B

Automatic and Manual Locking Mechanisms During SQL Operations

This appendix describes mechanisms that lock data either automatically or as specified by the user during SQL statements. For a general discussion of locking mechanisms in the context of data concurrency and consistency, see *Oracle Database Concepts*.

This appendix contains the following sections:

List of Nonblocking DDLs

Release 23

The following nonblocking DDLs are added in Release 23.3:

- alter table add column
- alter table set column unused
- alter table add constraint enable novalidate
- alter table drop constraint

Release 21

The following nonblocking DDLs are added in Release 21c:

- alter table modify default attributes tablespace
- alter table modify default attributes lob tablespace
- alter index modify default attributes tablespace
- alter table modify default attributes for partition tablespace
- alter table modify default attributes for partition lob tablespace
- alter index modify default attributes for partition tablespace

Release 12.2.0.2

List of Nonblocking DDLs Added in 12.2.0.2

- Alter table merge partition online
- alter table modify partition by .. online (to change the partitioning schema of a table)

Release 12.2.0.1

List of Nonblocking DDLs Added in 12.2.0.1

- alter table split partition [subpartition] online
- alter table move online (move of a non-partitioned table)
- alter table modify partition by .. online (to convert a non-partitioned table to partitioned state)

Release 12.1

The following nonblocking DDLs are added as of Release 12.1. Some nonblocking DDLs are downgraded to blocking in the presence of supplemental logging.

List of Nonblocking DDLs Added in 12.1

- drop index online
- alter index unusable online
- alter table move partition online
- alter table move subpartition online

List of Nonblocking DDLs Added in 12.1 that Downgrade to Blocking During Supplemental Logging

- alter table set unused column online
- alter table drop constraint online
- alter table modify column visible / invisible
- alter table add nullable column with default value

Release 11.2

The following nonblocking DDLs are added as of Release 11.2. Some nonblocking DDLs are downgraded to blocking in the presence of supplemental logging.

List of Nonblocking DDLs Added in 11.2

- create index online
- alter index rebuild online
- alter index rebuild partition online
- alter index rebuild subpartition online
- alter index visible / novisible

List of Nonblocking DDLs Added in 11.2 that Downgrade to Blocking During Supplemental Logging

- alter table add column not null with default value
- alter table add constraint enable novalidate
- alter table modify constraint validate
- alter table add column (without any default)

Automatic Locks in DML Operations

The purpose of a DML lock, also called a **data lock**, is to guarantee the integrity of data being accessed concurrently by multiple users. For example, a DML lock can prevent multiple customers from buying the last copy of a book available from an online bookseller. DML locks prevent destructive interference of simultaneous conflicting DML or DDL operations.

DML statements automatically acquire locks at both the table level and the row level. In the sections that follow, the acronym in parentheses after each type of lock or lock mode is the abbreviation used in the Locks Monitor of Oracle Enterprise Manager. Enterprise Manager might display "TM" for any table lock, rather than indicate the mode of table lock (such as RS or SRX).

The types of row and table locks are summarized here. For a more complete discussion of the types of row and table locks, see *Oracle Database Concepts*.

Row Locks (TX)

A **row lock**, also called a **TX lock**, is a lock on a single row of a table. A transaction acquires a row lock for each row modified by one of the following statements: `INSERT`, `UPDATE`, `DELETE`, `MERGE`, and `SELECT` ... `FOR UPDATE`. The row lock exists until the transaction commits or rolls back.

When a transaction obtains a row lock for a row, the transaction also acquires a table lock for the table in which the row resides. The table lock prevents conflicting DDL operations that would override data changes in a current transaction.

Table Locks (TM)

A transaction automatically acquires a table lock (**TM lock**) when a table is modified with the following statements: `INSERT`, `UPDATE`, `DELETE`, `MERGE`, and `SELECT` ... `FOR UPDATE`. These DML operations require table locks to reserve DML access to the table on behalf of a transaction and to prevent DDL operations that would conflict with the transaction. You can explicitly obtain a table lock using the `LOCK TABLE` statement, as described in "Manual Data Locking".

