

Using Cursors FOR UPDATE

Terminology

Directions: Identify the vocabulary word for each definition below:

1. _____ Declares that each row is locked as it is being fetched so other users can not modify the rows while the cursor is open.
2. _____ A keyword used to tell the Oracle server not to wait if the requested rows have already been locked by another user.

Try It/Solve It

In this Practice you will INSERT and later UPDATE rows in a new table: proposed_raises, which will store details of salary increases proposed for suitable employees. Create this table by executing the following SQL statement:

```
CREATE TABLE proposed_raises
(date_proposed    DATE,
date_approved    DATE,
employee_id      NUMBER(6),
department_id    NUMBER(4),
original_salary   NUMBER(8,2),
proposed_new_salary NUMBER(8,2));
```

1. Write a PL/SQL block which inserts a row into proposed_raises for each eligible employee. The eligible employees are those whose salary is below a chosen value. The salary value is passed as a parameter to the cursor. For each eligible employee, insert a row into proposed_raises with date_proposed = today's date, date_approved null, and proposed_new_salary 5% greater than the current salary. The cursor should LOCK the employees rows so that no-one can modify the employee data while we the cursor is open. Test your code using a chosen salary value of 5000.
2. SELECT from the proposed_raises table to see the results of your INSERT statements. There should be six rows. If you run your block in question 1 more than once, make sure the proposed_raises table is empty before each test.

```
SELECT * FROM proposed_raises;
```

```
[ DELETE FROM proposed_raises; ]
```

3. Before starting this question, ensure that there are six rows in `proposed_raises`. Now imagine that these proposed salary increases have been approved by company management.
 - A. Write and execute a PL/SQL block to read each row from the `proposed_raises` table. For each row, UPDATE the `date_approved` column with today's date. Use the `WHERE CURRENT OF...` syntax to UPDATE each row.
 - B. SELECT from the `proposed_raises` table to view the updated data.
 - C. Management has now decided that employees in department 50 cannot have a salary increase after all. Modify your code from question 3 to DELETE employees in department 50 from `proposed_raises`. This could be done by a simple DML statement (`DELETE FROM proposed_raises WHERE department_id = 50;`) but we want to do it using a FOR UPDATE cursor. Test your code, and view the `proposed_raises` table again to check that the rows have been deleted.
 - D. We are going to set up two sessions into the same schema. From one of the sessions we will manually update an employee row *NOT COMMITING*. From the other session we will try to update everyone's salary, again *NOT COMMITING*. You should see the difference between NOWAIT and WAIT when using FOR UPDATE.

IMPORTANT NOTE: in each of these sessions, do NOT leave the SQL Commands screen to visit another Application Express page (for example Object Browser or Home). If you leave SQL Commands, your updates will automatically be rolled back, releasing all locks being held.

In preparation, create a copy of the employees table by executing the following SQL statement. You should use the `upd_emps` table for the rest of this exercise.

```
CREATE TABLE upd_emps AS SELECT * FROM employees;
```

- 1) Open a second Application Express session in a new browser window and connect to your schema. Ensure that Autocommit is disabled in BOTH your sessions (uncheck the check box in the top left corner of the SQL Commands window).
- 2) In your first session, update `employee_id` 200 (Jennifer Whalen)'s first name to Jenny.
DO NOT COMMIT. You now have a lock on row 200 that will last indefinitely.
- 3) In your second session, write a PL/SQL block to give every employee in `upd_emps` a \$1 salary raise. Your cursor should be declared FOR UPDATE NOWAIT. Execute your code. What happens?
- 4) Still in your second session, modify your block to remove the NOWAIT attribute from the cursor declaration. Re-execute the block. What happens this time?

- 5) After waiting a minute or so, switch to your first session and COMMIT the update to Jennifer Smith's row. Then switch back to your second session. What happened?
- 6) Clean up by COMMITTING the updates in your second session.

Extension Exercise

1. For this question you will also need two separate Application Express sessions in two browser windows, with Autocommit turned off in both sessions. You will also need a copy of the departments table. Create this copy by executing the following SQL statement:

```
CREATE TABLE upd_depts AS SELECT * FROM departments;
```

- A. Modify your \$1 salary raise block from the previous question so that the cursor SELECTS from a join of upd_emps and upd_depts.
- B. Run the block in your second session. It should execute successfully, updating all the upd_emps rows but not committing.
- C. Switch to your first session and try to update an upd_depts row using the following statement. What happens and why ?


```
UPDATE upd_depts SET department_name = 'Accounting'  
WHERE department_id = 50;
```
- D. Release all the locks by committing in your second session and then in your first session.
- E. How would you prevent your cursor from locking upd_depts rows unnecessarily? Modify your block to avoid the unnecessary locking, and rerun the test you did in steps b and c. What happens this time ?
- F. Clean up by COMMITTING your updates in both sessions and re-enabling Autocommit.