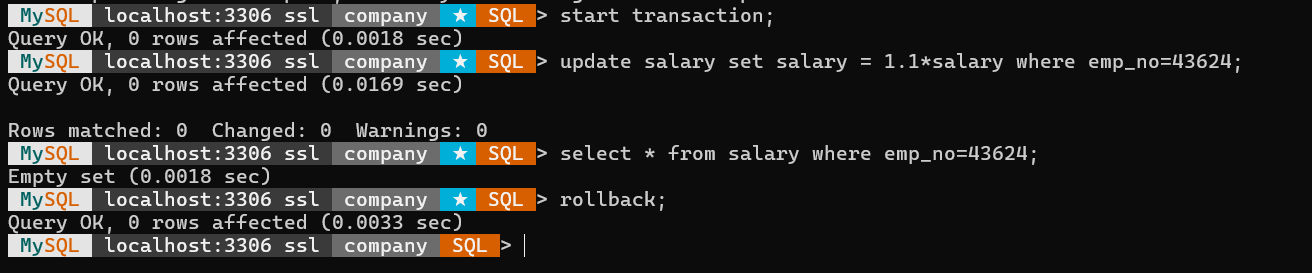
**CO527: Advanced Database Systems**

**Lab 03 - Transaction Processing**

Ranage R.D.P.R.

E/19/310



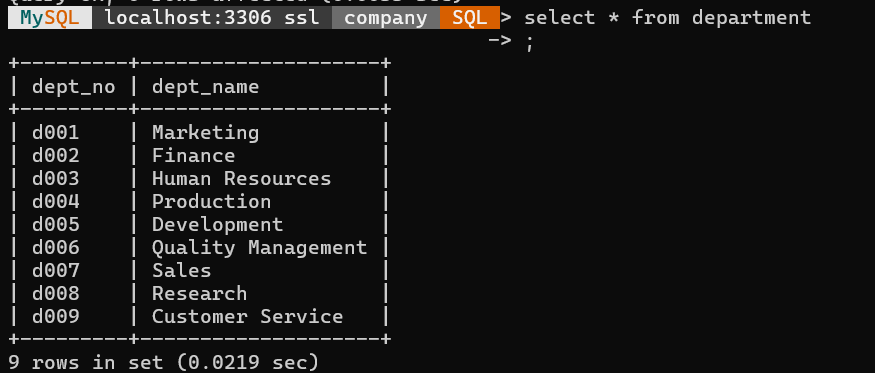
**Concurrent Accesses**

**I. Issue a select query to view the current status of the departments table in both sessions.**

Session 1



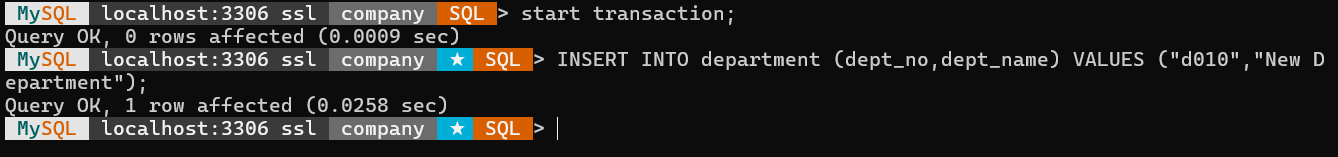
Session 2



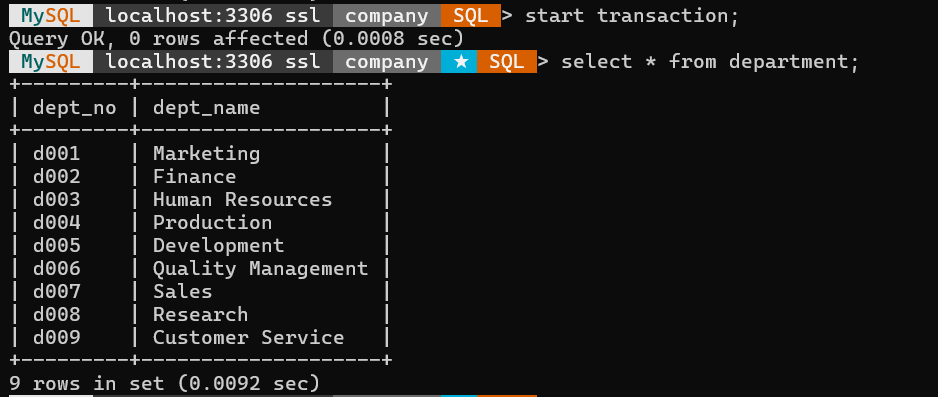
**II. Now, start transaction running start transaction in both sessions.**

