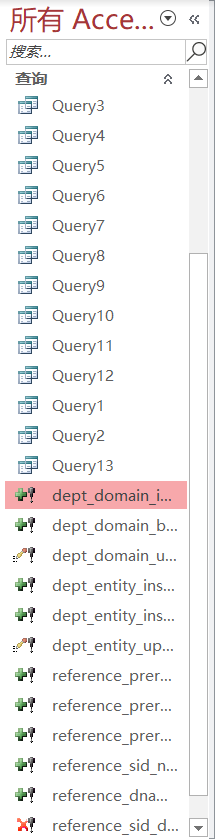
**Assignment Four**

**Report**

-

09020328 Liang Wang

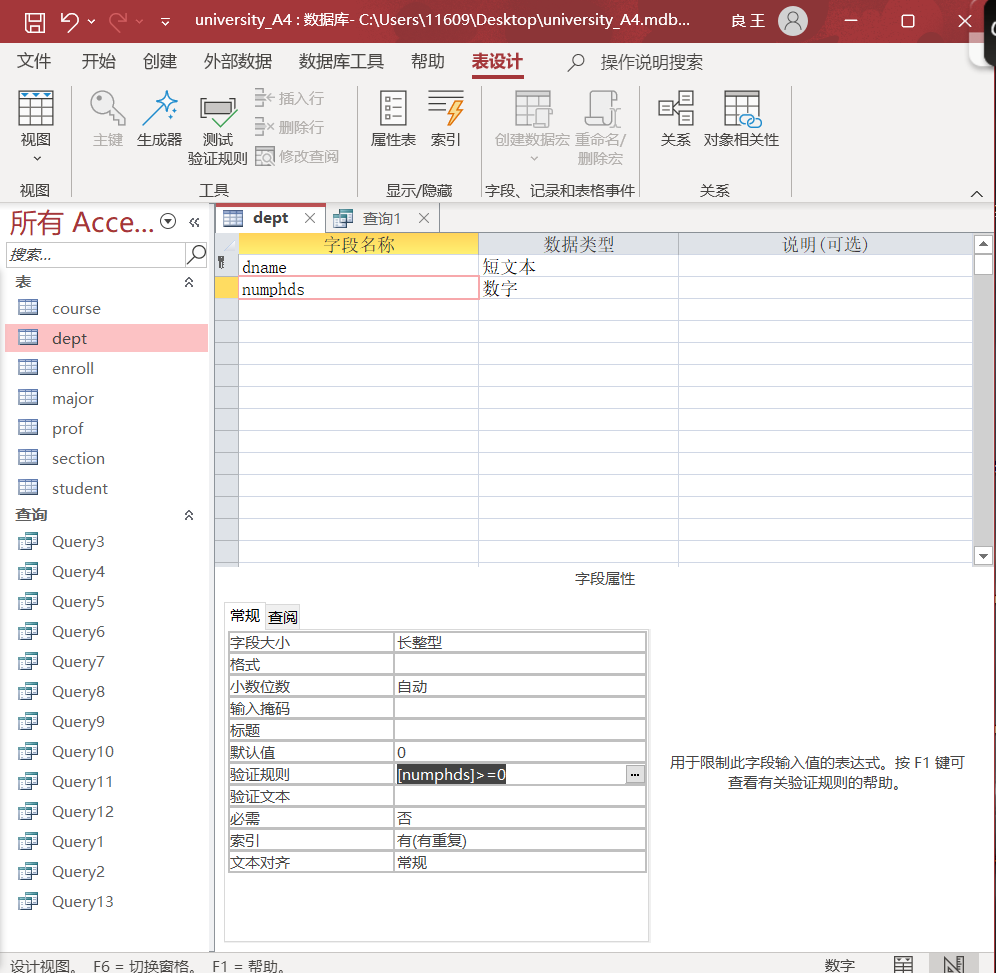
# **First: Domain Integrity Constraint**

 Note: It is substantially recommended that adopting the strategy - “creating time” of sorting.

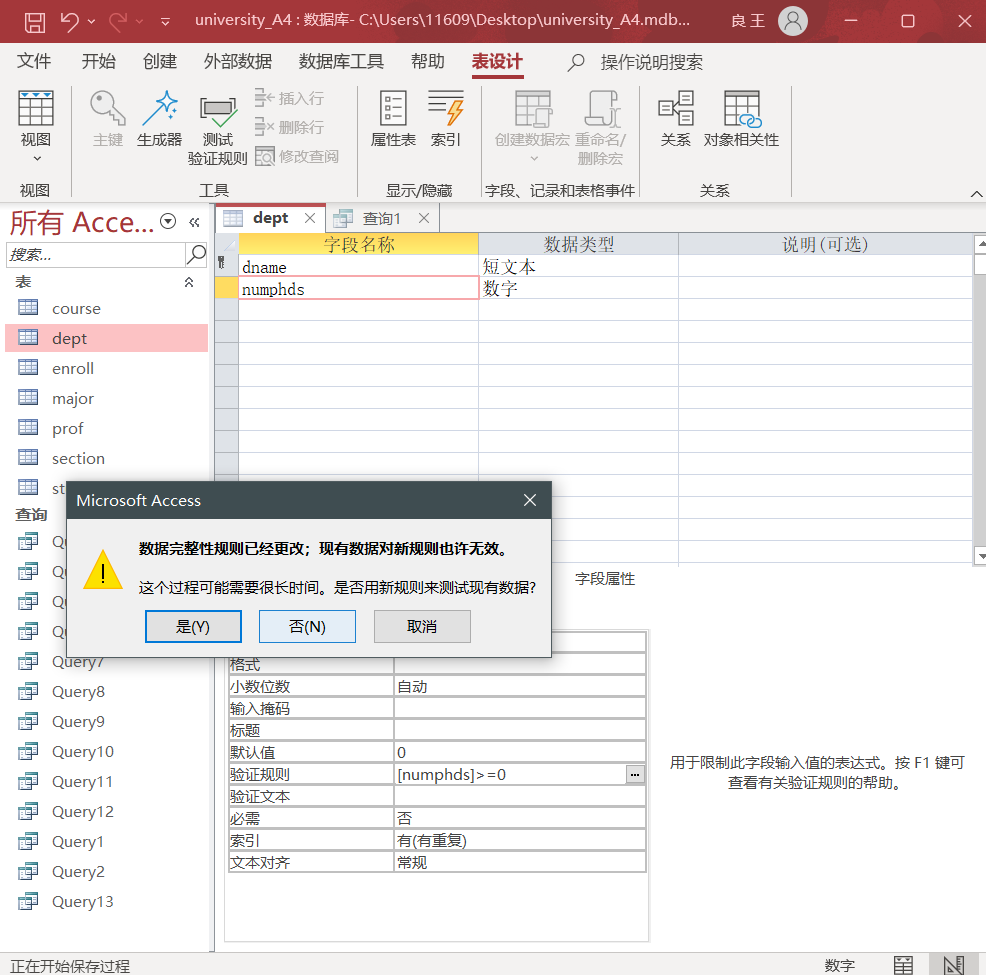
This part resolves around how domain integrity constraint helps to check the validity of data(obey some kind of rules) to be inserted or updated.

