13

Managing Data Concurrency
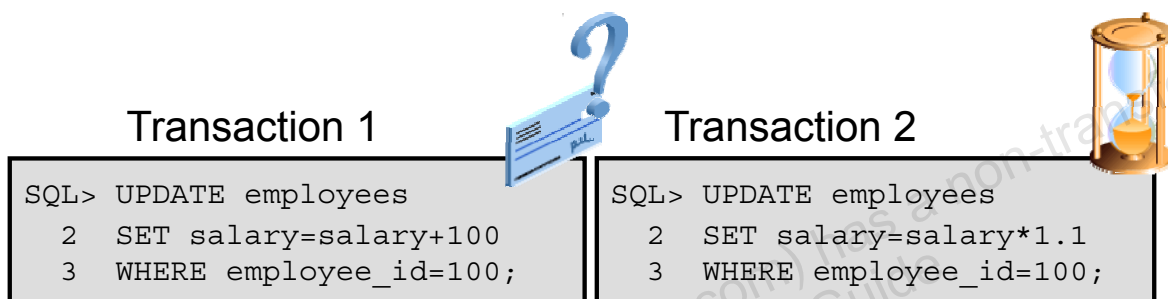
ORACLE

Objectives

After completing this lesson, you should be able to:

- Describe the locking mechanism and how Oracle manages data concurrency
- Monitor and resolve locking conflicts

Oracle Database 12c: Admin, Install and Upgrade Accelerated   13 - 2

Locks

- Prevent multiple sessions from changing the same data at the same time
- Are automatically obtained at the lowest possible level for a given statement
- Do not escalate

Transaction 1

```
SQL> UPDATE employees
  2  SET salary=salary+100
  3  WHERE employee_id=100;
```

Transaction 2

```
SQL> UPDATE employees
  2  SET salary=salary*1.1
  3  WHERE employee_id=100;
```

Before the database server allows a session to modify data, the session must first lock the data that is being modified. A lock gives the session exclusive control over the data so that no other transaction can modify the locked data until the lock is released.

Transactions can lock individual rows of data, multiple rows, or even entire tables. Oracle Database supports both manual and automatic locking. For automatically acquired locks, the lowest possible level of locking is chosen to minimize potential conflicts with other transactions.
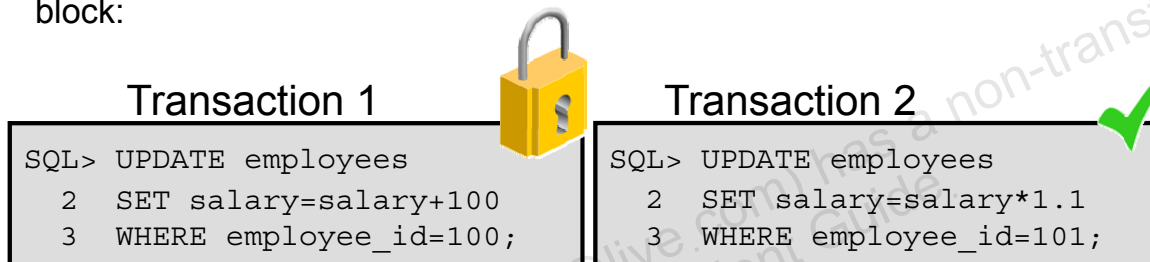
Note: There are many types of locks used by the Oracle server to maintain internal consistency. In this course, you will be concentrating only on locking used to protect rows and tables.

Locking Mechanism

- High level of data concurrency:
    - Row-level locks for inserts, updates, and deletes
    - No locks required for queries
- Automatic queue management
- Locks held until the transaction ends (with a commit or rollback operation)

Example
Assume that the rows for EMPLOYEE_ID 100 and 101 reside in the same block:

Transaction 1	Transaction 2

`SQL> UPDATE employees`
`  2   SET salary=salary+100`
`  3   WHERE employee_id=100;`
	`SQL> UPDATE employees`
	`  2   SET salary=salary*1.1`
	`  3   WHERE employee_id=101;`

The locking mechanism is designed to provide the maximum possible degree of data concurrency within the database. Transactions that modify data acquire row-level locks rather than block-level or table-level locks. Modifications to objects (such as table moves) obtain object-level locks rather than whole database or schema locks.

