

What is NuSphere's Gemini?

By NuSphere Corporation – June 2001

TABLE OF CONTENTS

So what is a Gemini table?

The History of Gemini

Gemini Features and Functionality

The Concepts and Definitions of the Gemini Programming Model

When to Use Gemini Tables

Gemini

About NuSphere

The NuSphere's Gemini database brings together the MySQL server, the #1 open source database, and the power of Gemini, NuSphere's table type integrated into the MySQL server. Gemini tables provide row-level locking, robust transaction support, and reliable crash recovery.

Gemini, as the table handler, is a key component of the database.

So what is Gemini?

Gemini is a "transaction-safe" table handler for the MySQL database. It is ideal for developers who are building web applications that require a database to handle the heavy multi-user updates typically found in transaction processing, yet also require excellent performance for read-intensive operations.

When a Gemini database developer sets the table type to use Gemini, `CREATE TABLE [name] TYPE = GEMINI;` they will be able to take advantage of several capabilities:

Row-level locking: delivers maximum transaction concurrency in a heavy update environment. The transaction, locking, and recovery mechanisms are tightly integrated to eliminate unnecessary administration overhead.

Full ACID (Atomic, Consistent, Isolation, and Durable) transactions: these properties work with a programming model that includes support for statement atomicity and all four standard isolation levels – *Read Uncommitted*, *Read Committed*, *Repeatable Read*, and *Serializable* – all defined in the SQL standard.

Crash recovery mechanism: in the event of a system failure, the Gemini table handler will automatically return the database to the *known and consistent* state it was when the system failure or restart occurred.

The History of Gemini

The Gemini table type was derived from a successful commercial database and uses the storage kernel technology tightly integrated with the MySQL database server. The maturity of Gemini tables allows Gemini to deliver a reliable solution for those looking to build cost-effective, transaction-based web applications. Today, millions of people in production environments worldwide use Gemini technology.

Gemini Features and Functionality

NuSphere's Gemini table type in the MySQL database delivers speed, power, and performance. The Gemini table handler's key features include:

- Support for optimization statistics used by the MySQL optimizer, including: table cardinality, index range estimates, and multi-component selectivity to insure optimal query performance.
- The exact cardinality information for each table is maintained so the `SELECT COUNT (*) FROM table-name` always returns an immediate answer.

- Support for index-only queries. This means that when index data is sufficient enough to resolve a query, no record data is read – that is for non-character data types.
- Gemini uses block-based I/O (input/output) for better performance. (There is no performance penalty for using `VARCHAR` fields, and the maximum record size is currently 32K.)
- Row-level locking. Rather than locking at the table level, which may result in data not being accessible, Gemini locking is done at record or row level unless a table lock has been explicitly requested. Consequently, when a row is inserted, or updated, in a table, other rows can be manipulated without waiting for the inserted row to be committed.
- Durable transactions are backed by a crash recovery mechanism that returns the database to the known and consistent state prior to the unexpected system failure or system restart. Gemini's *Reliable Master Replication* feature ensures that the master database will survive a system failure and recover all committed transactions processed prior to an unexpected failure or restart.
- Support for all isolation levels and statement atomicity as defined in the SQL standard.

The Concepts and Definitions of the Gemini Programming Model

To fully understand the Gemini technology, it will help to understand the concepts and programming model that make Gemini critical to your development projects.

Below are descriptions and definitions of the major features of Gemini tables, including:

- ACID Transactions
- Transaction COMMIT and ROLLBACK
- Statement Atomicity

- Crash Recovery
- Isolation Levels
- Row-Level Locking

1. ACID Transactions

In the context of database transactions, ACID is an acronym for **A**tomic, **C**onsistent, **I**solation, and **D**urable. Transactions provide a simple model of success or failure. A transaction either commits (i.e. all its actions happen), or it aborts (i.e. all its actions are undone). This all-or-nothing quality makes for a simple programming model. The attributes of an ACID transaction are:

Atomicity

A transaction allows for the grouping of one or more changes to tables and rows in the database to form an atomic or indivisible operation. That is, either all of the changes occur or none of them do. If for any reason the transaction cannot be completed, everything this transaction changed can be restored to the state it was in prior to the start of the transaction via a rollback operation.

Consistency

Transactions always operate on a consistent view of the data and when they end always leave the data in a consistent state. Data may be said to be consistent as long as it conforms to a set of invariants, such as no two rows in the customer table have the same customer id and all orders have an associated customer row. While a transaction executes these invariants may be violated, but no other transaction will be allowed to see these inconsistencies, and all such inconsistencies will have been eliminated by the time the transaction ends.

- Isolation** To a given transaction, it should appear as though it is running all by itself on the database. The effects of concurrently running transactions are invisible to this transaction, and the effects of this transaction are invisible to others until the transaction is committed.
- Durability** Once a transaction is committed, its effects are guaranteed to persist even in the event of subsequent system failures. Until the transaction commits, not only are any changes made by that transaction not durable, but are guaranteed not to persist in the face of a system failure, as crash recovery will rollback their effects.