Prerequisites:

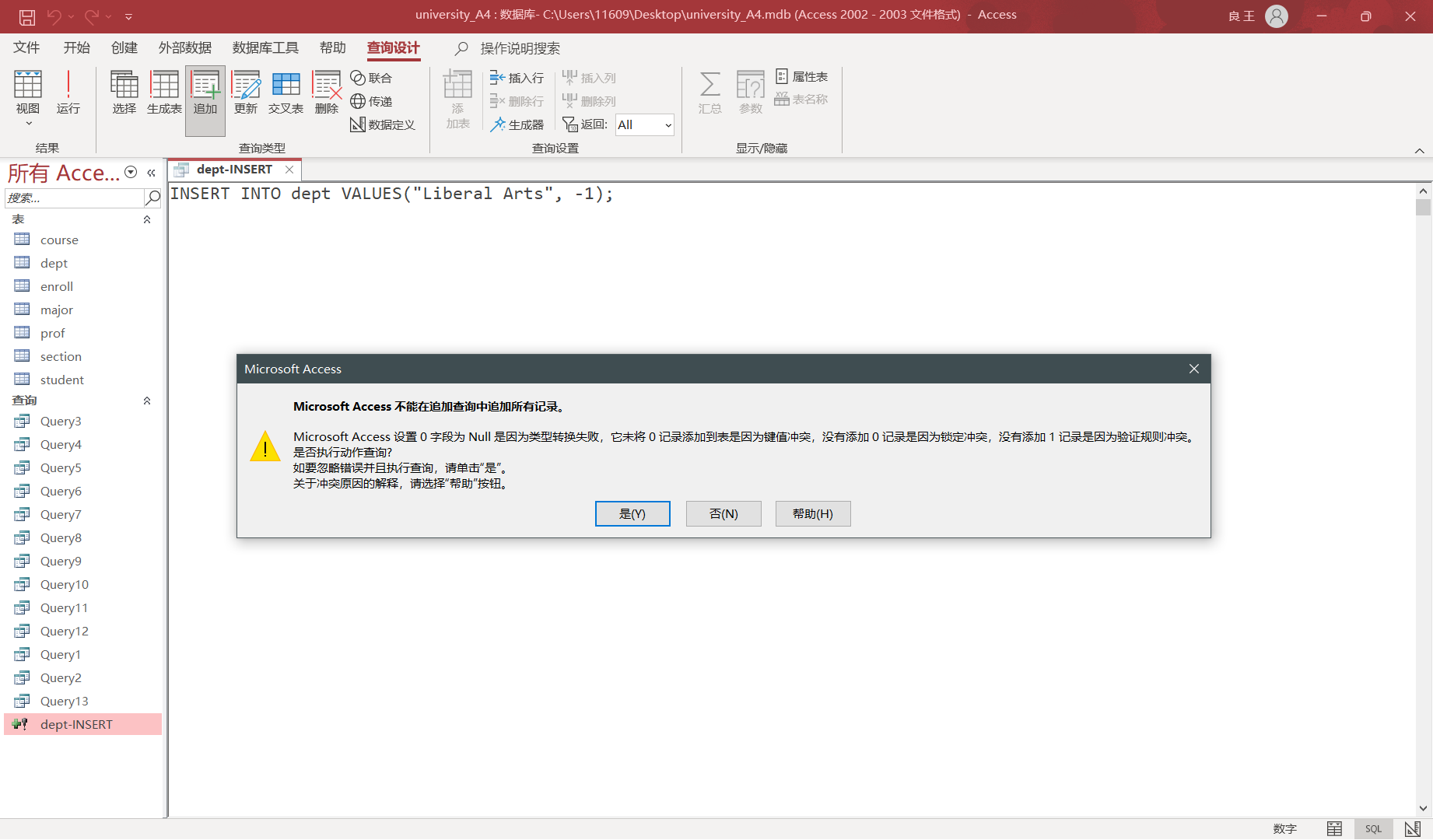
Set the checking rule of dept table to “[numphds]>=0 ”



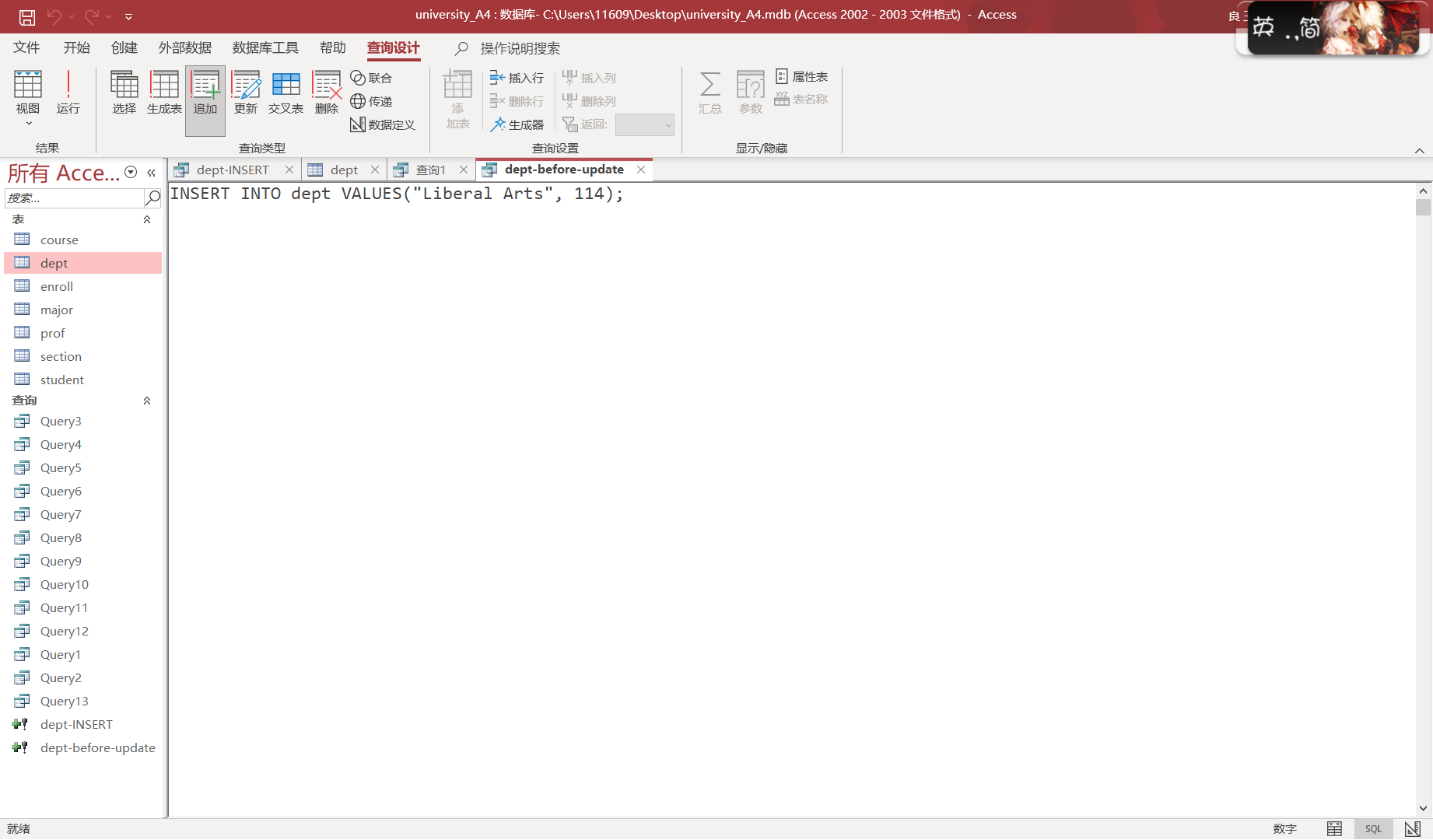
The process should pop a window suggesting that some data may not obey the new set rule. Allow the test under new rule and the verification of domain integrity constraint shall begin after the procedure.