**III. Insert a new row into the departments table from the 1st session and check if the changes are visible in the second session.**

Session 1



Session 2

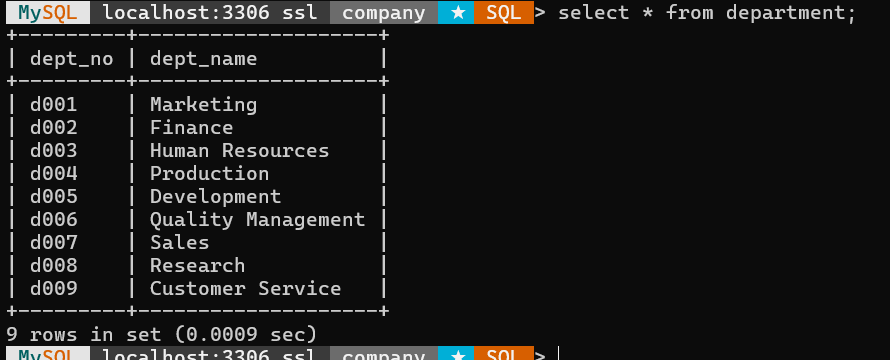


**IV. Commit changes in the 1st command window and check if you can see the updates done at 1st window in 2nd command window.**

Session 1



Session 2



**V. Explain your observations before and after running the commit in the 1st window.**

The output from session 02 remained unchanged.

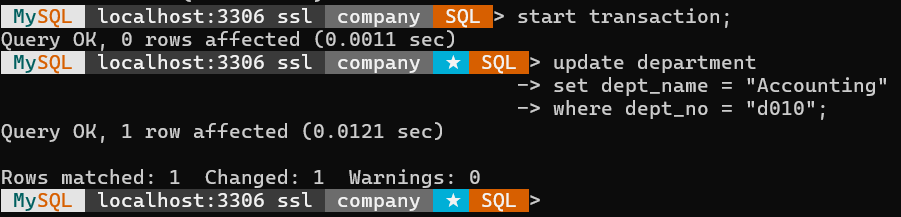
The session 02 modifications were not displayed before committing the session 01 changes as well as after committing the changes.

This was a result of session 2 still being in process. (The transaction hasn't been completed yet.) The changes were visible following session 02's commit.

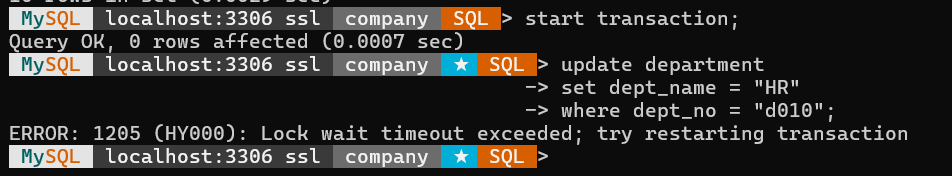
**Concurrent Updates**

**I. Try to do a concurrent update to the same row in departments table during two transactions.**

Session 1



Session 2



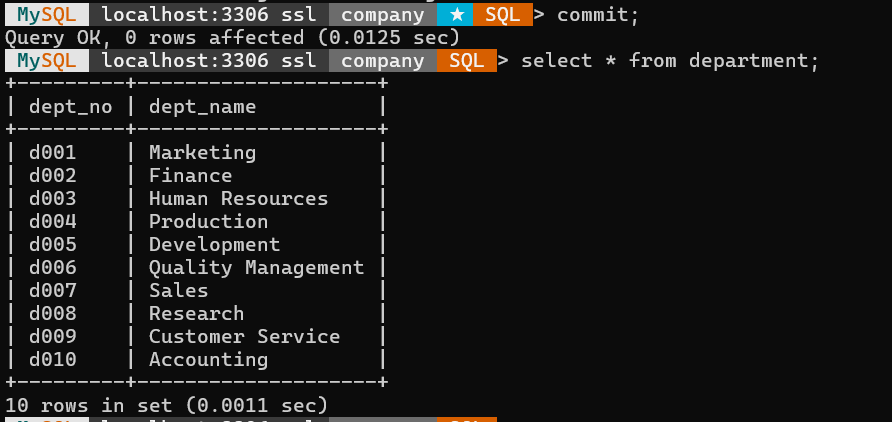
**II. Explain what happens before ending any of the transactions.**

Because a UPDATE statement locks a field, session 01 forced session 02 to wait in line while locking the field that needed to be changed.