The simplicity of ACID transactions is especially important in a distributed database environment where the transactions are being made simultaneously.

2. Transaction COMMIT and ROLLBACK

As stated above, a transaction is a series of changes being made independently or simultaneously to data stored within a table. Unless otherwise directed, the MySQL server considers each statement a transaction in itself. Multiple updates can be accomplished by placing them in a single statement, however they are limited to a single table. Business-critical applications tend to require a more robust use of transaction concepts. Take for example, a system that processes an order:

A row may be inserted in an *Order* table, additional rows may be added to an *Order-Line* table, updates may be made to *Inventory* tables, etc. It is important that if the order completes (or commits), the necessary changes are made to ALL the tables affected by the change; likewise if the order fails, none of the changes to the tables should occur.

To facilitate this requirement, the MySQL server supports syntax to start a transaction called `BEGIN WORK`. All statements that occur after the `BEGIN WORK` statement are grouped into a single transaction. The end of a transaction occurs when a `COMMIT` (success) or `ROLLBACK` (failure) statement is encountered. Once the `COMMIT` or `ROLLBACK` statement is encountered, the system will return back to the state it was before the `BEGIN WORK` statement was made. This behavior causes each transaction to be treated as a single statement when in fact database developers might want multiple statements to be part of the same transaction.

To permanently turn off the behavior where every statement is a transaction, the MySQL server added a variable called `AUTOCOMMIT`. The `AUTOCOMMIT` variable can have two values, 1 and 0. The value, or mode, where every statement is a transaction is when `AUTOCOMMIT` is set to 1 (`AUTOCOMMIT=1`). When `AUTOCOMMIT` is set to 0 (`AUTOCOMMIT=0`), every statement becomes part of the same transaction until the transaction ends by either a `COMMIT` or `ROLLBACK`. Once a transaction completes, a new transaction is immediately started and the process repeats thus giving you faster and more reliable transaction support.

So what would this look like? Below is an example of the SQL statements you might find in a typical programming model:

```
BEGIN WORK;  
  INSERT INTO order VALUES ...;  
  INSERT INTO order-lines VALUES ...;  
  INSERT INTO order-lines VALUES ...;  
  INSERT INTO order-lines VALUES ...;  
  UPDATE inventory WHERE ...;  
COMMIT;
```

The example shows how to use the `BEGIN WORK` statement to start a transaction. Now, if the variable `AUTOCOMMIT` is set to 0 (`AUTOCOMMIT=0`), then the next transaction would have

already begun, but in this case, the `BEGIN WORK` commits the current transaction and then starts a new one.

3. Statement Atomicity

It is critical that data retrieved and data stored be atomic. If it isn't, DBAs and other database users will spend countless hours manually trying to sync up data and recover data that may have been lost when a transaction failed. As mentioned above, when running with `AUTO COMMIT` set to 1 (`AUTO COMMIT=1`), each statement executes as a single transaction. When a statement has an error, then all changes made by the statement must be undone. Transactions support this behavior. Non-transaction safe table handlers would have a partial statement update where some of the changes from the statement would be contained in the database and other changes from the statement would not.

Gemini tables are "transaction-safe" tables. And we recommend that if Gemini tables are selected, that in order to maintain *statement atomicity* all tables used by an application be Gemini tables. If non-Gemini tables are used within an application, then ACID transactions and *statement atomicity* cannot be guaranteed. While there are clearly cases where mixing tables is appropriate, it should be done with careful consideration of the impact on transaction consistency and data recovery.

4. Crash Recovery

Since a transaction must be completed (or committed) in its entirety, transactions are the basis for database recovery. And recovery is what supports the Durability attribute of the ACID transaction. Gemini uses a separate file called the *Recovery Log* located in the `$DATADIR` directory. This file, called `gemini.rl`, maintains the integrity of all the data stored within Gemini tables. In addition, this file is used to rollback transactions in support of the roll back statement.

In the event of a system failure, the next time the MySQL server is started, Gemini will automatically go through its crash recovery process. The result of crash recovery is that all the Gemini tables will contain the latest “committed” changes, but any transactions that were open at the time of the crash will have been rolled back to the state they were in prior to the system failure. Since the result of a transaction may affect multiple tables, *replication* becomes an important part of crash recovery. Gemini has a *Reliable Master Replication* process whereby the master database will survive a system failure and recover all committed transactions processed prior to the unexpected problem or restart.

