEM_MAIN
...
Customers.DBA.ASIQ_IDX_T207_I10_HG               19      IQ_SYSTEM_MAIN
...
```

sp_iqspaceused Procedure

Shows information about space available and space used in the IQ store, IQ temporary store, RLV store, and IQ global and local shared temporary stores.

Syntax

```
sp_iqspaceused(out mainKB            unsigned bigint,
               out mainKBUsed        unsigned bigint,
               out tempKB            unsigned bigint,
               out tempKBUsed        unsigned bigint,
               out shTempTotalKB     unsigned bigint,
               out shTempTotalKBUsed unsigned bigint,
               out shTempLocalKB     unsigned bigint,
```

```
                out shTempLocalKBUsed unsigned bigint,
                out rlvLogKB          unsigned bigint,
                out rlvLogKBUsed      unsigned bigint)
```

Applies to
Simplex and multiplex.

Privileges
Requires the ALTER DATABASE, MANAGE ANY DBSPACE, or MONITOR system privileges. Users without one of these system privileges must be granted `EXECUTE` permission.

Usage
sp_iqspaceused returns several values as unsigned bigint out parameters. This system stored procedure can be called by user-defined stored procedures to determine the amount of main, temporary, and RLV store space in use.

Description
sp_iqspaceused returns a subset of the information provided by **sp_iqstatus**, but allows the user to return the information in SQL variables to be used in calculations.

If run on a multiplex database, this procedure applies to the server on which it runs. Also returns space used on IQ_SHARED_TEMP.

Column Name	Description
mainKB	The total IQ main store space, in kilobytes.
mainKBUsed	The number of kilobytes of IQ main store space used by the database. Secondary multiplex nodes return '(Null)'.
tempKB	The total IQ temporary store space, in kilobytes.
tempKBUsed	The number of kilobytes of total IQ temporary store space in use by the database.
shTempTotalKB	The total IQ global shared temporary store space, in kilobytes.
shTempLocalKB	The total IQ local shared temporary store space, in kilobytes.
shTempLocalKBUsed	The number of kilobytes of IQ local shared temporary store space in use by the database.
rlvLogKB	The total RLV store space, in kilobytes.

Column Name	Description
rlvLogKBUsed	The number of kilobytes of RLV store space in use by the database.

Example

sp_iqspaceused requires seven output parameters. Create a user-defined stored procedure **myspace** that declares the seven output parameters, then calls **sp_iqspaceused**:

```
create or replace procedure dbo.myspace()
begin
     declare mt unsigned bigint;
     declare mu unsigned bigint;
     declare tt unsigned bigint;
     declare tu unsigned bigint;
     declare gt unsigned bigint;
     declare gu unsigned bigint;
     declare lt unsigned bigint;
     declare lu unsigned bigint;
     declare rvlt unsigned bigint;
     declare rvlu unsigned bigint;
    call sp_iqspaceused(mt,mu,tt,tu,gt,gu,lt,lu, rvlt, rvlu);
     select cast(mt/1024 as unsigned bigint) as mainMB,
     cast(mu/1024 as unsigned bigint) as mainusedMB,
     mu*100/mt as mainPerCent,
     cast(tt/1024 as unsigned bigint) as tempMB,
     cast(tu/1024 as unsigned bigint) as tempusedMB,
     tu*100/tt as tempPerCent,
     cast(gt/1024 as unsigned bigint) as shTempTotalKB,
     if gu=0 then 0 else gu*100/gt endif as globalshtempPerCent,
     cast(lt/1024 as unsigned bigint) as shTempLocalMB,
     cast(lu/1024 as unsigned bigint) as shTempLocalKBUsed,
     if lt=0 then 0 else lu*100/lt endif as localshtempPerCent,
     cast(rvlt/1024 as unsigned bigint) as rlvLogKB,
     cast(rvlu/1024 as unsigned bigint) as rlvLogKBUsed;
end
```

To display the output of **sp_iqspaceused**, execute **myspace**:

```
myspace
```

sp_iqstatistics Procedure

Returns serial number, name, description, value, and unit specifier for each available statistic, or a specified statistic.

Syntax

sp_iqstatistics [*stat_name*]

Usage

Parameter	Description
stat_name	(Optional) VARCHAR parameter specifying the name of a statistic.

Privileges

Requires the MANAGE ANY STATISTICS system privilege. Users without MANAGE ANY STATISTICS system privilege must be granted EXECUTE permission to run the stored procedure.

Description

When **stat_name** is provided, **sp_iqstatistics** returns one row for the given statistic, or zero rows if the name is invalid. When invoked without any parameter, **sp_iqstatistics** returns all statistics.

Result Set

Column name	Data type	Description
stat_num	UNSIGNED INTEGER	Serial number of a statistic
stat_name	VARCHAR(255)	Name of statistic
stat_desc	VARCHAR(255)	Description of statistic
stat_value	LONG VARCHAR	Value of statistic
stat_unit	VARCHAR(128)	Unit specifier

The following statistics may be returned:

stat_num	stat_name	stat_desc	stat_unit
0	CpuTotalTime	Total CPU time in seconds consumed by the SAP Sybase IQ server since last server startup	Second
1	CpuUserTime	CPU user time in seconds consumed by the SAP Sybase IQ server since last server startup	Second

stat_num	stat_name	stat_desc	stat_unit
2	CpuSystemTime	CPU system time in seconds consumed by the SAP Sybase IQ server since last server startup	Second
3	ThreadsFree	Number of SAP Sybase IQ threads free	N/A
4	ThreadsInUse	Number of SAP Sybase IQ threads in use	N/A
5	MemoryAllocated	Allocated memory in megabytes	MB
6	MemoryMaxAllocated	Max allocated memory in megabytes	MB
7	MainCacheCurrentSize	Main cache current size in megabytes	MB
8	MainCacheFinds	Main cache total number of lookup requests	N/A
9	MainCacheHits	Main cache total number of hits	N/A
10	MainCachePagesPinned	Main cache number of pages pinned	Page
11	MainCachePagesPinnedPercentage	Percentage of main cache pages pinned	%
12	MainCachePagesDirtyPercentage	Percentage of main cache pages dirtied	%
13	MainCachePagesInUsePercentage	Percentage of main cache pages in use	%
14	TempCacheCurrentSize	Temporary cache current size in megabytes	MB
15	TempCacheFinds	Temporary cache total number of lookup requests	N/A
16	TempCacheHits	Temporary cache total number of hits	N/A
17	TempCachePagesPinned	Temporary cache number of pages pinned	Page
18	TempCachePagesPinnedPercentage	Percentage of temporary cache pages pinned	%
19	TempCachePagesDirtyPercentage	Percentage of temporary cache pages dirtied	%
20	TempCachePagesInUsePercentage	Percentage of temporary cache pages in use	%

stat_num	stat_name	stat_desc	stat_unit
21	MainStoreDiskReads	Number of kilobytes read from main store	KB
22	MainStoreDiskWrites	Number of kilobytes written to main store	KB
23	TempStoreDiskReads	Number of kilobytes read from main store	KB
24	TempStoreDiskWrites	Number of kilobytes written to main store	KB
25	ConnectionsTotalConnections	Total number of connections since server startup	N/A
26	ConnectionsTotalDisonnections	Total number of disconnections since server startup	N/A
27	ConnectionsActive	Number of active connections	N/A
28	OperationsWaiting	Number of operations waiting for SAP Sybase IQ resource governor	N/A
29	OperationsActive	Number of active concurrent operations admitted by SAP Sybase IQ resource governor	N/A
30	OperationsActiveLoadTableStatements	Number of active LOAD TABLE statements	N/A

Examples

Displays a single statistic, the total CPU time:

```
sp_iqstatistics 'CpuTotalTime'
```

Displays all statistics for MainCache%:

```
SELECT * from sp_iqstatistics() WHERE stat_name LIKE 'MainCache%'
```

stat_num	stat_name	stat_desc	stat_value	stat_unit
7	MainCacheCurrentSize	Main cache current size in megabytes	64	mb

stat_num	stat_name	stat_desc	stat_value	stat_unit
8	MainCacheFinds	Main cache total number of lookup requests	95303	
9	MainCacheHits	Main cache total number of hits	95283	
10	MainCachePagesPinned	Main cache number of pages pinned	0	page
11	MainCachePagesPinnedPercentage	Percentage of main cache pages pinned	0	%
12	MainCachePagesDirtyPercentage	Percentage of main cache pages dirtied	0.39	%
13	MainCachePagesInUsePercentage	Percentage of main cache pages in use	4.44	%

sp_iqstatus Procedure

Displays a variety of SAP Sybase IQ status information about the current database.

Syntax

```
sp_iqstatus
```

Applies to

Simplex and multiplex.

Privileges

Requires the ALTER DATABASE, MANAGE ANY DBSPACE, MONITOR, or SERVER OPERATOR system privilege. Users without one of these system privileges must be granted `EXECUTE` permission.

Description

Shows status information about the current database, including the database name, creation date, page size, number of dbspace segments, block usage, buffer usage, I/O, backup information, and so on.

sp_iqstatus displays an out-of-space status for main and temporary stores. If a store runs into an out-of-space condition, **sp_iqstatus** shows `Y` in the store's out-of-space status display value.

Memory used by the row-level versioning (RLV) store can be monitored with **sp_iqstatus**. The **RLV memory limit** row displays the memory limit as specified by the `-iqrlvmem` server option, or the sa_server_option `rlv_memory_mb`. The `RLV memory used` row displays the amount of memory used by the RLV store.

sp_iqspaceused returns a subset of the same information as provided by **sp_iqstatus**, but allows the user to return the information in SQL variables to be used in calculations.

To display space that can be reclaimed by dropping connections, use **sp_iqstatus** and add the results from the two returned rows:

```
(DBA)> select * from sp_iqstatus() where name like '%Versions:%'
Execution time: 6.25 seconds
Name                 Value
---------------------------
Other Versions: 2 = 1968Mb
Active Txn Versions: 1 = C:2175Mb/D:2850Mb

(First 2 rows)
```

The above example output shows that one active write transaction created 2175MB and destroyed 2850 MB of data. The total data consumed in transactions and not yet released is 4818MB, or 1968MB + 2850MB = 4818MB.

sp_iqstatus omits blocks that will be deallocated at the next checkpoint. These blocks do however, appear in **sp_iqdbspace** output as type X.

In a multiplex, this procedure also lists information about the shared IQ store and IQ temporary store. If **sp_iqstatus**shows a high percentage of main blocks in use on a multiplex server, run **sp_iqversionuse** to see which versions are being used and the amount of space that can be recovered by releasing versions.

Example

Note: This example includes a sample user dbspace named `iq_main` that may not be present in your own databases.

The following output is from the **sp_iqstatus** stored procedure:

```
SAP Sybase IQ (TM)                Copyright (c) 1992-2013 by Sybase,
Inc.
                                                All rights reserved.
Version:                                        16.0.0.160/120507/D/ELAN/
Sun_x64/OS 5.10/
                                                64bit/2012-05-07 17:36:36
Time Now:                                       2013-05-16 09:53:13.590
Build Time:                                     2013-05-07 17:36:36
File Format:                                    23 on 03/18/1999
```

```
Server mode:                              IQ Multiplex Coordinator Server
Catalog Format:                              2
Stored Procedure Revision:                   1
Page Size:                                   131072/8192blksz/16bpp
Number of Main DB Files :                    3
Main Store Out Of Space:                     N
Number of Shared Temp DB Files:              0
Shared Temp Store Out Of Space:              N
Number of Local Temp DB Files :              1
Local Temp Store Out Of Space:               N
DB Blocks: 1-640000                          IQ_SYSTEM_MAIN
DB Blocks: 1045440-130101439                 iqmain1
DB Blocks: 2090880-2346879                   iqmain2
Local Temp Blocks: 1-384000                  IQ_SYSTEM_TEMP
Create Time:                                 2013-05-08 15:54:15.549
Update Time:                                 2013-05-16 09:53:00.077
Local Temp Blocks:  1-1600                   IQ_SYSTEM_TEMP
Create Time:                                 2013-05-08 15:54:15.549
Update Time:                                 2013-05-16 09:53:00.077
Main IQ Buffers:                             510, 64Mb
Temporary IQ Buffers:                        510, 64Mb
Main IQ Blocks Used:                      157379 of 1126400, 13%=1229Mb,
Max Block#: 2128363
Shared Temporary IQ Blocks Used:            0 of 0, 0%=0Mb, Max Block#: 0
Local Temporary IQ Blocks Used:              81 of 358400, 0%=0Mb, Max
Block#: 81
Main Reserved Blocks Available:              25600 of 25600, 100%=200Mb
Shared Temporary Reserved Blocks Available:  0 of 0, 0%=0Mb
Local Temporary Reserved Blocks Available:   25600 of 25600,
100%=200Mb
IQ Dynamic Memory:                           Current: 178mb, Max: 178mb
Main IQ Buffers:                             Used: 99, Locked: 0
Temporary IQ Buffers:                        Used: 5, Locked: 0
Main IQ I/O:                              I: L60904/P29 O: C5463/D11343/
P9486
                                             D:5450 C:51.3
Temporary IQ I/O:                         I: L12526/P0 O: C165/D319/P157
D:160 C:100.0
Other Versions:                              6 = 0Mb
Active Txn Versions:                         0 = C:0Mb/D:0Mb
Last Full Backup ID:                         0
Last Full Backup Time:
Last Backup ID:
Last Backup Type:                            None
Last Backup Time:
DB Updated:                                  1
Blocks in next ISF Backup:                   0 Blocks: =0Mb
Blocks in next ISI Backup:                   0 Blocks: =0Mb
Main Tlvlog Size:                         Pages: 1, Recs: 193, Replays:
0/0
DB File Encryption Status:                   OFF
```

The following is a key to understanding the `Main IQ I/O` and `Temporary IQ I/O` output codes:

- `I:` Input
- `L:` Logical pages read (“Finds”)
- `P:` Physical pages read
- `O:` Output
- `C:` Pages created
- `D:` Pages dirtied
- `P:` Physically written
- `D:` Pages destroyed
- `C:` Compression ratio

sp_iqsysmon Procedure

Monitors multiple components of SAP Sybase IQ, including the management of buffer cache, memory, threads, locks, I/O functions, and CPU utilization.

Batch Mode Syntax

```
sp_iqsysmon start_monitor
sp_iqsysmon stop_monitor [, "section(s)" ]
or
sp_iqsysmon "time-period" [, "section(s)" ]
```

File Mode Syntax

```
sp_iqsysmon start_monitor, 'filemode' [, "monitor-options" ]
sp_iqsysmon stop_monitor
```

Privileges

Requires the MONITOR system privilege. Users without the MONITOR system privilege must be granted `EXECUTE` permission to run the stored procedure.

Batch Mode Usage

Parameter	Description
start_monitor	Starts monitoring.
stop_monitor	Stops monitoring and displays the report.
time-period	The time period for monitoring, in the form HH:MM:SS.

Parameter	Description
section(s)	The abbreviation for one or more sections to be shown by **sp_iqsysmon**. If you specify more than one section, separate the section abbreviations using spaces, and enclose the list in single or double quotes. The default is to display all sections. For sections related to the IQ store, you can specify main or temporary store by prefixing the section abbreviation with “m” or “t”, respectively. Without the prefix, both stores are monitored. For example, if you specify “mbufman”, only the IQ main store buffer manager is monitored. If you specify “mbufman tbufman” or “bufman”, both the main and temporary store buffer managers are monitored.

Report Section or IQ Component	Abbreviation
Buffer manager	(m/t)bufman
Buffer pool	(m/t)bufpool
Prefetch management	(m/t)prefetch
Free list management	(m/t)freelist
Buffer allocation	(m/t)bufalloc
Memory management	memory
Thread management	threads
CPU utilization	cpu
Transaction management	txn
Server context statistics	server
Catalog statistics	catalog

Note: The SAP Sybase IQ components Disk I/O and Lock Manager are not currently supported by **sp_iqsysmon**.

File Mode Usage

Parameter	Description
start_monitor	Starts monitoring.
stop_monitor	Stops monitoring and writes the remaining output to the log file.
filemode	Specifies that **sp_iqsysmon** is running in file mode. In file mode, a sample of statistics appear for every interval in the monitoring period. By default, the output is written to a log file named *dbname.connid-iqmon*. Use the **file_suffix** option to change the suffix of the output file. See the *monitor_options* parameter for a description of the **file_suffix** option.
monitor_options	The *monitor_options* string

The *monitor_options* string can include one or more options:

Table 1. monitor_options string options

monitor_options String Option	Description
-interval *seconds*	Specifies the reporting interval, in seconds. A sample of monitor statistics is output to the log file after every interval. The default is every 60 seconds, if the **-interval** option is not specified. The minimum reporting interval is 2 seconds. If the interval specified for this option is invalid or less than 2 seconds, the interval is set to 2 seconds. The first display shows the counters from the start of the server. Subsequent displays show the difference from the previous display. You can usually obtain useful results by running the monitor at the default interval of 60 seconds during a query with performance problems or during a time of day that generally has performance problems. A very short interval may not provide meaningful results. The interval should be proportional to the job time; 60 seconds is usually more than enough time.

monitor_options String Option	Description
-file_suffix *suffix*	Creates a monitor output file named `dbname.connid-suffix`. If you do not specify the **-file_suffix** option, the suffix defaults to `iqmon`. If you specify the **-file_suffix** option and do not provide a suffix or provide a blank string as a suffix, no suffix is used.
-append or **-truncate**	Directs **sp_iqsysmon** to append to the existing output file or truncate the existing output file, respectively. Truncate is the default. If both options are specified, the option specified later in the string takes precedence.
-section *section(s)*	Specifies the abbreviation of one or more sections to write to the monitor log file. The default is to write all sections. The abbreviations specified in the sections list in file mode are the same abbreviations used in batch mode. When more than one section is specified, spaces must separate the section abbreviations. If the **-section** option is specified with no sections, none of the sections are monitored. An invalid section abbreviation is ignored and a warning is written to the IQ message file.

Usage Syntax Examples

Syntax	Result
sp_iqsysmon start_monitor sp_iqsysmon stop_monitor	Starts the monitor in batch mode and displays all sections for the main and temporary stores
sp_iqsysmon start_monitor sp_iqsysmon stop_monitor “mbufman mbufpool”	Starts the monitor in batch mode and displays the Buffer Manager and Buffer Pool statistics for themain store
sp_iqsysmon “00:00:10”, “mbufpool memory”	Runs the monitor in batch mode for 10 seconds and displays the consolidated statistics at the end of the time period
sp_iqsysmon start_monitor, ‘filemode’, “-interval 5 -sections mbufpool memory” sp_iqsysmon stop_monitor	Starts the monitor in file mode and writes statistics for Main Buffer Pool and Memory Manager to the log file every 5 seconds

Description

The **sp_iqsysmon** stored procedure monitors multiple components of SAP Sybase IQ, including the management of buffer cache, memory, threads, locks, I/O functions, and CPU utilization.

The **sp_iqsysmon** procedure supports two modes of monitoring:

- Batch mode –

  In batch mode, **sp_iqsysmon** collects the monitor statistics for the period between starting and stopping the monitor or for the time period specified in the *time-period* parameter. At the end of the monitoring period, **sp_iqsysmon** displays a list of consolidated statistics.

  sp_iqsysmon in batch mode is similar to the SAP® Sybase Adaptive Server Enterprise procedure **sp_sysmon**.
- File mode –

  In file mode, **sp_iqsysmon** writes the sample statistics in a log file for every interval period between starting and stopping the monitor.

  The first display in file mode shows the counters from the start of the server. Subsequent displays show the difference from the previous display.

  sp_iqsysmon in file mode is similar to the **IQ UTILITIES** command **START MONITOR** and **STOP MONITOR** interface.

Batch Mode Examples

Prints monitor information after 10 minutes:

```
sp_iqsysmon "00:10:00"
```

Prints only the Memory Manager section of the **sp_iqsysmon** report after 5 minutes:

```
sp_iqsysmon "00:05:00", memory
```

Starts the monitor, executes two procedures and a query, stops the monitor, then prints only the Buffer Manager section of the report:

```
sp_iqsysmon start_monitor
go
execute proc1
go
execute proc2
go
select sum(total_sales) from titles
go
sp_iqsysmon stop_monitor, bufman
go
```

Prints only the Main Buffer Manager and Main Buffer Pool sections of the report after 20 minutes:

```
sp_iqsysmon "00:02:00", "mbufman mbufpool"
```

File Mode Examples

Truncates and writes information to the log file every 2 seconds between starting the monitor and stopping the monitor:

```
sp_iqsysmon start_monitor, 'filemode', '-interval 2'
.
.
.
sp_iqsysmon stop_monitor
```

Appends output for only the Main Buffer Manager and Memory Manager sections to an ASCII file with the name `dbname.connid-testmon`. For the database `iqdemo`, writes results in the file `iqdemo.2-testmon`:

```
sp_iqsysmon start_monitor, 'filemode',
"-file_suffix testmon -append -section mbufman memory"
.
.
.
sp_iqsysmon stop_monitor
```

Example

Run the monitor in batch mode for 10 seconds and display the consolidated statistics at the end of the time period

```
sp_iqsysmon "00:00:10", "mbufpool memory"

==============================
Buffer Pool (Main)
==============================
STATS-
NAME          TOTAL  NONE  BTREEV  BTREEF  BV  VDO  DBEXT  DBID  SORT
MovedToMRU        0     0       0       0   0    0      0     0     0
MovedToWash       0     0       0       0   0    0      0     0     0
RemovedFromLRU    0     0       0       0   0    0      0     0     0
RemovedFromWash   0     0       0       0   0    0      0     0     0
RemovedInScanMode 0     0       0       0   0    0      0     0     0

STORE  GARRAY  BARRAY  BLKMAP  HASH  CKPT  BM  TEST  CMID  RIDCA  LOB
0      0       0       0       0     0     0   0     0     0      0
0      0       0       0       0     0     0   0     0     0      0
0      0       0       0       0     0     0   0     0     0      0
0      0       0       0       0     0     0   0     0     0      0
0      0       0       0       0     0     0   0     0     0      0

STATS-NAME                    VALUE
Pages                         127   ( 100.0 %)
InUse                         4     ( 3.1 %)
Dirty                         1     ( 0.8 %)
Pinned                        0     ( 0.0 %)
Flushes                       0
FlushedBufferCount            0
GetPageFrame                  0
GetPageFrameFailure           0
```

```
GotEmptyFrame                   0
Washed                          0
TimesSweepersWoken              0

washTeamSize                    0
WashMaxSize                     26    ( 20.5 %)
washNBuffers                    4      ( 3.1 %)
washNDirtyBuffers               1      ( 0.8
%)
                   washSignalThreshold            3       ( 2.4 %)
washNActiveSweepers             0
washIntensity                   1

==============================
Memory Manager
==============================
STATS-NAME                    VALUE
MemAllocated                  43616536      (  42594 KB)
MemAllocatedMax               43735080      (  42710 KB)
MemAllocatedEver              0             (      0 KB)
MemNAllocated                 67079
MemNAllocatedEver             0
MemNTimesLocked               0
MemNTimesWaited               0             (   0.0 %)
```

sp_iqtable Procedure

Displays information about tables in the database.

Syntax1

```
sp_iqtable ( [ table_name ],[table_owner ],[table_type ] )
```

Syntax2

```
sp_iqtable [table_name='tablename'],
[table_owner='tableowner' ],[table_type='tabletype' ]
```

Privileges

No specific system privileges are required to run the procedure.

Usage: Syntax1

If you do not specify either of the first two parameters, but specify the next parameter in the sequence, you must substitute NULL for the omitted parameters. For example, `sp_iqtable NULL,NULL,TEMP` and `sp_iqtable NULL,dbo,SYSTEM`.

Note: The *table_type* values ALL and VIEW must be enclosed in single quotes in Syntax1.

Usage: Syntax2

The parameters can be specified in any order. Enclose them in single quotes.

The allowed values for the *table_type* parameter:

table_type value	Information displayed
TEMP	Global temporary tables
VIEW	Views
ALL	IQ tables, global temporary tables, and views
any other value	IQ tables

Description

Specifying one parameter returns only the tables that match that parameter. Specifying more than one parameter filters the results by all of the parameters specified. Specifying no parameters returns all SAP Sybase IQ tables in the database. There is no method for returning the names of local temporary tables.

Column name	Description
table_name	The name of the table.
table_type	BASE – a base table. MAT VIEW - a materialized view. (SA tables only) GBL TEMP - a global temporary table. PARTITION - a table partition (this table is for internal use only and cannot be used by SAP Sybase IQ users). VIEW – a view.
table_owner	The owner of the table
server_type	IQ – an object created in the IQ store. SA – an object created in the SA store. All views are created in the SA store.
location	TEMP – IQ temporary store. MAIN – IQ store. SYSTEM – catalog store.
dbspace_id	Number that identifies the dbspace.
isPartitioned	'Y' if the column belongs to a partitioned table and has one or more partitions whose dbspace is different from the table partition's dbspace, 'N' if the column's table is not partitioned or each partition of the column resides in the same dbspace as the table partition.
remarks	User comments added with the **COMMENT** statement.

Column name	Description
table_constraints	Constraints against the table.
isRLV	Indicates if the table is RLV-enabled.

sp_iqtable Procedure Examples

sp_iqtable output examples.

The following variations in syntax both return information about the table `Departments`:

```
sp_iqtable ('Departments')
```

```
sp_iqtable table_name='Departments'
```

Table_name	Table_type	Table_owner
Departments	BASE	GROUPO

Server_type	location	dbspace_id
IQ	Main	16387

isPartitioned	Remarks	table_constraints
N	contains the names and heads of the various departments in the sporting goods company	(NULL)

The following variations in syntax both return all tables that are owned by table owner `GROUPO`:

```
sp_iqtable NULL,GROUPO
```

```
sp_iqtable table_owner='GROUPO'
```

Table_name	Table_type	Table_owner	Server_type	location
Contacts	BASE	GROUPO	IQ	Main
Customers	BASE	GROUPO	IQ	Main
Departments	BASE	GROUPO	IQ	Main
Employees	BASE	GROUPO	IQ	Main
FinancialCodes	BASE	GROUPO	IQ	Main
FinancialData	BASE	GROUPO	IQ	Main

Table_name	Table_type	Table_owner	Server_type	location
Products	BASE	GROUPO	IQ	Main
SalesOrders	BASE	GROUPO	IQ	Main
SalesOrderItems	BASE	GROUPO	IQ	Main

dbspace_id	isPartitioned	Remarks	table_constraints
16387	N	names, addresses, and telephone numbers of all people with whom the company wishes to retain contact information	(NULL)
16387	N	customers of the sporting goods company	(NULL)
16387	N	contains the names and heads of the various departments in the sporting goods company	(NULL)
16387	N	contains information such as names, salary, hire date and birthday	(NULL)
16387	N	types of revenue and expenses that the sporting goods company has	(NULL)
16387	N	revenues and expenses of the sporting goods company	(NULL)
16387	N	products sold by the sporting goods company	(NULL)
16387	N	individual items that make up the sales orders	(NULL)
16387	N	sales orders that customers have submitted to the sporting goods company	(NULL)

sp_iqtablesize Procedure

Returns the size of the specified table.

Syntax

```
sp_iqtablesize ( table_owner.table_name )
```

Privileges

User must be a table owner or have the MANAGE ANY DBSPACE or ALTER ANY TABLE system privilege. Users without one of these system privileges must be granted `EXECUTE` permission to run the stored procedure.

Description

Returns the total size of the table in KBytes and NBlocks (IQ blocks). Also returns the number of pages required to hold the table in memory, and the number of IQ pages that are compressed when the table is compressed (on disk). You must specify the *table_name* parameter with this procedure. If you are the owner of *table_name*, then you do not have to specify the *table_owner* parameter.

Column name	Description
Ownername	Name of owner
Tablename	Name of table
Columns	Number of columns in the table
KBytes	Physical table size in KB
Pages	Number of IQ pages needed to hold the table in memory
CompressedPages	Number of IQ pages that are compressed, when the table is compressed (on disk)
NBlocks	Number of IQ blocks
RlvLogPages	Number of IQ pages needed to hold the RLV table log information on disk
RlvLogKBytes	Size of the RLV table log, in kilobytes.

`Pages` is the total number of IQ pages for the table. The unit of measurement for pages is IQ page size. All in-memory buffers (buffers in the IQ buffer cache) are the same size.

IQ pages on disk are compressed. Each IQ page on disk uses 1 to 16 blocks. If the IQ page size is 128KB, then the IQ block size is 8KB. In this case, an individual on-disk page could be 8, 16, 24, 32, 40, 48, 56, 64, 72, 80, 88, 96, 104, 112, 120, or 128 KB.

If you divide the `KBytes` value by page size, you see the average on-disk page size.

Note: SAP Sybase IQ always reads and writes an entire page, not blocks. For example, if an individual page compresses to 88K, then IQ reads and writes the 88K in one I/O. The average page is compressed by a factor of 2 to 3.

`NBlocks` is `Kbytes` divided by IQ block size.

`CompressedPages` is the number of pages that are compressed. For example, if `Pages` is 1000 and `CompressedPages` is 992, this means that 992 of the 1000 pages are compressed. `CompressedPages` divided by `Pages` is usually near 100%, because most pages compress. An empty page is not compressed, since SAP Sybase IQ does not write empty pages. IQ pages compress well, regardless of the fullness of the page.

Example

```
call sp_iqtablesize ('dba.t1')
```

Owner-name	Table-name	Col-umns	KBytes	Pa-ges	Compres-sedPages	NBlocks	RlvLog-Pag-es	RlvLog-Bytes
DBA	t1	3	192	5	4	24	96	12288

sp_iqtransaction Procedure

Shows information about transactions and versions.

Syntax

```
sp_iqtransaction
```

Applies to

Simplex and multiplex.

Privileges

Requires the MONITOR system privilege. Users without the MONITOR system privilege must be granted `EXECUTE` permission to run the stored procedure.

Description

sp_iqtransaction returns a row for each transaction control block in the SAP Sybase IQ transaction manager. The columns Name, Userid, and ConnHandle are the connection properties **Name**, **Userid**, and **Number**, respectively. Rows are ordered by TxnID.

sp_iqtransaction output does not include connections without transactions in progress. To include all connections, use **sp_iqconnection**.

Note: Although you can use **sp_iqtransaction** to identify users who are blocking other users from writing to a table, **sp_iqlocks** is a better choice for this purpose.

Column Name	Description
Name	The name of the application.
Userid	The user ID for the connection.

Column Name	Description
TxnID	The transaction ID of this transaction control block. The transaction ID is assigned during **begin transaction**. It appears in the `.iqmsg` file by the BeginTxn, CmtTxn, and PostCmtTxn messages, and is the same as the Txn ID Seq that is logged when the database is opened.
CmtID	The ID assigned by the transaction manager when the transaction commits. For active transactions, the CmtID is zero.
VersionID	For simplex and multiplex nodes, a value of 0 indicates that the transaction is unversioned, and the VersionID has not been assigned. For the multiplex coordinator, the VersionID is assigned after the transaction establishes table locks. Multiplex secondary servers receive the VersionID from the coordinator. The VersionID is used internally by the SAP Sybase IQ in-memory catalog and the IQ transaction manager to uniquely identify a database version to all nodes within a multiplex database.
State	The state of the transaction control block. This variable reflects internal SAP Sybase IQ implementation details and is subject to change in the future. Currently, transaction states are `NONE`, `ACTIVE`, `ROLLING_BACK`, `ROLLED_BACK`, `COMMITTING`, `COMMITTED`, and `APPLIED`. `NONE`, `ROLLING_BACK`, `ROLLED_BACK`, `COMMITTING` and `APPLIED` are transient states with a very small life span. `ACTIVE` indicates that the transaction is active. `COMMITTED` indicates that the transaction has completed and is waiting to be `APPLIED`, at which point a version that is invisible to any transaction is subject to garbage collection. Once the transaction state is `ROLLED_BACK`, `COMMITTED`, or `APPLIED`, ceases to own any locks other than those held by open cursors.
ConnHandle	The ID number of the connection.
IQConnID	The ten-digit connection ID that is included as part of all messages in the `.iqmsg` file. This is a monotonically increasing integer unique within a server session.
MainTableKBCr	The number of kilobytes of IQ store space created by this transaction.
MainTableKBDr	The number of kilobytes of IQ store space dropped by this transaction, but which persist on disk in the store because the space is visible in other database versions or other savepoints of this transaction.

Column Name	Description
TempTableKBCr	The number of kilobytes of IQ temporary store space created by this transaction for storage of IQ temporary table data.
TempTableKBDr	The number of kilobytes of IQ temporary table space dropped by this transaction, but which persist on disk in the IQ temporary store because the space is visible to IQ cursors or is owned by other savepoints of this transaction.
TempWorkSpaceKB	For ACTIVE transactions, a snapshot of the work space in use at this instant by this transaction, such as sorts, hashes, and temporary bitmaps. The number varies depending on when you run **sp_iqtransaction**. For example, the query engine might create 60MB in the temporary cache but release most of it quickly, even though query processing continues. If you run **sp_iqtransaction** after the query finishes, this column shows a much smaller number. When the transaction is no longer active, this column is zero. For ACTIVE transactions, this column is the same as the TempWorkSpaceKB column of **sp_iqconnection**.
TxnCreateTime	The time the transaction began. All SAP Sybase IQ transactions begin implicitly as soon as an active connection is established or when the previous transaction commits or rolls back.
CursorCount	The number of open SAP Sybase IQ cursors that reference this transaction control block. If the transaction is ACTIVE, it indicates the number of open cursors created within the transaction. If the transaction is COMMITTED, it indicates the number of hold cursors that reference a database version owned by this transaction control block.
SpCount	The number of savepoint structures that exist within the transaction control block. Savepoints may be created and released implicitly. Therefore, this number does not indicate the number of user-created savepoints within the transaction.
SpNumber	The active savepoint number of the transaction. This is an implementation detail and might not reflect a user-created savepoint.
MPXServerName	Indicates if an active transaction is from an internode communication (INC) connection. If from INC connection, the value is the name of the multiplex server where the transaction originates. NULL if not from an INC connection. Always NULL if the transaction is not active.
GlobalTxnID	The global transaction ID associated with the current transaction, 0 (zero) if none.

Column Name	Description
VersioningType	The snapshot versioning type of the transaction; either table-level (the default), or row-level. Row-level snapshot versioning (RLV) applies only to RLV-enabled tables. Once a transaction is started, this value cannot change.
Blocking	Indicates if connection blocking is enabled (True) or disabled (False). You set connection blocking using the **BLOCKING** database option. If true, the transaction blocks, meaning it waits for a conflicting lock to release before it attempts to retry the lock request.
BlockingTimeout	Indicates the time, in milliseconds, a transaction waits for a locking conflict to clear. You set the timeout threshold using the **BLOCKING_TIMEOUT** database option. A value of 0 (default) indicates that the transaction waits indefinitely.

Example

Example **sp_iqtransaction** output:

```
Name   Userid  TxnID  CmtID VersionID    State    ConnHandle  IQConnID
====== ====== ====== ====== =========  ========== ===========
========
red2      DBA  10058  10700     10058  Active     419740283       14


MainTableKBCr      MainTableKBDr    TempTableKBCr TempTableKBDr
============= ================== ================ =============
           0                  0            65824             0


TempWorkSpaceKB TxnCreateTime               CursorCount SpCount
SpNumber
==============
=======================     =========== ======= ========
       0        2013-03-26 13:17:27.612            1       3        2


MPXServerName  GlobalTxnID   VersioningType  Blocking
BlockingTimeout
=============  ===========   ==============  ========
===============
        (NULL)          0     Row-level       True
0
```

sp_iqwho Procedure

Displays information about all current users and connections, or about a particular user or connection.

Syntax

```
sp_iqwho [ { connhandle | user-name } [, arg-type ] ]
```

Privileges

Requires the DROP CONNECTION, MONITOR, and SERVER OPERATOR system privileges. Users without these system privileges must be granted `EXECUTE` permission to run the stored procedure.

Description

The **sp_iqwho** stored procedure displays information about all current users and connections, or about a particular user or connection.

Column name	Description
ConnHandle	The SA connection handle
IQConnID	The SAP Sybase IQ specific connection ID
Userid	The name of the user that opened the connection “ConnHandle”
BlockedOn	The connection on which a particular connection is blocked; 0 if not blocked on any connection
BlockUserid	The owner of the blocking connection; NULL if there is no blocking connection
ReqType	The type of the request made through the connection; DO_NOTHING if no command is issued
IQCmdType	The type of SAP Sybase IQ command issued from the connection; NONE if no command is issued
IQIdle	The time in seconds since the last SAP Sybase IQ command was issued through the connection; in case of no last SAP Sybase IQ command, the time since ‘01-01-2000’ is displayed
SAIdle	The time in seconds since the last SA request was issued through the connection; in case of no last SA command, the time since ‘01-01-2000’ is displayed
IQCursors	The number of active cursors in the connection; 0 if no cursors
IQThreads	The number of threads with the connection. At least one thread is started as soon as the connection is opened, so the minimum value for IQThreads is 1.
TempTableSpaceKB	The size of temporary table space in kilobytes; 0 if no temporary table space is used
TempWorkSpaceKB	The size of temporary workspace in kilobytes; 0 if no temporary workspace is used

Table 2. Mapping of sp_who and sp_iqwho columns

sp_who column	sp_iqwho column
fid	Family to which a lock belongs; omitted, as not applicable to SAP Sybase IQ
spid	ConnHandle, IQConnID
status	IQIdle, SAIdle
loginame	Userid
orignamе	User alias; omitted, as not applicable to SAP Sybase IQ
hostname	Name of the host on which the server is running; currently not supported
blk_spid	BlockedOn
dbname	Omitted, as there is one server and one database for SAP Sybase IQ and they are the same for every connection
cmd	ReqType, IQCmdType
block_xloid	BlockUserid

Usage

Parameter	Description
connhandle	An integer representing the connection ID. If this parameter is specified, **sp_iqwho** returns information only about the specified connection. If the specified connection is not open, no rows are displayed in the output.
user-name	A char(255) parameter representing a user login name. If this parameter is specified, **sp_iqwho** returns information only about the specified user. If the specified user has not opened any connections, no rows are displayed in the output. If the specified user name does not exist in the database, **sp_iqwho** returns the error message `"User` *`user-name`* `does not exist"`

Parameter	Description
arg-type	The *arg-type* parameter is optional and can be specified only when the first parameter has been specified. The only value for *arg-type* is **"user"**. If the *arg-type* value is specified as **"user"**, **sp_iqwho** interprets the first parameter as a user name, even if the first parameter is numeric. If any value other than **"user"** is specified for *arg-type*, **sp_iqwho** returns the error `"Invalid parameter."` Enclose the *arg-type* value in double quotes.

If no parameters are specified, **sp_iqwho** displays information about all currently active connections and users.

Either a connection handle or a user name can be specified as the first **sp_iqwho** parameter. The parameters *connhandle* and *user-name* are exclusive and optional. Only one of these parameters can be specified at a time. By default, if the first parameter is numeric, the parameter is assumed to be a connection handle. If the first parameter is not numeric, it is assumed to be a user name.

SAP Sybase IQ allows numeric user names. The *arg-type* parameter directs **sp_iqwho** to interpret a numeric value in the first parameter as a user name. For example:

```
sp_iqwho 1, "user"
```

When the *arg-type* **"user"** is specified, **sp_iqwho** interprets the first parameter 1as a user name, not as a connection ID. If a user named `1` exists in the database, **sp_iqwho** displays information about connections opened by user `1`.

Syntax	Output
sp_iqwho	Displays all active connections
sp_iqwho 3	Displays information about connection 3
sp_iqwho "DBA"	Displays connections opened by user `DBA`
sp_iqwho 3, "user"	Interprets 3 as a user name and displays connections opened by user 3. If user 3 does not exist, returns the error `"User 3 does not exist"`
sp_iqwho non-existing-user	Returns error `"User non-existing-user does not exist"`
sp_iqwho 3, "xyz"	Returns the error `"Invalid parameter: xyz"`

sp_iqwho Procedure Example

Use the example as a reference for **sp_iqwho** usage.

Display all active connections:

```
ConnHandle IQConnID Userid ReqType      IQCmdType            Blocked
On BlockUserid IQCursors

12         118      DBA    CURSOR_OPEN  IQUTILITYOPENCURSOR  0
    (NULL)      0
13         119      shweta DO_NOTHING   NONE                 0
    (NULL)      0


IQThreads IQIdle   SAIdle TempTableSpaceKB TempWorkSpaceKB
1         1          0       0             0
1         16238757   470     0             0
```

sp_iqwho compatibility with Adaptive Server Enterprise

The SAP Sybase IQ **sp_iqwho** stored procedure incorporates the SAP Sybase IQ equivalents of columns displayed by the Adaptive Server Enterprise **sp_who** procedure.

Some Adaptive Server Enterprise columns are omitted, as they are not applicable to SAP Sybase IQ.

Server Startup Options

The database startup utility **start_iq** starts an SAP Sybase IQ network database server. The switches for the parameters related to row-level versioning are listed here.

For a complete description of all available switches, see the *Utility Guide*.

-iqrlvmem start_iq Server Option

Specifies the amount of memory, in megabytes, available to the RLV store.

Syntax
-iqrlvmem *size*

Default
2048 (megabytes)

Remarks
If you specify 0 or an invalid value, then the default (2048 MB) is used.

Usage

-iqrlvmem is used at server startup to tell the server how much memory to reserve for row-level versioning.

SQL Statements

Lists the SQL statements that relate to in-memory row-level versioning.

ALTER DBSPACE Statement

Changes the read/write mode, changes the size, or extends an existing dbspace.

Syntax

```
ALTER DBSPACE dbspace-name
{ ADD new-file-spec  [, new-file-spec ... ]
| DROP FILE logical-file-name [, FILE logical-file-name ... ]
  |  RENAME TO newname |  RENAME 'new-file-pathname'
  |  READONLY | READWRITE
  |  ONLINE  | OFFLINE
  |  STRIPING{ ON |  OFF }
  |  STRIPESIZEKB size-in-KB
ALTER FILE file-name
{ READONLY  |  [ FORCE ] READWRITE }
  | SIZE file-size [ KB | MB | GB | TB ]
  | ADD file-size  [ KB  |  MB  |  GB |  TB  | PAGES ] }
RENAME PATH  'new-file-pathname'
RENAME TO newname
```

Parameters

- **new-file-spec –**

  ```
  FILE logical-file-name 'file-path' iq-file-opts
  ```

- **iq-file-opts –**

  ```
  [ [ SIZE ] file-size ]
  …[ KB | MB  | GB  | TB ] ]
     [ RESERVE reserve-size [ KB | MB | GB | TB ] ]
  ```

Examples

- **Example 1 –** Change the mode of a dbspace called `DspHist` to **READONLY**.

  ```
  ALTER DBSPACE DspHist READONLY
  ```

- **Example 2 –** Add 500MB to the dbspace `DspHist` by adding the file `FileHist3` of size 500MB.

```
ALTER DBSPACE DspHist
ALTER FILE FileHist3 ADD 500MB
```

- **Example 3 –** On a UNIX system, add two 500MB files to the dbspace DspHist.

```
ALTER DBSPACE DspHist ADD
FILE FileHist3 '/History1/data/file3' SIZE 500MB,
FILE FileHist4 '/History1/data/file4' SIZE 500
```

- **Example 4 –** Increase the size of the dbspace `IQ_SYSTEM_TEMP` by 2GB.

```
ALTER DBSPACE IQ_SYSTEM_TEMP ADD 2 GB
```

- **Example 5 –** Remove two files from dbspace DspHist. Both files must be empty.

```
ALTER DBSPACE DspHist
DROP FILE FileHist2, FILE FileHist4
```

- **Example 6 –** Increase the size of the dbspace `IQ_SYSTEM_MAIN` by 1000 pages. (**ADD** defaults to pages.)

```
ALTER DBSPACE IQ_SYSTEM_MAIN ADD 1000
```

Usage

ALTER DBSPACE changes the read-write mode, changes the online/offline state, alters the file size, renames the dbspace name, file logical name or file path, or sets the dbspace striping parameters. For details about existing dbspaces, run **sp_iqdbspace** procedure, **sp_iqdbspaceinfo** procedure, **sp_iqfile** procedure, **sp_iqdbspaceobjectinfo**, and **sp_iqobjectinfo**. Dbspace and dbfile names are always case-insensitive. The physical file paths are case-sensitive, if the database is **CASE RESPECT** and the operating system supports case-sensitive files. Otherwise, the file paths are case-insensitive.

Enclose dbspace and dbfile names either in no quotes or in double quotes. Enclose the physical file path to the dbfile in single quotes.

In Windows, if you specify a path, any backslash characters (\) must be doubled if they are followed by an n or an x. This prevents them being interpreted as a newline character (\n) or as a hexadecimal number (\x), according to the rules for strings in SQL. It is safer to always double the backslash.

- **ADD FILE clause –** adds one or more files to the specified dbspace. The dbfile name and the physical file path are required for each file and must be unique. You can add files to IQ main, IQ shared temporary, or IQ temporary dbspaces. You may add a file to a read-only dbspace, but the dbspace remains read-only. You can add files to multiplex shared temporary dbspaces only in read-only mode (the default for **ADD FILE**).

A catalog dbspace may contain only one file, so **ADD FILE** may not be used on catalog dbspaces.

For an RLV dbspace, use **ADD FILE** on simplex servers only. You cannot add a file to a multiplex RLV dbspace.

- **DROP FILE clause –** removes the specified file from an IQ dbspace. The file must be empty. You cannot drop the last file from the specified dbspace. Instead use DROP DBSPACE if the dbspace contains only one file. Rename TO clause—Renames the *dbspace-name* to a new name. The new name must be unique in the database. You cannot rename `IQ_SYSTEM_MAIN`, `IQ_SYSTEM_MSG`, `IQ_SYSTEM_TEMP`, `IQ_SHARED_TEMP`, or `SYSTEM`.
- **RENAME clause –** renames the pathname of the dbspace that contains a single file. It is semantically equivalent to the **ALTER FILE RENAME PATH** clause. An error is returned if the dbspace contains more than one file.
- **READONLY clause –** changes any dbspace except `IQ_SYSTEM_MAIN`, `IQ_SYSTEM_TEMP`, `IQ_SYSTEM_MSG`, `IQ_SHARED_TEMP`, and `SYSTEM` to read-only. Disallows DML modifications to any object currently assigned to the dbspace. Can only be used for dbspaces in the IQ main store.
- **READWRITE clause –** changes the dbspace to read-write. The dbspace must be online. Can only be used for dbspaces in the IQ main store.
- **ONLINE clause –** puts an offline dbspace and all associated files online. Can only be used for dbspaces in the IQ main store.
- **OFFLINE clause –** puts an online read-only dbspace and all associated files offline. (Returns an error if the dbspace is read-write, offline already, or not of the IQ main store.) Can only be used for dbspaces in the IQ main store.
- **STRIPING clause –** changes the disk striping on the dbspace as specified. When disk striping is set ON, data is allocated from each file within the dbspace in a round-robin fashion. For example, the first database page written goes to the first file, the second page written goes to the next file within given dbspace, and so on. Read-only dbspaces are skipped.
- **STRIPESIZEKB clause –** specifies the number of kilobytes (KB) to write to each file before the disk striping algorithm moves to the next stripe for the specified dbspace.
- **ALTER FILE READONLY –** changes the specified file to read-only. The file must be associated with an IQ main dbspace. You cannot change files in IQ_SHARED_TEMP to READONLY status.
- **ALTER FILE READWRITE –** changes specified IQ main or temporary store dbfile to read-write. The file must be associated with an IQ main or temporary dbspace.
- **ALTER FILE FORCE READWRITE –** changes the status of the specified shared temporary store dbfile to read-write, although there may be known file status problems on secondary nodes. The file may be associated with an IQ main, shared temporary, or temporary dbspace, but because new dbfiles in `IQ_SYSTEM_MAIN` and user main are created read-write, this clause only affects shared temporary dbspaces.

- **ALTER FILE SIZE clause –** specifies the new size of the file in units of kilobytes (KB), megabytes (MB), gigabytes (GB), or terabytes (TB). The default is megabytes. You can increase the size of the dbspace only if the free list (an allocation map) has sufficient room and if the dbspace has sufficient reserved space. You can decrease the size of the dbspace only if the portion to be truncated is not in use.
- **ALTER FILE ADD clause –** extends the size of the file in units of pages, kilobytes (KB), megabytes (MB), gigabytes (GB), or terabytes (TB). The default is MB. You can ADD only if the free list (an allocation map) has sufficient room and if the dbspace has sufficient reserved space.
- **ALTER FILE RENAME PATH clause –** renames the file pathname associated with the specified file. This clause merely associates the file with the new file path instead of the old path. The clause does not actually change the operating system file name. You must change the file name through your operating system. The dbspace must be offline to rename the file path. The new path is used when the dbspace is altered online or when the database is restarted.

You may not rename the path of a file in `IQ_SYSTEM_MAIN`, because if the new path were not accessible, the database would be unable to start. If you need to rename the path of a file in `IQ_SYSTEM_MAIN`, make the file read-only, empty the file, drop the file, and add the file again with the new file path name.

- **ALTER FILE RENAME TO clause –** renames the specified file's logical name to a new name. The new name must be unique in the database.

Side effects:

- Automatic commit
- Automatic checkpoint
- A mode change to **READONLY** causes immediate relocation of the internal database structures on the dbspace to one of the read-write dbspaces.

Standards

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Not supported by Adaptive Server Enterprise.

Permissions

Requires the MANAGE ANY DBSPACE system privilege.

ALTER TABLE Statement

Modifies a table definition.

Syntax

Syntax 1