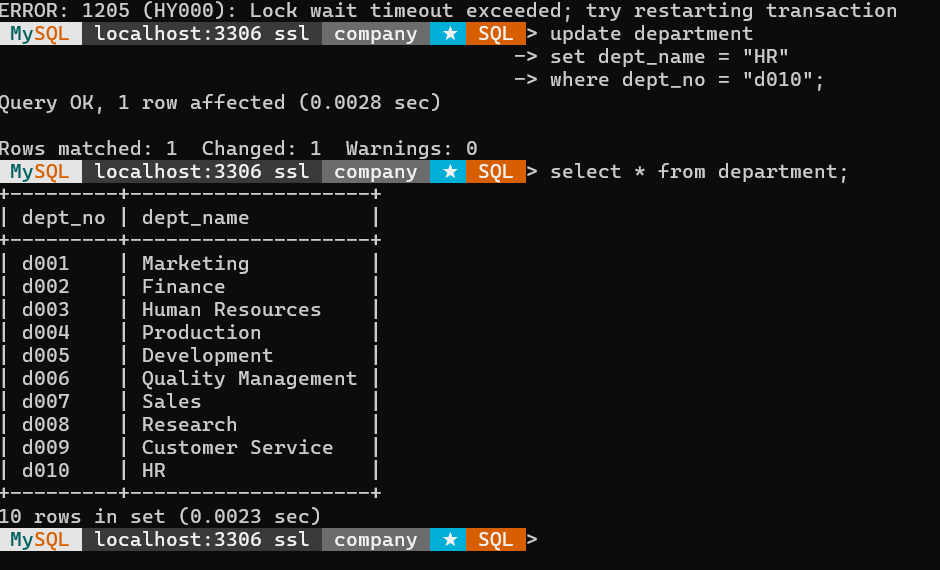
whenever a row in session 01 is updated. Once the commit is complete, this row is locked. Therefore, until session 01 commits, session 02 is not permitted to change the same row.

**III. What happens when you commit your changes in the 1st session?**

Session 1



Session 2



After committing the session 1, it is possible to update the same row in the session 2.

**What to Turn In**

**Use your imagination and words to write a scenario where using transactions is essential and then create the required tables and test how the transaction will effect your tables,**

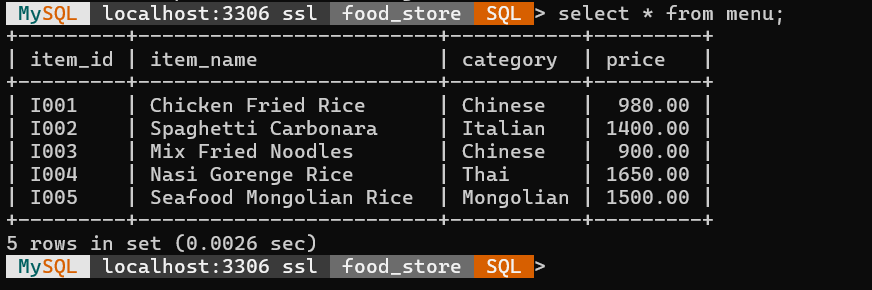
**1. during the transaction execution.**

**2. after rollback statement.**

**3. after the commit statement.**

**4. during 2 concurrent transactions, both of them update a record and both of them commit it.**

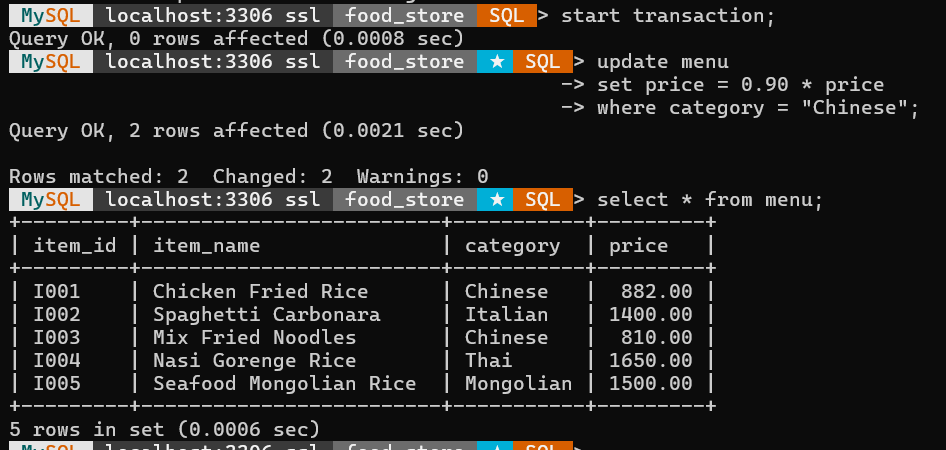
Simple database for a food store was created.



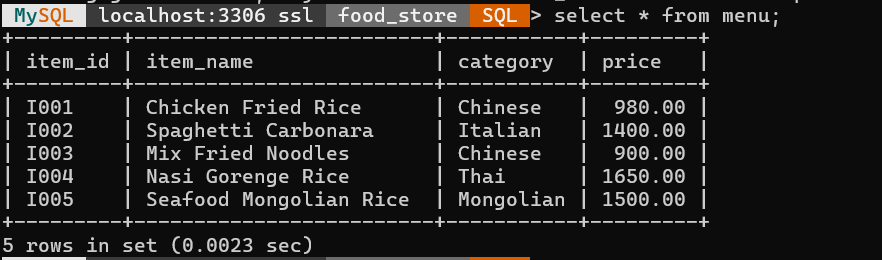
**1. Transaction**

First add 10% discount for all Chinese food and then rollback it

Session 1



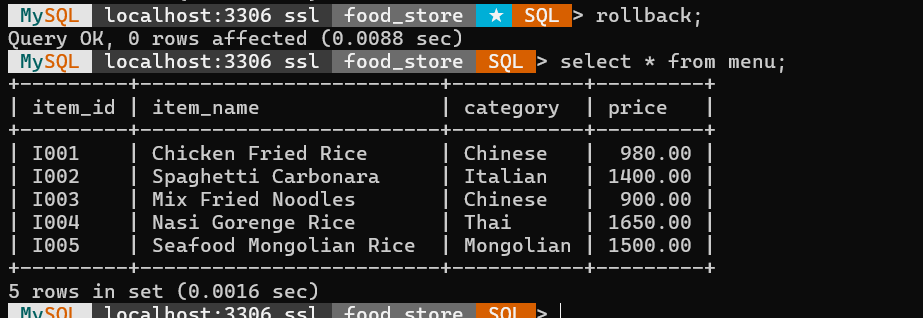
Session 2



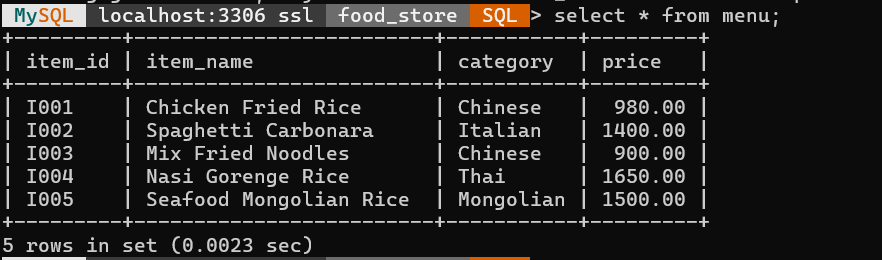
Here updates cannot be seen

**2. Rollback**

Session 1(After rolling back)



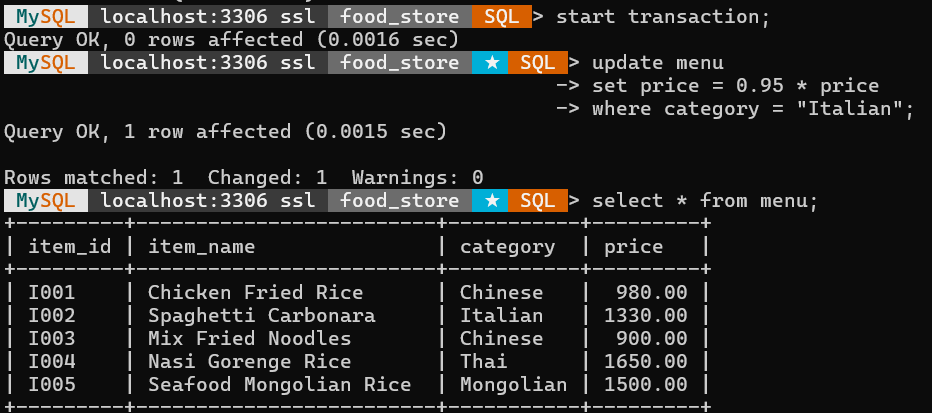
Session 2



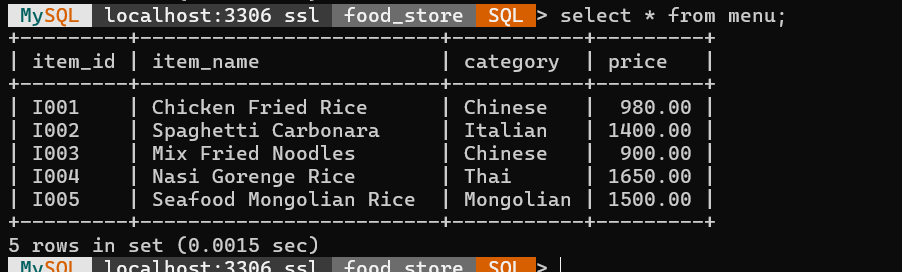
**3. Commit**

Add a 5% discount to Italian foods and check it from the 2nd session without commit

Session 1

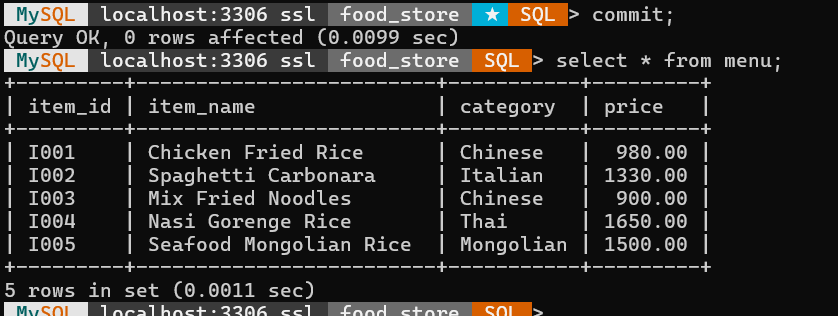


Session 2

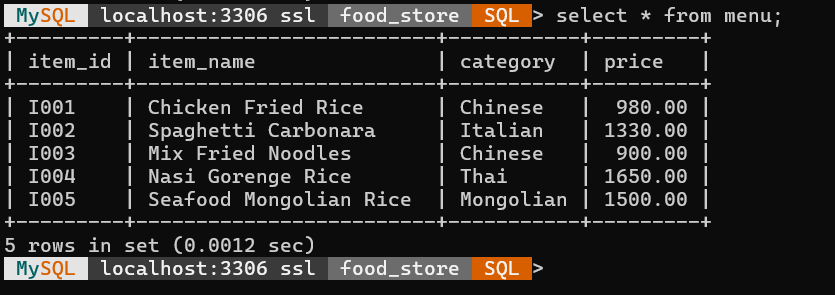


After committing,

Session 1



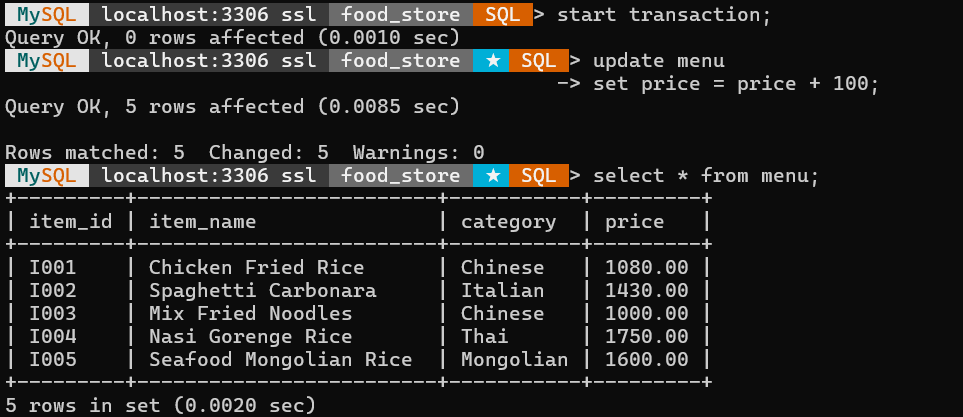
Session 2



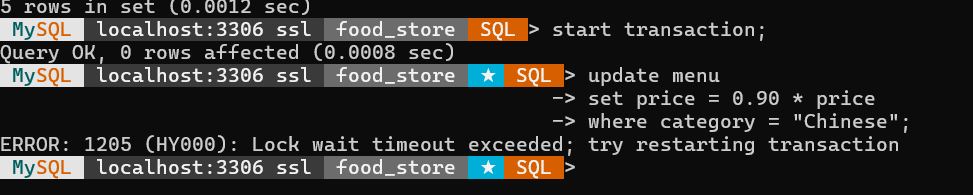
**4. Concurrent Updates**

First in the session 1, the prices of all items are increased by 100. Then, without committing the session 1 transaction, session 2 tries to update the same rows.

Session 1



Session 2



But session 2 cannot update the same row because session 1 has not been committed yet.

After session 1 is committed, session 2 can update.

Session 1



Session 2

