

Locking



Locking

**Progress® Version 9.1D SQL-92 Server
Edition 1, March 2003**

Contents

INTRODUCTION.....	3
LOCKS	3
TRANSACTIONS.....	4
WHAT IS DEFINED IN THE 1992 SQL STANDARD	4
LOCK LEVEL	7
LOCK MODE	7
HOW LOCK LEVELS AND LOCK MODES INTERACT TO PROVIDE THE BEHAVIOR DESCRIBED IN THE 1992 SQL STANDARD	8
MATCHING THE TRANSACTION ISOLATION LEVELS BEHAVIORS:	8
LOCK ACQUISITION	10
INFORMATION SCHEMA LOCK	10
TABLE/RECORD LOCK.....	10
AN EXAMPLE:.....	11
IN A NUTSHELL:	11
LOCK VISIBILITY	11
OTHER USEFUL INFORMATION.....	14
AUTO-COMMIT AND THE INFORMATION SCHEMA LOCK	14
TABLE LOCKS AND THE 4GL	14

Introduction

The intent of this white paper is to convey information regarding database locks as they apply to transactions in general and the more specific case of how they are implemented by the Progress® SQL-92 server. We'll begin with a general overview discussing why locks are needed and how they affect transactions. Transactions and locking are outlined in the SQL standard so no introduction would be complete without discussing the guidelines set forth here. Once we have a grasp on the general concepts of locking we'll dive into lock modes, such as table and record locks and their effect on different types of database operations. Next, the subject of timing will be introduced, when locks are obtained and when they are released. From here we'll get into lock contention and deadlocks, which are multiple operations or transactions all attempting to get locks on the same resource at the same time. And to conclude our discussion on locking we'll take a look at how we can see locks in our application so we know which transactions obtain which types of locks. Finally, this white paper describes differences in locking behavior between previous and current versions of Progress and differences in locking behavior when both 4GL and SQL92 clients are accessing the same resources.

To avoid being redundant, this white paper makes reference to Progress product documentation, which is available at www.progress.com/documentation.

Locks

Why do we lock database objects?

The answer to why we lock is simple; if we didn't there would be no consistency. Consistency provides us with successive, reliable, and uniform results without which applications such as banking and reservation systems, manufacturing, chemical, and industrial data collection and processing could not exist. Imagine a banking application where two clerks attempt to update an account balance at the same time: one credits the account and the other debits the account. While one clerk reads the account balance of \$200 to credit the account \$100, the other clerk has already completed the debit of \$100 and updated the account balance to \$100. When the first clerk finishes the credit of \$100 to the balance of \$200 and updates the balance to \$300 it will be as if the debit never happened. Great for the customer; however the bank wouldn't be in business for long.

What objects are we locking?

What database objects get locked is not as simple to answer as why they're locked. From a user perspective, objects such as the information schema¹, user tables, and user records are locked while being accessed to maintain consistency. There are other lower level objects that require locks that are handled by the RDBMS; however, they are not visible to the user. For the purposes of this discussion we will focus on the objects that the user has visibility of and control over.

¹ Schema as defined by the Webster's dictionary reads as follows, "A diagrammatic representation; an outline or model". As applied to a Relational Database Management Systems, schema is a term used to represent the metadata; tables, fields and indexes. The term "schema" used unqualified in the context of SQL may have a different meaning depending on whether or not you're a seasoned SQL user. A SQL Catalog is a container for one or more schema, at a minimum there is the Information Schema that contains the systems base tables or metadata. There can also be one or more user defined schema contained within a SQL Catalog. For the seasoned Progress 4GL user there is only one schema, a singular object representing the metadata; tables, fields and indexes. For this discussion we will refer to this object by its correct SQL name, the Information Schema. For more information about the SQL Catalog, and the SQL Schema please refer to Melton and Simon's book entitled "Understanding the New SQL: A Complete Guide".