Data queries do not require a lock, and a query succeeds even if someone has locked the data (always showing the original, prelock value reconstructed from undo information).

When multiple transactions need to lock the same resource, the first transaction to request the lock obtains it. Other transactions wait until the first transaction completes. The queue mechanism is automatic and requires no administrator interaction.

All locks are released as transactions are completed (that is, when a COMMIT or ROLLBACK is issued). In the case of a failed transaction, the same background process that automatically rolls back any changes from the failed transaction releases all locks held by that transaction.

Data Concurrency

Time:	Transaction 1	UPDATE hr.employees SET salary=salary+100 WHERE employee_id=100;
	Transaction 2	UPDATE hr.employees SET salary=salary+100 WHERE employee_id=101;
09:00:00	Transaction 3	UPDATE hr.employees SET salary=salary+100 WHERE employee_id=102;
	…	…
	Transaction *x*	UPDATE hr.employees SET salary=salary+100 WHERE employee_id=*xxx*;

The lock mechanism defaults to a fine-grained, row-level locking mode. Different transactions can be updating different rows within the same table without interfering with one another.

Although the default model is to lock at the row level, Oracle Database supports manual locking at higher levels if needed:

```
SQL> LOCK TABLE employees IN EXCLUSIVE MODE;
Table(s) Locked.
```

With the preceding statement, any other transaction that tries to update a row in the locked table must wait until the transaction that issued the lock request completes. EXCLUSIVE is the strictest lock mode. The following are the other lock modes:

- **ROW SHARE:** Permits concurrent access to the locked table but prohibits sessions from locking the entire table for exclusive access

- **ROW EXCLUSIVE:** Is the same as ROW SHARE, but also prohibits locking in SHARE mode. The ROW EXCLUSIVE locks are automatically obtained when updating, inserting, or deleting data. ROW EXCLUSIVE locks allow multiple readers and one writer.

- **SHARE:** Permits concurrent queries but prohibits updates to the locked table. A SHARE lock is required (and automatically requested) to create an index on a table. However, online index creation requires a ROW SHARE lock that is used when building the index.

Share locks allow multiple readers and no writers. Share locks are also used transparently when deleting or updating rows in a parent table that has a child table with foreign key constraints on the parent.

- **SHARE ROW EXCLUSIVE:** Is used to query a whole table and to allow others to query rows in the table, but prohibits others from locking the table in SHARE mode or updating rows
- **EXCLUSIVE:** Permits queries on the locked table but prohibits any other activity on it. An EXCLUSIVE lock is required to drop a table.

Like any request for a lock, manual lock statements wait until all sessions that either already have locks or have previously requested locks release their locks. The LOCK command accepts a special argument that controls the waiting behavior, NOWAIT.

NOWAIT returns control to you immediately if the specified table is already locked by another session:

```
SQL> LOCK TABLE hr.employees IN SHARE MODE NOWAIT;
LOCK TABLE hr.employees IN SHARE MODE NOWAIT
                 *
ERROR at line 1:
ORA-00054: resource busy and acquire with NOWAIT specified
```

It is usually not necessary to manually lock objects. The automatic locking mechanism provides the data concurrency needed for most applications. Oracle recommends that you avoid using manual locks, especially when developing applications. Severe performance issues often occur from unnecessarily high locking levels.