A table lock can be held in any of the following modes:

- A **row share lock (RS)**, also called a **subshare table lock (SS)**, indicates that the transaction holding the lock on the table has locked rows in the table and intends to update them. An SS lock is the least restrictive mode of table lock, offering the highest degree of concurrency for a table.
- A **row exclusive lock (RX)**, also called a **subexclusive table lock (SX)**, indicates that the transaction holding the lock has updated table rows or issued `SELECT` ... `FOR UPDATE`. An SX lock allows other transactions to query, insert, update, delete, or lock rows concurrently in the same table. Therefore, SX locks allow multiple transactions to obtain simultaneous SX and SS locks for the same table.
- A **share table lock (S)** held by one transaction allows other transactions to query the table (without using `SELECT` ... `FOR UPDATE`) but allows updates only if a single transaction holds the share table lock. Multiple transactions may hold a share table lock concurrently, so holding this lock is not sufficient to ensure that a transaction can modify the table.
- A **share row exclusive table lock (SRX)**, also called a **share-subexclusive table lock (SSX)**, is more restrictive than a share table lock. Only one transaction at a time can acquire an SSX lock on a given table. An SSX lock held by a transaction allows other transactions to query the table (except for `SELECT` ... `FOR UPDATE`) but not to update the table.

- An **exclusive table lock (X)** is the most restrictive mode of table lock, allowing the transaction that holds the lock exclusive write access to the table. Only one transaction can obtain an X lock for a table.

See Also:

"Manual Data Locking"

Locks in DML Operations

Oracle Database automatically obtains row-level and table-level locks on behalf of DML operations. The type of operation determines the locking behavior. Table B-1 summarizes the information in this section.

Note:

The implicit SX locks shown for the DML statements in Table B-1 can sometimes be exclusive (X) locks for a short time owing to side effects from constraints.

Table B-1 Summary of Locks Obtained by DML Statements

SQL Statement	Row Locks	Table Lock Mode	RS	RX	S	SRX	X
`SELECT` ... `FROM` *`table`*`...`	—	none	Y	Y	Y	Y	Y
`INSERT INTO` *`table`* ...	Yes	SX	Y	Y	N	N	N
`UPDATE` *`table`* ...	Yes	SX	Y[1]	Y[1]	N	N	N
`MERGE INTO` *`table`* ...	Yes	SX	Y	Y	N	N	N
`DELETE FROM` *`table`* ...	Yes	SX	Y[1]	Y[1]	N	N	N
`SELECT` ... `FROM` *`table`* `FOR UPDATE OF` ...	Yes	SX	Y[1]	Y[1]	N	N	N
`LOCK TABLE` *`table`* `IN` ...	—						
`ROW SHARE MODE`		SS	Y	Y	Y	Y	N
`ROW EXCLUSIVE MODE`		SX	Y	Y	N	N	N
`SHARE MODE`		S	Y	N	Y	N	N
`SHARE ROW EXCLUSIVE MODE`		SSX	Y	N	N	N	N
`EXCLUSIVE MODE`		X	N	N	N	N	N

1 Yes, if no conflicting row locks are held by another transaction. Otherwise, waits occur.

Locks When Rows Are Queried

A query can be explicit, as in the `SELECT` statement, or implicit, as in most `INSERT`, `MERGE`, `UPDATE`, and `DELETE` statements. The only DML statement that does not necessarily include a query component is an `INSERT` statement with a `VALUES` clause. Because queries only read data, they are the SQL statements least likely to interfere with other SQL statements.

The following characteristics apply to a query *without* the `FOR UPDATE` clause:

- The query acquires no data locks. Therefore, other transactions can query and update a table being queried, including the specific rows being queried. Because queries without the `FOR UPDATE` clause do not acquire any data locks to block other operations, such queries are often referred to as **nonblocking queries**.
- The query does not have to wait for any data locks to be released. Therefore, the query can always proceed. An exception to this rule is that queries may have to wait for data locks in some very specific cases of pending distributed transactions.

Locks When Rows Are Modified

Some databases use a lock manager to maintain a list of locks in memory. Oracle Database, in contrast, stores lock information in the data block that contains the locked row. Each row lock affects only a single row.

Oracle Database uses a queuing mechanism for acquisition of row locks. If a transaction requires a row lock, and if the row is not already locked, then the transaction acquires a lock in the row's data block. The transaction itself has an entry in the **interested transaction list (ITL)** section of the block header. Each row modified by this transaction points to a copy of the transaction ID stored in the ITL. Thus, 100 rows in the same block modified by a single transaction require 100 row locks, but all 100 rows reference a single transaction ID.

When a transaction ends, the transaction ID remains in the ITL section of the data block header. If a new transaction wants to modify a row, then it uses the transaction ID to determine whether the lock is active. If the lock is active, then the session of the new transaction asks to be notified when the lock is released; otherwise, the new transaction acquires the lock.

The characteristics of `INSERT`, `UPDATE`, `DELETE`, and `SELECT` ... `FOR UPDATE` statements are as follows:

- A transaction containing a DML statement acquires exclusive row locks on the rows modified by the statement. Therefore, other transactions cannot update or delete the locked rows until the locking transaction either commits or rolls back.
- In addition to these row locks, a transaction containing a DML statement that modifies data also requires at least a subexclusive table lock (SX) on the table that contains the affected rows. If the transaction already holds an S, SRX, or X table lock for the table, which are more restrictive than an SX lock, then the SX lock is not needed and is not acquired. If the containing transaction already holds only an SS lock, however, then Oracle Database automatically converts the SS lock to an SX lock.
- A transaction that contains a DML statement does not require row locks on any rows selected by a subquery or an implicit query.

  In the following sample `UPDATE` statement, the `SELECT` statement in parentheses is a subquery, whereas the `WHERE a > 5` clause is an implicit query:

  ```
  UPDATE t SET x = ( SELECT y FROM t2 WHERE t2.z = t.z ) WHERE a > 5;
  ```

  A subquery or implicit query inside a DML statement is guaranteed to be consistent as of the start of the query and does not see the effects of the DML statement of which it forms a part.
- A query in a transaction can see the changes made by previous DML statements in the same transaction, but not the uncommitted changes of other transactions.

See Also:

Oracle Database Concepts for information on locks in foreign keys

Automatic Locks in DDL Operations

A **data dictionary (DDL) lock** protects the definition of a schema object while it is acted upon or referred to by an ongoing DDL operation. For example, when a user creates a procedure, Oracle Database automatically acquires DDL locks for all schema objects referenced in the procedure definition. The DDL locks prevent these objects from being altered or dropped before procedure compilation is complete.

Oracle Database acquires a DDL lock automatically on behalf of any DDL transaction requiring it. Users cannot explicitly request DDL locks. Only individual schema objects that are modified or referenced are locked during DDL operations. The whole data dictionary is never locked.

DDL operations also acquire DML locks on the schema object to be modified.

Exclusive DDL Locks

An **exclusive DDL lock** prevents other session from obtaining a DDL or DML lock.

Most DDL operations require exclusive DDL locks to prevent destructive interference with other DDL operations that might modify or reference the same schema object. For example, a `DROP TABLE` operation is not allowed to drop a table while an `ALTER TABLE` operation is adding a column to it, and vice versa. However, a query against the table is not blocked.

Exclusive DDL locks last for the duration of DDL statement execution and automatic commit. During the acquisition of an exclusive DDL lock, if another DDL lock is already held on the schema object by another operation, then the acquisition waits until the older DDL lock is released and then proceeds.

Share DDL Locks

A **share DDL lock** for a resource prevents destructive interference with conflicting DDL operations, but allows data concurrency for similar DDL operations.

For example, when a `CREATE PROCEDURE` statement is run, the containing transaction acquires share DDL locks for all referenced tables. Other transactions can concurrently create procedures that reference the same tables and acquire concurrent share DDL locks on the same tables, but no transaction can acquire an exclusive DDL lock on any referenced table.

A share DDL lock lasts for the duration of DDL statement execution and automatic commit. Thus, a transaction holding a share DDL lock is guaranteed that the definition of the referenced schema object is constant for the duration of the transaction.

Breakable Parse Locks

A **parse lock** is held by a SQL statement or PL/SQL program unit for each schema object that it references. Parse locks are acquired so that the associated shared SQL area can be invalidated if a referenced object is altered or dropped. A parse lock is called a **breakable parse lock** because it does not disallow any DDL operation and can be broken to allow conflicting DDL operations.

A **parse lock** is acquired in the shared pool during the parse phase of SQL statement execution. The lock is held as long as the shared SQL area for that statement remains in the shared pool.

Manual Data Locking

Oracle Database always performs locking automatically to ensure data concurrency, data integrity, and statement-level read consistency. However, you can override the Oracle default locking mechanisms. This can be useful in situations such as the following:

- When your application requires consistent data for the duration of the transaction, not reflecting changes by other transactions, you can achieve transaction-level read consistency by using explicit locking, read-only transactions, serializable transactions, or by overriding default locking.
- When your application requires that a transaction have exclusive access to a resource so that the transaction does not have to wait for other transactions to complete, you can explicitly lock the data for the duration of the transaction.

You can override automatic locking at two levels:

- **Transaction**. You can override transaction-level locking with the following SQL statements:
  - `SET TRANSACTION ISOLATION LEVEL`
  - `LOCK TABLE`
  - `SELECT` ... `FOR UPDATE`

  Locks acquired by these statements are released after the transaction commits or rolls back.
- **Session**. A session can set the required transaction isolate level with an `ALTER SESSION SET ISOLATION LEVEL` statement.

Note:

When overriding Oracle default locking, the database administrator or application developer should ensure that data integrity is guaranteed, data concurrency is acceptable, and deadlocks are not possible or, if possible, are appropriately handled. For more information on these criteria, see *Oracle Database Concepts*.