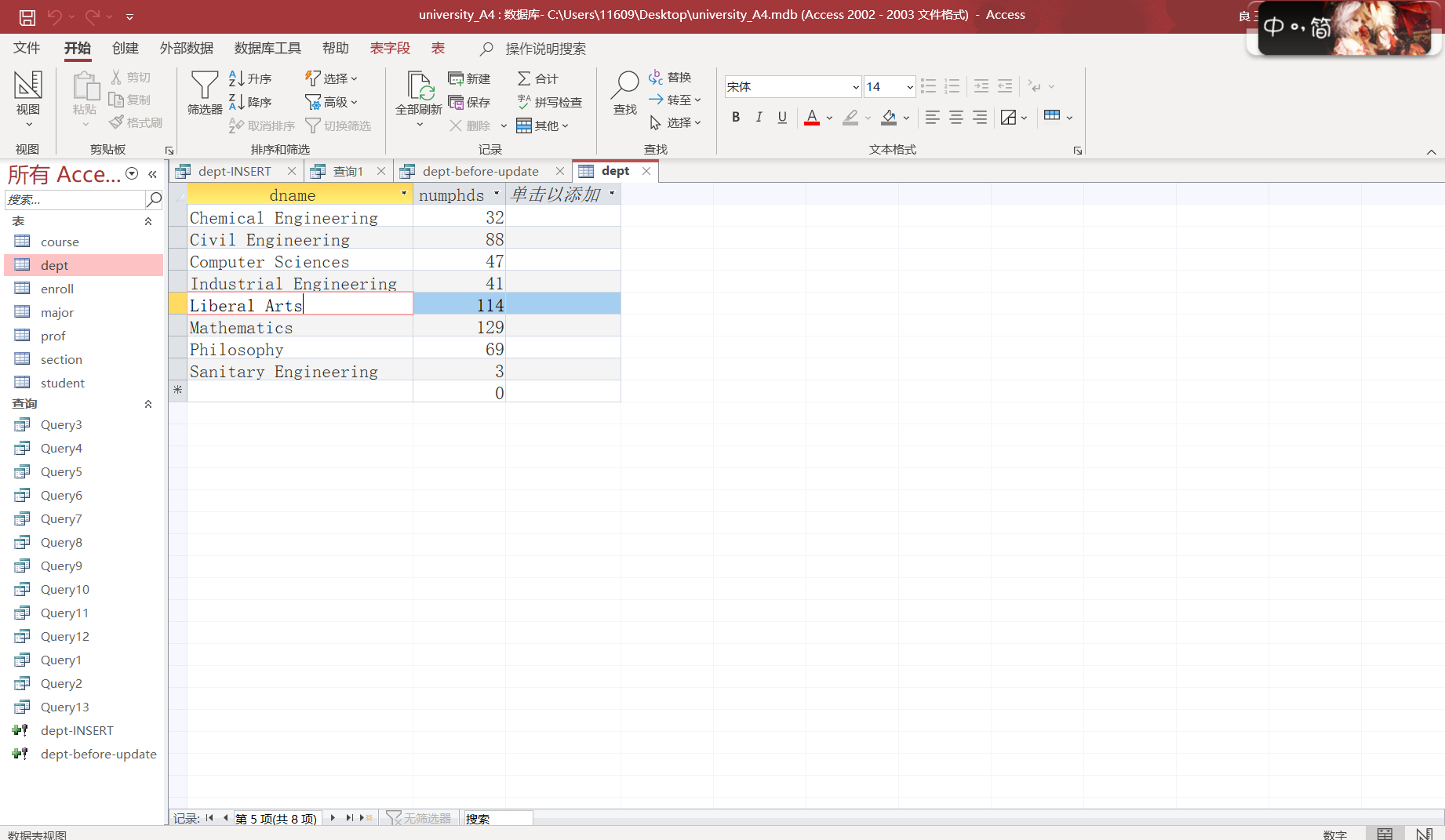
Adopt the sql statement below which offers an invalid value “-1” contradicting to the rule “>=0”. A warning pops up to illustrate the conflict with the rule has stopped the process to continue, which itself proved the correctness of domain integrity constraint.