```
ALTER TABLE [ owner.]table-name
|{ ENABLE | DISABLE } RLV STORE
{ alter-clause, ... }
```

Syntax 2

```
ALTER TABLE table_name ALTER OWNER TO new_owner
    [ { PRESERVE | DROP } PERMISSIONS ]
    [ { PRESERVE | DROP } FOREIGN KEYS ]
```

Parameters

- **alter-clause –**

```
ADD create-clause
    | ALTER column-name column-alteration
    | ALTER [ CONSTRAINT constraint-name ] CHECK ( condition )
    | DROP drop-object
    | RENAME rename-object
    | move-clause
    | SPLIT PARTITION range-partition-name
        INTO ( range-partition-decl-1, range-partition-decl-2
    | SPLIT SUBPARTITION range-subpartition-name
        INTO ( subrange-partition-decl-1, subrange-partition-
decl-2 )
    | MERGE PARTITION partition-name-1 INTO partition-name-2
    | MERGE SUBPARTITION subrange-partition-name-1 INTO subrange-
partition-name-2
    | UNPARTITION
    | PARTITION BY
      range-partitioning-scheme
        | hash-partitioning-scheme
        | composite-partitioning-scheme
```

- **create-clause –**

```
column-name column-definition [ column-constraint ]
    | table-constraint
    | [ PARTITION BY | SUBPARTITION BY ] range-partitioning-scheme
```

- **column-definition –**

```
 column-name data-type [ NOT NULL | NULL ]
    [ IN dbspace-name ]
    [ DEFAULT default-value | IDENTITY ]
```

- **column-constraint –**

```
[ CONSTRAINT constraint-name ]
   { UNIQUE
```

```
      | PRIMARY KEY
      | REFERENCES table-name [ (column-name ) ] [ actions ]
      | CHECK ( condition )
      | IQ UNIQUE ( integer )
   }
```

- **table-constraint –**

```
[ CONSTRAINT constraint-name ]
    { UNIQUE ( column-name [ , … ] )
      | PRIMARY KEY ( column-name [ , … ] )
      | foreign-key-constraint
      | CHECK ( condition )
   }
```

- **foreign-key-constraint –**

```
FOREIGN KEY [ role-name ] [ ( column-name [ , … ] ) ]
   ... REFERENCES table-name [ ( column-name [ , … ] ) ]
   ... [ actions ]
```

- **actions –**

```
[ ON   { UPDATE | DELETE }    { RESTRICT } ]
```

- **column-alteration –**

```
{ column-data-type | alterable-column-attribute } [ alterable-column-attribute … ]

    | ADD [ constraint-name ] CHECK ( condition )
    | DROP { DEFAULT | CHECK | CONSTRAINT constraint-name }
```

- **alterable-column-attribute –**

```
[ NOT ] NULL
   | DEFAULT default-value
   | [ CONSTRAINT constraint-name ] CHECK { NULL |( condition )
     }
```

- **default-value –**

```
CURRENT { DATABASE |DATE |REMOTE USER |TIME |TIMESTAMP | USER |
PUBLISHER )
   | string
   | global variable
   | [ - ] number
   | ( constant-expression )
   | built-in-function ( constant-expression )
   | AUTOINCREMENT
   | NULL
   | TIMESTAMP
   | LAST USER
   | USER
```

- **drop-object –**

```
{ column-name
   | CHECK constraint-name
   | CONSTRAINT
   | UNIQUE ( index-columns-list )
   | PRIMARY KEY
```

```
    | FOREIGN KEY fkey-name
    | [  PARTITION | SUBPARTITION  ] range-partition-name
  }
```

- **rename-object –**

```
new-table-name
    | column-name TO new-column-name
    | CONSTRAINT constraint-name TO new-constraint-name
   | [ PARTITION | SUBPARTITION ] range-partition-name TO new-range-
partition-name
```

- **move-clause –**

```
{ ALTER column-name
        MOVE
        { PARTITION ( range-partition-name TO new-dbspace-name)
          | SUBPARTITION ( range-partition-name TO new-dbspace-name
            |  TO new-dbspace-name }
         }
        | MOVE PARTITION range-partition-name TO new-dbspace-name
       | MOVE SUBPARTITION range-partition-name TO new-dbspace-name
        | MOVE TO new-dbspace-name
        | MOVE METADATA TO new-dbspace-name
   }
```

- **range-partitioning-scheme –**

```
RANGE ( partition-key )
     ( range-partition-decl [,range-partition-decl ...] )
```

- **hash-partitioning-scheme –**

```
HASH ( partition-key [, partition-key,...] )
```

- **composite-partitioning-schem: –**

```
hash-partitioning scheme SUBPARTITION range-partitioning-scheme
```

- **partition-key –**

```
column-name
```

- **range-partition-decl –**

```
range-partition-name VALUES <= ( {constant |  MAX } ) [ IN dbspace-
name ]
```

Examples

- **Example 1 –** Add a new column to the `Employees` table showing which office they work in:

```
ALTER TABLE Employees
ADD office CHAR(20)
```

- **Example 2 –** Drop the `office` column from the `Employees` table:

```
ALTER TABLE Employees
DROP office
```

- **Example 3 –** Add a column to the `Customers` table assigning each customer a sales contact:

```
ALTER TABLE Customers
ADD SalesContact INTEGER
REFERENCES Employees (EmployeeID)
```

- **Example 4 –** Add a new column `CustomerNum` to the `Customers` table and assigns a default value of 88:

```
ALTER TABLE Customers
ADD CustomerNum INTEGER DEFAULT 88
```

- **Example 5 –** Only **FP** indexes for `c2`, `c4`, and `c5`, are moved from dbspace `Dsp3` to `Dsp6`. **FP** index for `c1` remains in `Dsp1`. **FP** index for `c3` remains in `Dsp2`. The primary key for `c5` remains in `Dsp4`. **DATE** index `c4_date` remains in `Dsp5`.

```
CREATE TABLE foo (
        c1 INT IN Dsp1,
        c2 VARCHAR(20),
        c3 CLOB IN Dsp2,
        c4 DATE,
        c5 BIGINT,
        PRIMARY KEY (c5) IN Dsp4) IN Dsp3);

    CREATE DATE INDEX c4_date ON foo(c4) IN Dsp5;
    ALTER TABLE foo
        MOVE TO Dsp6;
```

- **Example 6 –** Move only **FP** index `c1` from dbspace `Dsp1` to `Dsp7`:

```
ALTER TABLE foo ALTER c1 MOVE TO Dsp7
```

- **Example 7 –** This example illustrates the use of many **ALTER TABLE** clauses to move, split, rename, and merge partitions.

  Create a partitioned table:

```
CREATE TABLE bar (
        c1 INT,
        c2 DATE,
        c3 VARCHAR(10))
    PARTITION BY RANGE(c2)
    (p1 VALUES <= ('2005-12-31') IN dbsp1,
     p2 VALUES <= ('2006-12-31') IN dbsp2,
     P3 VALUES <= ('2007-12-31') IN dbsp3,
     P4 VALUES <= ('2008-12-31') IN dbsp4);

    INSERT INTO bar VALUES(3, '2007-01-01', 'banana nut');
    INSERT INTO BAR VALUES(4, '2007-09-09', 'grape jam');
    INSERT INTO BAR VALUES(5, '2008-05-05', 'apple cake');
```

  Move partition `p2` to `dbsp5`:

```
ALTER TABLE bar MOVE PARTITION p2 TO DBSP5;
```

  Split partition `p4` into 2 partitions:

```
ALTER TABLE bar SPLIT PARTITION p4 INTO
        (P41 VALUES <= ('2008-06-30') IN dbsp4,
         P42 VALUES <= ('2008-12-31') IN dbsp4);
```

This **SPLIT PARTITION** reports an error, as it requires data movement. Not all existing rows are in the same partition after split.

```
ALTER TABLE bar SPLIT PARTITION p3 INTO
        (P31 VALUES <= ('2007-06-30') IN dbsp3,
         P32 VALUES <= ('2007-12-31') IN dbsp3);
```

This error is reported:

```
No data move is allowed, cannot split partition p3.
```

This **SPLIT PARTITION** reports an error, because it changes the partition boundary value:

```
ALTER TABLE bar SPLIT PARTITION p2 INTO
        (p21 VALUES <= ('2006-06-30') IN dbsp2,
         P22 VALUES <= ('2006-12-01') IN dbsp2);
```

This error is reported:

```
Boundary value for the partition p2 cannot be changed.
```

Merge partition `p3` into `p2`. An error is reported as a merge from a higher boundary value partition into a lower boundary value partition is not allowed.

```
ALTER TABLE bar MERGE PARTITION p3 into p2;
```

This error is reported:

```
Partition 'p2' is not adjacent to or before partition 'p3'.
```

Merge partition `p2` into `p3`:

```
ALTER TABLE bar MERGE PARTITION p2 INTO P3;
```

Rename partition `p1` to `p1_new`:

```
ALTER TABLE bar RENAME PARTITION p1 TO p1_new;
```

Unpartition table `bar`:

```
ALTER TABLE bar UNPARTITION;
```

Partition table `bar`. This command reports an error, because all rows must be in the first partition.

```
ALTER TABLE bar PARTITION BY RANGE(c2)
    (p1 VALUES <= ('2005-12-31') IN dbsp1,
     P2 VALUES <= ('2006-12-31') IN DBSP2,
     P3 VALUES <= ('2007-12-31') IN dbsp3,
     P4 VALUES <= ('2008-12-31') IN dbsp4);
```

This error is reported:

```
All rows must be in the first partition.
```

Partition table bar:

```
ALTER TABLE bar PARTITION BY RANGE(c2)
    (p1 VALUES <= ('2008-12-31') IN dbsp1,
     P2 VALUES <= ('2009-12-31') IN dbsp2,
     P3 VALUES <= ('2010-12-31') IN dbsp3,
     P4 VALUES <= ('2011-12-31') IN dbsp4);
```

- **Example 8 –** Change a table tab1 so that it is no longer registered for in-memory real-time updates in the RLV store.

```
ALTER TABLE tab1 DISABLE RLV STORE
```

Usage

The ALTER TABLE statement changes table attributes (column definitions and constraints) in a table that was previously created. The syntax allows a list of alter clauses; however, only one table constraint or column constraint can be added, modified, or deleted in each ALTER TABLE statement. ALTER TABLE is prevented whenever the statement affects a table that is currently being used by another connection. ALTER TABLE can be time consuming, and the server does not process requests referencing the same table while the statement is being processed.

Note: You cannot alter local temporary tables, but you can alter global temporary tables when they are in use by only one connection.

SAP Sybase IQ enforces REFERENCES and CHECK constraints. Table and/or column check constraints added in an ALTER TABLE statement are evaluated, only if they are defined on one of the new columns added, as part of that alter table operation. For details about CHECK constraints, see *CREATE TABLE Statement*.

If SELECT * is used in a view definition and you alter a table referenced by the SELECT *, then you must run ALTER VIEW <*viewname*> RECOMPILE to ensure that the view definition is correct and to prevent unexpected results when querying the view.

ADD column-definition [**column-constraint**] — Add a new column to the table.

- The table must be empty to specify NOT NULL. The table might contain data when you add an IDENTITY or DEFAULT AUTOINCREMENT column. If the column has a default IDENTITY value, all rows of the new column are populated with sequential values. You can also add FOREIGN constraint as a column constraint for a single column key. The value of the IDENTITY/DEFAULT AUTOINCREMENT column uniquely identifies every row in a table.
- The IDENTITY/DEFAULT AUTOINCREMENT column stores sequential numbers that are automatically generated during inserts and updates. DEFAULT AUTOINCREMENT columns are also known as IDENTITY columns. When using IDENTITY/DEFAULT AUTOINCREMENT, the column must be one of the integer data types, or an exact numeric

type, with scale 0. See *CREATE TABLE Statement* for more about column constraints and `IDENTITY/DEFAULT AUTOINCREMENT` columns.

- **IQ UNIQUE** constraint — Defines the expected cardinality of a column and determines whether the column loads as `Flat FP` or `NBit FP`. An `IQ UNIQUE(`*n*`)` value explicitly set to 0 loads the column as `Flat FP`. Columns without an `IQ UNIQUE` constraint implicitly load as `NBit` up to the limits defined by the `FP_NBIT_AUTOSIZE_LIMIT`, `FP_NBIT_LOOKUP_MB`, and `FP_NBIT_ROLLOVER_MAX_MB` options:
  - `FP_NBIT_AUTOSIZE_LIMIT` limits the number of distinct values that load as `NBit`
  - `FP_NBIT_LOOKUP_MB` sets a threshold for the total `NBit` dictionary size
  - `FP_NBIT_ROLLOVER_MAX_MB` sets the dictionary size for implicit `NBit` rollovers from `NBit` to `Flat FP`
  - `FP_NBIT_ENFORCE_LIMITS` enforces `NBit` dictionary sizing limits. This option is OFF by default

  Using `IQ UNIQUE` with an *n* value less than the `FP_NBIT_AUTOSIZE_LIMIT` is not necessary. Auto-size functionality automatically sizes all low or medium cardinality columns as `NBit`. Use `IQ UNIQUE` in cases where you want to load the column as `Flat FP` or when you want to load a column as `NBit` when the number of distinct values exceeds the `FP_NBIT_AUTOSIZE_LIMIT`.

Note:

- Consider memory usage when specifying high `IQ UNIQUE` values. If machine resources are limited, avoid loads with `FP_NBIT_ENFORCE_LIMITS='OFF'` (default).
  Prior to SAP Sybase IQ 16.0, an `IQ UNIQUE` *n* value > 16777216 would rollover to `Flat FP`. In 16.0, larger `IQ UNIQUE` values are supported for tokenization, but may require significant memory resource requirements depending on cardinality and column width.
- BIT, BLOB,and CLOB data types do not support `NBit` dictionary compression. If `FP_NBIT_IQ15_COMPATIBILITY='OFF'`, a non-zero `IQ UNIQUE` column specification in a CREATE TABLE or ALTER TABLE statement that includes these data types returns an error.

{ ENABLE | DISABLE } RLV STORE — Registers this table with the RLV store for real-time in-memory updates. This value overrides the value of the database option **BASE_TABLES_IN_RLV**. Requires the CREATE TABLE system privilege and `CREATE` permissions on the RLV store dbspace to set this value to `ENABLE`.

Note: The `{ ENABLE | DISABLE } RLV STORE` clause is not supported for IQ temporary tables.

ALTER *column-name* **column-alteration** — Change the column definition:

- **SET DEFAULT** *default-value* — Change the default value of an existing column in a table. You can also use the `MODIFY` clause for this task, but `ALTER` is `ISO/ANSI SQL` compliant, and `MODIFY` is not. Modifying a default value does not change any existing values in the table.
- **DROP DEFAULT** — Remove the default value of an existing column in a table. You can also use the `MODIFY` clause for this task, but `ALTER` is `ISO/ANSI SQL` compliant, and `MODIFY` is not. Dropping a default does not change any existing values in the table.
- **ADD** — Add a named `constraint` or a `CHECK` condition to the column. The new constraint or condition applies only to operations on the table after its definition. The existing values in the table are not validated to confirm that they satisfy the new constraint or condition.
- **CONSTRAINT** *column-constraint-name* — The optional column constraint name lets you modify or drop individual constraints at a later time, rather than having to modify the entire column constraint.
- [**CONSTRAINT** *constraint-name*] **CHECK** (*condition*) — Use this clause to add a `CHECK` constraint on the column.
- **SET COMPUTE** (*expression*) — Change the expression associated with a computed column. The values in the column are recalculated when the statement is executed, and the statement fails if the new expression is invalid.
- **DROP COMPUTE** — Change a column from being a computed column to being a non-computed column. This statement does not change any existing values in the table.

ADD table-constraint — Add a constraint to the table.

You can also add a foreign key constraint as a table constraint for a single-column or multicolumn key. If `PRIMARY KEY` is specified, the table must not already have a primary key created by the `CREATE TABLE` statement or another `ALTER TABLE statement`. See *CREATE TABLE Statement* for a full explanation of table constraints.

Note: You cannot MODIFY a table or column constraint. To change a constraint, DELETE the old constraint and ADD the new constraint.

DROP *drop-object* — Drops a table object.

- **DROP** *column-name* — Drop the column from the table. If the column is contained in any multicolumn index, uniqueness constraint, foreign key, or primary key, then the index, constraint, or key must be deleted before the column can be deleted. This does not delete `CHECK` constraints that refer to the column. An `IDENTITY/DEFAULT AUTOINCREMENT` column can only be deleted if `IDENTITY_INSERT` is turned off and the table is not a local temporary table.
- **DROP CHECK** — Drop all check constraints for the table. This includes both table check constraints and column check constraints.
- **DROP CONSTRAINT** *constraint-name* — Drop the named constraint for the table or specified column.

- **DROP UNIQUE** (*column-name, ...*) — Drop the unique constraints on the specified column(s). Any foreign keys referencing the unique constraint (rather than the primary key) are also deleted. Reports an error if there are associated foreign-key constraints. Use `ALTER TABLE` to delete all foreign keys that reference the primary key before you delete the primary key constraint.
- **DROP PRIMARY KEY** — Drop the primary key. All foreign keys referencing the primary key for this table are also deleted. Reports an error if there are associated foreign key constraints. If the primary key is unenforced, `DELETE` returns an error if associated unenforced foreign key constraints exist.
- **DROP FOREIGN KEY** *role-name* — Drop the foreign key constraint for this table with the given role name. Retains the implicitly created non-unique `HG` index for the foreign key constraint. Users can explicitly remove the `HG` index with the `DROP INDEX` statement.
- **DROP** [**PARTITION** | **SUBPARTITION**] — Drop the specified partition. The rows in partition `P1` are deleted and the partition definition is dropped. You cannot drop the last partition because dropping the last partition would transform a partitioned table to a non-partitioned table. (To merge a partitioned table, use an `UNPARTITION` clause instead.) For example:

```
CREATE TABLE foo (c1 INT, c2 INT)
    PARTITION BY RANGE (c1)
    (P1 VALUES <= (100) IN dbsp1,
     P2 VALUES <= (200) IN dbsp2,
     P3 VALUES <= (MAX) IN dbsp3
    ) IN dbsp4);
LOAD TABLE ….
ALTER TABLE DROP PARTITION P1;
```

  Use **DROP SUBPARTITION**for tables partitioned by a *composite-partitioning-scheme*.

RENAME *rename-object*

- **RENAME** *new-table-name* — Change the name of the table to the *new-table-name*. Any applications using the old table name must be modified. Also, any foreign keys that were automatically assigned the same name as the old table name do not change names.
- **RENAME** *column-name* **TO** *new-column-name* — Change the name of the column to *new-column-name*. Any applications using the old column name must be modified.
- **RENAME** [**PARTITION** | **SUBPARTITION**] — Rename an existing partition or sub-partition.
- **RENAME** *constraint-name* **TO** *new-constraint-name* — Change the name of the constraint to *new-constraint-name*. Any applications using the old constraint name must be modified.

MOVE clause — Moves a table object.

A table object can only reside in one dbspace. Any type of `ALTER MOVE` blocks any modification to the table for the entire duration of the move.

- **MOVE TO** — Move all table objects including columns, indexes, unique constraints, primary key, foreign keys, and metadata resided in the same dbspace as the table is mapped

to the new dbspace. The ALTER Column MOVE TO clause cannot be requested on a partitioned table.

- **MOVE TABLE METADATA** — Move the metadata of the table to a new dbspace. For a partitioned table, MOVE TABLE METADATA also moves metadata that is shared among partitions.
- **MOVE PARTITION** — Move the specified partition to the new dbspace.
- **MOVE SUBPARTITION** — Move the specified range subpartition of an existing hash-range partitioned table to the new dbspace.

PARTITION BY — Partitions a non-partitioned table. A non-partitioned table can be partitioned, if all existing rows belong to the first partition. You can specify a different dbspace for the first partition than the dbspace of the column or table. But existing rows are not moved. Instead, the proper dbspace for the column/partition is kept in SYS.ISYSIQPARTITIONCOLUMN for existing columns. Only the default or max identity column(s) that are added later for the first partition are stored in the specified dbspace for the first partition.

SAP Sybase IQ supports range, hash, and composite partitioning schemes:

- **PARTITION BY** *RANGE* — Maps data to partitions based on a range of partition keys established for each partition.
- **PARTITION BY** *HASH* — Maps data to partitions based on partition key values processed by a system-defined function. An existing table with rows can only be made hash-range partitioned if all the hash partition keys hash to a single subpartition; in practice, this means only empty tables can always be altered.
- **PARTITION BY** *composite-partitioning-scheme* — Subpartition rows after partitioning by RANGE or HASH. This method provides the benefits of combined partitioning methods.

Note: Range-partitions and composite partitioning schemes, like hash-range partitions, require the separately licensed VLDB Management option.

SUBPARTITION — Subpartition range or hash partitioned tables by range or hash partitioning strategy. In a *create-clause*, use SUBPARTITION to subpartition tables that are partitioned by a *composite-partitioning-scheme*.

- **SUBPARTITION BY RANGE** — Adds a new range subpartition to an existing hash-range partitioned table. The new range subpartition will be logically partitioned by hash with the same hash partitioned keys as the existing hash-range partitioned table.
- **DROP SUBPARTITION** — Delete rows and drops the sub-partition definition from an existing hash-range partitioned table. The specified range subpartition is dropped
- **MOVE PARTITION** — Moves the specified range subpartition of an existing hash-range partitioned table to the new dbspace.
- **MOVE SUBPARTITION** — Moves the column of the specified hash-range partition to the specified dbspace.
- **SPLIT PARTITION** — Splits the specified range subpartition of a hash-range partitioned table into two partitions. Split the specified partition into two partitions. A partition can be

split only if no data must be moved. All existing rows of the partition to be split must remain in a single partition after the split. merges range-subpartition-name-1 into range-subpartition-name-2 of a hash-range partitioned table.

The boundary value for *partition-decl-1* must be less than the boundary value of *partition-name* and the boundary value for *partition-decl-2* must be equal to the boundary value of *partition-name*. You can specify different names for the two new partitions. The old *partition-name* can only be used for the second partition, if a new name is not specified.

- **SPLIT SUBPARTITION** — Splits the specified range subpartition of a hash-range partitioned table into two partitions.

MERGE PARTITION — Merge *partition-name-1* into *partition-name-2*. Two partitions can be merged if they are adjacent partitions and the data resides on the same dbspace. You can only merge a partition with a lower partition value into the adjacent partition with a higher partition value. Note that the server does not check `CREATE` permission on the dbspace into which the partition is merged. For an example of how to create adjacent partitions, see *CREATE TABLE Statement* examples.

RENAME [**PARTITION** | **SUBPARTITION**] — Rename an existing `PARTITION` or `SUBPARTITION`.

UNPARTITION — Remove partitions from a partitioned table. Each column is placed in a single dbspace. Note that the server does not check `CREATE` permission on the dbspace to which data of all partitions is moved. `ALTER TABLE UNPARTITION` blocks all database activities.

ALTER OWNER – Change the owner of a table. The **ALTER OWNER** clause may not be used in conjunction with any other [alter-clause] clauses of the ALTER TABLE statement.

- [**PRESERVE** | **DROP] PERMISSIONS** – If you do not want the new owner to have the same privileges as the old owner, use the DROP permissions clause (default) to drop all explicitly-granted privileges that allow a user access to the table. Implicitly-granted privileges given to the owner of the table are given to the new owner and dropped from the old owner.

  [**PRESERVE** | **DROP] FOREIGN KEYS** If you want to prevent the new owner from accessing data in referenced tables, use the DROP FOREIGN KEYS clause (default) to drop all foreign keys within the table, as well as all foreign keys referring to the table. Use of the PRESERVE FOREIGN KEYS clause with the DROP PERMISSIONS clause fails unless all referencing tables are owned by the new owner.
- The **ALTER TABLE ALTER OWNER** statement fails if:
  - Another table with the same name as the original table exists and is owned by the new user.
  - The PRESERVE FOREIGN KEYS and PRESERVE PERMISSIONS clauses are both specified and there is a foreign key owned by a user other than the new table owner referencing the table that relies on implicitly-granted permissions (such as those given

to the owner of a table). To avoid this failure, explicitly grant SELECT permissions to the referring table's original owner, or drop the foreign keys.
- The PRESERVE FOREIGN KEYS clause is specified, but the PRESERVE PERMISSIONS clause is NOT, and there is a foreign key owned by a user other than the new table owner referencing the table. To avoid this failure, drop the foreign keys.
- The PRESERVE FOREIGN KEYS clause is specified and the table contains a foreign key that relies on implicitly-granted permissions (such as those given to the owner of a table). To avoid this failure, explicitly GRANT SELECT permissions to the new owner on the referenced table, or drop the foreign keys.
- The table contains a column with a default value that refers to a sequence, and the USAGE permission of the sequence generator relies on implicitly-granted permissions (such as those given to the owner of a sequence). To avoid this failure, explicitly grant USAGE permission on the sequence generator to the new owner of the table.
- Enabled materialized views that depend on the original table exist.

Side effects:

- Automatic commit. The `ALTER` and `DROP` options close all cursors for the current connection. The Interactive SQL data window is also cleared.
- A checkpoint is carried out at the beginning of the `ALTER TABLE` operation.
- Once you alter a column or table, any stored procedures, views or other items that refer to the altered column no longer work.

Standards

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Some clauses are supported by Adaptive Server Enterprise.

Permissions

Syntax 1

The system privileges required for syntax 1 varies depending upon the clause used.

Clause	Privilege Required
Add	Requires one of: • ALTER ANY TABLE system privilege • ALTER ANY OBJECT system privilege • ALTER privilege on the underlying table • You own the underlying table UNIQUE, PRIMARY KEY, FOREIGN KEY, or IQ UNIQUE column constraint – Requires above along with REFERENCE privilege on the underlying table. FOREIGN KEY table constraint requires above along with one of: • CREATE ANY INDEX system privilege • CREATE ANY OBJECT system privilege • REFERENCE privilege on the base table Hash partition, range partition, or hash-range partition requires above along with one of: • CREATE ANY OBJECT system privilege • CREATE permission on the dbspaces where the partitions are being created
Alter	Requires one of: • ALTER ANY TABLE system privilege • ALTER ANY OBJECT system privilege • ALTER permission on the table • You own the table. To alter a primary key or unique constraint, also requires REFERENCE permission on the table.

Clause	Privilege Required
Drop	Drop a column with no constraints – Requires one of: • ALTER ANY OBJECT system privilege • ALTER ANY TABLE system privilege • ALTER permission on the underlying table • You own the underlying table Drop a column or table with a constraint requires above along with REFERENCE permission if using ALTER permission. Drop a partition on table owned by self – None required. Drop a partition on table owned by other users – Requires one of: • ALTER ANY TABLE system privilege • ALTER ANY OBJECT system privilege • ALTER permission on the table
RENAME	Requires one of: • ALTER ANY TABLE system privilege • ALTER ANY OBJECT system privilege • ALTER permission on the table • You own the table
Move	Requires one of: • ALTER ANY TABLE system privilege • ALTER ANY OBJECT system privilege • MANAGE ANY DBSPACE system privilege • ALTER privilege on the underlying table • You own the underlying table Also requires one of the following: • CREATE ANY OBJECT system privilege • CREATE privilege on the dbspace to which the partition is being moved

Clause	Privilege Required
Split Partition or Subpartition	Partition on table owned by self – None required. Partition on table owned by other users – Requires one of: • SELECT ANY TABLE system privilege • SELECT privilege on table Also requires one of: • ALTER ANY TABLE system privilege • ALTER ANY OBJECT system privilege • ALTER privilege on the table
Merge Partition or Subpartition Unpartition	Table owned by self – None required. Table owned by other users – Requires one of: • ALTER ANY TABLE system privilege • ALTER ANY OBJECT system privilege • ALTER privilege on the table
Partition By	Hash partition, range partition, or hash-range partition – Requires one of: • CREATE ANY OBJECT system privilege • CREATE permission on the dbspaces where the partitions are being created Also requires one of: • ALTER ANY TABLE system privilege • ALTER ANY OBJECT system privilege • ALTER permission on the table • You own the table
Enable or disable RLV store	Requires one of: • ALTER ANY TABLE system privilege • ALTER ANY OBJECT system privilege

Syntax 2

Requires one of:

- ALTER ANY TABLE system privilege
- ALTER ANY OBJECT system privilege
- ALTER privilege on the table
- You own the table

See also

- *CREATE TABLE Statement* on page 164

CREATE DBSPACE Statement

Creates a new dbspace and the associated dbfiles for the IQ main store, catalog store, or RLV store.

Syntax

Syntax 1

Use for catalog store dbspaces only (SQL Anywhere (SA) dbspaces).