Transactions

Now that we know why and what we lock, let's talk a bit about when we lock. A transaction is a unit of work; there is a well-defined beginning and end to each unit of work. At the beginning of each transaction certain locks are obtained and at the end of each transaction they are released. During any given transaction, the RDBMS², on behalf of the user, can escalate, deescalate, and even release locks as required. We'll talk about this in more detail later when we discuss lock modes. The aforementioned is all-true in the case of a normal, successful transaction; however in the case of an abnormally terminated transaction things are handled a bit differently. When a transaction fails, for any reason, the action performed by the transaction needs to be backed out, the change undone. To accomplish this most RDBMS use what are known as "save points"³. A save point marks the last known good point prior to the abnormal termination; typically this is the beginning of the transaction. It's the RDBMS's job to undo the changes back to the previous save point as well as ensuring the proper locks are held until the transaction is completely undone. So, as you can see, transactions that are in the process to be undone (rolled back) are still transactions nonetheless and still need locks to maintain data consistency.

Locking certain objects for the duration of a transaction ensures database consistency and isolation from other concurrent transactions, preventing the banking situation we described previously. Transactions are the basis for the **ACID**⁴ properties:

- **ATOMICITY** guarantees that all operations within a transaction are performed or none of them are performed.
- **CONSISTENCY** is the concept that allows an application to define consistency points and validate the correctness of data transformations from one state to the next.
- **ISOLATION** guarantees that concurrent transactions have no effect on each other.
- **DURABILITY** guarantees that all transaction updates are preserved.

What is Defined in the 1992 SQL Standard

The 1992 SQL standard does not specify how a SQL implementation should provide for data consistency and concurrency via various locking schemes. However it does specify what the expected behavior should be for active transactions in different situations. Each behavior is identified with a specific name referred to as: "Transaction isolation level" and offers varying degrees of isolation and concurrency while accessing a database.

To clearly identify expected behaviors for each transaction isolation levels, the 1992 SQL standard starts by describing different issues occurring while accessing data in a concurrent mode. These issues are called phenomena and are permitted or prevented by each isolation level resulting in varying degrees of isolation and concurrency.

² Relational Database Management System, a type of database management system (DBMS) that stores data in the form of related tables

³ The ability of the client application to set user defined save points is not part of the SQL92 standard however it is an extension to the core SQL99 standard. The Progress database development team is currently giving thought to implementing user defined save points in a future release.

⁴ For a full explanation and discussion on the ACID database properties please refer to Jim Gray and Andreas Reuter's book entitled "Transaction Processing: Concepts and Techniques".

The three phenomena described in the 1992 SQL standard are as follows:

Dirty read — Occurs when one user is updating / inserting a record while a different user is reading it, but that work is not committed to the database.

Hypotheses: We assume there is no transaction control mechanism used by the RDBMS which could mean that the RDBMS is not locking any records while a user is accessing the database.

Scenario:

```
User 1 executes:
    INSERT INTO pub.State (state, state_name, region)
    VALUES ('AB', 'Abcdefghij', 'ABCD');
User 2 executes:
    SELECT * FROM pub.State
User 2 sees: state 'AB'
User 1 executes:
    ROLLBACK WORK
User 2 has seen data that did not really exist.
```

Conclusion: To prevent this phenomenon, User 2 must guarantee that the records being accessed are not currently being accessed from a different transaction started by a different user.

Non repeatable read — Occurs when one user is repeating a read operation on the same records but has updated values.

Hypotheses: We assume there is no transaction control mechanism used by the RDBMS or that the RDBMS is able to check that a record about to be retrieved is used in a transaction held by a different user. This could mean that the RDBMS is not locking any records while a user is accessing the database, or that there is a mechanism to check if a record is locked by a different user.

Scenario:

```
User 1 executes:
    SELECT * FROM pub.State
User 2 executes:
    UPDATE pub.State
    SET state_name = 'hello world'
    WHERE state = 'AK';
    COMMIT WORK;
User 1 re-executes:
    SELECT * FROM pub.State
User 1 has now updated records in the result set.
```

Conclusion: To prevent this phenomenon, the RDBMS needs to use some sort of mechanism preventing any other user from updating the records User 1 has already read. This mechanism should be used until User 1 indicates that the read operation is complete.

Phantom Read — Occurs when one user is repeating a read operation on the same records but has new records in his result set.

Hypotheses: We assume there is no transaction control mechanism used by the RDBMS or that the RDBMS is able to check that a record about to be retrieved is used in a transaction held by a different user and that it holds locks on records retrieved until the user indicates that the transaction is complete. This could mean that the RDBMS is not locking any records while a user is accessing the database, or that there is a mechanism to check if a record is locked by a different user and that there is a mechanism holding locks on a record until the end of the transaction.