5. Isolation Levels

Simply stated isolation means that each transaction runs as though there are no concurrent transactions. The Isolation level refers to the locks, shared or exclusive, that the SET command applies to a Gemini table or row. The four Isolation levels supported by Gemini, include:

- READ UNCOMMITTED
- READ COMMITTED
- REPEATABLE READ
- SERIALIZABLE

Gemini delivers four isolation levels, three of which apply only to *shared* locks obtained by select statements, excluding select FOR UPDATE. Statements that get *exclusive* locks always retain those locks until a transaction COMMIT or ROLLBACK is

encountered. By default, Gemini tables operate at the READ COMMITTED level but you can override the default using a simple command. Example:

```
SET SESSION TRANSACTION ISOLATION LEVEL READ UNCOMMITTED
```

To get a better understanding of how isolation levels affect transactions, a description of what behavior an isolation level will have on your database follows:

READ UNCOMMITTED Does not obtain any locks when reading rows. If a row is locked by another process/transaction that has a stricter isolation level, i.e. READ COMMITTED, the READ UNCOMMITTED query will not wait until the locks are released before reading the row. More than likely, you will get an error if you attempt to update a row while running at this isolation level.

READ COMMITTED Locks the requested rows long enough to copy the row from the database block to the client row buffer. If a READ COMMITTED query finds that a row is locked exclusively by another process/transaction, it will wait until either the row has been released or the lock timeout value has expired.

REPEATABLE READ Locks all the rows needed to satisfy the query. Locks are held until the transaction ends (COMMIT or ROLLBACK). If a REPEATABLE READ query finds that a row is locked exclusively by another process/transaction, it will wait until either the row has been released or the lock timeout value has expired.

SERIALIZABLE

Locks the table that contains the rows needed to satisfy the query. This lock is held until the transaction ends (COMMIT or ROLLBACK). If a SERIALIZABLE query finds that a row is exclusively locked by another process, it will wait until either the row has been released or the lock timeout value has expired.

For developers, the statements that get exclusive locks are `INSERT`, `UPDATE`, `DELETE` and `SELECT [content] FOR UPDATE`. Select statements without the "FOR UPDATE" qualifier get shared locks. A shared lock will allow other not "FOR UPDATE" select statements to read the same rows, but block anyone trying to update the row from accessing it. Rows or tables with exclusive locks block access to the row from other processes/transactions. Access to the row will be released once the process/transaction has been completed. Database experts suggest that the higher the Isolation level (SERIALIZABLE being the highest), the more likely it is to have concurrent locks and therefore lock conflicts. In such cases, they recommend that you adjust the `-O gemini_lock_table_size` accordingly.

6. Row-Level Locking

The MySQL server's MyISAM table handler runs at a SERIALIZABLE isolation level. When the MySQL server is coupled with the Gemini table type, it will now allow for concurrent transactions to a table. Gemini uses row-level locking so that multiple requests can be made on the same table. This means if one person/session needs to *update* a row, only that row/record is locked, allowing other users to perform transactions on other rows/records. NuSphere's Gemini is ideal for web applications that require heavy multi-user updates and transaction processing.

You can maintain different tables within the same application that allow for row-level locking or table-level locking. Database experts recommend that in order to avoid lock table overflow, SQL statements that require applying locks to a large number of rows be run at the SERIALIZABLE isolation level or should be covered by a lock table statement. Just make sure to pre-allocate the appropriate memory for the lock table.

When to Use Gemini Tables

By now, you probably understand the difference between Gemini and the MySQL server's MyISAM table type. Here are some general guidelines about when to employ Gemini “transaction-safe” tables and when to use other “non-transaction” safe tables.

Data Access Trend	Example	Table Type	Reason
Read only	Inventory item descriptions	MyISAM	Frequent access with infrequent updates. Less overhead and faster.
Critical data	Inventory info	Gemini	Frequent access and updates. Demands crash recovery.
High concurrency	Orders for a customer	Gemini	Critical data that relies heavily on ACID transactions and row-level locking.
Heavy update	Customer account info	Gemini	Critical data that demands frequent updates, row-level locking and crash recovery.

The Gemini Table Type

With Gemini, your applications get the performance, scalability and reliability you need to grow your business. With an API that matches that of the MySQL server table handler, Gemini appli-

cations can accommodate any database schema, record format, index key format and server architecture.

About NuSphere Corporation

NuSphere, a subsidiary of Progress Software Corporation (NASDAQ: PRGS), delivers the first Internet Application Platform (IAP) based on open source components, providing an integrated foundation that allows companies to deploy reliable, cost-effective, enterprise-class applications across Windows, UNIX and Linux environments. NuSphere® Advantage is an integrated software suite that pairs the reliability and cost-effectiveness of PHP, Apache, Perl and open source databases with new technology for building business-critical web applications and web services. Based in Bedford, Mass., the company's commercial software services include technical support, consulting and training. For more information, visit www.nusphere.com or call +1-781-280-4600.

NuSphere is a registered trademark in Australia, Norway, Hong Kong, Switzerland, and the European Community; NuSphere and PHPEd are trademarks of NuSphere Corporation in the U.S. and other countries. Any other trademarks or service marks contained herein are the property of their respective owners.

MySQL AB distributes the MySQL database pursuant to the applicable GNU General Public License that is available as of the date of this publication at <http://www.fsf.org/licenses/gpl.txt> and all of the terms and disclaimers contained therein. NuSphere Corporation is not affiliated with MySQL AB. The products and services of NuSphere Corporation are not sponsored or endorsed by MYSQL AB.