```
CREATE DBSPACE dbspace-name AS file-path CATALOG STORE
```

Syntax 2

Use for IQ main store dbspaces.

```
CREATE DBSPACE dbspace-name USING file-specification
[ IQ STORE ] iq-dbspace-opts
```

Syntax 3

Use for RLV dbspaces.

```
CREATE DBSPACE dbspace-name USING file-specification
RLV STORE
```

Parameters

- **file-specification –** { *single-path-spec* | *new-file-spec* [, ...] }
- **single-path-spec –** *'file-path'* | *iq-file-opts*
- **new-file-spec – FILE** *logical-file-name* | *'file-path' iq-file-opts*
- **iq-file-opts –** [[**SIZE**] *file-size*] …[**KB** | **MB** | **GB** | **TB**]] [**RESERVE** *size* …[**KB** | **MB** | **GB** | **TB**]]
- **iq-dbspace-opts –** [**STRIPING**] {ON | OFF}] …[**STRIPESIZEKB** *sizeKB*]

Examples

- **Example 1 –** Create a dbspace called `DspHist` for the IQ main store with two files on a UNIX system. Each file is 1GB in size and can grow 500MB:

  ```
  CREATE DBSPACE DspHist USING FILE
  FileHist1 '/History1/data/file1'
  SIZE 1000 RESERVE 500,
  FILE FileHist2 '/History1/data/file2'
  SIZE 1000 RESERVE 500;
  ```

- **Example 2 –** Create a second catalog dbspace called `DspCat2`:

```
CREATE DBSPACE DspCat2 AS
'catalog_file2'
CATALOG STORE;
```

- **Example 3 –** Creates an IQ main dbspace called `EmpStore1` for the IQ store (three alternate syntax examples):

```
CREATE DBSPACE EmpStore1
USING FILE EmpStore1
'EmpStore1.IQ' SIZE 8 MB IQ STORE;
```

```
CREATE DBSPACE EmpStore1
USING FILE EmpStore1
'EmpStore1.IQ' 8 IQ STORE;
```

```
CREATE DBSPACE EmpStore1
USING FILE EmpStore1
'EmpStore1.IQ' 8;
```

- **Example 4 –** Creates a RLV store dbspace called d1:

```
CREATE DBSPACE d1
USING FILE f1
'f1.iq' SIZE 100 RLV STORE;
```

Usage

CREATE DBSPACE creates a new dbspace for the IQ main store, catalog store, or RLV store. The dbspace you add can be on a different disk device than the initial dbspace, allowing you to create stores that are larger than one physical device.

`Syntax 1` creates a dbspace for the catalog store, where both dbspace and dbfile have the same logical name. Each dbspace in the catalog store has a single file.

`new-file-spec` creates a dbspace for the IQ main store. You can specify one or more dbfiles for the IQ main store. The dbfile name and physical file path are required for each file, and must be unique.

The dbspace name and dbfile names are always case-insensitive. The physical file paths have the case sensitivity of the operating system if the database is **CASE RESPECT**, and are case-insensitive if the database is **CASE IGNORE**.

You cannot create a dbspace for an IQ temporary store. A single temporary dbspace, `IQ_SYSTEM_TEMP`, is created when you create a new database or upgrade one that was created in a version earlier than SAP Sybase IQ 15.3. You can add additional files to the `IQ_SYSTEM_TEMP` dbspace using the **ALTER DBSPACE ADD FILE** syntax.

Note: Creating a RLV dbspace containing a minimum of one file is a prerequisite for RLV storage. Before enabling RLV storage on a simplex server, check that the RLV dbspace exists.

RESERVE clause—Specifies the size in kilobytes (KB), megabytes (MB), gigabytes (GB), or terabytes (TB) of space to reserve, so that the dbspace can be increased in size in the future.

The *size* parameter can be any number greater than 0; megabytes is the default. You cannot change the reserve after the dbspace dbfile is created.

When **RESERVE** is specified, the database uses more space for internal (free list) structures. If reserve size is too large, the space needed for the internal structures can be larger than the specified size, which results in an error.

You can create a unique path in any of these ways:

- Specify a different extension for each file (for example, `mydb.iq`)
- Specify a different file name (for example, `mydb2.iq`)
- Specify a different path name (for example, `/iqfiles/main/iq`) or different raw partitions

---

Warning! On UNIX platforms, to maintain database consistency, specify file names that are links to different files. SAP Sybase IQ cannot detect the target where linked files point. Even if the file names in the command differ, make sure they do not point to the same operating system file.

---

dbspace-name and *dbfile-name* are internal names for dbspaces and dbfiles. *filepath* is the actual operating system file name of the dbfile, with a preceding path where necessary. `filepath` without an explicit directory is created in the same directory as the catalog store of the database. Any relative directory is relative to the catalog store.

SIZE clause—Specifies the size, from 0 to 4 terabytes, of the operating system file specified in *filepath*. The default depends on the store type and block size. For the IQ main store, the default number of bytes equals 1000* the block size. You cannot specify the **SIZE** clause for the catalog store.

A **SIZE** value of 0 creates a dbspace of minimum size, which is 8MB for the IQ main store.

For raw partitions, do not explicitly specify **SIZE**. SAP Sybase IQ automatically sets this parameter to the maximum raw partition size, and returns an error if you attempt to specify another size.

STRIPESIZEKB clause—Specifies the number of kilobytes (KB) to write to each file before the disk striping algorithm moves to the next stripe for the specified dbspace.

If you do not specify striping or stripe size, the default values of the options `DEFAULT_DISK_STRIPING` and `DEFAULT_KB_PER_STRIPE` apply.

A database can have as many as (32KB - 1) dbspaces, including the initial dbspaces created when you create the database. However, your operating system might limit the number of files per database.

Side effects:

- Automatic commit
- Automatic checkpoint.

Standards

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Not supported by Adaptive Server Enterprise.

Permissions

Requires the MANAGE ANY DBSPACE system privilege.

CREATE TABLE Statement

Creates a new table in the database or on a remote server.

Syntax

```
CREATE [ { GLOBAL | LOCAL } TEMPORARY ] TABLE
  [ IF NOT EXISTS ] [ owner. ]table-name
… ( column-definition [ column-constraint ] …
[ , column-definition [ column-constraint ] …]
[ , table-constraint ] … )
|{ ENABLE | DISABLE } RLV STORE
|[WITH NULLS NOT DISTINCT]
…[ IN dbspace-name ]
…[ ON COMMIT { DELETE | PRESERVE } ROWS
| NOT TRANSACTIONAL ]
[ AT location-string ]
[PARTITION BY
   range-partitioning-scheme
   | hash-partitioning-scheme
   | composite-partitioning-scheme ]
```

Parameters

- **column-definition –**

```
column-name data-type
     [ [ NOT ] NULL ]
     [ DEFAULT default-value | IDENTITY ]
     [ PARTITION | SUBPARTITION ( partition-name IN   dbspace-name
[ , ... ] ) ]
```

- **default-value –**

```
special-value
    | string
    | global variable
    | [ - ] number
    | ( constant-expression )
    | built-in-function( constant-expression )
    | AUTOINCREMENT
    | CURRENT DATABASE
    | CURRENT REMOTE USER
    | NULL
```

```
    | TIMESTAMP
    | LAST USER
```

- **special-value –**

```
CURRENT
  { DATE
    | TIME
    | TIMESTAMP
    | USER
    | PUBLISHER }
    | USER
```

- **column-constraint –**

```
[ CONSTRAINT constraint-name ] {
     { UNIQUE
        | PRIMARY KEY
        | REFERENCES table-name [ ( column-name ) ] [ action ]
      }
      [ IN dbspace-name ]
      | CHECK ( condition )
      | IQ UNIQUE ( integer )
    }
```

- **table-constraint –**

```
[ CONSTRAINT constraint-name ]
  {
    { UNIQUE ( column-name [ , column-name ] … )
      | PRIMARY KEY ( column-name [ , column-name ] … )
     }
    [ IN dbspace-name ]
      | foreign-key-constraint
      | CHECK ( condition )
      | IQ UNIQUE ( integer )
   }
```

- **foreign-key-constraint –**

```
FOREIGN KEY [ role-name ] [ ( column-name [ , column-name ] … ) ]
   …REFERENCES table-name [ ( column-name [ , column-name ] … ) ]
   …[ actions ] [ IN dbspace-name ]
```

- **actions –**

```
[ ON { UPDATE | DELETE } RESTRICT ]
```

- **location-string –**

```
{ remote-server-name. [ db-name ].[ owner ].object-name
      | remote-server-name; [ db-name ];[ owner ];object-name }
```

- **Partitioning-scheme –**

```
{ range-partitioning-scheme
      | hash-partitioning-scheme
      | composite-partitioning-scheme
    }
```

- **range-partitioning-scheme –**

```
RANGE( partition-key ) ( range-partition-decl [,range-partition-
decl ... ] )
```

- **partition-key –**

```
column-name
```

- **range-partition-decl –**

```
   VALUES <= ( {constant-expr |  MAX } [ , { constant-expr |
MAX }]... )
    [ IN dbspace-name ]
```

- **hash-partitioning-scheme –**

```
HASH  ( partition-key [ , partition-key, … ] )
```

- **composite-partitioning-scheme –**

```
hash-partitioning-scheme  SUBPARTITION  range-partitioning-scheme
```

Examples

- **Example 1 –** Create a table named `SalesOrders2` with five columns. Data pages for columns `FinancialCode`, `OrderDate`, and `ID` are in dbspace `Dsp3`. Data pages for integer column `CustomerID` are in dbspace `Dsp1`. Data pages for CLOB column `History` are in dbspace `Dsp2`. Data pages for the primary key, HG for `ID`, are in dbspace `Dsp4`:

```
CREATE TABLE SalesOrders2 (
FinancialCode CHAR(2),
CustomerID int IN Dsp1,
History CLOB IN Dsp2,
OrderDate TIMESTAMP,
ID BIGINT,
PRIMARY KEY(ID) IN Dsp4
) IN Dsp3
```

- **Example 2 –** Create a table `fin_code2` with four columns. Data pages for columns `code`, `type`, and `id` are in the default dbspace, which is determined by the value of the database option `DEFAULT_DBSPACE`. Data pages for `CLOB` column `description` are in dbspace `Dsp2`. Data pages from foreign key fk1, HG for c1 are in dbspace `Dsp4`:

```
CREATE TABLE fin_code2 (
code INT,
type CHAR(10),
description CLOB IN Dsp2,
id BIGINT,
FOREIGN KEY fk1(id) REFERENCES SalesOrders(ID) IN Dsp4
)
```

- **Example 3 –** Create a table `t1` where partition `p1` is adjacent to `p2` and partition `p2` is adjacent to `p3`:

```
CREATE TABLE t1 (c1 INT, c2 INT)
PARTITION BY RANGE(c1)
(p1 VALUES <= (0), p2 VALUES <= (10), p3 VALUES <= (100))
```

- **Example 4 –** Create a RANGE partitioned table `bar` with six columns and three partitions, mapping data to partitions based on dates:

```
CREATE TABLE bar (
        c1 INT IQ UNIQUE(65500),
        c2 VARCHAR(20),
        c3 CLOB PARTITION (P1 IN Dsp11, P2 IN Dsp12,
           P3 IN Dsp13),
        c4 DATE,
        c5 BIGINT,
        c6 VARCHAR(500) PARTITION (P1 IN Dsp21,
           P2 IN Dsp22),
        PRIMARY KEY (c5) IN Dsp2) IN Dsp1
        PARTITION BY RANGE (c4)
        (P1 VALUES <= ('2006/03/31') IN Dsp31,
         P2 VALUES <= ('2006/06/30') IN Dsp32,
         P3 VALUES <= ('2006/09/30') IN Dsp33
        ) ;
```

Data page allocation for each partition:

Partition	Dbspaces	Columns
P1	Dsp31	c1, c2, c4, c5
P1	Dsp11	c3
P1	Dsp21	c6
P2	Dsp32	c1, c2, c4, c5
P2	Dsp12	c3
P2	Dsp22	c6
P3	Dsp33	c1, c2, c4, c5, c6
P3	Dsp13	c3
P1, P2, P3	Dsp1	lookup store of c1 and other shared data
P1, P2, P3	Dsp2	primary key (HG for c5)

- **Example 5 –** Create a `HASH` partitioned (`table tbl42`) that includes a `PRIMARY KEY` (column c1) and a `HASH PARTITION KEY` (columns c4 and c3).

```
CREATE TABLE tbl42 (
    c1 BIGINT NOT NULL,
    c2 CHAR(2) IQ UNIQUE(50),
```

```
    c3 DATE IQ UNIQUE(36524),
    c4 VARCHAR(200),
  PRIMARY KEY (c1)
    )
  PARTITION BY HASH ( c4, c3 )
```

- **Example 6 –** Create a hash-ranged partitioned table with a `PRIMARY KEY` (column c1), a hash partition key (columns c4 and c2) and a range subpartition key (column c3).

```
CREATE TABLE tbl42 (
    c1 BIGINT NOT NULL,
    c2 CHAR(2) IQ UNIQUE(50),
    c3 DATE,
    c4 VARCHAR(200),
  PRIMARY KEY (c1)) IN Dsp1

  PARTITION BY HASH ( c4, c2 )
  SUBPARTITION BY RANGE ( c3 )
  ( P1 VALUES <= (2011/03/31) IN Dsp31,
    P2 VALUES <= (2011/06/30) IN Dsp32,
    P3 VALUES <= (2011/09/30) IN Dsp33) ;
```

- **Example 7 –** Create a table for a library database to hold information on borrowed books:

```
CREATE TABLE borrowed_book (
date_borrowed DATE NOT NULL,
date_returned DATE,
book       CHAR(20)
           REFERENCES library_books (isbn),
CHECK( date_returned >= date_borrowed )
)
```

- **Example 8 –** Create table `t1` at the remote server `SERVER_A` and create a proxy table named `t1` that is mapped to the remote table:

```
CREATE TABLE t1
( a  INT,
  b  CHAR(10))
AT 'SERVER_A.db1.joe.t1'
```

- **Example 9 –** Create table `tab1` that contains a column `c1` with a default value of the special constant `LAST USER`:

```
CREATE TABLE tab1(c1 CHAR(20) DEFAULT LAST USER)
```

- **Example 10 –** Create a local temporary table `tab1` that contains a column `c1`:

```
CREATE LOCAL TEMPORARY TABLE tab1(c1 int) IN IQ_SYSTEM_TEMP
```

  The example creates `tab1` in the IQ_SYSTEM_TEMP dbspace in the following cases:

  - `DQP_ENABLED` logical server policy option is set ON but there are no read-write files in IQ_SHARED_TEMP
  - `DQP_ENABLED` option is OFF, `TEMP_DATA_IN_SHARED_TEMP` logical server policy option is ON, but there are no read-write files in IQ_SHARED_TEMP

- Both the DQP_ENABLED option and the TEMP_DATA_IN_SHARED_TEMP option are set OFF

The example creates the same table tab1 in the IQ_SHARED_TEMP dbspace in the following cases:

- DQP_ENABLED is ON and there are read-write files in IQ_SHARED_TEMP
- DQP_ENABLED is OFF, TEMP_DATA_IN_SHARED_TEMP is ON, and there are read-write files in IQ_SHARED_TEMP

- **Example 11 –** Create a table tab1 that is enabled to use row-level versioning, and real-time storage in the in-memory RLV store.

```
CREATE TABLE tab1 ( c1 INT, c2 CHAR(25) ) ENABLE RLV STORE
```

Usage

You can create a table for another user by specifying an owner name. If GLOBAL TEMPORARY or LOCAL TEMPORARY is not specified, the table is referred to as a base table. Otherwise, the table is a temporary table.

A created global temporary table exists in the database like a base table and remains in the database until it is explicitly removed by a DROP TABLE statement. The rows in a temporary table are visible only to the connection that inserted the rows. Multiple connections from the same or different applications can use the same temporary table at the same time and each connection sees only its own rows. A given connection inherits the schema of a global temporary table as it exists when the connection first refers to the table. The rows of a temporary table are deleted when the connection ends.

When you create a local temporary table, omit the owner specification. If you specify an owner when creating a temporary table, for example, CREATE TABLE dbo.#temp(col1 int), a base table is incorrectly created.

An attempt to create a base table or a global temporary table will fail, if a local temporary table of the same name exists on that connection, as the new table cannot be uniquely identified by owner.table.

You can, however, create a local temporary table with the same name as an existing base table or global temporary table. References to the table name access the local temporary table, as local temporary tables are resolved first.

For example, consider this sequence:

```
CREATE TABLE t1 (c1 int);
INSERT t1 VALUES (9);

CREATE LOCAL TEMPORARY TABLE t1 (c1 int);
INSERT t1 VALUES (8);

SELECT * FROM t1;
```

The result returned is 8. Any reference to `t1` refers to the local temporary table `t1` until the local temporary table is dropped by the connection.

In a procedure, use the `CREATE LOCAL TEMPORARY TABLE` statement, instead of the `DECLARE LOCAL TEMPORARY TABLE` statement, when you want to create a table that persists after the procedure completes. Local temporary tables created using the `CREATE LOCAL TEMPORARY TABLE` statement remain until they are either explicitly dropped, or until the connection closes.

Local temporary tables created in `IF` statements using `CREATE LOCAL TEMPORARY TABLE` also persist after the `IF` statement completes.

SAP Sybase IQ does not support the `CREATE TABLE ENCRYPTED` clause for table-level encryption of SAP Sybase IQ tables. However, the `CREATE TABLE ENCRYPTED` clause is supported for SQL Anywhere tables in an SAP Sybase IQ database.

IF NOT EXISTS — If the named object already exists, no changes are made and an error is not returned.

{ ENABLE | DISABLE } RLV STORE — Registers this table with the RLV store for real-time in-memory updates. This value overrides the value of the database option **BASE_TABLES_IN_RLV**. Requires the CREATE TABLE system privilege and CREATE permissions on the RLV store dbspace to set this value to **ENABLE**.

Note: The `{ ENABLE | DISABLE } RLV STORE` clause is not supported for IQ temporary tables.

`WITH NULLS NOT DISTINCT` clause can only be specified if you are declaring the index to be `UNIQUE` and allows you to specify that `NULL`s in index keys are not unique. See the `UNIQUE` clause for more information.

IN — Specifies in which database file (dbspace) the table is to be created. Specify `SYSTEM` with this clause to put either a permanent or temporary table in the catalog store. Specify `IQ_SYSTEM_TEMP` to store temporary user objects (tables, partitions, or table indexes) in `IQ_SYSTEM_TEMP` or, if the `TEMP_DATA_IN_SHARED_TEMP` option is set '`ON`', and the `IQ_SHARED_TEMP` dbspace contains `RW` files, in `IQ_SHARED_TEMP`. (You cannot specify the `IN` clause with `IQ_SHARED_TEMP`.) All other use of the `IN` clause is ignored. By default, all permanent tables are placed in the main IQ store, and all temporary tables are placed in the temporary IQ store. Global temporary and local temporary tables can never be in the IQ store.

The following syntax is unsupported:

```
CREATE LOCAL TEMPORARY TABLE tab1(c1 int) IN IQ_SHARED_TEMP
```

The `IN` clause in the *column-definition*, *column-constraint*, *table-constraint*, and *foreign-key* clauses specify the dbspace where the object is to be created. If the `IN` clause is omitted, SAP Sybase IQ creates the object in the dbspace where the table is assigned.

For more information about dbspaces, see *CREATE DBSPACE Statement* .

ON COMMIT — Allowed for temporary tables only. By default, the rows of a temporary table are deleted on COMMIT.

NOT TRANSACTIONAL — Allowed only for temporary tables. A table created using NOT TRANSACTIONAL is not affected by either COMMIT or ROLLBACK.

Note: The NOT TRANSACTIONAL clause is not supported for IQ temporary tables.

The NOT TRANSACTIONAL clause provides performance improvements in some circumstances because operations on nontransactional temporary tables do not cause entries to be made in the rollback log. For example, NOT TRANSACTIONAL might be useful if procedures that use the temporary table are called repeatedly with no intervening COMMIT or ROLLBACK statements.

The parenthesized list following the CREATE TABLE statement can contain these clauses in any order:

AT — Used to create a table at the remote location specified by *location-string*. The local table that is created is a proxy table that maps to the remote location. Tables used as proxy tables must have names of 30 characters or less. The AT clause supports the semicolon (;) as a delimiter. If a semicolon is present anywhere in the *location-string*, the semicolon is the field delimiter. If no semicolon is present, a period is the field delimiter. This allows file names and extensions to be used in the database and owner fields.

Semicolon field delimiters are used primarily with server classes not currently supported; however, you can also use them in situations where a period would also work as a field delimiter. For example, this statement maps the table proxy_a to the SQL Anywhere database mydb on the remote server myasa:

```
CREATE TABLE proxy_a1
AT 'myasa;mydb;;a1'
```

Foreign-key definitions are ignored on remote tables. Foreign-key definitions on local tables that refer to remote tables are also ignored. Primary key definitions are sent to the remote server if the server supports primary keys.

In a simplex environment, you cannot create a proxy table that refers to a remote table on the same node. In a multiplex environment, you cannot create a proxy table that refers to the remote table defined within the multiplex.

For example, in a simplex environment, if you try to create proxy table proxy_e which refers to base table Employees defined on the same node, the CREATE TABLE ... AT statement is rejected with an error message. In a multiplex environment, the CREATE TABLE .... AT statement is rejected if you create proxy table proxy_e from any node (coordinator or secondary) that refers to remote table Employees defined within a multiplex.

column-definition — Defines a column in the table. Allowable data types are described in *Reference: Building Blocks, Tables, and Procedures > SQL Data Types*. Two columns in the

same table cannot have the same name. If NOT NULL is specified, or if the column is in a UNIQUE or PRIMARY KEY constraint, the column cannot contain any NULL values. You can create up to 45,000 columns; however, there might be performance penalties with more than 10,000 columns in a table. The limit on the number of columns per table that allow NULLs is approximately 8*(database-page-size - 30).

- **DEFAULT default-value**—When defining a column for a table, you can specify a default value for the column using the DEFAULT keyword in the CREATE TABLE (and ALTER TABLE) statement. If a DEFAULT value is specified for a column, this DEFAULT value is used as the value of the column in any INSERT (or LOAD) statement that does not specify a value for the column.
- **DEFAULT AUTOINCREMENT** — The value of the DEFAULT AUTOINCREMENT column uniquely identifies every row in a table. Columns of this type are also known as IDENTITY columns, for compatibility with Adaptive Server Enterprise. The IDENTITY/DEFAULT AUTOINCREMENT column stores sequential numbers that are automatically generated during inserts and updates. When using IDENTITY or DEFAULT AUTOINCREMENT, the column must be one of the integer data types, or an exact numeric type, with scale 0. The column value might also be NULL. You must qualify the specified table name with the owner name.

  ON inserts into the table. If a value is not specified for the IDENTITY/DEFAULT AUTOINCREMENT column, a unique value larger than any other value in the column is generated. If an INSERT specifies a value for the column, it is used; if the specified value is not larger than the current maximum value for the column, that value is used as a starting point for subsequent inserts.

  Deleting rows does not decrement the IDENTITY/AUTOINCREMENT counter. Gaps created by deleting rows can only be filled by explicit assignment when using an insert. The database option IDENTITY_INSERT must be set to the table name to perform an insert into an IDENTITY/AUTOINCREMENT column.

  For example, this creates a table with an IDENTITY column and explicitly adds some data to it:

  ```
  CREATE TABLE mytable(c1 INT IDENTITY);
  SET TEMPORARY OPTION IDENTITY_INSERT = "DBA".mytable;
  INSERT INTO mytable VALUES(5);
  ```

  After an explicit insert of a row number less than the maximum, subsequent rows without explicit assignment are still automatically incremented with a value of one greater than the previous maximum.

  You can find the most recently inserted value of the column by inspecting the @@identity global variable.
- **IDENTITY** — A Transact-SQL-compatible alternative to using the AUTOINCREMENT default. In SAP Sybase IQ, the identity column may be created using either the IDENTITY or the DEFAULT AUTOINCREMENT clause.

table-constraint — Helps ensure the integrity of data in the database. There are four types of integrity constraints:

- **UNIQUE** constraint — Identifies one or more columns that uniquely identify each row in the table. No two rows in the table can have the same values in all the named columns. A table may have more than one unique constraint.
- **PRIMARY KEY** constraint—Is the same as a `UNIQUE` constraint except that a table can have only one primary-key constraint. You cannot specify the `PRIMARY KEY` and `UNIQUE` constraints for the same column. The primary key usually identifies the best identifier for a row. For example, the customer number might be the primary key for the customer table.
- **FOREIGN KEY** constraint — Restricts the values for a set of columns to match the values in a primary key or uniqueness constraint of another table. For example, a foreign-key constraint could be used to ensure that a customer number in an invoice table corresponds to a customer number in the customer table.

  Note: You cannot create foreign-key constraints on local temporary tables. Global temporary tables must be created with `ON COMMIT PRESERVE ROWS`.
- **CHECK** constraint — Allows arbitrary conditions to be verified. For example, a check constraint could be used to ensure that a column called `Gender` contains only the values male or female. No row in a table is allowed to violate a constraint. If an `INSERT` or `UPDATE` statement would cause a row to violate a constraint, the operation is not permitted and the effects of the statement are undone.

  Column identifiers in column check constraints that start with the symbol '@' are placeholders for the actual column name. A statement of the form:

  ```
  CREATE TABLE t1(c1 INTEGER CHECK (@foo < 5))
  ```

  is exactly the same as this statement:

  ```
  CREATE TABLE t1(c1 INTEGER CHECK (c1 < 5))
  ```

  Column identifiers appearing in table check constraints that start with the symbol '@'are not placeholders.

If a statement would cause changes to the database that violate an integrity constraint, the statement is effectively not executed and an error is reported. (Effectively means that any changes made by the statement before the error was detected are undone.)

SAP Sybase IQ enforces single-column `UNIQUE` constraints by creating an `HG` index for that column.

Note: You cannot define a column with a `BIT` data type as a `UNIQUE` or `PRIMARY KEY` constraint. Also, the default for columns of `BIT` data type is to not allow `NULL` values; you can change this by explicitly defining the column as allowing `NULL` values.

column-constraint — Restricts the values the column can hold. Column and table constraints help ensure the integrity of data in the database. If a statement would cause a violation of a constraint, execution of the statement does not complete, any changes made by the statement before error detection are undone, and an error is reported. Column constraints are abbreviations for the corresponding table constraints. For example, these are equivalent:

```
CREATE TABLE Products (
    product_num integer UNIQUE
)
CREATE TABLE Products (
    product_num integer,
    UNIQUE ( product_num )
)
```

Column constraints are normally used unless the constraint references more than one column in the table. In these cases, a table constraint must be used.

IQ UNIQUE constraint — Defines the expected cardinality of a column and determines whether the column loads as `Flat FP` or `NBit FP`. An `IQ UNIQUE(n)` value explicitly set to 0 loads the column as `Flat FP`. Columns without an `IQ UNIQUE` constraint implicitly load as `NBit` up to the limits defined by the `FP_NBIT_AUTOSIZE_LIMIT`, `FP_NBIT_LOOKUP_MB`, and `FP_NBIT_ROLLOVER_MAX_MB` options:

- `FP_NBIT_AUTOSIZE_LIMIT` limits the number of distinct values that load as `NBit`
- `FP_NBIT_LOOKUP_MB` sets a threshold for the total `NBit` dictionary size
- `FP_NBIT_ROLLOVER_MAX_MB` sets the dictionary size for implicit `NBit` rollovers from `NBit` to `Flat FP`
- `FP_NBIT_ENFORCE_LIMITS` enforces `NBit` dictionary sizing limits. This option is OFF by default

Using `IQ UNIQUE` with an *n* value less than the `FP_NBIT_AUTOSIZE_LIMIT` is not necessary. Auto-size functionality automatically sizes all low or medium cardinality columns as `NBit`. Use `IQ UNIQUE` in cases where you want to load the column as `Flat FP` or when you want to load a column as `NBit` when the number of distinct values exceeds the `FP_NBIT_AUTOSIZE_LIMIT`.

Note:

- Consider memory usage when specifying high `IQ UNIQUE` values. If machine resources are limited, avoid loads with `FP_NBIT_ENFORCE_LIMITS='OFF'` (default). Prior to SAP Sybase IQ 16.0, an `IQ UNIQUE` *n* value > 16777216 would rollover to `Flat FP`. In 16.0, larger `IQ UNIQUE` values are supported for tokenization, but may require significant memory resource requirements depending on cardinality and column width.
- BIT, BLOB,and CLOB data types do not support `NBit` dictionary compression. If `FP_NBIT_IQ15_COMPATIBILITY='OFF'`, a non-zero `IQ UNIQUE` column specification in a CREATE TABLE or ALTER TABLE statement that includes these data types returns an error.

Integrity Constraints

UNIQUE or **UNIQUE** (*column-name*, …)—No two rows in the table can have the same values in all the named columns. A table may have more than one unique constraint.

There is a difference between a *unique constraint* and a *unique index*. Columns of a unique index are allowed to be `NULL`, while columns in a unique constraint are not. A foreign key can

reference either a primary key or a column with a unique constraint, but not a unique index, because it can include multiple instances of NULL.

PRIMARY KEY or **PRIMARY KEY** (*column-name*, …) — The primary key for the table consists of the listed columns, and none of the named columns can contain any NULL values. SAP Sybase IQ ensures that each row in the table has a unique primary key value. A table can have only one PRIMARY KEY.

When the second form is used (PRIMARY KEY followed by a list of columns), the primary key is created including the columns in the order in which they are defined, not the order in which they are listed.

When a column is designated as PRIMARY KEY, FOREIGN KEY, or UNIQUE, SAP Sybase IQ creates a High_Group index for it automatically. For multicolumn primary keys, this index is on the primary key, not the individual columns. For best performance, you should also index each column with a HG or LF index separately.

REFERENCES *primary-table-name* [(*primary-column-name*)] — Defines the column as a foreign key for a primary key or a unique constraint of a primary table. Normally, a foreign key would be for a primary key rather than an unique constraint. If a primary column name is specified, it must match a column in the primary table which is subject to a unique constraint or primary key constraint, and that constraint must consist of only that one column. Otherwise the foreign key references the primary key of the second table. Primary key and foreign key must have the same data type and the same precision, scale, and sign. Only a non unique single-column HG index is created for a single-column foreign key. For a multicolumn foreign key, SAP Sybase IQ creates a non unique composite HG index. The maximum width of a multicolumn composite key for a unique or non unique HG index is 1KB.

A temporary table cannot have a foreign key that references a base table and a base table cannot have a foreign key that references a temporary table. Local temporary tables cannot have or be referenced by a foreign key.

FOREIGN KEY [*role-name*] [(...)] **REFERENCES** *primary-table-name* [(...)] — Defines foreign-key references to a primary key or a unique constraint in another table. Normally, a foreign key would be for a primary key rather than an unique constraint. (In this description, this other table is called the primary table.)

If the primary table column names are not specified, the primary table columns are the columns in the table's primary key. If foreign key column names are not specified, the foreign-key columns have the same names as the columns in the primary table. If foreign-key column names are specified, then the primary key column names must be specified, and the column names are paired according to position in the lists.

If the primary table is not the same as the foreign-key table, either the unique or primary key constraint must have been defined on the referenced key. Both referenced key and foreign key must have the same number of columns, of identical data type with the same sign, precision, and scale.

The value of the row's foreign key must appear as a candidate key value in one of the primary table's rows unless one or more of the columns in the foreign key contains nulls in a null allows foreign key column.

Any foreign-key column not explicitly defined is automatically created with the same data type as the corresponding column in the primary table. These automatically created columns cannot be part of the primary key of the foreign table. Thus, a column used in both a primary key and foreign key must be explicitly created.

role-name is the name of the foreign key. The main function of *role-name* is to distinguish two foreign keys to the same table. If no *role-name* is specified, the role name is assigned as follows:

1. If there is no foreign key with a *role-name* the same as the table name, the table name is assigned as the *role-name*.
2. If the table name is already taken, the *role-name* is the table name concatenated with a zero-padded 3-digit number unique to the table.

The referential integrity action defines the action to be taken to maintain foreign-key relationships in the database. Whenever a primary key value is changed or deleted from a database table, there may be corresponding foreign key values in other tables that should be modified in some way. You can specify an `ON DELETE` clause, followed by the `RESTRICT` clause:

RESTRICT — Generates an error if you try to update or delete a primary key value while there are corresponding foreign keys elsewhere in the database. Generates an error if you try to update a foreign key so that you create new values unmatched by a candidate key. This is the default action, unless you specify that `LOAD` optionally reject rows that violate referential integrity. This enforces referential integrity at the statement level.

If you use `CHECK ON COMMIT` without specifying any actions, then `RESTRICT` is implied as an action for `DELETE`. SAP Sybase IQ does not support `CHECK ON COMMIT`.

A global temporary table cannot have a foreign key that references a base table and a base table cannot have a foreign key that references a global temporary table. Local temporary tables cannot have or be referenced by a foreign key.

CHECK (*condition*) — No row is allowed to fail the condition. If an `INSERT` statement would cause a row to fail the condition, the operation is not permitted and the effects of the statement are undone.

The change is rejected only if the condition is `FALSE`; in particular, the change is allowed if the condition is `UNKNOWN`. `CHECK` condition is not enforced by SAP Sybase IQ.

Note: If possible, do not define referential integrity foreign key-primary key relationships in SAP Sybase IQ unless you are certain there are no orphan foreign keys.

Remote Tables

Foreign-key definitions are ignored on remote tables. Foreign-key definitions on local tables that refer to remote tables are also ignored. Primary-key definitions are sent to the remote server if the server supports it.

TABLE PARTITIONING

Table partitioning can improve performance by dividing large tables into smaller, more manageable storage objects. Partitions share the same logical attributes of the parent table, but can be placed in separate dbspaces and managed individually. SAP Sybase IQ supports several table partitioning schemes:

- *hash-partitions*
- *range-partitions*
- *composite-partitions*

Note: Range-partitions and composite partitioning schemes, like hash-range partitions, require the separately licensed VLDB Management option.

A *partition-key* is the column or columns that contain the table partitioning keys. Partition keys can contain `NULL` and `DEFAULT` values, but cannot contain:

- `LOB` (`BLOB` or `CLOB`) columns
- `BINARY`, or `VARBINARY` columns
- `CHAR` or `VARCHAR` columns whose length is over 255 bytes
- `BIT` columns
- `FLOAT`/`DOUBLE`/`REAL` columns

PARTITION BY RANGE — Partitions rows by a range of values in the partitioning column. Range partitioning is restricted to a single partition key column and a maximum of 1024 partitions. In a *range-partitioning-scheme*, the *partition-key* is the column that contains the table partitioning keys:

```
range-partition-decl:
  partition-name VALUES <= ( {constant-expr |  MAX } [ , { constant-
expr |  MAX }]... )
    [ IN dbspace-name ]
```

The *partition-name* is the name of a new partition on which table rows are stored. Partition names must be unique within the set of partitions on a table. The *partition-name* is required.

VALUE clause — Specifies the inclusive upper bound for each partition (in ascending order). The user must specify the partitioning criteria for each range partition to guarantee that each row is distributed to only one partition. `NULL`s are allowed for the partition column and rows with `NULL` as partition key value belong to the first table partition. However, `NULL` cannot be the bound value.

There is no lower bound (`MIN` value) for the first partition. Rows of `NULL` cells in the first column of the partition key will go to the first partition. For the last partition, you can either specify an inclusive upper bound or `MAX`. If the upper bound value for the last partition is not

MAX, loading or inserting any row with partition key value larger than the upper bound value of the last partition generates an error.

MAX — Denotes the infinite upper bound and can only be specified for the last partition.

IN — In the *partition-decl*, specifies the dbspace on which rows of the partition should reside.

These restrictions affect partitions keys and bound values for range partitioned tables:

- Partition bounds must be constants, not constant expressions.
- Partition bounds must be in ascending order according to the order in which the partitions were created. That is, the upper bound for the second partition must be higher than for the first partition, and so on.

  In addition, partition bound values must be compatible with the corresponding partition-key column data type. For example, VARCHAR is compatible with CHAR.
- If a bound value has a different data type than that of its corresponding partition key column, SAP Sybase IQ converts the bound value to the data type of the partition key column, with these exceptions:
- Explicit conversions are not allowed. This example attempts an explicit conversion from INT to VARCHAR and generates an error:

```
CREATE TABLE Employees(emp_name VARCHAR(20))
PARTITION BY RANGE(emp_name)
(p1 VALUES <=(CAST (1 AS VARCHAR(20))),
p2 VALUES <= (CAST (10 AS VARCHAR(20)))
```

- Implicit conversions that result in data loss are not allowed. In this example, the partition bounds are not compatible with the partition key type. Rounding assumptions may lead to data loss and an error is generated:

```
CREATE TABLE emp_id (id INT) PARTITION BY RANGE(id) (p1 VALUES <=
(10.5), p2 VALUES <= (100.5))
```

- In this example, the partition bounds and the partition key data type are compatible. The bound values are directly converted to float values. No rounding is required, and conversion is supported:

```
CREATE TABLE id_emp (id FLOAT)
PARTITION BY RANGE(id) (p1 VALUES <= (10),
p2 VALUES <= (100))
```

- Conversions from non-binary data types to binary data types are not allowed. For example, this conversion is not allowed and returns an error:

```
CREATE TABLE newemp (name BINARY)
PARTITION BY RANGE(name)
(p1 VALUES <= ("Maarten"),
p2 VALUES <= ("Zymmerman")
```

- NULL cannot be used as a boundary in a range-partitioned table.
- The row will be in the first partition if the cell value of the 1st column of the partition key evaluated to be NULL. SAP Sybase IQ supports only single column partition keys, so any NULL in the partition key distributes the row to the first partition.

PARTITION BY HASH — Hash partitioning maps data to partitions based on partition-key values processed by an internal hashing function. Hash partition keys are restricted to a

maximum of eight columns with a combined declared column width of 5300 bytes or less. For hash partitions, the table creator determines only the partition key columns; the number and location of the partitions are determined internally.

In a hash-partitioning declaration, the *partition-key* is a column or group of columns, whose composite value determines the partition where each row of data is stored:

```
hash-partitioning-scheme:
  HASH  ( partition-key [ , partition-key, … ] )
```

Restrictions

- You can only hash partition a base table. Attempting to partitioning a global temporary table or a local temporary table raises an error.
- You cannot add, drop, merge, or split a hash partition.
- You cannot add or drop a column from a hash partition key.

Hash-Range Partitions — Hash-range partitioning is a composite partitioning scheme that subpartitions a hash-partitioned table by range. In a hash-range-partitioning-scheme declaration, a `SUBPARTITION BY RANGE` clause adds a new range subpartition to an existing hash-range partitioned table:

```
hash-range-partitioning-scheme:
PARTITION BY HASH  ( partition-key [ , partition-key, … ] )
   [ SUBPARTITION BY RANGE  ( range-partition-decl [ , range-partition-
decl ... ] ) ]
```

The hash partition specifies how the data is logically distributed and colocated; the range subpartition specifies how the data is physically placed. The new range subpartition is logically partitioned by hash with the same hash partition keys as the existing hash-range partitioned table. The range subpartition key is restricted to one column.

Side Effects

- Automatic commit

Standards

- SQL—Vendor extension to `ISO/ANSI SQL` grammar.

  These are vendor extensions:
  - The { **IN** | **ON** } *dbspace-name* clause
  - The **ON COMMIT** clause
  - Some of the default values
- Sybase—Supported by Adaptive Server Enterprise, with some differences.
  - Temporary tables— You can create a temporary table by preceding the table name in a **CREATE TABLE** statement with a pound sign (#). These temporary tables are SAP Sybase IQ declared temporary tables, which are available only in the current connection. For information about declared temporary tables, see *DECLARE LOCAL TEMPORARY TABLE Statement*.

- Physical placement—Physical placement of a table is carried out differently in SAP Sybase IQ and in Adaptive Server Enterprise. The **ON** *segment-name* clause supported by Adaptive Server Enterprise is supported in SAP Sybase IQ, but *segment-name* refers to an IQ dbspace.
- Constraints—SAP Sybase IQ does not support named constraints or named defaults, but does support user-defined data types that allow constraint and default definitions to be encapsulated in the data type definition. It also supports explicit defaults and CHECK conditions in the **CREATE TABLE** statement.
- `NULL` default—By default, columns in Adaptive Server Enterprise default to `NOT NULL`, whereas in SAP Sybase IQ the default setting is `NULL`, to allow `NULL` values. This setting can be controlled using the `ALLOW_NULLS_BY_DEFAULT` option. See *ALLOW_NULLS_BY_DEFAULT Option [TSQL]*. To make your data definition statements transferable, explicitly specify `NULL` or `NOT NULL`.

Permissions

Table Type	Privileges Required
Base table in the IQ main store	Table owned by self – Requires CREATE privilege on the dbspace where the table is created. Also requires one of: • CREATE TABLE system privilege. • CREATE ANY OBJECT system privilege. Table owned by any user – Requires CREATE privilege on the dbspace where the table is created. Also requires one of: • CREATE ANY TABLE system privilege. • CREATE ANY OBJECT system privilege.
Global temporary table	Table owned by self – Requires one of: • CREATE TABLE system privilege. • CREATE ANY OBJECT system privilege. Table owned by any user – Requires one of: • CREATE ANY TABLE system privilege. • CREATE ANY OBJECT system privilege.

Table Type	Privileges Required
Proxy table	Table owned by self – Requires one of: • CREATE PROXY TABLE system privilege. • CREATE ANY TABLE system privilege. • CREATE ANY OBJECT system privilege. Table owned by any user – Requires one of: • CREATE ANY TABLE system privilege. • CREATE ANY OBJECT system privilege.

See also

- *CREATE DBSPACE Statement* on page 161

DELETE Statement

Deletes rows from the database.

Syntax

```
DELETE [ FROM ] [ owner.]table-name
…[ FROM table-list ]
…[ WHERE search-condition ]
```

Examples

- **Example 1 –** Remove employee 105 from the database:

```
DELETE
FROM Employees
WHERE EmployeeID = 105
```

- **Example 2 –** Remove all data prior to 1993 from the `FinancialData` table:

```
DELETE
FROM FinancialData
WHERE Year < 1993
```

- **Example 3 –** Remove all names from the `Contacts` table if they are already present in the `Customers` table:

```
DELETE
FROM Contacts
FROM Contacts, Customers
WHERE Contacts.Surname = Customers.Surname
AND Contacts.GivenName = Customers.GivenName
```

Usage

DELETE deletes all the rows from the named table that satisfy the search condition. If no **WHERE** clause is specified, all rows from the named table are deleted.

DELETE can be used on views provided the **SELECT** statement defining the view has only one table in the **FROM** clause and does not contain a **GROUP BY** clause, an aggregate function, or involve a **UNION** operation.

The optional second **FROM** clause in the **DELETE** statement allows rows to be deleted based on joins. If the second **FROM** clause is present, the **WHERE** clause qualifies the rows of this second **FROM** clause. Rows are deleted from the table name given in the first **FROM** clause.

Note: You cannot use the **DELETE** statement on a join virtual table. If you attempt to delete from a join virtual table, an error is reported.

Correlation Name Resolution

This statement illustrates a potential ambiguity in table names in **DELETE** statements with two **FROM** clauses that use correlation names:

```
DELETE
FROM table_1
FROM table_1 AS alias_1, table_2 AS alias_2
WHERE ...
```

The table `table_1` is identified without a correlation name in the first **FROM** clause, but with a correlation name in the second **FROM** clause. In this case, `table_1` in the first clause is identified with `alias_1` in the second clause; there is only one instance of **table_1** in this statement.

This is an exception to the general rule that where a table is identified with a correlation name and without a correlation name in the same statement, two instances of the table are considered.

Consider this example:

```
DELETE
FROM table_1
FROM table_1 AS alias_1, table_1 AS alias_2
WHERE ...
```

In this case, there are two instances of `table_1` in the second **FROM** clause. There is no way of identifying which instance the first **FROM** clause should be identified with. The usual rules of correlation names apply, and `table_1` in the first **FROM** clause is identified with neither instance in the second clause: there are three instances of `table_1` in the statement.

Standards

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Supported by Adaptive Server Enterprise, including the vendor extension.

Permissions

Requires DELETE privilege on the table.

DROP Statement

Removes objects from the database.

Syntax

```
DROP
{ DBSPACE dbspace-name
| { DATATYPE [ IF EXISTS ]
| DOMAIN [ IF EXISTS ] } datatype-name
| EVENT [ IF EXISTS ]  event-name
| INDEX [ IF EXISTS ]  [ [ owner].table-name.]index-name
| MESSAGE message-number
| TABLE [ IF EXISTS ]  [ owner.]table-name
| VIEW [ IF EXISTS ]  [ owner.]view-name
| MATERIALIZED VIEW [ IF EXISTS ]  [ owner.]view-name
| PROCEDURE [ IF EXISTS ]  [ owner.]procedure-name
| FUNCTION [ IF EXISTS ]  [ owner.]function-name }
```

Examples

- **Example 1 –** Drop the `Departments` table from the database:

  ```
  DROP TABLE Departments
  ```

- **Example 2 –** Drop the `emp_dept` view from the database:

  ```
  DROP VIEW emp_dept
  ```

Usage

DROP removes the definition of the indicated database structure. If the structure is a dbspace, then all tables with any data in that dbspace must be dropped or relocated prior to dropping the dbspace; other structures are automatically relocated. If the structure is a table, all data in the table is automatically deleted as part of the dropping process. Also, all indexes and keys for the table are dropped by **DROP TABLE**.

Use the **IF EXISTS** clause if you do not want an error returned when the **DROP** statement attempts to remove a database object that does not exist.

DROP INDEX deletes any explicitly created index. It deletes an implicitly created index only if there are no unique or foreign-key constraints or associated primary key.

DROP INDEX for a nonunique **HG** index fails if an associated unenforced foreign key exists.

Warning! Do not delete views owned by the DBO user. Deleting such views or changing them into tables might cause problems.

DROP TABLE, **DROP INDEX**, and **DROP DBSPACE** are prevented whenever the statement affects a table that is currently being used by another connection.

DROP TABLE is prevented if the primary table has foreign-key constraints associated with it, including unenforced foreign-key constraints

DROP TABLE is also prevented if the table has an IDENTITY column and IDENTITY_INSERT is set to that table. To drop the table you must clear IDENTITY_INSERT, that is, set IDENTITY_INSERT to '' (an empty string), or set to another table name.

A foreign key can have either a nonunique single or a multicolumn **HG** index. A primary key may have unique single or multicolumn **HG** indexes. You cannot drop the **HG** index implicitly created for an existing foreign key, primary key, and unique constraint.

The four initial dbspaces are `SYSTEM`, `IQ_SYSTEM_MAIN`, `IQ_SYSTEM_TEMP`, and `IQ_SYSTEM_MSG`. You cannot drop these initial dbspaces, but you may drop dbspaces from the IQ main store or catalog store, which may contain multiple dbspaces, as long as at least one dbspace remains with readwrite mode.

You must drop tables in the dbspace before you can drop the dbspace. An error is returned if the dbspace still contains user data; other structures are automatically relocated when the dbspace is dropped. You can drop a dbspace only after you make it read-only.

Note: A dbspace may contain data at any point after it is used by a command, thereby preventing a **DROP DBSPACE** on it.

DROP PROCEDURE is prevented when the procedure is in use by another connection.

DROP DATATYPE is prevented if the data type is used in a table. You must change data types on all columns defined on the user-defined data type to drop the data type. It is recommended that you use **DROP DOMAIN** rather than **DROP DATATYPE**, as **DROP DOMAIN** is the syntax used in the ANSI/ISO SQL3 draft.

Side Effects

- Automatic commit. Clears the Data window in **dbisql**. **DROP TABLE** and **DROP INDEX** close all cursors for the current connection.
- Local temporary tables are an exception; no commit is performed when one is dropped.

Standards

- SQL—ISO/ANSI SQL compliant.
- Sybase—Supported by Adaptive Server Enterprise.

Permissions

DBSPACE clause – Requires the DROP ANY OBJECT system privilege and user must be the only connection to the database.

DOMAIN clause – Requires one of:

- DROP DATATYPE system privilege.
- DROP ANY OBJECT system privilege.
- You own the object.

FUNCTION clause – Requires one of:

- DROP ANY PROCEDURE system privilege.
- DROP ANY OBJECT system privilege.
- You own the function.

INDEX clause – Requires one of:

- DROP ANY INDEX system privilege.
- DROP ANY OBJECT system privilege.
- REFERENCE privilege on the underlying table being indexed.
- You own the underlying table being indexed.

DBA or users with the appropriate privilege can drop an index on tables that are owned other users without using a fully-qualified name. All other users must provide a fully-qualified index name to drop an index on a base table owned by the DBA.

MATERIALIZED VIEW clause – Requires one of:

- DROP ANY MATERIALIZED VIEW system privilege.
- DROP ANY OBJECT system privilege.
- You own the materialized view.

PROCEDURE clause – Requires one of:

- DROP ANY PROCEDURE system privilege.
- DROP ANY OBJECT system privilege.
- You own the procedure.

TABLES clause – Requires one of:

- DROP ANY TABLE system privilege.
- DROP ANY OBJECT system privilege.
- You own the table.

Global temporary tables cannot be dropped unless all users that have referenced the temporary table have disconnected.

VIEW clause – Requires one of:

- DROP ANY VIEW system privilege.
- DROP ANY OBJECT system privilege.
- You own the view.

All other clauses – Requires one of:

- DROP ANY OBJECT system privilege.
- You own the object.

INSERT Statement

Inserts a single row or a selection of rows, from elsewhere in the current database, into the table. This command can also insert a selection of rows from another database into the table.

Syntax

Syntax 1

```
INSERT [ INTO ] [ owner.]table-name [ ( column-name [, …] ) ]
   ... VALUES ( [ expression | DEFAULT,… ) ]
or
INSERT [ INTO ] [ owner.]table-name DEFAULT VALUES
```

Syntax 2

```
INSERT [ INTO ] [ owner.]table-name [ ( column-name [, …] ) ]
... insert-load-options insert-select-load-options
... select-statement
```

Syntax 3

```
INSERT [ INTO ] [ owner.]table-name[ ( column-name [, …] ) ]
... insert-load-options insert-select-load-options
LOCATION 'servername.dbname'
[ location-options ]
... { { select-statement } | 'select statement' }
```

Parameters

- **insert-load-options –**

```
[ LIMIT number-of-rows ]
  [ NOTIFY number-of-rows ]
  [ SKIP number-of-rows ]
```

- **insert-select-load-options –**

```
[ WORD SKIP number ]
  [ IGNORE CONSTRAINT constrainttype [, …] ]
  [ MESSAGE LOG 'string' ROW LOG 'string' [ ONLY LOG logwhat
[, …] ] ]
  [ LOG DELIMITED BY 'string' ]
```

- **constrainttype –**

```
{ CHECK integer
     | UNIQUE integer
     | NULL integer
     | FOREIGN KEY integer
     | DATA VALUE integer
     | ALL integer
   }
```

- **logwhat –**

```
{ CHECK
    | ALL
    | NULL
    | UNIQUE
    | DATA VALUE
    | FOREIGN KEY
    | WORD
  }
```

- **location-options –**

```
[ ENCRYPTED PASSWORD ]
  [ PACKETSIZE packet-size ]
  [ QUOTED_IDENTIFIER { ON | OFF } ]
  [ ISOLATION LEVEL { READ UNCOMMITTED | READ
COMMITTED | SERIALIZABLE } ]
```

Examples

- **Example 1 –** Add an Eastern Sales department to the database:

```
INSERT INTO Departments
(DepartmentID, DepartmentName, DepartmentHeadID)
VALUES (600, 'Eastern Sales', 501)
```

- **Example 2 –** Fill the table `dept_head` with the names of department heads and their departments:

```
INSERT INTO dept_head (name, dept)
  NOTIFY 20
  SELECT Surname || ' ' || GivenName
  AS name,
  dept_name
FROM Employees JOIN Departments
  ON EmployeeID= DepartmentHeadID
```

- **Example 3 –** Insert data from the `l_shipdate` and `l_orderkey` columns of the `lineitem` table from the SAP Sybase IQ database `iqdet` on the remote server `detroit` into the corresponding columns of the `lineitem` table in the current database:

```
INSERT INTO lineitem
  (l_shipdate, l_orderkey)
  LOCATION 'detroit.iqdet'
  PACKETSIZE 512
  ' SELECT l_shipdate, l_orderkey
FROM lineitem '
```

- **Example 4 –** The INSERT statement permits a list of values allowing several rows to be inserted at once.

```
INSERT into t1 values( 10, 20, 30 ), ( 11, 21, 31 ), ( 12, 22,
32 )
```

Usage

Syntax 1 allows the insertion of a single row with the specified expression values. If the list of column names is not specified, the values are inserted into the table columns in the order they were created (the same order as retrieved with **SELECT ***). The row is inserted into the table at an arbitrary position. (In relational databases, tables are not ordered.)

Syntax 2 allows the user to perform a mass insertion into a table using the results of a fully general **SELECT** statement. Insertions are done in an arbitrary order unless the **SELECT** statement contains an **ORDER BY** clause. The columns from the select list are matched ordinally with the columns specified in the column list, or sequentially in the order in which the columns were created.

Note: The **NUMBER(*)** function is useful for generating primary keys with Syntax 2 of the **INSERT** statement.

Syntax 3 **INSERT...LOCATION** is a variation of Syntax 2 that allows you to insert data from an Adaptive Server Enterprise or SAP Sybase IQ database. The *servername.dbname* specified in the **LOCATION** clause identifies the remote server and database for the table in the **FROM** clause. To use Syntax 3, the Adaptive Server Enterprise or SAP Sybase IQ remote server to which you are connecting must exist in the SAP Sybase Open Client `interfaces` or `sql.ini` file on the local machine.

In queries using Syntax 3, you can insert a maximum of 2147483647 rows.

The **SELECT** statement can be delimited by either curly braces or straight single quotation marks.

Note: Curly braces represent the start and end of an escape sequence in the ODBC standard, and might generate errors in the context of ODBC or Sybase Control Center. The workaround is to use single quotes to escape the **SELECT** statement.

The local SAP Sybase IQ server connects to the server and database you specify in the **LOCATION** clause. The results from the queries on the remote tables are returned and the local server inserts the results in the current database. If you do not specify a server name in the **LOCATION** clause, SAP Sybase IQ ignores any database name you specify, since the only choice is the current database on the local server.

When SAP Sybase IQ connects to the remote server, **INSERT...LOCATION** uses the remote login for the user ID of the current connection, if a remote login has been created with **CREATE EXTERNLOGIN** and the remote server has been defined with a **CREATE SERVER** statement. If the remote server is not defined, or if a remote login has not been created for the user ID of the current connection, SAP Sybase IQ connects using the user ID and password of the current connection.

Note: If you rely on the user ID and password of the current connection, and a user changes the password, you must stop and restart the server before the new password takes effect on the

remote server. Remote logins created with **CREATE EXTERNLOGIN** are unaffected by changes to the password for the default user ID.

Creating a remote login with the **CREATE EXTERNLOGIN** statement and defining a remote server with a **CREATE SERVER** statement sets up an external login and password for **INSERT...LOCATION** such that any user can use the login and password in any context. This avoids possible errors due to inaccessibility of the login or password, and is the recommended way to connect to a remote server.

For example, user `russid` connects to the SAP Sybase IQ database and executes this statement:

```
INSERT local_SQL_Types LOCATION 'ase1.ase1db'
{SELECT int_col FROM SQL_Types};
```

On server `ase1`, there exists user ID `ase1user` with password `sybase`. The owner of the table `SQL_Types` is `ase1user`. The remote server is defined on the IQ server as:

```
CREATE SERVER ase1 CLASS 'ASEJDBC'
USING 'system1:4100';
```

The external login is defined on the IQ server as:

```
CREATE EXTERNLOGIN russid TO ase1 REMOTE LOGIN ase1user IDENTIFIED BY
sybase;
```

INSERT...LOCATION connects to the remote server `ase1` using the user ID `ase1user` and the password `sybase` for user `russid`.

Use the **ENCRYPTED PASSWORD** parameter to specify the use of Open Client Library default password encryption when connecting to a remote server. If **ENCRYPTED PASSWORD** is specified and the remote server does not support Open Client Library default password encryption, an error is reported indicating that an invalid user ID or password was used.

When used as a remote server, SAP Sybase IQ supports TDS password encryption. The SAP Sybase IQ server accepts a connection with an encrypted password sent by the client. For information on connection properties to set for password encryption, see *Software Developer's Kit 15.5 > Open Client Client-Library/C Reference Manual > Client-Library Topics > Security features > Adaptive Server Enterprise security features > Security handshaking: encrypted password* for Open Server 15.5.

Note: Password encryption requires Open Client 15.0. TDS password encryption requires Open Client 15.0 ESD #7 or later.

To enable the SAP Sybase IQ server to accept a jConnect connection with an encrypted password, set the jConnect **ENCRYPT_PASSWORD** connection property to true.

The **PACKETSIZE** parameter specifies the TDS packet size in bytes. The default TDS packet size on most platforms is 512 bytes. If your application is receiving large amounts of text or bulk data across a network, then a larger packet size might significantly improve performance.

The value of *packet-size* must be a multiple of 512 either equal to the default network packet size or between the default network packet size and the maximum network packet size. The maximum network packet size and the default network packet size are multiples of 512 in the range 512 – 524288 bytes. The maximum network packet size is always greater than or equal to the default network packet size.

If **INSERT...LOCATION PACKETSIZE** *packet-size* is not specified or is specified as zero, then the default packet size value for the platform is used.

When **INSERT...LOCATION** is transferring data between an SAP Sybase IQ server and a remote SAP Sybase IQ or Adaptive Server Enterprise server, the value of the **INSERT...LOCATION** TDS **PACKETSIZE** parameter is always 512 bytes, even if you specify a different value for **PACKETSIZE**.

Note: If you specify an incorrect packet size (for example 933, which is not a multiple of 512), the connection attempt fails with an Open Client **ct_connect** "Connection failed" error. Any unsuccessful connection attempt returns a generic "Connection failed" message. The Adaptive Server Enterprise error log might contain more specific information about the cause of the connection failure.

Use the **QUOTED_IDENTIFIER** parameter to specify the setting of the `QUOTED_IDENTIFIER` option on the remote server. The default setting is 'OFF.' You set **QUOTED_IDENTIFIER** to 'ON' only if any of the identifiers in the **SELECT** statement are enclosed in double quotes, as in this example using 'c1':

```
INSERT INTO foo
LOCATION 'ase.database'
QUOTED_IDENTIFIER ON {select "c1" from xxx};
```

Use the **ISOLATION LEVEL** parameter to specify an isolation level for the connection to a remote server.

Isolation level	Characteristics
READ UNCOMMITTED	• Isolation level 0 • Read permitted on row with or without write lock • No read locks are applied • No guarantee that concurrent transaction will not modify row or roll back changes to row
READ COMMITTED	• Isolation level 1 • Read only permitted on row with no write lock • Read lock acquired and held for read on current row only, but released when cursor moves off the row • No guarantee that data will not change during transaction

Isolation level	Characteristics
SERIALIZABLE	• Isolation level 3 • Read only permitted on rows in result without write lock • Read locks acquired when cursor is opened and held until transaction ends

SAP Sybase IQ does not support the Adaptive Server Enterprise data type `TEXT`, but you can execute **INSERT...LOCATION** (Syntax 3) from both an IQ `CHAR` or `VARCHAR` column whose length is greater than 255 bytes, and from an ASE database column of data type `TEXT`. ASE `TEXT` and `IMAGE` columns can be inserted into columns of other SAP Sybase IQ data types, if SAP Sybase IQ supports the internal conversion. By default, if a remote data column contains over 2GB, SAP Sybase IQ silently truncates the column value to 2GB.

Warning! SAP Sybase IQ does not support the Adaptive Server Enterprise data types `UNICHAR`, `UNIVARCHAR`, or `UNITEXT`. An **INSERT...LOCATION** command from `UNICHAR` or `UNITEXT` to `CHAR` or `CLOB` columns in the ISO_BINENG collation may execute without error; if this happens, the data in the columns may be inconsistent. An error is reported in this situation, only if the conversion fails.

Users must be specifically licensed to use the large object functionality of the Unstructured Data Analytics Option.

Note: If you use **INSERT...LOCATION** to insert data selected from a **VARBINARY** column, set **ASE_BINARY_DISPLAY** to OFF on the remote database.

INSERT...LOCATION (Syntax 3) does not support the use of variables in the **SELECT** statement.

Inserts can be done into views, provided the **SELECT** statement defining the view has only one table in the **FROM** clause and does not contain a **GROUP BY** clause, an aggregate function, or involve a **UNION** operation.

Character strings inserted into tables are always stored in the case they are entered, regardless of whether the database is case-sensitive or not. Thus, a string “Value” inserted into a table is always held in the database with an uppercase V and the remainder of the letters lowercase. **SELECT** statements return the string as 'Value.' If the database is not case-sensitive, however, all comparisons make 'Value' the same as 'value,' 'VALUE,'' and so on. Further, if a single-column primary key already contains an entry Value, an **INSERT** of value is rejected, as it would make the primary key not unique.

Whenever you execute an **INSERT...LOCATION** statement, SAP Sybase IQ loads the localization information needed to determine language, collation sequence, character set, and date/time format. If your database uses a nondefault locale for your platform, you must set an environment variable on your local client to ensure that SAP Sybase IQ loads the correct information.

If you set the LC_ALL environment variable, SAP Sybase IQ uses its value as the locale name. If LC_ALL is not set, SAP Sybase IQ uses the value of the LANG environment variable. If neither variable is set, SAP Sybase IQ uses the default entry in the locales file.

Use the `(DEFAULT)`, `DEFAULT VALUES` or `VALUES()` clauses in an to insert rows with all default values. Assuming that there are 3 columns in table t2, these examples are semantically equivalent:

```
INSERT INTO t2 values (DEFAULT, DEFAULT, DEFAULT);
```

```
INSERT INTO t2 DEFAULT VALUES;
```

```
INSERT INTO t2() VALUES();
```

INSERT...VALUES also supports multiple rows. The following example inserts 3 rows into table t1:

```
CCREATE TABLE t1(c1 varchar(30));
INSERT INTO t1 VALUES  ('morning'),('afternoon'),
    ('evening');
```

SAP Sybase IQ treats all load/inserts as full-width inserts. Columns not explicitly specified on the load/insert statement, the value loaded will either be the column's `DEFAULT` value (if one is defined) or `NULL` (if no `DEFAULT` value is defined for the column).

The **LIMIT** option specifies the maximum number of rows to insert into the table from a query. The default is 0 for no limit. The maximum is 2GB -1.

The **NOTIFY** option specifies that you be notified with a message each time the number of rows are successfully inserted into the table. The default is every 100,000 rows.

The **SKIP** option lets you define a number of rows to skip at the beginning of the input tables for this insert. The default is 0.

For information on the *insert-select-load-options* **WORD SKIP**, **IGNORE CONSTRAINT**, **MESSAGE LOG**, **ROW LOG**, and **LOG DELIMITED BY** and the *constrainttype* and *logwhat* parameters, see the *LOAD TABLE Statement*.

An **INSERT** on a multicolumn index must include all columns of the index.

SAP Sybase IQ supports column DEFAULT values for **INSERT...VALUES**, **INSERT...SELECT**, and **INSERT...LOCATION**. If a DEFAULT value is specified for a column, this DEFAULT value is used as the value of the column in any **INSERT** (or **LOAD**) statement that does not specify a value for the column.

An **INSERT** from a stored procedure or function is not permitted, if the procedure or function uses **COMMIT**, **ROLLBACK**, or some **ROLLBACK TO SAVEPOINT** statements.

The result of a **SELECT…FROM** may be slightly different from the result of an **INSERT… SELECT…FROM** due to an internal data conversion of an imprecise data type, such as `DOUBLE` or `NUMERIC`, for optimization during the insert. If a more precise result is required,

a possible workaround is to declare the column as a DOUBLE or NUMERIC data type with a higher precision.

Standards

- SQL—ISO/ANSI SQL compliant.
- Sybase—Supported by Adaptive Server Enterprise (excluding the *insert-load-options*).

Permissions

Requires INSERT privilege on the table.

See also

- *LOAD TABLE Statement* on page 193

LOAD TABLE Statement

Imports data into a database table from an external file.

Syntax

```
LOAD [ INTO ] TABLE [ owner.]table-name
... ( load-specification [, …] )
...  { FROM | USING [ CLIENT ] FILE }
{ 'filename-string' | filename-variable } [, …]
... [ CHECK CONSTRAINTS { ON | OFF } ]
... [ DEFAULTS { ON | OFF } ]
... [ QUOTES OFF ]
... ESCAPES OFF
... [ FORMAT { ascii | binary | bcp } ]
... [ DELIMITED BY 'string' ]
... [ STRIP { OFF | RTRIM } ]
... [ WITH CHECKPOINT { ON | OFF } ]
... [ BYTE ORDER { NATIVE | HIGH | LOW } ]
... [ LIMIT number-of-rows ]
... [ NOTIFY number-of-rows ]
... [ ON FILE ERROR { ROLLBACK | FINISH | CONTINUE } ]
... [ PREVIEW { ON | OFF } ]
... [ ROW DELIMITED BY 'delimiter-string' ]
... [ SKIP number-of-rows ]
... [ HEADER SKIP number [ HEADER DELIMITED BY 'string' ] ]
... [ WORD SKIP number ]
... [ ON PARTIAL INPUT ROW { ROLLBACK | CONTINUE } ]
... [ IGNORE CONSTRAINT constrainttype [, …] ]
... [ MESSAGE LOG 'string' ROW LOG 'string' [ ONLY LOG logwhat [, …] ]
... [ LOG DELIMITED BY 'string' ]
```

Parameters

- **load-specification –**

```
{ column-name [ column-spec ]
        | FILLER ( filler-type )
      }
```

- **column-spec –**

```
{ ASCII ( input-width )
        | BINARY [ WITH NULL BYTE ]
        | PREFIX { 1 | 2 | 4 }
        | 'delimiter-string'
        | DATE [ FORMAT ] ( input-date-format ) [, input-date-
format, …]
        | DATETIME [ FORMAT ] ( input-datetime-format [, input-
datetime-format, …]  )
        | ENCRYPTED ( data-type 'key-string' [, 'algorithm-
string' ] )
        | DEFAULT default-value
        } [ NULL ( { BLANKS | ZEROS | 'literal',  …}
      )
```

- **filler-type –**

```
{ input-width
        | PREFIX { 1 | 2 | 4 }
        | 'delimiter-string'
      }
```

- **constrainttype –**

```
{ CHECK integer
        | UNIQUE integer
        | NULL integer
        | FOREIGN KEYinteger
        | DATA VALUE integer
        | ALL integer
      }
```

- **logwhat –**

```
{ CHECK
        | ALL
        | NULL
        | UNIQUE
        | DATA VALUE
        | FOREIGN KEY
        | WORD
    }
```

Examples

- **Example 1 –** Load data from one file into the `Products` table on a Windows system. A tab is used as the column delimiter following the `Description` and `Color` columns:

```
LOAD TABLE Products
( ID ASCII(6),
FILLER(1),
Name   ASCII(15),
FILLER(1),
```

```
Description   '\x09',
Size   ASCII(2),
FILLER(1),
Color   '\x09',
Quantity   PREFIX 2,
UnitPrice   PREFIX 2,
FILLER(2) )
FROM 'C:\\mydata\\source1.dmp'
QUOTES OFF
ESCAPES OFF
BYTE ORDER LOW
NOTIFY 1000
```

- **Example 2 –** Load data from a file `a.inp` on a client computer:

```
LOAD TABLE t1(c1,c2,filler(30))
USING CLIENT FILE 'c:\\client-data\\a.inp'
QUOTES OFF ESCAPES OFF
IGNORE CONSTRAINT UNIQUE 0, NULL 0
MESSAGE LOG 'c:\\client-data\\m.log'
ROW LOG 'c:\\client-data\\r.log'ONLY LOG UNIQUE
```

- **Example 3 –** Load data from two files into the `product_new` table (which allows NULL values) on a UNIX system. The tab character is the default column delimiter, and the newline character is the row delimiter:

```
LOAD TABLE product_new
( id,
name,
description,
size,
color   '\x09'   NULL( 'null', 'none', 'na' ),
quantity   PREFIX 2,
unit_price   PREFIX 2 )
FROM '/s1/mydata/source2.dump',
'/s1/mydata/source3.dump'
QUOTES OFF
ESCAPES OFF
FORMAT ascii
DELIMITED BY '\x09'
ON FILE ERROR CONTINUE
ROW DELIMITED BY '\n'
```

- **Example 4 –** Ignore 10 word-length violations; on the 11th, deploy the new error and roll back the load:

```
load table PTAB1(
       ck1          ','  null ('NULL') ,
       ck3fk2c2     ','  null ('NULL') ,
       ck4          ','  null ('NULL') ,
       ck5          ','  null ('NULL') ,
       ck6c1        ','  null ('NULL') ,
       ck6c2        ','  null ('NULL') ,
       rid          ','  null ('NULL')  )
FROM 'ri_index_selfRI.inp'
       row delimited by '\n'
       LIMIT 14   SKIP 10
```

```
       IGNORE CONSTRAINT UNIQUE 2, FOREIGN KEY 8
       word skip 10 quotes off escapes off strip
       off
```

- **Example 5 –** Load data into table `t1` from the **BCP** character file `bcp_file.bcp` using the **FORMAT BCP** load option:

```
LOAD TABLE t1 (c1, c2, c3)
FROM 'bcp_file.bcp'
FORMAT BCP
...
```

- **Example 6 –** Load default values 12345 into `c1` using the **DEFAULT** load option, and load `c2` and `c3` with data from the `LoadConst04.dat` file:

```
LOAD TABLE t1 (c1 DEFAULT '12345 ', c2, c3, filler(1))
FROM 'LoadConst04.dat'
STRIP OFF
QUOTES OFF
ESCAPES OFF
DELIMITED BY ',';
```

- **Example 7 –** Load `c1` and `c2` with data from the file `bcp_file.bcp` using the **FORMAT BCP** load option and set `c3` to the value 10:

```
LOAD TABLE t1 (c1, c2, c3 DEFAULT '10')
FROM 'bcp_file.bcp'
FORMAT BCP
QUOTES OFF
ESCAPES OFF;
```

- **Example 8 –** This code fragment ignores one header row at the beginning of the data file, where the header row is delimited by '&&':

```
LOAD TABLE
...HEADER SKIP 1 HEADER DELIMITED by '&&'
```

- **Example 9 –** This code fragment ignores 2 header rows at the beginning of the data file, where each header row is delimited by '\n':

```
LOAD TABLE
...HEADER SKIP 2
```

- **Example 10 –** Load a file into a RLV-enabled table.

  Load data into RLV-enabled table `rvt1` from the **BCP** character file `bcp_file.bcp` using the **FORMAT BCP** load option:

```
LOAD TABLE rvt1 (c1, c2, c3)
FROM 'bcp_file.bcp'
FORMAT BCP
...
```

Usage

The **LOAD TABLE** statement allows efficient mass insertion into a database table from a file with ASCII or binary data.

The **LOAD TABLE** options also let you control load behavior when integrity constraints are violated and to log information about the violations.

You can use **LOAD TABLE** on a temporary table, but the temporary table must have been declared with **ON COMMIT PRESERVE ROWS**, or the next **COMMIT** removes the rows you have loaded.

You can also specify more than one file to load data. In the **FROM** clause, specify each `filename-string` separated by commas. Because of resource constraints, SAP Sybase IQ does not guarantee that all the data can be loaded. If resource allocation fails, the entire load transaction is rolled back. The files are read one at a time, and processed in the order specified in the **FROM** clause. Any **SKIP** or **LIMIT** value only applies in the beginning of the load, not for each file.

Note: When loading a multiplex database, use absolute (fully qualified) paths in all file names. Do not use relative path names.

LOAD TABLE supports loading of large object (LOB) data.

SAP Sybase IQ supports loading from both ASCII and binary data, and it supports both fixed- and variable-length formats. To handle all of these formats, you must supply a *load-specification* to tell SAP Sybase IQ what kind of data to expect from each "column" or field in the source file. The *column-spec* lets you define these formats:

- ASCII with a fixed length of bytes. The *input-width* value is an integer indicating the fixed width in bytes of the input field in every record.
- Binary or non-binary fields that use a number of PREFIX bytes (1, 2, or 4) to specify the length of the input.

  There are two parts related to a **PREFIX** clause:
  - Prefix value – always a binary value.
  - Associated data bytes – always character format; never binary format.

  If the data is unloaded using the extraction facility with the `TEMP_EXTRACT_BINARY` option set ON, you must use the **BINARY WITH NULL BYTE** parameter for each column when you load the binary data.
- Variable-length characters delimited by a separator. You can specify the terminator as hexadecimal ASCII characters. The *delimiter-string* can be any string of up to 4 characters, including any combination of printable characters, and any 8-bit hexadecimal ASCII code that represents a nonprinting character. For example, specify:
  - '\x09' to represent a tab as the terminator.
  - '\x00' for a null terminator (no visible terminator as in "C" strings).
  - '\x0a' for a newline character as the terminator. You can also use the special character combination of '\n' for newline.

  Note: The delimiter string can be from 1 to 4 characters long, but you can specify only a single character in the **DELIMITED BY** clause. For **BCP**, the delimiter can be up to 10 characters.

- DATE or DATETIME string as ASCII characters. You must define the *input-date-format* or *input-datetime-format* of the string using one of the corresponding formats for the date and datetime data types supported by SAP Sybase IQ. Use **DATE** for date values and **DATETIME** for datetime and time values.

Table 3. Formatting Dates and Times

Option	Meaning
yyyy or YYYY yy or YY	Represents number of year. Default is current year.
mm or MM	Represents number of month. Always use leading zero or blank for number of the month where appropriate, for example, '05' for May. `DATE` value must include a month. For example, if the **DATE** value you enter is 1998, you receive an error. If you enter '03', SAP Sybase IQ applies the default year and day and converts it to '1998-03-01'.
dd or DD jjj or JJJ	Represents number of day. Default day is 01. Always use leading zeros for number of day where appropriate, for example, '01' for first day. J or j indicates a Julian day (1 to 366) of the year.
hh HH	Represents hour. Hour is based on 24-hour clock. Always use leading zeros or blanks for hour where appropriate, for example, '01' for 1 am. '00' is also valid value for hour of 12 a.m.
nn	Represents minute. Always use leading zeros for minute where appropriate, for example, '08' for 8 minutes.
ss[.ssssss]	Represents seconds and fraction of a second.
aa	Represents the a.m. or p.m. designation.
pp	Represents the p.m. designation only if needed. (This is an incompatibility with SAP Sybase IQ versions earlier than 12.0; previously, “pp” was synonymous with “aa”.)
hh	SAP Sybase IQ assumes zero for minutes and seconds. For example, if the **DATETIME** value you enter is '03', SAP Sybase IQ converts it to '03:00:00.0000'.
hh:nn or hh:mm	SAP Sybase IQ assumes zero for seconds. For example, if the time value you enter is '03:25', SAP Sybase IQ converts it to '03:25:00.0000'.

Table 4. Sample DATE and DATETIME Format Options

Input data	Format specification
12/31/98	**DATE** ('MM/DD/YY')
19981231	**DATE** ('YYYYMMDD')
123198140150	**DATETIME** ('MMDDYYhhnnss')
14:01:50 12-31-98	**DATETIME** ('hh:nn:ss MM-DD-YY')

Input data	Format specification
18:27:53	**DATETIME** ('hh:nn:ss')
12/31/98 02:01:50AM	**DATETIME** ('MM/DD/YY hh:nn:ssaa')

SAP Sybase IQ has built-in load optimizations for common date, time, and datetime formats. If your data to be loaded matches one of these formats, you can significantly decrease load time by using the appropriate format.

You can also specify the date/time field as an ASCII fixed-width field (as described above) and use the FILLER(1) option to skip the column delimiter.

The NULL portion of the *column-spec* indicates how to treat certain input values as NULL values when loading into the table column. These characters can include BLANKS, ZEROS, or any other list of literals you define. When specifying a NULL value or reading a NULL value from the source file, the destination column must be able to contain NULLs.

ZEROS are interpreted as follows: the cell is set to NULL if (and only if) the input data (before conversion, if ASCII) is all binary zeros (and not character zeros).

- If the input data is character zero, then:
  1. NULL (ZEROS) never causes the cell to be NULL.
  2. NULL ('0') causes the cell to be NULL.
- If the input data is binary zero (all bits clear), then:
  1. NULL (ZEROS) causes the cell to be NULL.
  2. NULL ('0') never causes the cell to be NULL.

For example, if your **LOAD** statement includes `col1 date('yymmdd') null(zeros)` and the date is 000000, you receive an error indicating that 000000 cannot be converted to a `DATE`(4). To get **LOAD TABLE** to insert a NULL value in `col1` when the data is 000000, either write the NULL clause as `null('000000')`, or modify the data to equal binary zeros and use NULL(ZEROS).

If the length of a `VARCHAR` cell is zero and the cell is not NULL, you get a zero-length cell. For all other data types, if the length of the cell is zero, SAP Sybase IQ inserts a NULL. This is ANSI behavior. For non-ANSI treatment of zero-length character data, set the **NON_ANSI_NULL_VARCHAR** database option.

Use the **DEFAULT** option to specify a load default column value. You can load a default value into a column, even if the column does not have a default value defined in the table schema. This feature provides more flexibility at load time.

- The **LOAD TABLE DEFAULTS** option must be ON in order to use the default value specified in the **LOAD TABLE** statement. If the **DEFAULTS** option is OFF, the specified load default value is not used and a NULL value is inserted into the column instead.

- The **LOAD TABLE** command must contain at least one column that needs to be loaded from the file specified in the **LOAD TABLE** command. Otherwise, an error is reported and the load is not performed.
- The specified load default value must conform to the supported default values for columns and default value restrictions. The **LOAD TABLE DEFAULT** option does not support **AUTOINCREMENT**, **IDENTITY**, or **GLOBAL AUTOINCREMENT** as a load default value.
- The **LOAD TABLE DEFAULT** *default-value* must be of the same character set as that of the database.
- Encryption of the default value is not supported for the load default values specified in the **LOAD TABLE DEFAULT** clause.
- A constraint violation caused by evaluation of the specified load default value is counted for each row that is inserted in the table.

Another important part of the *load-specification* is the **FILLER** option. This option indicates you want to skip over a specified field in the source input file. For example, there may be characters at the end of rows or even entire fields in the input files that you do not want to add to the table. As with the *column-spec* definition, **FILLER** specifies ASCII fixed length of bytes, variable length characters delimited by a separator, and binary fields using PREFIX bytes.

The *filename-string* is passed to the server as a string. The string is therefore subject to the same formatting requirements as other SQL strings. In particular:

- To indicate directory paths in Windows systems, the backslash character \ must be represented by two backslashes. Therefore, the statement to load data from the file `c:\temp\input.dat` into the `Employees` table is:

```
LOAD TABLE Employees
FROM 'c:\\temp\\input.dat' ...
```

- The path name is relative to the database server, not to the client application. If you are running the statement on a database server on some other computer, the directory names refers to directories on the server machine, not on the client machine.

Descriptions of each statement clause follow:

USING— **USING FILE** loads one or more files from the server. This clause is synonymous with specifying the **FROM** *filename* clause. **USING CLIENT FILE** bulk loads one or more files from a client. The character set of the file on the client side must be the same as the server collation. SAP Sybase IQ serially processes files in the file list. Each file is locked in read mode as it is processed, then unlocked. Client-side bulk loading incurs no administrative overhead, such as extra disk space, memory or network-monitoring daemon requirements.

When bulk loading large objects, the **USING CLIENT FILE** clause applies to both primary and secondary files.

During client-side loads, the **IGNORE CONSTRAINT** log files are created on the client host and any error while creating the log files causes the operation to roll back.

Client-side bulk loading is supported by Interactive SQL and ODBC/JDBC clients using the Command Sequence protocol. It is not supported by clients using the TDS protocol. For data

security over a network, use Transport Layer Security. To control who can use client-side bulk loads, use the secure feature (**-sf**) server startup switch, the ALLOW_READ_CLIENT_FILE database option, and/or the READCLIENTFILE access control.

The **LOAD TABLE FROM** clause is deprecated, but may be used to specify a file that exists on the server.

This example loads data from the file a.inp on a client computer.

```
LOAD TABLE t1(c1,c2,filler(30))
USING CLIENT FILE 'c:\\client-data\\a.inp'
QUOTES OFF ESCAPES OFF
IGNORE CONSTRAINT UNIQUE 0, NULL 0
MESSAGE LOG 'c:\\client-data\\m.log'
ROW LOG 'c:\\client-data\\r.log'
ONLY LOG UNIQUE
```

CHECK CONSTRAINTS—This option defaults to ON. When you specify **CHECK CONSTRAINTS ON**, check constraints are evaluated and you are free to ignore or log them.

Setting **CHECK CONSTRAINTS OFF** causes SAP Sybase IQ to ignore all check constraint violations. This can be useful, for example, during database rebuilding. If a table has check constraints that call user-defined functions that are not yet created, the rebuild fails unless this option is set to OFF.

This option is mutually exclusive to the following options. If any of these options are specified in the same load, an error results:

- **IGNORE CONSTRAINT ALL**
- **IGNORE CONSTRAINT CHECK**
- **LOG ALL**
- **LOG CHECK**

DEFAULTS—If the **DEFAULTS** option is ON (the default) and the column has a default value, that value is used. If the **DEFAULTS** option is OFF, any column not present in the column list is assigned NULL.

The setting for the **DEFAULTS** option applies to all column DEFAULT values, including AUTOINCREMENT.

QUOTES—This parameter is optional and the default is ON. With **QUOTES** turned on, **LOAD TABLE** expects input strings to be enclosed in quote characters. The quote character is either an apostrophe (single quote) or a quotation mark (double quote). The first such character encountered in a string is treated as the quote character for the string. String data must be terminated with a matching quote.

With **QUOTES ON**, column or row delimiter characters can be included in the column value. Leading and ending quote characters are assumed not to be part of the value and are excluded from the loaded data value.

To include a quote character in a value with **QUOTES ON**, use two quotes. For example, this line includes a value in the third column that is a single quote character:

```
'123 High Street, Anytown', '(715)398-2354',''''
```

With **STRIP** turned on (the default), trailing blanks are stripped from values before they are inserted. Trailing blanks are stripped only for non-quoted strings. Quoted strings retain their trailing blanks. Leading blank or TAB characters are trimmed only when the **QUOTES** setting is ON.

The data extraction facility provides options for handling quotes (`TEMP_EXTRACT_QUOTES`, `TEMP_EXTRACT_QUOTES_ALL`, and `TEMP_EXTRACT_QUOTE`). If you plan to extract data to be loaded into an IQ main store table and the string fields contain column or row delimiter under default ASCII extraction, use the `TEMP_EXTRACT_BINARY` option for the extract and the **FORMAT binary** and **QUOTES OFF** options for **LOAD TABLE**.

Limits:

- **QUOTES ON** applies only to column-delimited ASCII fields.
- With **QUOTES ON**, the first character of a column delimiter or row terminator cannot be a single or double quote mark.
- The **QUOTES** option does not apply to loading binary large object (`BLOB`) or character large object (`CLOB`) data from the secondary file, regardless of its setting. A leading or trailing quote is loaded as part of `CLOB` data. Two consecutive quotes between enclosing quotes are loaded as two consecutive quotes with the **QUOTES ON** option.
- Adaptive Server Enterprise **BCP** does not support the **QUOTES** option. All field data is copied in or out equivalent to the **QUOTES OFF** setting. As **QUOTES ON** is the default setting for the SAP Sybase IQ **LOAD TABLE** statement, you must specify **QUOTES OFF** when importing ASE data from **BCP** output to an SAP Sybase IQ table.

Exceptions:

- If **LOAD TABLE** encounters any nonwhite characters after the ending quote character for an enclosed field, this error is reported and the load operation is rolled back:

  ```
  Non-SPACE text found after ending quote character for
  an enclosed field.
  SQLSTATE: QTA14    SQLCODE: -1005014L
  ```

- With **QUOTES ON**, if a single or double quote is specified as the first character of the column delimiter, an error is reported and the load operation fails:

  ```
  Single or double quote mark cannot be the 1st character
  of column delimiter or row terminator with QUOTES option
  ON.
  SQLSTATE: QCA90    SQLCODE: -1013090L
  ```

ESCAPES—If you omit a *column-spec* definition for an input field and **ESCAPES** is ON (the default), characters following the backslash character are recognized and interpreted as special characters by the database server. You can include newline characters as the combination \n, and other characters as hexadecimal ASCII codes, such as \x09 for the tab character. A sequence of two backslash characters (\\ ) is interpreted as a single backslash. For SAP Sybase IQ, you must set **ESCAPES** OFF.

FORMAT—SAP Sybase IQ supports ASCII and binary input fields. The format is usually defined by the *column-spec* described above. If you omit that definition for a column, by default SAP Sybase IQ uses the format defined by this option. Input lines are assumed to have **ascii** (the default) or **binary** fields, one row per line, with values separated by the column delimiter character.

SAP Sybase IQ also accepts data from BCP character files as input to the **LOAD TABLE** command.

- The BCP data file loaded into SAP Sybase IQ tables using the **LOAD TABLE FORMAT BCP** statement must be exported (**BCP OUT**) in cross-platform file format using the **-c** option.
- For **FORMAT BCP**, the default column delimiter for the **LOAD TABLE** statement is <tab> and the default row terminator is <newline>.
- For **FORMAT BCP**, the last column in a row must be terminated by the row terminator, not by the column delimiter. If the column delimiter is present before the row terminator, then the column delimiter is treated as a part of the data.
- Data for columns that are not the last column in the load specification must be delimited by the column delimiter only. If a row terminator is encountered before a column delimiter for a column that is not the last column, then the row terminator is treated as a part of the column data.
- Column delimiter can be specified via the **DELIMITED BY** clause. For **FORMAT BCP**, the delimiter must be less than or equal to 10 characters in length. An error is returned, if the delimiter length is more than 10.
- For **FORMAT BCP**, the load specification may contain only column names, **NULL**, and **ENCRYPTED**. An error is returned, if any other option is specified in the load specification.

  For example, these **LOAD TABLE** load specifications are valid:

```
LOAD TABLE x( c1, c2 null(blanks), c3 )
FROM 'bcp_file.bcp'
FORMAT BCP
...
```

```
LOAD TABLE x( c1 encrypted(bigint,'KEY-ONE','aes'), c2, c3 )
FROM 'bcp_file.bcp'
FORMAT BCP
...
```

DELIMITED BY—If you omit a column delimiter in the *column-spec* definition, the default column delimiter character is a comma. You can specify an alternative column delimiter by providing a single ASCII character or the hexadecimal character representation. The **DELIMITED BY** clause is:

```
... DELIMITED BY '\x09' ...
```

To use the newline character as a delimiter, you can specify either the special combination '\n' or its ASCII value '\x0a'. Although you can specify up to four characters in the column-spec *delimiter-string*, you can specify only a single character in the **DELIMITED BY** clause.

STRIP—The **STRIP** clause specifies whether unquoted values should have trailing blanks stripped off before they are inserted. The **LOAD TABLE** command accepts these **STRIP** keywords:

- STRIP OFF—Do not strip off trailing blanks.
- STRIP RTRIM—Strip trailing blanks.
- STRIP ON—Deprecated. Use **STRIP RTRIM**.

With **STRIP** turned on (the default), SAP Sybase IQ strips trailing blanks from values before inserting them. This is effective only for `VARCHAR` data. **STRIP OFF** preserves trailing blanks.

Trailing blanks are stripped only for unquoted strings. Quoted strings retain their trailing blanks. If you do not require blank sensitivity, you can use the **FILLER** option as an alternative to be more specific in the number of bytes to strip, instead of all the trailing spaces. **STRIP OFF** is more efficient for SAP Sybase IQ, and it adheres to the ANSI standard when dealing with trailing blanks. (`CHAR` data is always padded, so the **STRIP** option only affects `VARCHAR` data.)

The `STRIP` option applies only to variable-length non-binary data and does not apply to ASCII fixed-width inserts. For example, assume this schema:

```
CREATE TABLE t( c1 VARCHAR(3) );
LOAD TABLE t( c1 ',' ) ........ STRIP RTRIM    // trailing blanks
trimmed

LOAD TABLE t( c1 ',' ) ........ STRIP OFF     // trailing blanks not
trimmed

LOAD TABLE t( c1 ASCII(3) ) ... STRIP RTRIM   // trailing blanks not
trimmed
LOAD TABLE t( c1 ASCII(3) ) ... STRIP OFF      // trailing blanks
trimmed

LOAD TABLE t( c1 BINARY ) ..... STRIP RTRIM    // trailing blanks
trimmed
LOAD TABLE t( c1 BINARY ) ..... STRIP OFF      // trailing blanks
trimmed
```

Trailing blanks are always trimmed from binary data.

WITH CHECKPOINT—This option is useful only when loading SQL Anywhere tables in an SAP Sybase IQ database.

Use this clause to specify whether to perform a checkpoint. The default setting is OFF. If this clause is set to ON, a checkpoint is issued after successfully completing and logging the statement. If the server fails after a connection commits and before the next checkpoint, the data file used to load the table must be present for the recovery to complete successfully. However, if **WITH CHECKPOINT ON** is specified, and recovery is subsequently required, the data file need not be present at the time of recovery.

The data files are required, regardless of what is specified for this clause, if the database becomes corrupt and you need to use a backup and apply the current log file.

Warning! If you set the database option CONVERSION_ERROR to OFF, you may load bad data into your table without any error being reported. If you do not specify **WITH CHECKPOINT ON**, and the database needs to be recovered, the recovery may fail as CONVERSION_ERROR is ON (the default value) during recovery. It is recommended that you do not load tables when CONVERSION_ERROR is set to OFF and **WITH CHECKPOINT ON** is not specified.

See also *CONVERSION_ERROR Option [TSQL]*.

BYTE ORDER—Specifies the byte order during reads. This option applies to all binary input fields. If none are defined, this option is ignored. SAP Sybase IQ always reads binary data in the format native to the machine it is running on (default is **NATIVE**). You can also specify:

- **HIGH** when multibyte quantities have the high order byte first (for big endian platforms like Sun, IBM AIX, and HP).
- **LOW** when multibyte quantities have the low order byte first (for little endian platforms like Windows).

LIMIT—Specifies the maximum number of rows to insert into the table. The default is 0 for no limit. The maximum is 2^{31} - 1 (2147483647) rows.

NOTIFY—Specifies that you be notified with a message each time the specified number of rows is successfully inserted into the table. The default is every 100,000 rows. The value of this option overrides the value of the NOTIFY_MODULUS database option.

ON FILE ERROR—Specifies the action SAP Sybase IQ takes when an input file cannot be opened because it does not exist or you have incorrect permissions to read the file. You can specify one of the following:

- **ROLLBACK** aborts the entire transaction (the default).
- **FINISH** finishes the insertions already completed and ends the load operation.
- **CONTINUE** returns an error but only skips the file to continue the load operation.

Only one **ON FILE ERROR** clause is permitted.

PREVIEW—Displays the layout of input into the destination table including starting position, name, and data type of each column. SAP Sybase IQ displays this information at the start of the load process. If you are writing to a log file, this information is also included in the log.

ROW DELIMITED BY—Specifies a string up to 4 bytes in length that indicates the end of an input record. You can use this option only if all fields within the row are any of the following:

- Delimited with column terminators
- Data defined by the DATE or DATETIME *column-spec* options
- ASCII fixed length fields

You cannot use this option if any input fields contain binary data. With this option, a row terminator causes any missing fields to be set to NULL. All rows must have the same row

delimiters, and it must be distinct from all column delimiters. The row and field delimiter strings cannot be an initial subset of each other. For example, you cannot specify “*” as a field delimiter and “*#” as the row delimiter, but you could specify “#” as the field delimiter with that row delimiter.

If a row is missing its delimiters, SAP Sybase IQ returns an error and rolls back the entire load transaction. The only exception is the final record of a file where it rolls back that row and returns a warning message. On Windows, a row delimiter is usually indicated by the newline character followed by the carriage return character. You might need to specify this as the *delimiter-string* (see above for description) for either this option or **FILLER**.

SKIP—Defines the number of rows to skip at the beginning of the input tables for this load. The maximum number of rows to skip is 2^{31} - 1 (2147483647). The default is 0.

HEADER SKIP…HEADER DELIMITED BY—Specifies a number of lines at the beginning of the data file, including header rows, for **LOAD TABLE** to skip. All **LOAD TABLE** column specifications and other load options are ignored, until the specified number of rows is skipped.

- The number of lines to skip is greater than or equal to zero.
- Lines are determined by a 1 to 4 character delimiter string specified in the **HEADER DELIMITED BY** clause. The default **HEADER DELIMITED BY** string is the ‘\n’ character.
- The **HEADER DELIMITED BY** string has a maximum length of four characters. An error is returned, if the string length is greater than four or less than one.
- When a non-zero **HEADER SKIP** value is specified, all data inclusive of the **HEADER DELIMITED BY** delimiter is ignored, until the delimiter is encountered the number of times specified in the **HEADER SKIP** clause.
- All **LOAD TABLE** column specifications and other load options are ignored, until the specified number of rows has been skipped. After the specified number of rows has been skipped, the **LOAD TABLE** column specifications and other load options are applied to the remaining data.
- The "header" bytes are ignored only at the beginning of the data. When multiple files are specified in the **USING** clause, **HEADER SKIP** only ignores data starting from the first row of the first file, until it skips the specified number of header rows, even if those rows exist in subsequent files. **LOAD TABLE** does not look for headers once it starts parsing actual data.
- No error is reported, if **LOAD TABLE** processes all input data before skipping the number of rows specified by **HEADER SKIP**.

WORD SKIP—Allows the load to continue when it encounters data longer than the limit specified when the word index was created.

If a row is not loaded because a word exceeds the maximum permitted size, a warning is written to the `.iqmsg` file. WORD size violations can be optionally logged to the MESSAGE LOG file and rejected rows logged to the ROW LOG file specified in the **LOAD TABLE** statement.

- If the option is not specified, **LOAD TABLE** reports an error and rolls back on the first occurrence of a word that is longer than the specified limit.
- *number* specifies the number of times the "`Words exceeding the maximum permitted word length not supported`" error is ignored.
- 0 (zero) means there is no limit.

ON PARTIAL INPUT ROW—Specifies the action to take when a partial input row is encountered during a load. You can specify one of the following:

- **CONTINUE** issues a warning and continues the load operation. This is the default.
- **ROLLBACK** aborts the entire load operation and reports the error.