DML Locks

Transaction 1	Transaction 2
| ```
SQL> UPDATE employees
  2   SET salary=salary*1.1
  3   WHERE employee_id= 107;
1 row updated.
``` | ```
SQL> UPDATE employees
  2   SET salary=salary*1.1
  3   WHERE employee_id= 106;
1 row updated.
``` |

Each DML transaction must acquire *two* locks:

- `EXCLUSIVE` row lock on the row or rows being updated
- Table lock (`TM`) in `ROW EXCLUSIVE (RX)` mode on the table containing the rows

ORACLE

Each DML transaction obtains two locks:

- An `EXCLUSIVE` row lock on the row or rows being updated
- A table lock (`TM`) in `ROW EXCLUSIVE (RX)` mode on the table being updated. This prevents another session from locking the whole table (possibly to drop or truncate it) while the change is being made. This mode is also called a subexclusive table lock (`SX`).

A `ROW EXCLUSIVE` lock on the table prevents DDL commands from changing the dictionary metadata in the middle of an uncommitted transaction. This preserves dictionary integrity and read consistency across the life of a transaction.

Enqueue Mechanism

The enqueue mechanism keeps track of:
- Sessions waiting for locks
- Requested lock mode
- Order in which sessions requested the lock

Requests for locks are automatically queued. As soon as the transaction holding a lock is completed, the next session in line receives the lock.

The enqueue mechanism tracks the order in which locks are requested and the requested lock mode.

Sessions that already hold a lock can request that the lock be *converted* without having to go to the end of the queue. For example, suppose a session holds a `SHARE` lock on a table. The session can request that the `SHARE` lock be converted to an `EXCLUSIVE` lock. If no other transaction already has an `EXCLUSIVE` or `SHARE` lock on the table, the session holding the `SHARE` lock is granted an `EXCLUSIVE` lock without having to wait in the queue again.

Note: There are two categories of waiters for enqueues: those waiting without shared ownership and those with shared ownership that do not choose to escalate the lock level. Waiters in the second category are known as *converters*, which are always given priority over normal waiters even if they have been waiting less time.

Lock Conflicts

Transaction 1	Time	Transaction 2
UPDATE employees SET salary=salary+100 WHERE employee_id=100; 1 row updated.	9:00:00	UPDATE employees SET salary=salary+100 WHERE employee_id=101; 1 row updated.
UPDATE employees SET COMMISION_PCT=2 WHERE employee_id=101; Session waits enqueued due to lock conflict.	9:00:05	SELECT sum(salary) FROM employees; SUM(SALARY) ----------- 692634
Session still waiting!	16:30:00	Many selects, inserts, updates, and deletes during the last 7.5 hours, but no commits or rollbacks!
1 row updated. Session continues.	16:30:01	commit;

ORACLE

Lock conflicts occur often but are usually resolved through time and the enqueue mechanism. In certain rare cases, a lock conflict may require administrator intervention. In the example in the slide, transaction 2 obtains a lock on a single row at 9:00:00 and neglects to commit, leaving the lock in place. Transaction 1 attempts to update the entire table, requiring a lock on all rows, at 9:00:05. Transaction 1 is blocked by transaction 2 until transaction 2 commits at 16:30:01.

A user attempting to perform transaction 1 would almost certainly contact the administrator for help in this case, and the DBA would have to detect and resolve the conflict.

Possible Causes of Lock Conflicts

- Uncommitted changes
- Long-running transactions
- Unnecessarily high locking levels

ORACLE

The most common cause of lock conflicts is an uncommitted change, but there are a few other possible causes:

- **Long-running transactions:** Many applications use batch processing to perform bulk updates. These batch jobs are usually scheduled for times of low or no user activity, but in some cases, they may not have finished or may take too long to run during the low activity period. Lock conflicts are common when transaction and batch processing are being performed simultaneously.
- **Unnecessarily high locking levels:** Not all databases support row-level locking (Oracle added support for row-level locks in 1988 with release 6). Some databases still lock at the page or table level. Developers writing applications that are intended to run on many different databases often write their applications with artificially high locking levels so that Oracle Database behaves similarly to these less capable database systems. Developers who are new to Oracle also sometimes unnecessarily code in higher locking levels than are required by Oracle Database.

Detecting Lock Conflicts

- Select Blocking Sessions in the Performance menu.