Scenario:

```
User 1 executes:
    SELECT * FROM pub.State
User 2 executes:
    INSERT INTO pub.State (state, state_name, region)
    VALUES ('AB', 'Abcdefghij', 'ABCD');
    COMMIT WORK;
User 1 re-executes:
    SELECT * FROM pub.State
User 1 has new records in the result set.
```

Conclusion: To prevent this phenomenon, the RDBMS needs to use some sort of mechanism preventing any other user from inserting records in the table User 1 is currently accessing. This mechanism should be used until User 1 indicates that the read operation is complete.

In order to allow or prevent each of the above phenomena, 4 isolation levels are clearly identified:

- **READ UNCOMMITTED** (also called “dirty read”) — When this isolation level is used, a transaction can read uncommitted data that later may be rolled back. The standard requires that a transaction that uses this isolation level can only fetch data but can’t update, delete, or insert data.
- **READ COMMITTED**— With this isolation level dirty reads are not possible, but if the same row is read repeatedly during the same transaction, its contents may be changed or the entire row may be deleted by other transactions.
- **REPEATABLE READ** — This isolation level guarantees that a transaction can read the same row many times and it will remain intact. However, if a query with the same search criteria (the same WHERE clause) is executed more than once, each execution may return different set of rows. This may happen because other transactions are allowed to insert new rows that satisfy the search criteria or update some rows in such way that they now satisfy the search criteria.
- **SERIALIZABLE** — This isolation level guarantees that none of the above happens. In addition, it guarantees that transactions that use this level will be completely isolated from other transactions.

Table 1 gives a clear overview of what is described above. It identifies which phenomena are either permitted or prevented by each isolation level.

Table 1

	Dirty Read	Dirty Read	Dirty Read
Read Uncommitted	Permitted	Permitted	Permitted
Read Committed	Prevented	Permitted	Permitted
Repeatable Read	Prevented	Prevented	Permitted
Serializable	Prevented	Prevented	Prevented

Note that the isolation levels are ordered according to the phenomena they either permit or prevent —the first one (READ UNCOMMITTED) is the isolation level providing the highest level of concurrency but with the lowest level of consistency. Each subsequent level provides at least as much data consistency as the one before but will result in less concurrency.

As a general rule, the more data consistency that is provided by the isolation level used from an application, the less concurrency is allowed between this application and other applications connected to the same database.

Lock Level

In order to provide a reasonable level of concurrency, locking is performed in a hierarchy that requires several lock modes across the levels of the hierarchy. For the purpose of this discussion there are three levels in our locking hierarchy, records, tables and information schema. A record is at the lower level, a table is at the medium level, and the information schema is at the highest level.

Lock Mode

In general, lock modes are prioritized tokens in a queue that indicate what action is being taken. The intent to update a given record requires a different mode of lock than to actually update the record; likewise the intent to read a record requires a different mode of lock than to actually read the record. Lock modes are needed to facilitate concurrency and provide consistency; they indicate intent and are used to stage lock requests. Lock requests are generated on the user's behalf as a result of executing a transaction, such as a database read or write.

Progress provides 6 lock modes that are described as follows:

- ***NO-LOCK (NL)*** — you have no intentions of performing an update and accuracy of the resulting set of data is not important.
- ***INTENT SHARE (IS)*** — you intend to share-lock objects at the next lower level of granularity for this object (table). That is, you intend to get share locks on the rows of this table.
- ***INTENT EXCLUSIVE (IX)*** — you intend to exclusive-lock objects at the next lower level of granularity for this object (table). That is, you intend to get exclusive locks on the rows of this table.

- **SHARED (S)** — you want a share-lock on the object. Getting a share-lock on an object means that you implicitly get a share-lock on all of the objects that this object contains, i.e. all of the rows for this table.
- **SHARED WITH INTENT EXCLUSIVE (SIX)** — you want a share-lock on the table so no one else can modify, delete, or add rows except for you.
- **EXCLUSIVE (X)** — you want an exclusive-lock on the object. Getting an exclusive lock on an object means that you implicitly get an exclusive lock on all of the objects that this object contains, i.e. all of the rows for this table.

Note: Since rows (records) are the lowest level in the locking hierarchy, the intent locks (IS, IX and SIX) do not apply. At the information schema level there is currently no need to use any of these intents locks. In other words, they are only used at the table level.