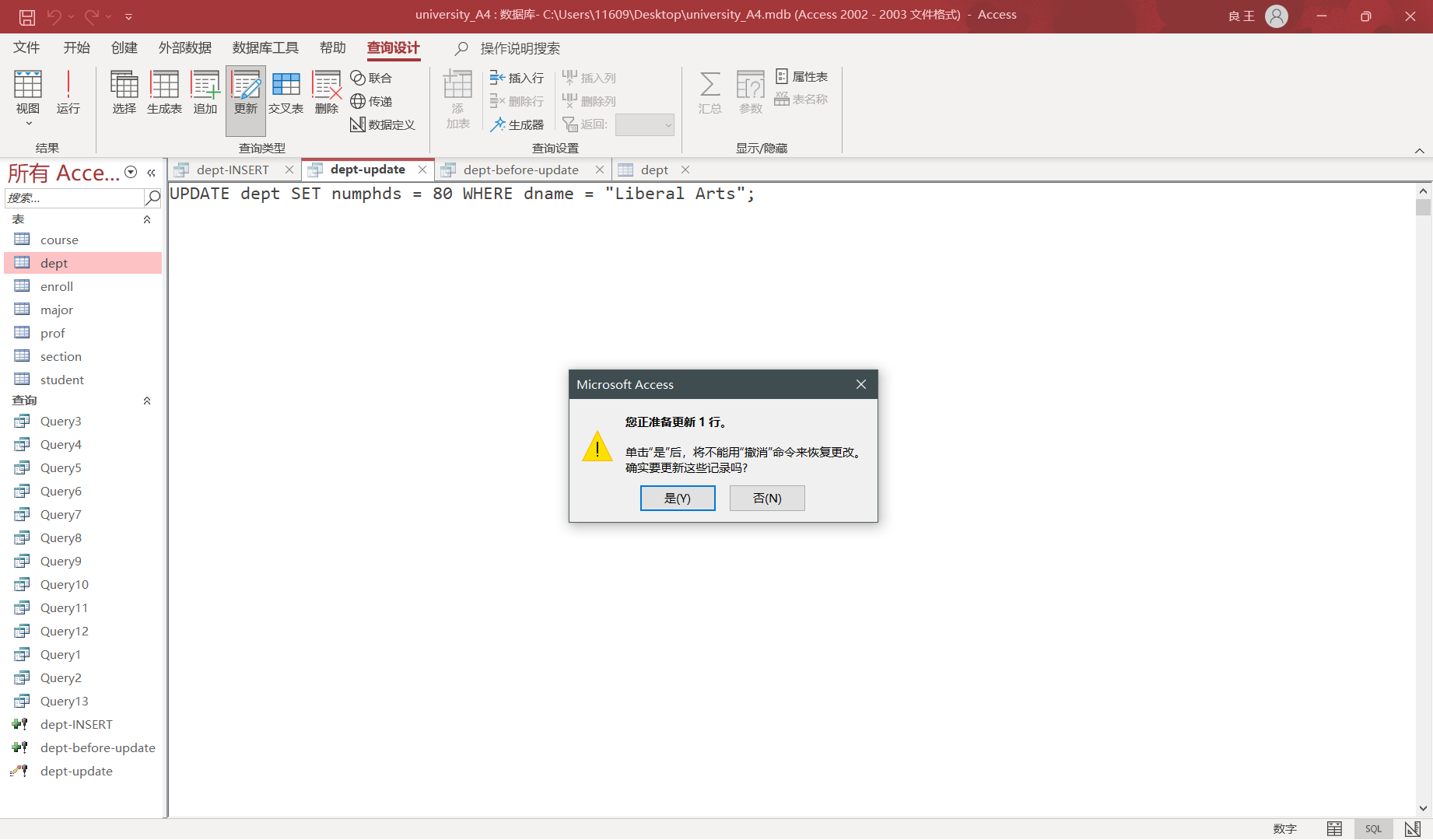
Besides “INSERT” sql, “UPDATE” can demonstrate the fact in a similar way. Here a valid data is appended by the insert sql.

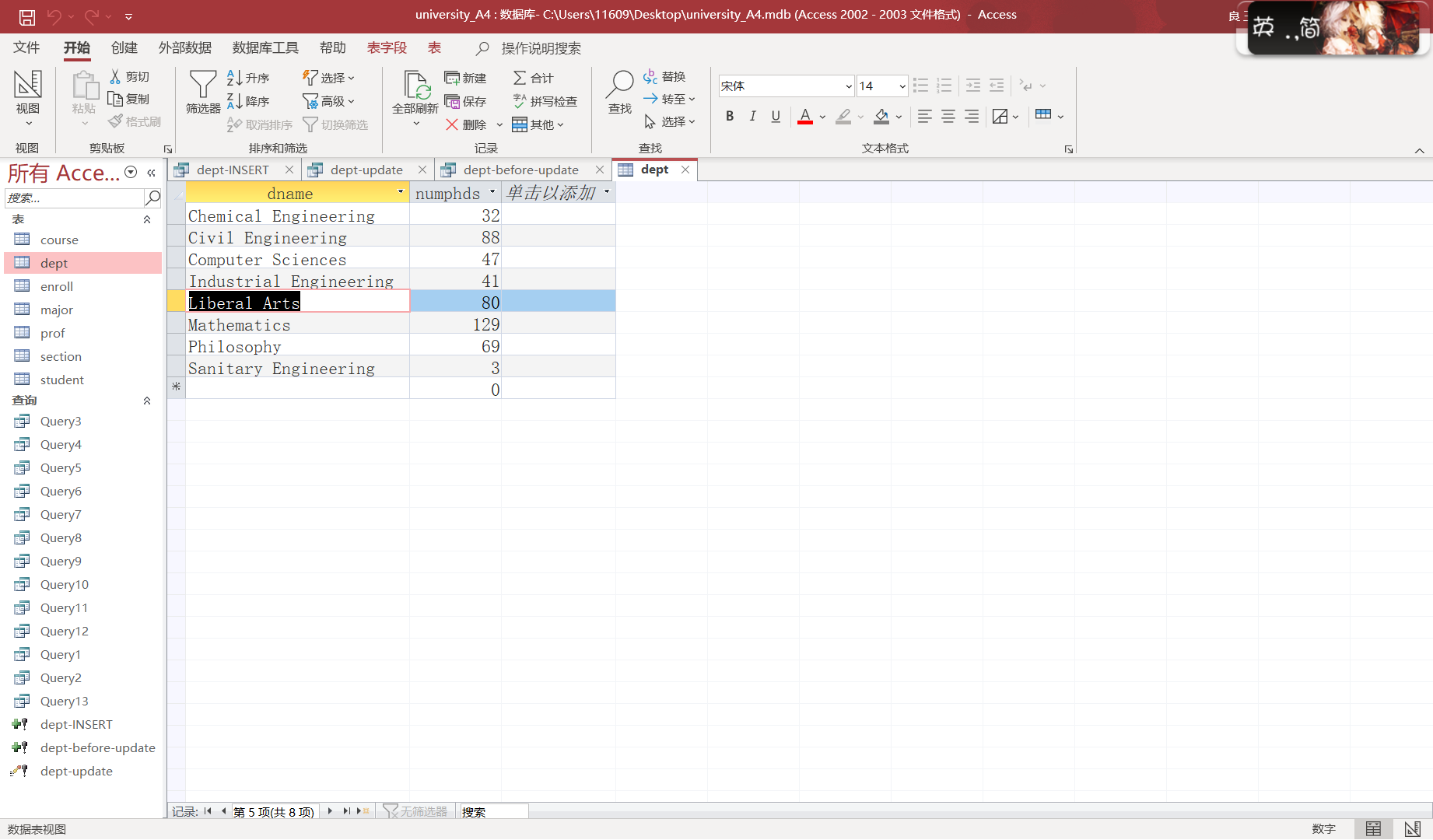


Check the dept table.

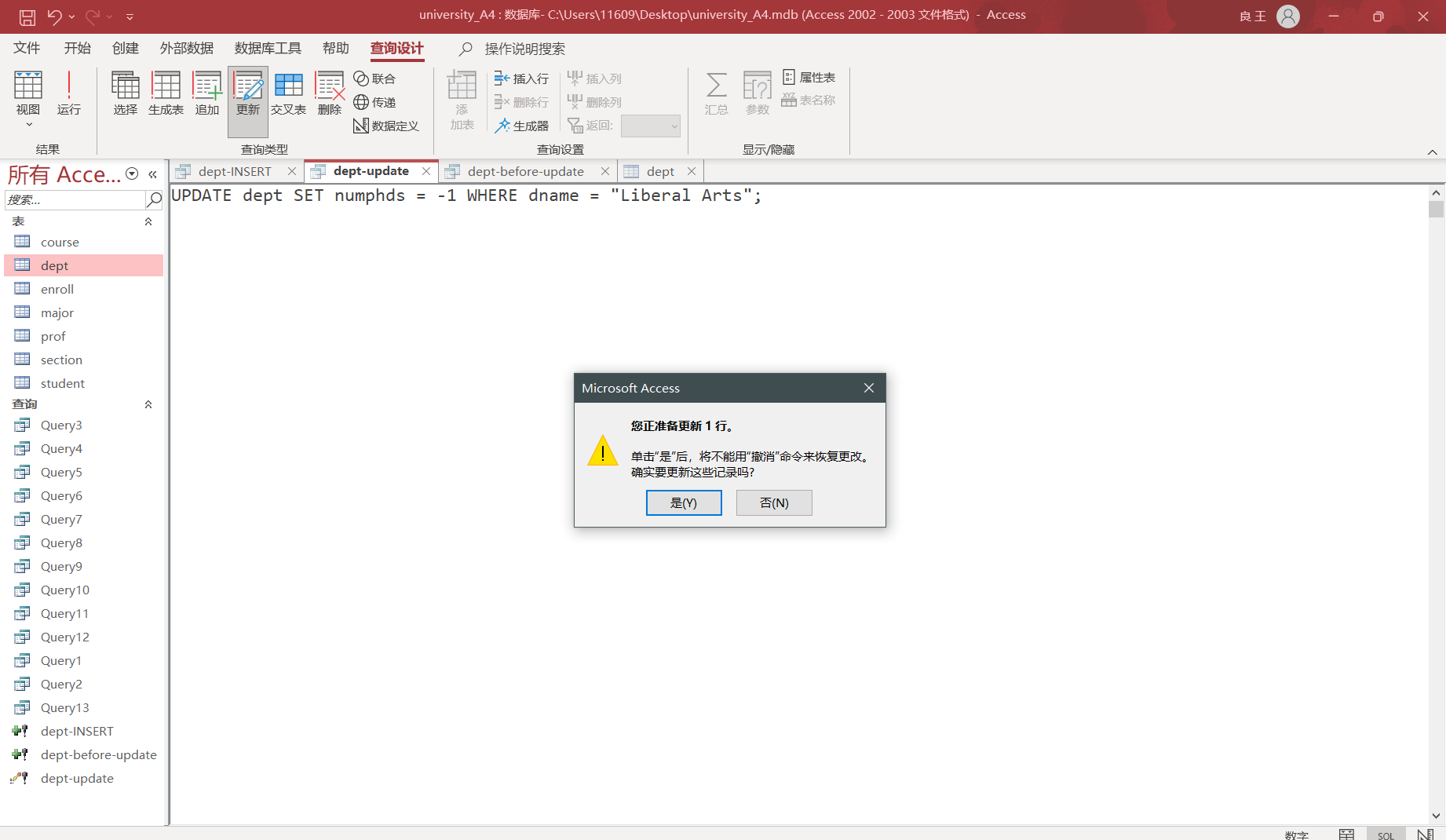


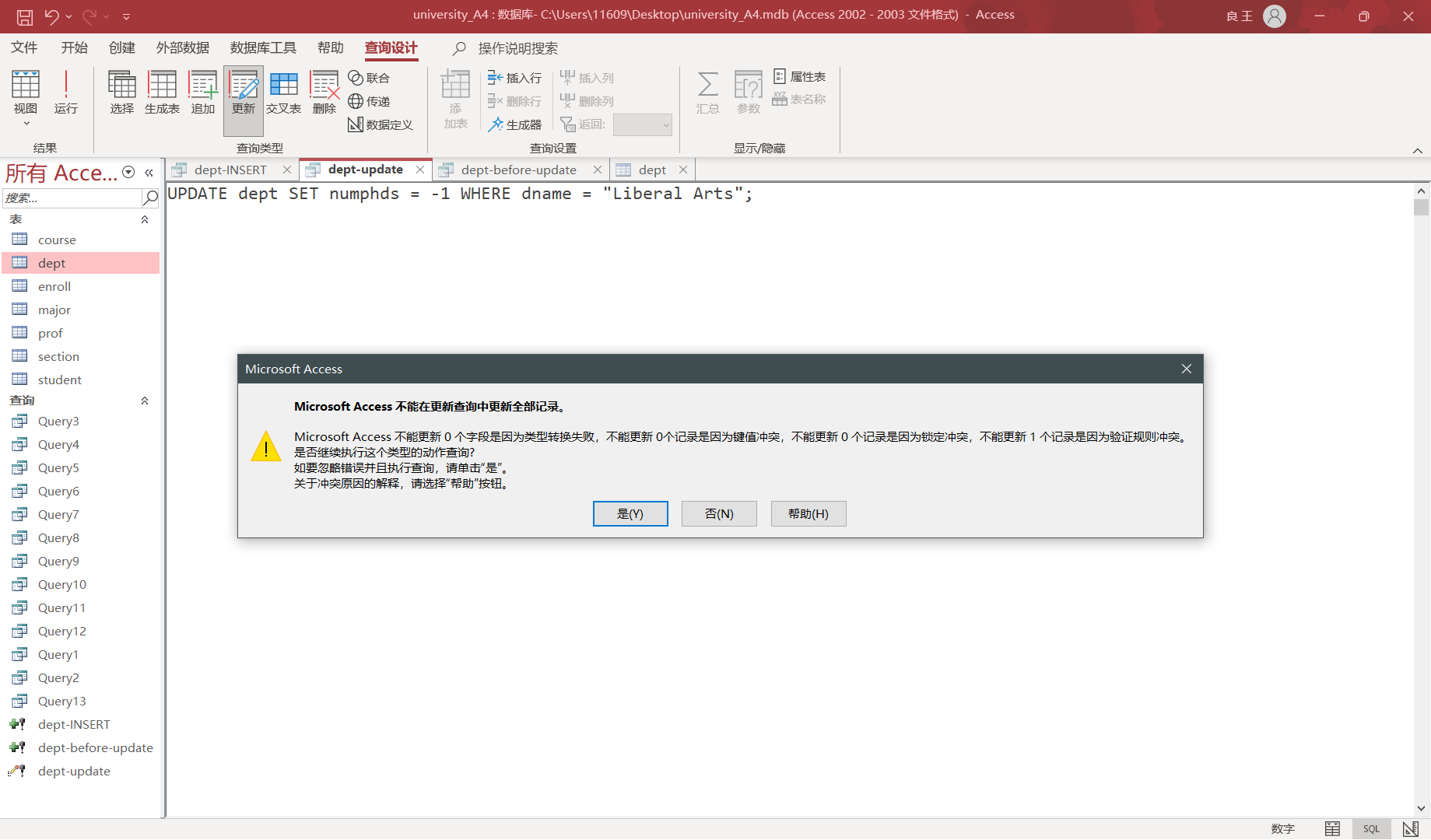
Normally, updating process should go smoothly like the following process.



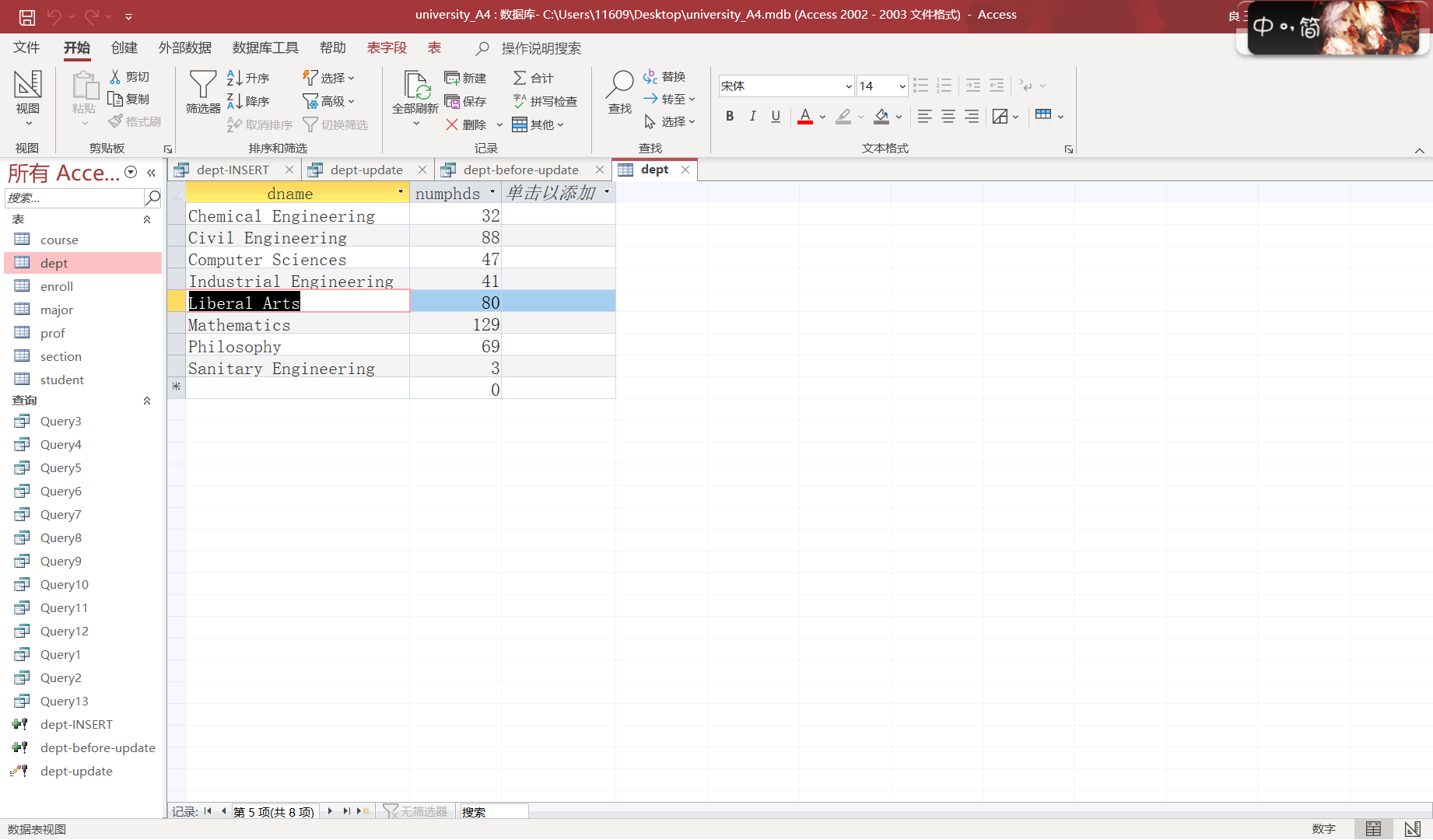


Under the set rule, updating with invalid data is not allowed.





The data remained still.

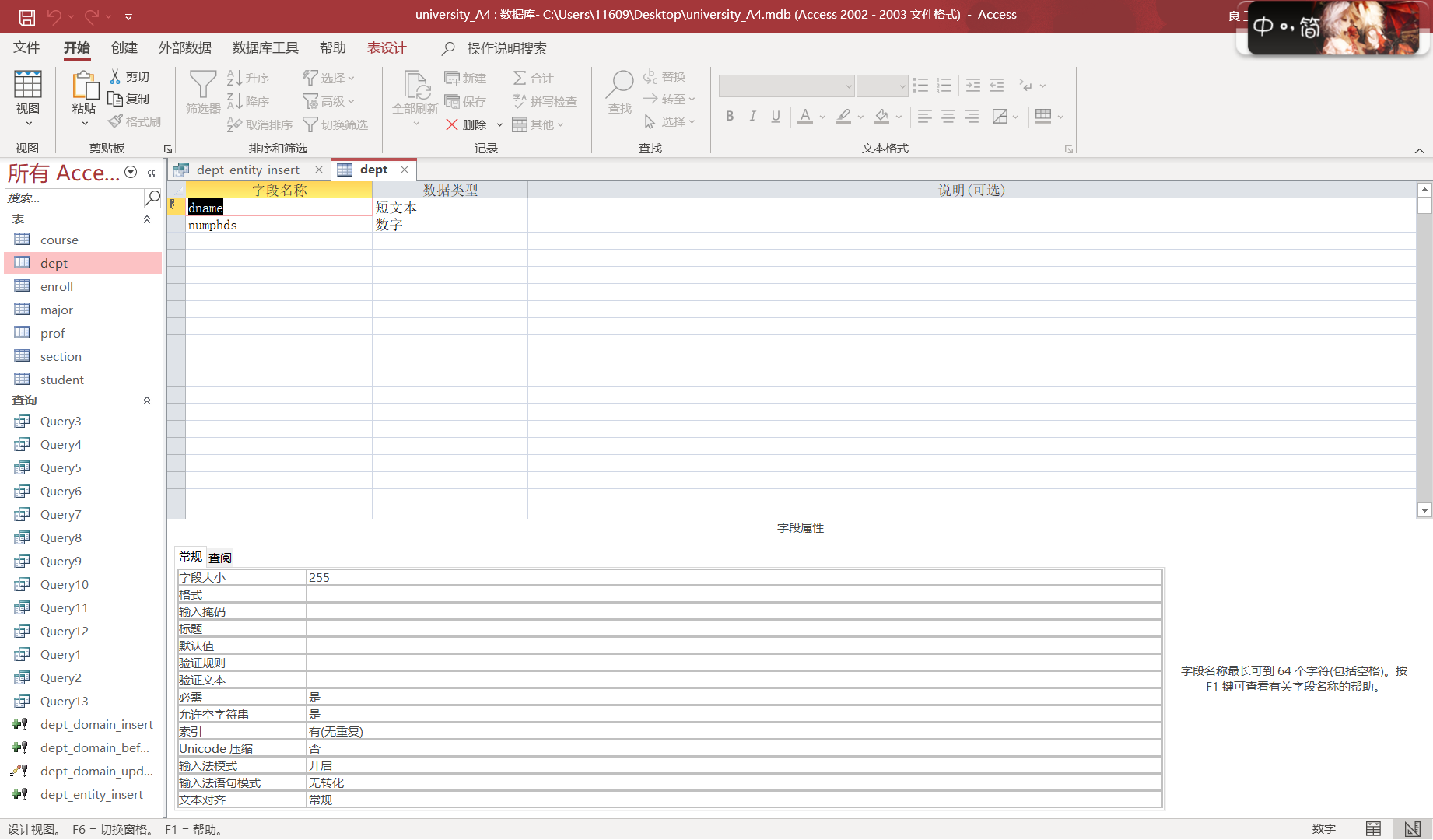


# **Second: Entity Integrity Constraint**

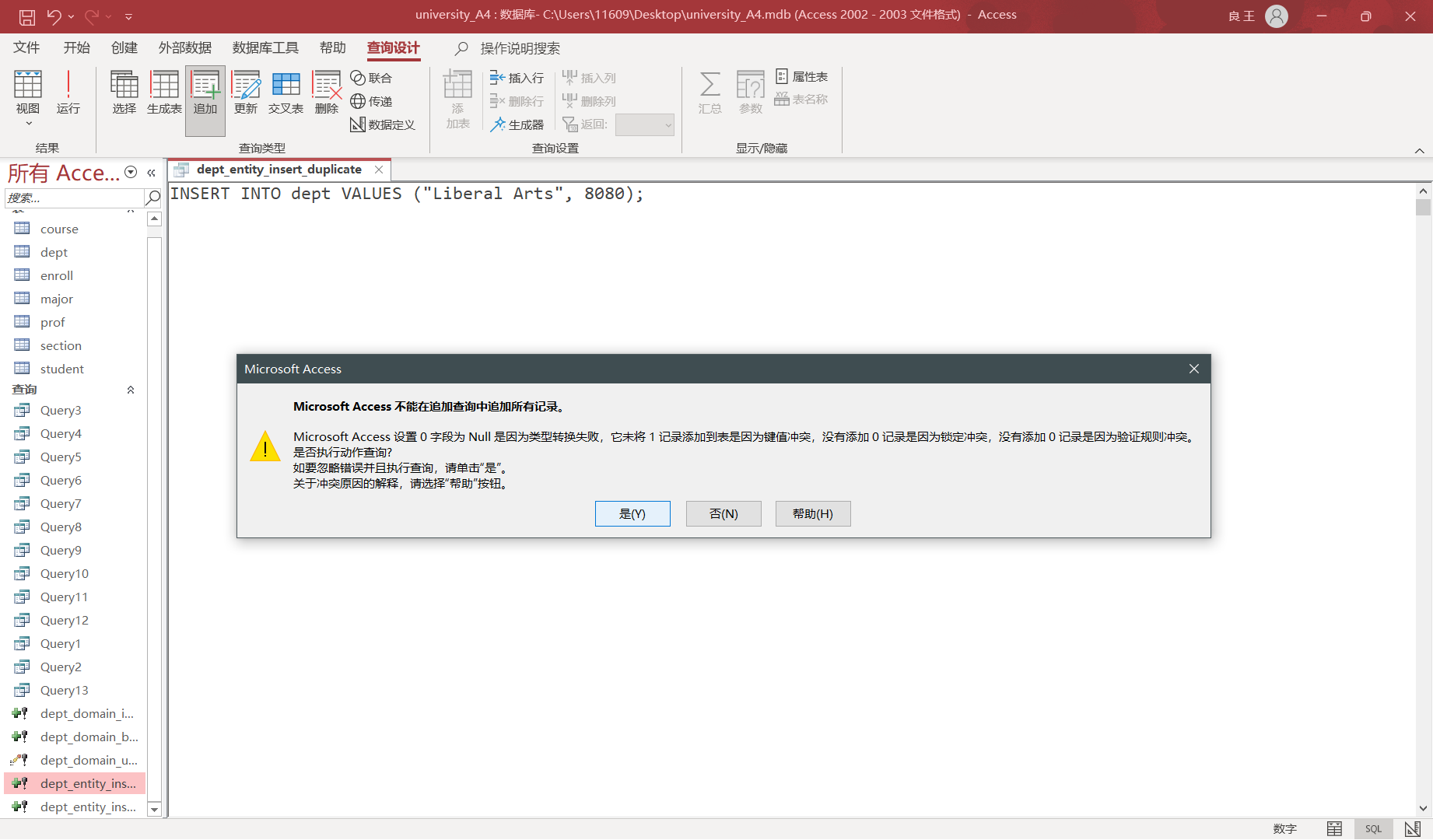
Prerequisites:

This part focuses on how entity integrity constraint is helpful in averting duplicate keys as well as null.