- Click the Session ID link to view information about the locking session.

Use the Blocking Sessions page in Enterprise Manager Cloud Control to locate lock conflicts. Conflicting lock requests are shown in hierarchical layout, showing the session holding the lock at the top and, below that, any sessions that are enqueued for the lock.

For each session involved in the conflict, you are given the username, session ID, and number of seconds for which the session has been waiting. Drill down to the SQL ID to see actual SQL statements that are currently being executed or requested by the session.

The Automatic Database Diagnostic Monitor (ADDM) also automatically detects lock conflicts and can advise you on inefficient locking trends.

Resolving Lock Conflicts

To resolve a lock conflict:
- Have the session holding the lock commit or roll back
- Terminate the session holding the lock (in an emergency)

To resolve a lock conflict, the session holding the lock must release it. The best way to have the session release the lock is to contact the user and ask that the transaction be completed.

In an emergency, it is possible for the administrator to terminate the session holding the lock. Remember that when a session is killed, all work within the current transaction is lost (rolled back). A user whose session is killed must log in again and redo all work since the killed session's last commit.

Users whose sessions have been killed receive the following error the next time they try to issue a SQL statement:

```
ORA-03135: connection lost contact
```

Note: PMON can automatically kill sessions due to idle timeout. This can be implemented by using profiles or Resource Manager.

Resolving Lock Conflicts by Using SQL

SQL statements can be used to determine the blocking session and kill it.

1
```
SQL> SELECT sid, serial#, username
  2  FROM v$session WHERE sid IN
  3  (SELECT blocking_session FROM v$session);
```

Result:

```
        SID    SERIAL# USERNAME
-------- ---------- --------
        144       8982 HR
```

2
```
SQL> ALTER SYSTEM KILL SESSION '144,8982' immediate;
```

Session manipulation can also be done by issuing SQL statements. The V$SESSION view contains details of all connected sessions. The value in BLOCKING_SESSION is the session ID of the session that is blocking. If you query for SID and SERIAL# (where SID matches a blocking session ID), you have the information needed to perform the KILL SESSION operation.

Note: Database Resource Manager can be used to automatically log out sessions that block others and are idle.

Deadlocks

Transaction 1		Transaction 2
UPDATE employees SET salary = salary x 1.1 WHERE employee_id = 1000;	9:00	UPDATE employees SET manager = 1342 WHERE employee_id = 2000;
UPDATE employees SET salary = salary x 1.1 WHERE employee_id = 2000;	9:15	UPDATE employees SET manager = 1342 WHERE employee_id = 1000;
ORA-00060: Deadlock detected while waiting for resource	9:16	

ORACLE

A deadlock is a special example of a lock conflict. Deadlocks arise when two or more sessions wait for data that has been locked by the other. Because each is waiting for the other, neither can complete their transaction to resolve the conflict.

Oracle Database automatically detects deadlocks and terminates the statement with an error. The proper response to that error is either commit or roll back, which releases any other locks in that session so that the other session can continue its transaction.

In the example in the slide, transaction 1 must either commit or roll back in response to the deadlock detected error. If it commits, it must resubmit the second update to complete its transaction. If it performs a rollback, it must resubmit both statements to complete its transaction.

Quiz

The lock mechanism defaults to a fine-grained, row-level locking mode.

a. True

b. False

Answer: a

Quiz

When a deadlock occurs, Oracle database automatically:

a. Waits 300 seconds before terminating both sessions

b. Terminates one statement with an error in one session

c. Terminates the statements with an error in both sessions

d. Takes no action by default and leaves it to the DBA

ORACLE

Answer: b

Summary

In this lesson, you should have learned how to:

- Describe the locking mechanism and how Oracle manages data concurrency
- Monitor and resolve locking conflicts

Oracle Database 12*c*: Admin, Install and Upgrade Accelerated   13 - 17

Practice 13

13-1: Resolving Lock Conflicts

Oracle Database 12*c*: Admin, Install and Upgrade Accelerated   13 - 18