**INT207 Lab 7**

***Concurrency Control***

***How Oracle Locks Data***

*Locks* are mechanisms that prevent destructive interaction between transactions accessing the same **resource**—either user objects such as tables and rows or system objects not visible to users, such as shared data structures in memory and data dictionary rows.

In all cases, Oracle ***automatically*** obtains necessary locks when executing SQL statements, so users need not be concerned with such details. Oracle automatically uses the lowest applicable level of restrictiveness to provide the highest degree of data concurrency yet also provide fail-safe data integrity.

* Row-level lock for DML operation: Insert, Update, Delete
* No locks required for query

However, Oracle also allows the user to lock data manually.

***Transactions and Data Concurrency***

Oracle provides data concurrency and integrity between transactions using its locking mechanisms. Because the locking mechanisms of Oracle are tied closely to transaction control, application designers need only define transactions properly, and Oracle automatically manages locking.

Keep in mind that Oracle locking is fully automatic and requires no user action. Implicit locking occurs for all SQL statements so that database users never need to lock any resource explicitly. Oracle's default locking mechanisms lock data at the lowest level of restrictiveness to guarantee data integrity while allowing the highest degree of data concurrency.

***DML Locks***

Each DML transaction obtains two locks:

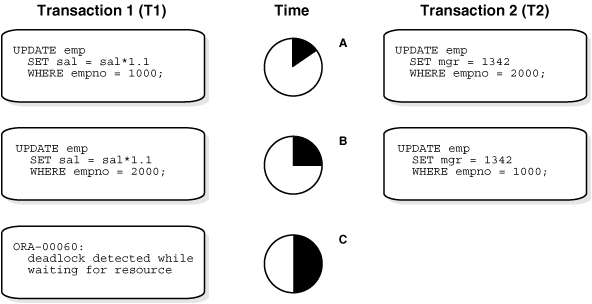
* A row-exclusive lock on the row or rows being updated. There will be one row-exclusive lock regardless of the number of rows changed.
* A shared table-level lock on the table being updated. This is to prevent another session from locking the whole table (possibly to drop or truncate it) while the change is being made.

Locks held ***until transaction ends*** (COMMIT or ROLLBACK operation)

#### Deadlocks

A **deadlock** can occur when two or more users are waiting for data locked *by each other*. Deadlocks prevent some transactions from continuing to work. The following [figure](http://download.oracle.com/docs/cd/B19306_01/server.102/b14220/consist.htm#i6885) is a hypothetical illustration of two transactions in a deadlock.

Oracle *automatically detects deadlock* situations and resolves them by rolling back one of the statements involved in the deadlock, thereby releasing one set of the conflicting row locks. A corresponding message also is returned to the transaction that undergoes statement-level rollback. The statement rolled back is the one belonging to the transaction that detects the deadlock. Usually, the signalled transaction should be rolled back explicitly, but it can retry the rolled-back statement after waiting.



#### Explicit (Manual) Data Locking

Oracle always performs locking automatically to ensure data concurrency, data integrity, and statement-level read consistency. However, you can override the Oracle default locking mechanisms. At the transaction level, transactions that include the following SQL statements override Oracle's default locking:

* The SET TRANSACTION ISOLATION LEVEL statement
* The LOCK TABLE statement (which locks either a table or, when used with views, the underlying base tables)
* The SELECT ... FOR UPDATE statement

***Locks acquired by these statements are released after the transaction commits or rolls back.***

***Savepoint In Transactions***

|  |  |
| --- | --- |
| Types | Statements |
| Transaction Control | COMMIT, ROLLBACK, SAVEPOINT |

You can declare intermediate markers called **savepoints** within the context of a transaction. Savepoints divide a long transaction into smaller parts.

**Syntax:**

|  |
| --- |
| SAVEPOINT *name;*  ROLLBACK TO [SAVEPOINT] *name;* |

Using savepoints, you can arbitrarily mark your work at any point within a long transaction. You then have the option later of rolling back work performed before the current point in the transaction but after a declared savepoint within the transaction. For example, you can use savepoints throughout a long complex series of updates, so if you make an error, you do not need to resubmit every statement.

When a transaction is rolled back to a savepoint, the following occurs:

1. Oracle rolls back only the statements run after the savepoint.
2. Oracle preserves the specified savepoint, but all savepoints that were established after the specified one are lost.
3. Oracle releases all table and row locks acquired since that savepoint but retains all data locks acquired previous to the savepoint.

The transaction remains active and can be continued.

| No | Transaction A (DBxxxx) | Transaction B (APPxxxx) |
| --- | --- | --- |
| 1 | SELECT SUM(salary)FROM STAFF; ***Ans.*** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | SELECT SUM(salary)FROM DBxxxx.STAFF; ***Ans.*** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
| 2 | UPDATE STAFFSET salary = 70000WHERE fname = 'David'; ***The result?*** |  |
| 3 | UPDATE STAFF  SET salary = 20000  WHERE fname = 'Susan';  ***The result?*** |  |
| 4 | SELECT SUM(salary)  FROM STAFF;  ***Ans.*** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | SELECT SUM(salary)FROM DBxxxx.STAFF; ***Ans.*** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
| 5 | Write SQL statement to mark an intermediate point ***called PointOne*** in the processing of the transaction.  ***Ans:*** |  |
| 6 | UPDATE STAFF  SET salary = 50000  WHERE fname = 'John';  ***The result?*** |  |
| 7 | SELECT SUM(salary)  FROM STAFF;  ***Ans.*** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | SELECT SUM(salary)FROM DBxxxx.STAFF; ***Ans.*** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
| 8 | Write SQL statement to mark an intermediate point ***called PointTwo*** in the processing of the transaction.  ***Ans:*** |  |
| 9 | UPDATE STAFF  SET salary = 110000  WHERE fname = 'Mary';  ***The result?*** |  |
| 10 | SELECT SUM(salary)  FROM STAFF;  ***Ans.*** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | SELECT SUM(salary)FROM DBxxxx.STAFF; ***Ans.*** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
| 11 | **Write SQL statement to discard the most recent UPDATE operation (NO. 9) without discarding the earlier INSERT or UPDATE operations.**  ***Ans:*** |  |
| 12 |  | UPDATE DBxxxx.STAFFSET branchno = 'B005'WHERE fname = 'Mary'; ***The result?*** |
| 13 | Commit; | Rollback; |
| 14 | SELECT SUM(salary) FROM STAFF; ***Ans.*** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | SELECT SUM(salary) FROM DBxxxx.STAFF; ***Ans.*** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |