Table of Contents

[1. Create Two Sessions 1](#_Toc58802838)

[2. Create Two Tables 1](#_Toc58802839)

[3. Insert Two Rows 1](#_Toc58802840)

[4. Update Deadlocker1 Table 1](#_Toc58802841)

[5. Update Deadlocker2 Table 1](#_Toc58802842)

[6. Check Update Command Being Executed By User System 2](#_Toc58802843)

[7. Update Deadlocker2 Table In First Session 2](#_Toc58802844)

[8. Deadlock Error Message 2](#_Toc58802845)

[9. Unlock Second Session 3](#_Toc58802846)

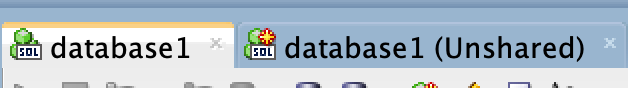
[10. Explain Who Deadlock Victim Was and Why 3](#_Toc58802847)

[11. Shared Locks 3](#_Toc58802848)

[12. Three-Session Deadlock 4](#_Toc58802849)

## 1. Create Two Sessions

Create two sessions connected as the same user to a transactional DBMS.

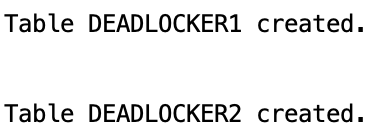


SQL Developer allows two sessions for one user within the same window. It just marks the other session as “Unshared”.

## 2. Create Two Tables

CREATE TABLE Deadlocker1 (x INTEGER);

CREATE TABLE Deadlocker2 (x INTEGER);



## 3. Insert Two Rows

Insert two rows in the tables with different data values

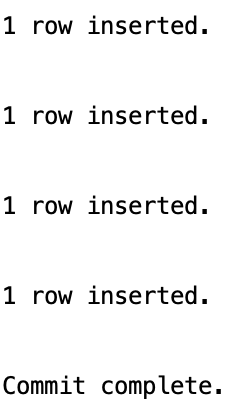
INSERT INTO Deadlocker1 VALUES(1);

INSERT INTO Deadlocker1 VALUES(2);

INSERT INTO Deadlocker2 VALUES(1);

INSERT INTO Deadlocker2 VALUES(2);

COMMIT YOUR TRANSACTION



## 4. Update Deadlocker1 Table

In the first session start a transaction by updating the first Deadlocker1 table. This procures an exclusive lock on the Dedlocker1 table, on the row with x=1.

UPDATE Deadlocker1 SET x=3 where x=1;



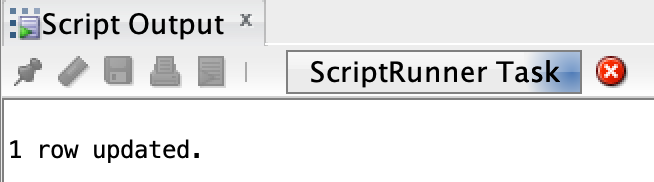
## 5. Update Deadlocker2 Table

In the second session start a new transaction by updating the Deadlocker2 table, updating the second row (where x=2), placing a lock on the second row of the Deadlocker2 table. Also update the Deadlocker1 table, trying to set the first column (which is now locked by the other transaction)

UPDATE Deadlocker2 SET x=3 where x=2;

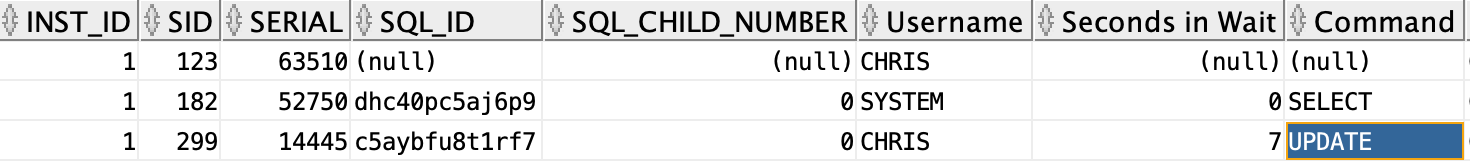
UPDATE Deadlocker1 SET x=4 where x=1;

* **Oracle Note:** In Oracle Developer you will see ScriptRunner Task window running once you run the above commands.



## 6. Check Update Command Being Executed By User System

At this point, the second session should be trying to execute the query. Let us take a look at the locks.

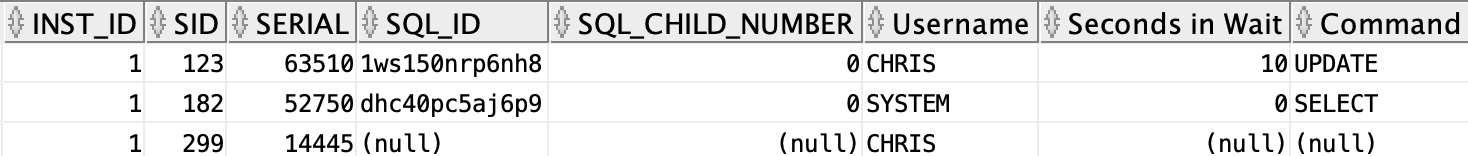


* In **Oracle SQL Developer:** Under Tools select “Monitor Sessions” and connect as user SYSTEM. A Session window will open. Look for the User who you logged in as (you can filter by the column) and note the UPDATE command running (under the Command Column).

## 7. Update Deadlocker2 Table In First Session

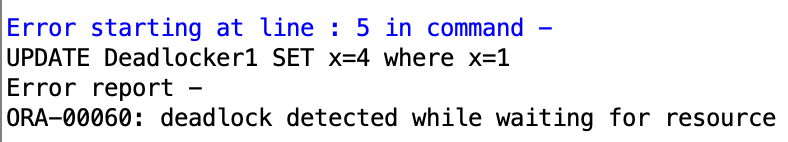
Now go back to the first session window, which is running the first transaction, and update the second Deadlocker2 table, updating the row where X=2.

UPDATE Deadlocker2 SET x=5 where x=2;



## 8. Deadlock Error Message

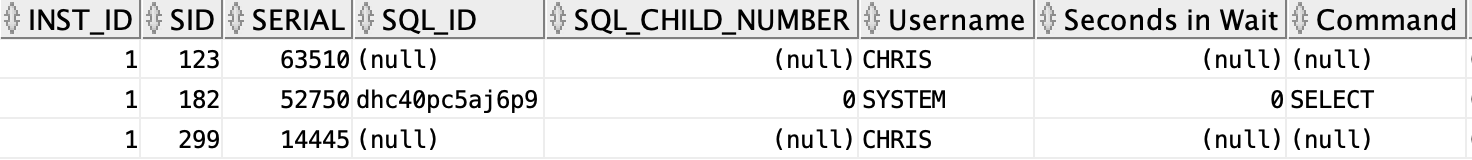
You will see an error message, which should tell you that the transaction was deadlocked for one of your transactions. Take a screenshot of this and attach it here.



## 9. Unlock Second Session

You can unlock the second session window by committing the transaction.





## 10. Explain Who Deadlock Victim Was and Why

[Optional] Explain which session was the deadlock victim and why this occurred. It may be helpful to create a little sequence table, and consider what is happening at each step, identifying when the deadlock occurs.

|  |  |  |
| --- | --- | --- |
| Session1 | Session2 | Explanation |
| UPDATE Deadlocker1 SET x=3 where x=1; |  | Locked Row |
|  | UPDATE Deadlocker2 SET x=3 where x=2; | Locked Row |
|  | UPDATE Deadlocker1 SET x=4 where x=1; | DeadLock Victim – Session1 has the row locked |

The deadlock victim was session 2 because session 1 already had the row locked.

## 11. Shared Locks

[Optional]: If you would like you can try the same thing with shared locks, by selecting the rows instead of updating them. You shouldn’t get a deadlock in a DBMS that supports shared locks. In a DBMS with only binary locks you would get a deadlock.

|  |  |  |
| --- | --- | --- |
| Session1 | Session2 | Explanation |
| SELECT \* FROM Deadlocker1 **WHERE x=3**; |  | Viewed Row |
|  | SELECT \* FROM Deadlocker2 **WHERE x=4**; | Viewed Row |
|  | SELECT \* FROM Deadlocker1 **WHERE x=3**; | Viewed Row |

## 12. Three-Session Deadlock

[Optional] For a more advanced variation you can create a three-session deadlock. Hint: This requires three rows in Deadlocker and multiple Deadlocker tables.

INSERT INTO Deadlocker1

VALUES (3);

INSERT INTO Deadlocker2

VALUES (5);

COMMIT;

INSERT INTO Deadlocker3

VALUES (15);

INSERT INTO Deadlocker3

VALUES (30);

INSERT INTO Deadlocker3

VALUES (45);

COMMIT;

Deadlocker1: 3, 2, 3

Deadlocker2: 1, 3, 5

Deadlocker3: 15, 30, 45

|  |  |  |  |
| --- | --- | --- | --- |
| Session1 | Session2 | Session3 | Explanation |
| UPDATE Deadlocker1 SET x=32 where x=2; |  |  | Lock Row |
|  | UPDATE Deadlocker2 SET x=0 where x=5; |  | Lock Row |
|  |  | UPDATE Deadlocker3 SET x=3 where x=15; | Lock Row |
| UPDATE Deadlocker3 SET x=21 where x=15; |  |  | Dead Lock |
|  | UPDATE Deadlocker1 SET x=100 where x=2; |  | Dead Lock |
|  |  | UPDATE Deadlocker2 SET x=20 where x=5; | DeadLock |