Table 2 depicts the lock compatibility matrix for these different lock modes. A checkmark indicates the requested lock can be granted. A cross indicates that the requested and the granted modes are not compatible so the requested lock could not be granted on the object.

This matrix applies to lock requests made by a user different than the holder of the lock.

Table 2

Requested Mode	Granted Mode						
	Lock Mode	None	IS	IX	S	SIX	X
	None	✓	✓	✓	✓	✓	✓
	IS	✓	✓	✓	✓	✓	✗
	IX	✓	✓	✓	✗	✗	✗
	S	✓	✓	✗	✓	✗	✗
	SIX	✓	✓	✗	✗	✗	✗
	X	✓	✗	✗	✗	✗	✗

How lock levels and lock modes interact to provide the behavior described in the 1992 SQL standard

Matching the transaction isolation levels behaviors:

Table 3 describes how the Progress SQL-92 server makes use of this locking scheme implementation in order to match the desired behavior described in the 1992 SQL standard.

This information is also provided here for a better understanding of how a caller might make use of this table lock implementation; however it is not intended to dictate how it is used.

Note that the information in the table applies to the requested lock strength based on the transaction isolation level in effect for a given transaction. This table does not take into consideration lock upgrades possibly resulting in a different lock strength actually being applied.

Table 3

	Insert / Update / Delete record Operation			Fetch / Select record Operation		
Isolation Level	Information Schema Lock	Table Lock	Record Lock	Information Schema Lock	Table Lock	Record Lock
Serializable	S	SIX	X	S	S	S
Repeatable Read	S	IX	X	S	IS	S
Read Committed	S	IX	X	S	IS	S
Read Uncommitted	S	NL	NL	S	NL	NL

Notes:

- As Table 3 indicates, there are no table or record locks acquired when the transaction isolation level is Read Uncommitted. In the Read Uncommitted transaction isolation level you maximize concurrency but may also read dirty data. Depending on your application, this may be acceptable.
- The primary difference between the Read Committed and Repeatable Read transaction isolation levels is that while in Repeatable Read, individual record locks are held for the duration of the transaction. For example, if your fetch criteria include all companies in the state of Idaho, each record in the result set will remain locked until all of the records meeting the criteria have been read. In the Read Committed transaction isolation level the record locks are released once the record has been read. So, as the name indicates, you will only read committed records, which may change once your result set is complete. This is known as a phantom read. If you were to re-read the same records, it's possible that records that were once visible no longer exist. To avoid this behavior, use the Repeatable Read transaction isolation level keeping in mind that as you progress towards the Serializable transaction isolation level you reduce your application's concurrency because locks are held for a longer duration.
- In the Serializable transaction isolation level a share lock on a table is held for the duration of the transaction, preventing any other transaction from updating the table.
- Any SQL operation that modifies the information schema automatically gets upgraded to the serializable transaction isolation level regardless of the user's current transaction setting.

Now that we've discussed lock modes / levels and how they affect transactions, let's take a look at lock acquisition, that is, how and when locks are acquired.

Lock Acquisition

How and when are locks acquired? So far we've talked about locking in general, how the SQL standard interprets locks via behavior, lock modes, and database objects that get locks applied to them such as tables, records, and information schema. Knowing which objects get locked and when goes a long way towards helping you develop applications that are more robust and predictable.

Since SQL uses the transaction isolation level exclusively to determine what lock mode is applied to which objects, it's extremely important to know how this translates into object locks and lock modes. This is the only way to communicate your application's intentions to the SQL engine. As you can see from Table 3 above, the strongest locks are held when the transaction isolation level is Serializable and the weakest locks are held when the transaction isolation level is Read Uncommitted. This also translates into application concurrency—the higher the transaction isolation level, the less concurrent your application will be.

Information Schema Lock

Every operation performed by the SQL server operates inside a transaction. Operating inside a transaction provides the ability to provide for the ACID properties we expect from a database. For each transaction, a information schema share-lock is acquired at the beginning of the transaction and released at the end of the transaction. This is true regardless of whether or not the transaction is committed successfully or terminated abnormally. Acquiring the information schema share-lock protects the information schema from being altered while the transaction is active. During the life of an active connection there can be many individual transactions begun and ended depending on the operations being performed. The first transaction is begun upon connection to the SQL server and is used to read the information schema. Once the information schema has been read, the transaction is ended. Each successive operation will then begin and end a transaction requiring, at a minimum, a share-lock on the information schema. While the connection is quiet, there is no active transaction and therefore no lock held on the information schema. If an operation is being performed that will modify the information schema, an exclusive lock on the information schema will be requested. For the exclusive lock on the information schema to be granted, there can be no other active transactions in the database. Once granted, the information schema lock is upgraded from a share to an exclusive lock. While this transaction is active, no other transaction can begin because the share-lock on the information schema cannot be granted while there is an outstanding exclusive lock on the information schema. Keep in mind that the lock on the information schema is above and beyond any locks obtained on tables and records via transaction isolation level settings for data manipulation operations.

Table/Record Lock

Lock acquisition for tables and records is straightforward given Tables 2 and 3 above and the information we covered so far regarding transactions. To get a record lock of sufficient strength for the operation being performed, you must first have a table lock of sufficient strength. Regardless of the current transaction isolation level, if the application's intent is to perform an operation other than a fetch, the lock mode will be in effect strengthened for the duration of that operation. That is to say, you are not prohibited from creating or updating records based on the transaction isolation level. It is the responsibility of the SQL implementation's RDBMS to provide sufficient lock escalation when an operation is being performed that requires lock upgrades.

What happens when two transactions attempt to update an object at the same time? Let's take a look at our banking example where two clerks, C1 and C2, are updating a customer's bank account balance at the same time. To update the customer's bank account balance requires an exclusive lock on two tables; the accounts table, T1, where the customer information is located and the transaction table, T2, where debits and credits are recorded. To perform the update C1 gets an exclusive lock on T1, and C2 gets an exclusive lock on T2.

C1 now tries to lock T2, and C2 tries to lock T1. Neither can proceed because each holds a resource locked exclusively that's required to perform the update. This is called a deadlock and is handled by the RDBMS via a lock wait timeout. The rule imposed by the SQL RDBMS is that a transaction will wait for a resource to become available for five seconds at which point the application will need to retry the operation. A well-designed application could prevent this type of deadlock by requiring a lock on T1 prior to requesting a lock on T2. This example was contrived to illustrate a deadlock scenario and does not represent any kind of programming best practices.

That's it in a nutshell. Simply put, the RDBMS translates SQL transaction isolation level intended behavior into locks on information schema, tables, and records of varying strengths to give the desired results.

An example:

Assuming that you want to have the behavior associated with the **“Read Committed”** transaction isolation level (as defined by the 1992 SQL standard), locking a record requires that a lock gets placed at the information schema level to prevent any information schema modification to occur while accessing the record. It also requires that an intent share (IS) lock be obtained on a table, before locking the record itself. Thus to share-lock a row of a table requires you to get a shared intent lock on the table and a shared lock on the information schema before locking a row in the table. Similarly, before getting an exclusive lock on a row of a table, you must get a shared lock on the information schema and an exclusive intent lock on the table first.

This also means (depending on your transaction isolation level) that:

- To get a Share (S) lock on a record, the table must be locked with an Intent Share (IS) lock or stronger. If the table holds a Share (S), Shared with Intent Exclusive (SIX), or Exclusive (X) lock, then record locks need not be obtained at all for that table.
- To get an Exclusive (X) lock on a record, the table must hold an Intent Exclusive (IX) lock or stronger. If the table holds an Exclusive (X) lock then record locks need not be obtained at all for that table.

In a nutshell:

The locking protocol follows two simple rules:

- **Acquire locks from the top down.** Acquire an information schema lock and then a table lock before acquiring any record locks.
- **Release locks from the bottom up.** Release record locks before releasing table locks and at last the information schema lock.

Now that we've discussed lock modes, levels, and lock acquisition and how they affect transactions, let's take a look at lock visibility, that is, how we can see locks that have been acquired.

Lock Visibility

How can the user see what locks are in effect at any given time? There are a couple ways to accomplish this with the tools available in any Progress installation. See the *Progress Database Administration Guide and Reference* for details. PROMON is probably the most widely used utility for monitoring locks. Using

Virtual System Tables (VSTs) is also an option. Basic PROMON provides “canned” reports that you can use to get a feel for what is happening with regards to locks.

The “Record Locking Table” option displays locks that are being held at any given time. This information can be used to see the locks that are currently being held in an attempt to resolve a locking conflict. The information presented includes the ID and Name of the user holding the lock, the lock chain ID, the Record ID, the Table number and the type of lock, and a flag indicating the state of the lock. Additional information on this and other PROMON tables can be found in the *Progress Database Administration Guide and Reference*. Here is sample output:

Record Locking Table:						
Usr	Name	Chain #	Rec-id	Table	Lock	Flags
44	jffj	REC 105	103	2	SHR	L
41	jffj	REC 105	103	2	EXCL	Q H
44	jffj	REC 269	10240	2	SHR	L
44	jffj	REC 270	10241	2	SHR	L
44	jffj	REC 307	10278	2	SHR	L
44	jffj	REC 686	10657	2	SHR	L
44	jffj	REC 707	705	2	SHR	L
44	jffj	REC 742	740	2	SHR	L
44	jffj	REC 771	769	2	SHR	L
44	jffj	REC 772	770	2	SHR	L
44	jffj	REC 774	772	2	SHR	L
44	jffj	REC 803	801	2	SHR	L
44	jffj	REC 836	834	2	SHR	L
44	jffj	REC 837	835	2	SHR	L
44	jffj	REC 867	865	2	SHR	L
42	jffj	REC 890	20832	4	EXCL	L
44	jffj	REC 900	898	2	SHR	L
44	jffj	REC 903	901	2	SHR	L
44	jffj	REC 941	10912	2	SHR	L

The “Locking and Waiting Statistics” option displays statistics regarding locks. The information in this report is cumulative for the life of the process. This information can give you a feel for your system’s concurrency with regards to locks. The first two lines display cumulative locking statistics. The information presented includes the Lock, whether a Lock or Wait, the User ID and Name of the user, the Record ID, the number of times a Transaction Lock was issued, and the total number of times a Information schema lock was obtained for that lock type. Here is sample information that this table provides:

Locking and Waiting:				
Type	Usr	Name	Record	Trans
Lock	999	TOTAL...	89796	824
Wait	999	TOTAL...	142	4
Lock	0	jffj	0	0
Wait	0	jffj	0	0
Lock	41	jffj	39466	311
Wait	41	jffj	25	1
Lock	42	jffj	19216	228
Wait	42	jffj	30	0
Lock	43	jffj	13832	103
Wait	43	jffj	30	1
Lock	44	jffj	8579	87
Wait	44	jffj	19	0
Lock	45	jffj	5550	45
Wait	45	jffj	22	2
Lock	46	jffj	0	0
Wait	46	jffj	0	0
Lock	47	jffj	2808	23
Wait	47	jffj	11	0
Lock	48	jffj	345	27
Wait	48	jffj	5	0

The “Transaction Control” option displays information regarding individual transactions. The information in this table is useful for identifying when transactions are active and which user is holding them. The information presented includes the User ID and Name of the user with the transaction, the Transaction ID, the Date and Time the transaction began, the Ready to commit state, whether a transaction is Limbo, and other Coordinator information with regards to distributed transactions.. Here is sample information that this table provides:

Transaction Control:									
Usr	Name	Trans	Login	Time	R-comm?	Limbo?	Crd?	Coord	Crd-task
41	jfj	8008	09/20/02	12:47	no	no	no		0
42	jfj	7939	09/20/02	12:47	no	no	no		0
43	jfj	7556	09/20/02	12:48	no	no	no		0
44	jfj	7973	09/20/02	12:48	no	no	no		0
45	jfj	7987	09/20/02	12:48	no	no	no		0
47	jfj	7988	09/20/02	12:48	no	no	no		0
48	jfj	7650	09/20/02	12:49	no	no	no		0
49	jfj	7985	09/20/02	12:49	no	no	no		0
50	jfj	8001	09/20/02	12:49	no	no	no		0

In addition to the basic PROMON displays, there are a couple of other displays under the PROMON R&D menu. You can access the R&D Menu by typing “r&d” at the main PROMON menu. There are activity, status, and other lock displays that can help you understand what is happening with regards to locking on the system.

Another option that provides information on locks is Progress Virtual System Tables (VSTs). VSTs are provided so that users can “roll their own” reports from data collected in the applicable VST. Of interest to anyone looking to understand more about locking are the following VSTs:

Lock Table Activity (_ActLock) — Displays lock-table activity, including the number of share, exclusive, upgrade, Rec Get, and redundant requests; the number of exclusive, Rec Get, share, and upgrade grants; the number of exclusive, Rec Get, share, and upgrade waits; the number of downgrades, transactions committed, cancelled requests, and database up time.

