ABS Tech Note: DAL17 INGRES Locking Issues (6/92)

Article Created: 30 June 1992

TOPIC ----------

This technical note describes special locking considerations when using DAL with INGRES.

DISCUSSION ----------

By default, INGRES provides "repeatable read" transactions. This means that during a single transaction, your application can repeat a DAL query two or more times and is guaranteed that the query results will be identical. Thus, no database changes made by other users during your transactions will be visible to your program. INGRES provides this high level of transaction integrity by locking the sections of the database that your query has accessed until your transaction ends with a COMMIT or ROLLBACK statement. During your transaction, no other user may update the locked portions of the database.

This default INGRES locking can reduce the amount of concurrent access to an INGRES database, especially when a transaction includes "think time" during which your program is waiting for user input. To increase concurrency, INGRES provides several locking parameters which can be set programmatically through the INGRES SET LOCKMODE statement. These excerpts from the INGRES SQL reference manual, pages 2-56 and 2-57, describe these parameters.

Description of lock modes:

SET LOCKMODE provides four different parameters to govern the nature of locking in an INGRES session:

- Level: This refers to the level of granularity desired when the table is accessed. You can specify any of the following locking levels:

page	Specifies locking at the level of the data page (subject to escalation criteria; see below)
table	Specifies table-level locking in the database
session	Specifies the current default for your INGRES session
system	Specifies that INGRES will start with page-level locking,

unless it estimates that more than Maxlocks pages will be referenced, in which case table-level locking will be used.

- Readlock: This refers to locking in situations where table access is required for reading data only (as opposed to updating data). You can specify any of the following Readlock modes:

nolock	Specifies no locking when reading data
shared	Specifies the default mode of locking when reading data
exclusive	Specifies exclusive locking when reading data (useful in "select-for-update" processing within a multi-statement transaction)
system	Specifies the general Readlock default for the INGRES system

- Maxlocks: This refers to an escalation factor, or number of locks on data pages, at which locking escalates from page-level to table-level. The number of locks available to you is dependent upon your system configuration. You can specify the following Maxlocks escalation factors:

n	A specific (integer) number of page locks to allow before escalating to table-level locking. The default "n" is 10, and "n" must be greater than 0.
session	Specifies the current Maxlocks default for your INGRES session
system	Specifies the general Maxlocks default for the INGRES system

  Note: If you specify page-level locking, and the number of locks granted during a query exceeds the system-wide lock limit, or if the operating system's locking resources are depleted, locking escalates to table-level. This escalation occurs automatically and is independent of the user.

- Timeout: This refers to a time limit, expressed in seconds, for which a lock request should remain pending. If INGRES cannot grant the lock request within the specified time, then the query that requested the lock aborts. You can specify the following timeout characteristics:

n	A specific (integer) number of seconds to wait for a lock (setting "n" to 0 requires INGRES to wait indefinitely for the lock)
session	Specifies the current timeout default for your INGRES session (which is also the INGRES default)
system	Specifies the general timeout default for the INGRES system

Against the backdrop of these SET LOCKMODE parameters and options are the INGRES system defaults for each of the parameters:

Level	dynamically determined by INGRES
Readlock	shared
Maxlocks	10
Timeout	0 (no timeout)

If you select the system option for any of the SET LOCKMODE parameters, the values above are automatically supplied. When you begin your INGRES session, the INGRES system defaults are in effect. If you override them with other values using the SET LOCKMODE command, you can revert to the system defaults easily.

Similarly, if you set session parameters (such as locking behavior for all user tables accessed by queries in your INGRES session), you can further set parameters for individual tables on an ad hoc basis. After setting the ad hoc locking behavior, you can return it to either the session defaults or to the INGRES system defaults.

INGRES tries to place a lock on every page it touches in a transaction. Depending on the Lockmode, this will lead to an exclusive, shared, or no lock condition on the page. INGRES also escalates locks (as with more than "Maxlocks," the whole table will be locked to prevent deadlocks). The default Lockmode in INGRES is shared. This means that several users can share the same page for "selects," but updates or inserts cannot take place until the page is held only by the updating user. If a user performs a "select * from xxx" type of query, then every physical page in the table is marked and the locks are not removed until the transaction is finished. This prevents other users from making updates to the same table. Concurrency will therefore be reduced.

However, concurrency is only an issue when there are write operations performed on the database on-line (as with updates or inserts). If no updates are going to take place in your application, you should use the "nolock" mode.

If you know in advance that your application will not be updating data in a transaction, it can use the "nolock" mode to eliminate the need for INGRES to lock parts of the database accessed by a query. Here is an example of a DAL sequence that uses this technique:

```
/* Open INGRES for "select" and non-updating operations only: */
open INGRES dbms;
open INGRES database "cl1demo";
execute in INGRES "set lockmode session where readlock = nolock";
select * from offices;
[continue with session]
```

If you do not want to turn off locking, you can still improve concurrency in a query-only application by making sure that a COMMIT statement is executed as soon as you have finished processing each set of query results. Do not perform two or more queries in sequence without an intervening COMMIT statement unless your application absolutely depends on there being no changes to the database between the queries.

Finally, if your application must update the database, it should avoid "think" periods during a transaction that locks the database whenever possible. For example, this statement sequence will work correctly when the lockmode is shared or exclusive, but may lock large parts of the database during the "think time:"

```
  select (my_columns) where bool_expr;
  /* think and make changes */
  update (my_columns) where bool_expr;
  commit;
```

An alternative approach that will improve concurrency is:

```
  /* Do the query the first time without locking */
  execute in INGRES "set lockmode session where level = system,
          readlock = nolock";
  select (my_columns) where bool_expr;
  printall;
  /* application puts data in buffer #1 */
  /* think and make changes and store updates in a buffer #3 */

  /* Turn on locking and do the query a second time */
  execute in INGRES "set lockmode session where level = page,
          readlock = exclusive";
  select (my_columns) where unique_id;
  printall;
  /* application puts data in buffer #2 */

  /* If data remains unchanged from first query, do the update */
  if buffer2 == buffer1 {
      update (my_columns)  where unique_id;    /* using contents of buffer
  #3 */
     commit;
  }
  else
  {
      /* warn the user that the data has changed since last select */
  }

  /* Turn locking back off again */
  execute in INGRES "set lockmode session where level = system, readlock =
  nolock";
```

The "unique_id" can be, for example, the employee number or another identifier that uniquely identifies the rows to be updated. The objective in an update process is to "touch" as few pages as possible, since all touched pages will be locked until the transaction commits. The reason for going through this lengthy process is to avoid having a page locked up for a long time.

Keywords: <None>

==========

19960215 11:05:19.00

Tech Info Library Article Number: 11644