```
Partial input record skipped at EOF.
SQLSTATE: QDC32    SQLSTATE: -1000232L
```

IGNORE CONSTRAINT—Specifies whether to ignore CHECK, UNIQUE, NULL, DATA VALUE, and FOREIGN KEY integrity constraint violations that occur during a load and the maximum number of violations to ignore before initiating a rollback. Specifying each *constrainttype* has the following result:

- CHECK *limit*—If *limit* specifies zero, the number of CHECK constraint violations to ignore is infinite. If CHECK is not specified, the first occurrence of any CHECK constraint violation causes the **LOAD** statement to roll back. If *limit* is nonzero, then the *limit* +1 occurrence of a CHECK constraint violation causes the load to roll back.
- UNIQUE *limit*—If *limit* specifies zero, then the number of UNIQUE constraint violations to ignore is infinite. If *limit* is nonzero, then the *limit* +1 occurrence of a UNIQUE constraint violation causes the load to roll back.
- NULL *limit*—If *limit* specifies zero, then the number of NULL constraint violations to ignore is infinite. If *limit* is nonzero, then the *limit* +1 occurrence of a NULL constraint violation causes the load to roll back.
- FOREIGN KEY *limit*—If *limit* specifies zero, the number of FOREIGN KEY constraint violations to ignore is infinite. If *limit* is nonzero, then the *limit* +1 occurrence of a FOREIGN KEY constraint violation causes the load to roll back.
- DATA VALUE *limit*—If the database option CONVERSION_ERROR = ON, an error is reported and the statement rolls back. If *limit* specifies zero, then the number of DATA VALUE constraint violations (data type conversion errors) to ignore is infinite. If *limit* is nonzero, then the *limit* +1 occurrence of a DATA VALUE constraint violation causes the load to roll back.
- ALL *limit*—If the database option `CONVERSION_ERROR = ON`, an error is reported and the statement rolls back. If *limit* specifies zero, then the cumulative total of all integrity constraint violations to ignore is infinite. If *limit* is nonzero, then load rolls back when the cumulative total of all ignored UNIQUE, NULL, DATA VALUE, and FOREIGN KEY integrity constraint violations exceeds the value of *limit*. For example, you specify this **IGNORE CONSTRAINT** option:

  ```
  IGNORE CONSTRAINT NULL 50, UNIQUE 100, ALL 200
  ```

The total number of integrity constraint violations cannot exceed 200, whereas the total number of NULL and UNIQUE constraint violations cannot exceed 50 and 100, respectively. Whenever any of these limits is exceeded, the **LOAD TABLE** statement rolls back.

Note: A single row can have more than one integrity constraint violation. Every occurrence of an integrity constraint violation counts towards the limit of that type of violation.

Tip: Set the **IGNORE CONSTRAINT** option limit to a nonzero value if you are logging the ignored integrity constraint violations. Logging an excessive number of violations affects the performance of the load.

If CHECK, UNIQUE, NULL, or FOREIGN KEY is not specified in the **IGNORE CONSTRAINT** clause, then the load rolls back on the first occurrence of each of these types of integrity constraint violation.

If DATA VALUE is not specified in the **IGNORE CONSTRAINT** clause, then the load rolls back on the first occurrence of this type of integrity constraint violation, unless the database option `CONVERSION_ERROR = OFF`. If `CONVERSION_ERROR = OFF`, a warning is reported for any DATA VALUE constraint violation and the load continues.

When the load completes, an informational message regarding integrity constraint violations is logged in the `.iqmsg` file. This message contains the number of integrity constraint violations that occurred during the load and the number of rows that were skipped.

MESSAGE LOG—Specifies the names of files in which to log information about integrity constraint violations and the types of violations to log. Timestamps indicating the start and completion of the load are logged in both the **MESSAGE LOG** and the **ROW LOG** files. Both **MESSAGE LOG** and **ROW LOG** must be specified, or no information about integrity violations is logged.

- If the **ONLY LOG** clause is not specified, no information on integrity constraint violations is logged. Only the timestamps indicating the start and completion of the load are logged.
- Information is logged on all integrity constraint-type violations specified in the **ONLY LOG** clause or for all word index-length violations if the keyword WORD is specified.
- If constraint violations are being logged, every occurrence of an integrity constraint violation generates exactly one row of information in the **MESSAGE LOG** file.

  The number of rows (errors reported) in the **MESSAGE LOG** file can exceed the **IGNORE CONSTRAINT** option limit, because the load is performed by multiple threads running in parallel. More than one thread might report that the number of constraint violations has exceeded the specified limit.
- If constraint violations are being logged, exactly one row of information is logged in the **ROW LOG** file for a given row, regardless of the number of integrity constraint violations that occur on that row.

The number of distinct errors in the **MESSAGE LOG** file might not exactly match the number of rows in the **ROW LOG** file. The difference in the number of rows is due to the parallel processing of the load described above for the **MESSAGE LOG**.

- The **MESSAGE LOG** and **ROW LOG** files cannot be raw partitions or named pipes.
- If the **MESSAGE LOG** or **ROW LOG** file already exists, new information is appended to the file.
- Specifying an invalid file name for the **MESSAGE LOG** or **ROW LOG** file generates an error.
- Specifying the same file name for the **MESSAGE LOG** and **ROW LOG** files generates an error.

Various combinations of the **IGNORE CONSTRAINT** and **MESSAGE LOG** options result in different logging actions.

Table 5. LOAD TABLE Logging Actions

IGNORE CONSTRAINT specified?	MESSAGE LOG specified?	Action
yes	yes	All ignored integrity constraint violations are logged, including the user specified limit, before the rollback.
no	yes	The first integrity constraint violation is logged before the rollback.
yes	no	Nothing is logged.
no	no	Nothing is logged. The first integrity constraint violation causes a rollback.

Tip: Set the **IGNORE CONSTRAINT** option limit to a nonzero value, if you are logging the ignored integrity constraint violations. If a single row has more than one integrity constraint violation, a row for each violation is written to the **MESSAGE LOG** file. Logging an excessive number of violations affects the performance of the load.

LOG DELIMITED BY—Specifies the separator between data values in the **ROW LOG** file. The default separator is a comma.

SAP Sybase IQ no longer returns an error message when **FORMAT BCP** is specified as a **LOAD TABLE** clause. In addition, these conditions are verified and proper error messages are returned:

- If the specified load format is not **ASCII**, **BINARY**, or **BCP**, SAP Sybase IQ returns the message "`Only ASCII, BCP and BINARY are supported LOAD formats.`"

- If the **LOAD TABLE** column specification contains anything other than column name, **NULL**, or **ENCRYPTED**, then SAP Sybase IQ returns the error message "`Invalid load specification for LOAD ... FORMAT BCP.`"
- If the column delimiter or row terminator size for the **FORMAT BCP** load is greater than 10 characters, then SAP Sybase IQ returns the message "`Delimiter '%2' must be 1 to %3 characters in length.`" (where %3 equals 10).

  Messages corresponding to error or warning conditions which can occur for **FORMAT BCP** as well as **FORMAT ASCII** are the same for both formats.
- If the load default value specified is **AUTOINCREMENT**, **IDENTITY**, or **GLOBAL AUTOINCREMENT**, SAP Sybase IQ returns the error "`Default value %2 cannot be used as a LOAD default value. %1`"
- If the **LOAD TABLE** specification does not contain any columns that need to be loaded from the file specified, SAP Sybase IQ returns the error "`The LOAD statement must contain at least one column to be loaded from input file.`" and the **LOAD TABLE** statement rolls back.
- If a load exceeds the limit on the maximum number of terms for a text document with **TEXT** indexes, SAP Sybase IQ returns the error "`Text document exceeds maximum number of terms. Support up to 4294967295 terms per document.`"

Standards

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Not applicable.

Permissions

The permissions required to execute a **LOAD TABLE** statement depend on the database server **-gl** command line option, as follows:

- **-gl ALL –** You must be the owner of the table, have ALTER or LOAD permission on the table, or have the ALTER ANY TABLE, LOAD ANY TALBE, or ALTER ANY OBJECT system privilege.
- **-gl DBA –** You must have the ALTER ANY TABLE, LOAD ANY TABLE, or ALTER ANY OBJECT system privilege.
- **-gl NONE –** Execution of the **LOAD TABLE** statement is not permitted.

For more information on the **-gl** command line option, please refer *Utility Guide > start_iq Database Server Startup Utility > start_iq Server Options*.

LOAD TABLE also requires a write lock on the table.

LOCK TABLE Statement

Prevents other concurrent transactions from accessing or modifying a table within the specified time.

Syntax

LOCK TABLE *table-list* [**WITH HOLD**] **IN** { **SHARE** | **WRITE | EXCLUSIVE** } **MODE** [**WAIT** *time*]

Parameters

- **table-list –** [*owner.*] *table-name* [, [*owner.*] *table-name*, …]

Examples

- **Example 1 –** Obtain a WRITE lock on the `Customers` and `Employees` tables, if available within 5 minutes and 3 seconds:

```
LOCK TABLE Customers, Employees IN WRITE MODE WAIT
'00:05:03'
```

- **Example 2 –** Wait indefinitely until the WRITE lock on the `Customers` and `Employees` tables is available, or an interrupt occurs:

```
LOCK TABLE Customers, Employees IN WRITE MODE WAIT
```

Usage

table-name—The table must be a base table, not a view. WRITE mode is only valid for IQ base tables. **LOCK TABLE** either locks all tables in the table list, or none. The table must not be enabled for row-level versioning (RLV). If obtaining a lock for a SQL Anywhere table, or when obtaining SHARE or EXCLUSIVE locks, you may only specify a single table. Standard SAP Sybase IQ object qualification rules are used to parse *table-name*.

WITH HOLD—If this clause is specified, the lock is held until the end of the connection. If the clause is not specified, the lock is released when the current transaction is committed or rolled back. Using the WITH HOLD clause in the same statement with WRITE MODE is unsupported and returns the error `SQLCODE=-131, ODBC 3 State="42000"`.

SHARE—Prevents other transactions from modifying the table, but allows them read access. In this mode, you can change data in the table as long as no other transaction has locked the row being modified, either indirectly, or explicitly by using **LOCK TABLE**.

WRITE—Prevents other transactions from modifying a list of tables. Unconditionally commits the connections outermost transaction. The transaction's snapshot version is established not by the **LOCK TABLE IN WRITE MODE** statement, but by the execution of the next command processed by SAP Sybase IQ.

WRITE mode locks are released when the transaction commits or rolls back, or when the connection disconnects.

EXCLUSIVE—Prevents other transactions from accessing the table. In this mode, no other transaction can execute queries, updates of any kind, or any other action against the table.

LOCK TABLE statements run on tables in the IQ main store on the coordinator do not affect access to those tables from connections on secondary servers. For example:

On a coordinator connection, issue the command:

```
LOCK TABLE coord1 WITH HOLD IN EXCLUSIVE MODE
```

sp_iqlocks on the coordinator confirms that the table `coord1` has an exclusive (E) lock.

The result of **sp_iqlocks** run on a connection on a secondary server does not show the exclusive lock on table `coord1`. The user on this connection can see updates to table `coord1` on the coordinator.

Other connections on the coordinator can see the exclusive lock on `coord1` and attempting to select from table `coord1` from another connection on the coordinator returns `User DBA has the row in coord1 locked.`

WAIT *time*— Wait options specify maximum blocking time for all lock types. This option is mandatory when lock mode is WRITE. When a time argument is given, the server locks the specified tables only if available within the specified time. The time argument can be specified in the format *hh:nn:ss:sss*. If a date part is specified, the server ignores it and converts the argument into a timestamp. When no time argument is given, the server waits indefinitely until a WRITE lock is available or an interrupt occurs.

LOCK TABLE on views is unsupported. Attempting to lock a view acquires a shared schema lock regardless of the mode specified in the command. A shared schema lock prevents other transactions from modifying the table schema.

The Transact-SQL (T-SQL) stored procedure dialect does not support **LOCK TABLE**. For example, this statement returns `Syntax error near LOCK`:

```
CREATE PROCEDURE tproc()
AS
BEGIN
COMMIT;
LOCK TABLE t1 IN SHARE MODE
INSERT INTO t1 VALUES(30)
END
```

The Watcom-SQL stored procedure dialect supports **LOCK TABLE**. The default command delimiter is a semicolon (;). For example:

```
CREATE PROCEDURE tproc()
AS
BEGIN
COMMIT;
LOCK TABLE t1 IN SHARE MODE
```

```
INSERT INTO t1 VALUES(30)
END
```

Standards

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- SAP Sybase—Supported in Adaptive Server Enterprise. The **WITH HOLD** clause is not supported in Adaptive Server Enterprise. Adaptive Server Enterprise provides a **WAIT** clause that is not supported in SQL Anywhere.

Permissions

To lock a table in SHARE mode, SELECT privileges are required.

To lock a table in EXCLUSIVE mode, you must be the table owner or have any of the following system privileges:

- ALTER ANY OBJECT
- INSERT ANY TABLE
- UPDATE ANY TABLE
- DELETE ANY TABLE
- ALTER ANY TABLE
- LOAD ANY TABLE
- TRUNCATE ANY TABLE

.

TRUNCATE Statement

Deletes all rows from a table or materialized view without deleting the table definition.

Syntax

Syntax 1

```
TRUNCATE
    TABLE [ owner.]table-name
    | MATERIALIZED VIEW owner.]materialized-view-name
```

Syntax 2

```
TRUNCATE TABLE   [ owner .]table
 [ PARTITION partition-name
    | SUBPARTITION subpartition-name ]
```

Examples

- **Example 1 –** Delete all rows from the `Sale` table:

```
TRUNCATE TABLE Sale
```

Usage

TRUNCATE is equivalent to a DELETE statement without a WHERE clause, except that each individual row deletion is not entered into the transaction log. After a TRUNCATE TABLE statement, the table structure and all of the indexes continue to exist until you issue a DROP TABLE statement. The column definitions and constraints remain intact, and permissions remain in effect.

The TRUNCATE statement is entered into the transaction log as a single statement, like data definition statements. Each deleted row is not entered into the transaction log.

The PARTITION clause specifies which partition to truncate, and does not affect data in other partitions. Use SUBPARTITION to truncate tables partitioned by a composite partitioning scheme.

Note: Specifying an RLV-enabled table in the PARTITION or SUBPARTITION clause results in an error.

Standards

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Supported by Adaptive Server Enterprise.

Permissions

Requires one of:

- TRUNCATE ANY TABLE system privilege.
- ALTER ANY TABLE system privilege.
- ALTER ANY OBJECT system privilege.
- TRUNCATE privilege on the table.
- You own the object.

For both temporary and base tables, you can execute **TRUNCATE TABLE** while other users have read access to the table. This behavior differs from SQL Anywhere, which requires exclusive access to truncate a base table. SAP Sybase IQ table versioning ensures that **TRUNCATE TABLE** can occur while other users have read access; however, the version of the table these users see depends on when the read and write transactions commit.

UPDATE Statement

Modifies existing rows of a single table, or a view that contains only one table.

Syntax

```
UPDATE table
... SET [column-name  = expression, …
```

```
... [ FROM table-expression, ]
... [ WHERE search-condition ]
... [ ORDER BY expression [ ASC | DESC ] , …]
```

```
FROM table-expression
```

Parameters

- **table-expression –** *table-spec* | *table-expression join-type table-spec* [**ON** *condition*] | *table-expression*, …

Examples

- **Example 1 –** Transfer employee Philip Chin (employee 129) from the sales department to the marketing department:

```
UPDATE Employees
SET DepartmentID = 400
WHERE EmployeeID = 129;
```

- **Example 2 –** The Marketing Department (400) increases bonuses from 4% to 6% of each employee's base salary:

```
UPDATE Employees
SET bonus = base * 6/100
WHERE DepartmentID =400;
```

- **Example 3 –** Each employee gets a pay increase with the department bonus:

```
UPDATE Employees
SET emp.Salary = emp.Salary + dept.bonus
FROM Employees emp, Departments dept
WHERE emp.DepartmentID = dept.DepartmentID;
```

- **Example 4 –** Another way to give each employee a pay increase with the department bonus:

```
UPDATE Employees
SET emp.salary = emp.salary + dept.bonus
FROM Employees emp JOIN Departments dept
ON emp.DepartmentID = dept.DepartmentID;
```

Usage

The table on which you use **UPDATE** may be a base table or a temporary table.

Defaults on updates are honored for current user, user and current timestamp, and timestamp only.

Each named column is set to the value of the expression on the right-hand side of the equal sign. Even *column-name* can be used in the expression—the old value is used.

The **FROM** clause can contain multiple tables with join conditions and returns all the columns from all the tables specified and filtered by the join condition and/or **WHERE** condition.

Using the wrong join condition in a **FROM** clause causes unpredictable results. If the **FROM** clause specifies a one-to-many join and the SET clause references a cell from the "many" side of the join, the cell is updated from the first value selected. In other words, if the join condition causes multiple rows of the table to be updated per row ID, the first row returned becomes the update result. For example:

```
UPDATE T1
SET T1.c2 = T2.c2
FROM T1 JOIN TO T2
ON T1.c1 = T2.c1
```

If table `T2` has more than one row per `T2.c1`, results might be as follows:

T2.c1	T2.c2	T2.c3
1	4	3
1	8	1
1	6	4
1	5	2

With no **ORDER BY** clause, `T1.c2` may be 4, 6, 8, or 9.

- With `ORDER BY T2.c3`, `T1.c2` is updated to 8.
- With `ORDER BY T2.c3 DESC`, `T1.c2` is updated to 6.

SAP Sybase IQ rejects any **UPDATE** statement in which the table being updated is on the null-supplying side of an outer join. In other words:

- In a left outer join, the table on the left side of the join cannot be missing any rows on joined columns.
- In a right outer join, the table on the right side of the join cannot be missing any rows on joined columns.
- In a full outer join, neither table can be missing any rows on joined columns.

For example, in this statement, table `T1` is on the left side of a left outer join, and thus cannot contain be missing any rows:

```
UPDATE T1
SET T1.c2 = T2.c4
FROM T1 LEFT OUTER JOIN T2
ON T1.rowid = T2.rowid
```

Normally, the order in which rows are updated does not matter. However, in conjunction with the **NUMBER(*)** function, an ordering can be useful to get increasing numbers added to the rows in some specified order. If you are not using the **NUMBER(*)** function, avoid using the **ORDER BY** clause, because the **UPDATE** statement performs better without it.

In an **UPDATE** statement, if the **NUMBER(*)** function is used in the **SET** clause and the **FROM** clause specifies a one-to-many join, **NUMBER(*)** generates unique numbers that increase, but do not increment sequentially due to row elimination.

You can use the **ORDER BY** clause to control the result from an **UPDATE** when the **FROM** clause contains multiple joined tables.

SAP Sybase IQ ignores the **ORDER BY** clause in searched **UPDATE** and returns a message that the syntax is not valid ANSI syntax.

If no **WHERE** clause is specified, every row is updated. If you specify a **WHERE** clause, SAP Sybase IQ updates only rows satisfying the search condition.

The left side of each **SET** clause must be a column in a base table.

Views can be updated provided the **SELECT** statement defining the view does not contain a **GROUP BY** clause or an aggregate function, or involve a **UNION** operation. The view should contain only one table.

Character strings inserted into tables are always stored in the case they are entered, regardless of whether the database is case-sensitive or not. Thus a character data type column updated with the string 'Value' is always held in the database with an uppercase V and the remainder of the letters lowercase. **SELECT** statements return the string as 'Value.' If the database is not case-sensitive, however, all comparisons make 'Value' the same as 'value,' 'VALUE,' and so on. The IQ server may return results in any combination of lowercase and uppercase, so you cannot expect case-sensitive results in a database that is case-insensitive (**CASE IGNORE**). Further, if a single-column primary key already contains an entry 'Value,' an **INSERT** of 'value' is rejected, as it would make the primary key not unique.

If the update violates any check constraints, the whole statement is rolled back.

SAP Sybase IQ supports scalar subqueries within the **SET** clause, for example:

```
UPDATE r
SET r.o= (SELECT MAX(t.o)
FROM t ... WHERE t.y = r.y),
r.s= (SELECT SUM(x.s)
FROM x ...
WHERE x.x = r.x)
WHERE r.a = 10
```

SAP Sybase IQ supports DEFAULT column values in **UPDATE** statements. If a column has a DEFAULT value, this DEFAULT value is used as the value of the column in any **UPDATE** statement that does not explicitly modify the value for the column.

See *CREATE TABLE Statement* for details about updating IDENTITY/AUTOINCREMENT columns, which are another type of DEFAULT column.

Standards

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—With these exceptions, syntax of the IQ **UPDATE** statement is generally compatible with the Adaptive Server Enterprise **UPDATE** statement Syntax 1: SAP Sybase IQ supports multiple tables with join conditions in the **FROM** clause.

Updates of remote tables are limited to SAP Sybase IQ syntax supported by CIS.

Permissions

Requires UPDATE privilege on the columns being modified.

See also

- *CREATE TABLE Statement* on page 164

Views

Use the system views to view the contents of the system tables.

SYSIQDBSPACE System View

Presents group information from `ISYSIQDBSPACE` in a readable format.

Column name	Column type	Description
dbspace_id	small int	Each dbspace in a database is assigned a unique number (dbspace ID)
last_modified	timestamp	Time at which the dbspace's read-write status was last modified
segment_type	char(8)	Segment type: Main, Temp or Msg
read_write	char(1)	'T' – read writable; 'F' – read only
online	char(1)	'T' – online; 'F' – offline
create_txn_id	unsigned bigint	Transaction ID that create the dbspace
alter_txn_id	unsigned bigint	Transaction ID that last modified read_write status
striping_on	char(1)	'T' – disk striping on; 'F' – disk striping off
stripe_size_kb	unsigned int	Number of kilobytes written to each file of the dbspace before the disk striping algorithm moves to the next dbfile
is_rlv_store	char(1)	'T' – dbspace is a RLV store dbspace; 'F' – dbspace is a MIN, SHARED_TEMP, or TEMPORARY store dbspace.

Constraints on underlying system table:

Primary key (dbspace_id)

Foreign key (dbspace_id) references SYS.ISYSDBSPACE(dbspace_id)

SYSIQRLVMERGEHISTORY System View

A log entry is added for each row-level versioning (RLV) enabled-table each time a merge between the RLV store and the IQ main store begins. Log entries are updated when the merge is complete.

Column Name	Column Type	Description
merge_id	unsigned bigint	Unique log entry identifier
table_id	unsigned int	Foreign key to the sys.systable system table
start_time	timestamp	Time the merge started
end_time	timestamp	Time the merge ended
status	char (9)	STARTED | COMPLETED | FAILED
return_code	tinyint	SQL code of the merge once completed
merge_type	char (9)	The cause of the merge trigger: AUTOMATIC | DML | DDL | SHUTDOWN | USER
merge_mode	char (12)	BLOCKING | NON-BLOCKING
merge_detail	varchar (255)	Additional information, if provided, such as error information
rows_inserted	unsigned bigint	Number of rows that were inserted as a result of the merge
rows_updated	unsigned bigint	Number of rows that were updated as a result of the merge
rows_deleted	unsigned bigint	Number of rows that were deleted as a result of the merge
rows_forwarded	unsigned bigint	Number of rows that were uncommitted at the time of the merge

SYSIQRVLOG System View

Presents group information from `ISYSIQRVLOG` in a readable format. Each row in the SYSIQRVLOG view corresponds to a log for a RLV-enabled table . The row with table_id 0 represents the server-wide commit log.

Column Name	Column Type	Description
stream_id	unsigned int	The log stream identifier.
table_id	unsigned int	Indicates the table the log stream belongs to. NULL indicates a commit log stream.
partition_low	unsigned int	Corresponds to the partition map in use when that log was last active.
partition_high	int	Corresponds to the partition map in use when that log was last active.
identity_location	unsigned bigint	Location of the log stream identity block.

SYSIQTAB System View

Presents group information from `ISYSIQTAB` in a readable format. Each row in the SYSIQTAB view describes an IQ table.

```
ALTER VIEW "SYS"."SYSIQTAB"
as select * from SYS.ISYSIQTAB
```

Note: This view replaces the deprecated system view SYSIQTABLE.

Column name	Column type	Description
table_id	unsigned int	Each table is assigned a unique number (the table number) that is the primary key.
block_map	hs_blockmapidentity	For internal use.
block_map_size	unsigned int	For internal use.
vdo	hs_vdoidentity	For internal use.
vdoid_size	unsigned int	For internal use.
info_location	hs_vdorecid	Not used. Always zero.

Column name	Column type	Description
info_recid_size	unsigned int	Not used. Always zero.
info_location_size	unsigned int	Not used. Always zero.
commit_txn_id	unsigned bigint	For internal use.
txn_id	unsigned bigint	For internal use.
update_time	timestamp	Last date and time the IQ table was modified.
is_rlv	char(1)	'T' – RLV storage is enabled on the table; 'F' – RLV storage is not enabled on the table.

Constraints on underlying system table:

Primary key (table_id)

Index

A

B

C

D

E

F

G

H

I

J

L

R

S

T

U

V

W