Lock Table Status File (_Lock) — Displays the status of the lock table, including the user number, the user name, lock type, record ID, number, flags, and chain.

Lock Request File (_LockReq) — Displays information about lock requests, including user name and number, record locks and waits, information schema locks and waits, and transaction locks and waits.

Record Locking Table File (_UserLock) — Displays the contents of the record locking table, such as user name, chain, number, record ID, lock type, and flags.

Other Useful Information

Based on a number of responses to customer inquiries, this section should provide additional insight into the specifics of why locking behavior for the SQL-92 server appears as it does.

Auto-Commit and the Information Schema Lock

In Progress Version 9.1D, the scope of the information schema lock for SQL92 was changed. Prior to Version 9.1D, a share-lock on the information schema was obtained upon connection and held for the duration of the connection. This meant that no information schema modifications could be performed while users other than the one performing the modification were connected to the database. This posed complications for customers using connection pools in that they would have to shut down the connection pool to perform an information schema operation. Beginning with Version 9.1D, the information schema share-lock has been moved from the scope of a connection to the scope of a transaction. No longer is the lock held for the duration of the connection.

Another aspect of the information schema lock is the auto-commit behavior. There have been a number of inquiries regarding the information schema lock being held even though there are no active clients. One way to be sure of this is to look at the Transaction Control Table using PROMON. Because of the way auto-commit works, if your client uses auto-commit they will always be inside of a transaction. With auto-commit active each time a statement is executed the previous statement is committed. This relieves the user of the task of committing every statement. So, unless you explicitly commit your transactions, or turn auto-commit off, it's likely that you will run into this situation.

Table Locks and the Progress 4GL

Prior to the introduction of the SQL92 Server and the background work started with Progress Version 9.0A, the Progress database engine did not support table locks. Consistency and concurrency were maintained with record locks. Because SQL relies on table and record locks to carry out the intent of transaction isolation levels, table locks were implemented in the database engine. Now, both the 4GL and SQL clients encounter table locks while executing transactions. From the 4GL point of view, table locks are somewhat transparent, and based on our previous discussion regarding table and record locking we already know how they affect the SQL client.

When we described the banking example above, we mentioned locking conflicts and what happens when two operations request the same resource at the same time. This is known as a dead lock. The SQL client will wait on a resource for five seconds before giving up, at which point the operation would need to be re-tried. The five-second wait is currently hard-coded and applies to all SQL clients. The Progress database development team is currently giving thought to changing this value in a future release. For 4GL clients, there is a Lock Wait Timeout (`-lkwtmpo`) parameter that specifies how long a client should wait for a resource. The current default value for the Lock Wait Timeout parameter is thirty minutes. Given that a SQL client can only wait five seconds and a 4GL client could wait as long as 30 minutes, if a SQL client has a lock on a table for which a 4GL client also has requested a lock, the SQL client will timeout and give up waiting long before the 4GL client.

The reason for providing this information is to make our customers aware that there are behavioral differences when both 4GL and SQL clients are active on the same database. Knowing that the Lock Wait Timeout values between the 4GL and SQL are different and that the 4GL is subject to table locks when SQL clients are active may help explain unfamiliar or unexpected locking behavior.

Worldwide and North American Headquarters

Progress Software Corporation, 14 Oak Park, Bedford, MA 01730 USA Tel: 781 280 4000 Fax: 781 280 4095

Europe/Middle East/Africa Headquarters

Progress Software Europe B.V. Schorpioenstraat 67 3067 GG Rotterdam, The Netherlands Tel: 31 10 286 5700 Fax: 31 10 286 5777

Latin American Headquarters

Progress Software Corporation, 2255 Glades Road, One Boca Place, Suite 300 E, Boca Raton, FL 33431 USA Tel: 561 998 2244 Fax: 561 998 1573

Asia/Pacific Headquarters

Progress Software Pty. Ltd., 1911 Malvern Road, Malvern East, 3145, Australia Tel: 61 39 885 0544 Fax: 61 39 885 9473

Progress is a registered trademark of Progress Software Corporation. All other trademarks, marked and not marked, are the property of their respective owners.

PROGRESS
SOFTWARE

www.progress.com

Specifications subject to change without notice.

© 2003 Progress Software Corporation.

All rights reserved.