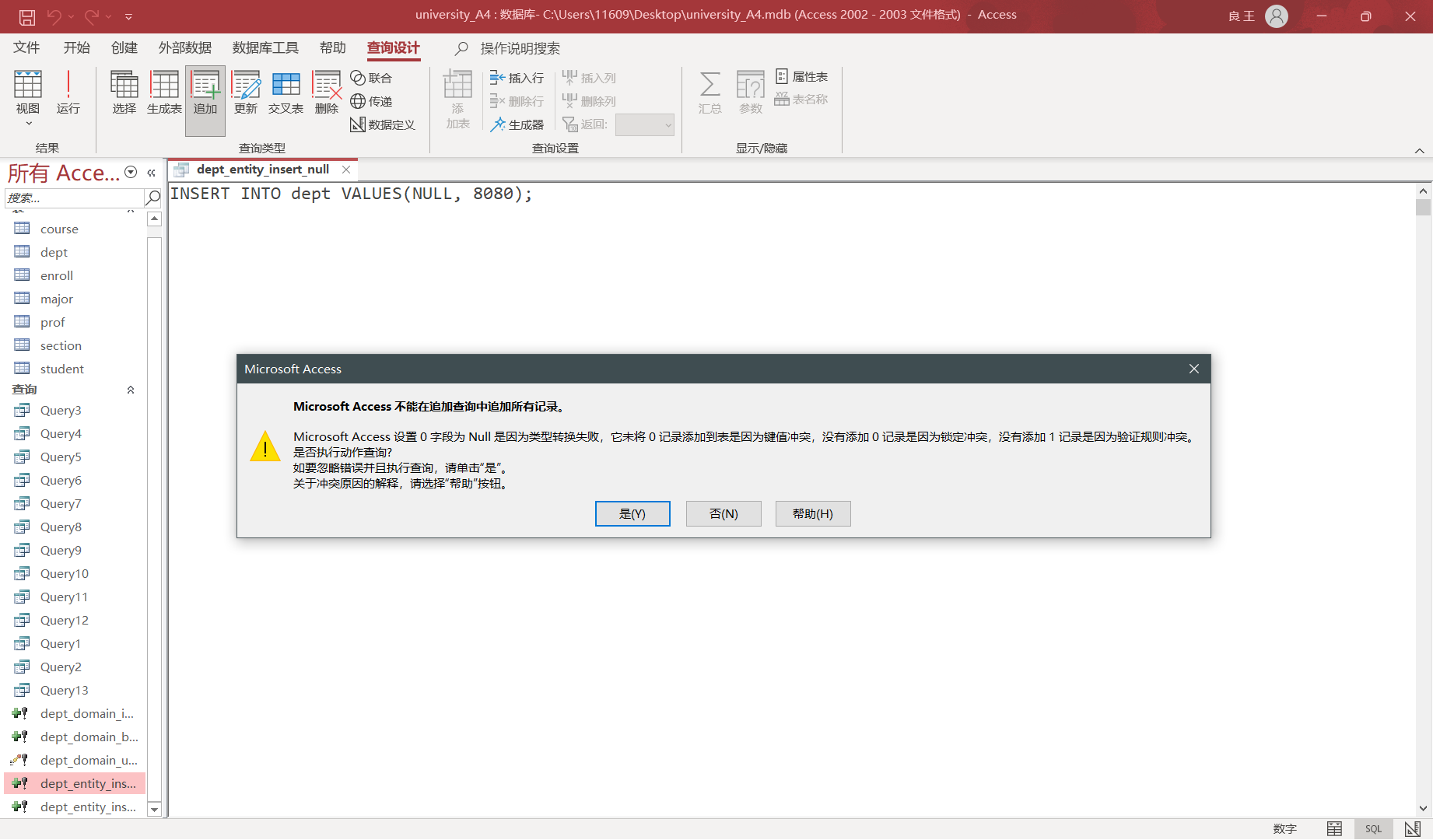
Set the appropriate field or fields as primary key. With the constraint, in theory, neither duplicate values nor null is allowed. The “dname” field in dept table alone is set as the primary key, the sole thing worth attention here.



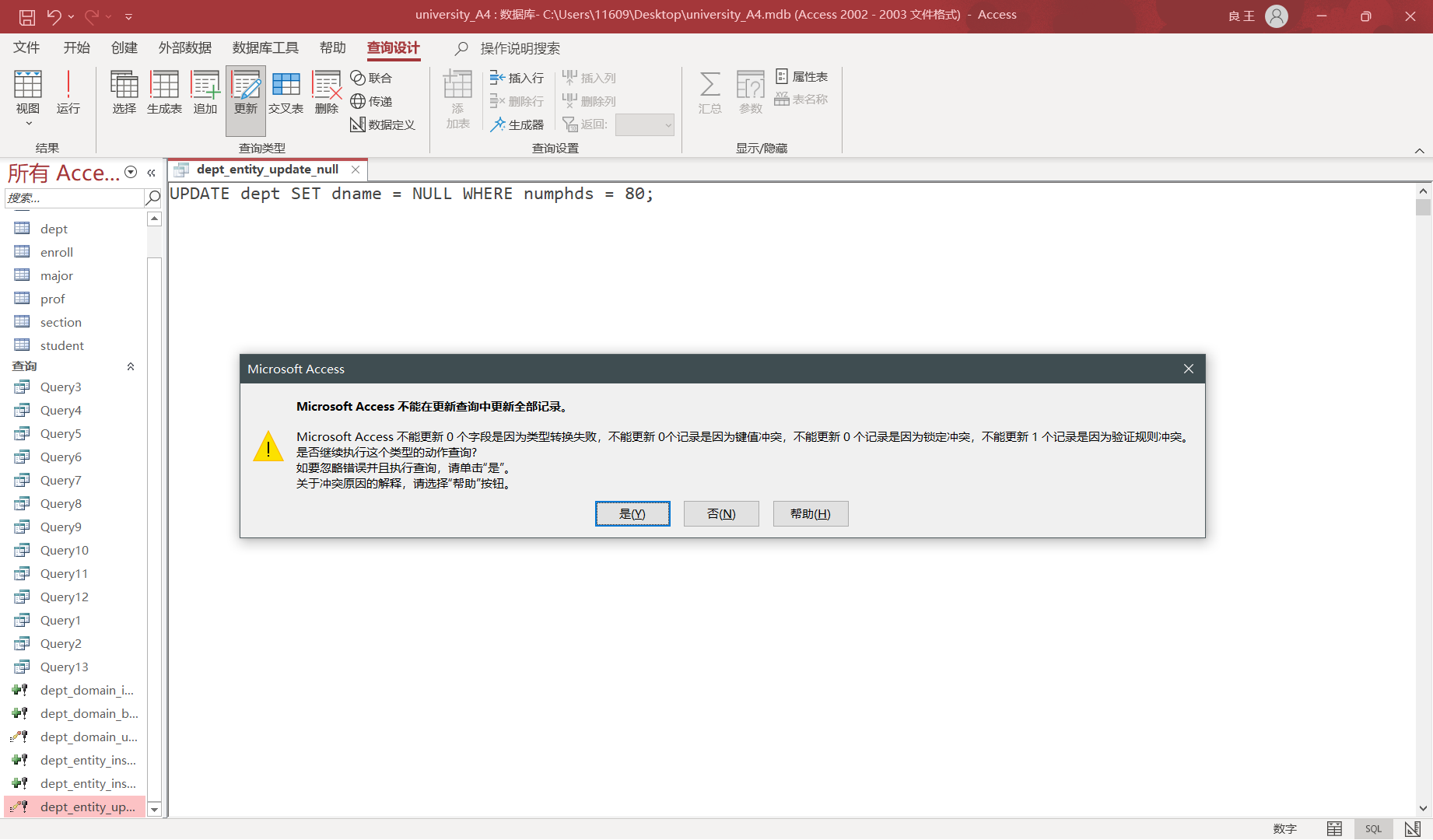
Test it with duplicate key, and got rejected.



The same goes for the null.



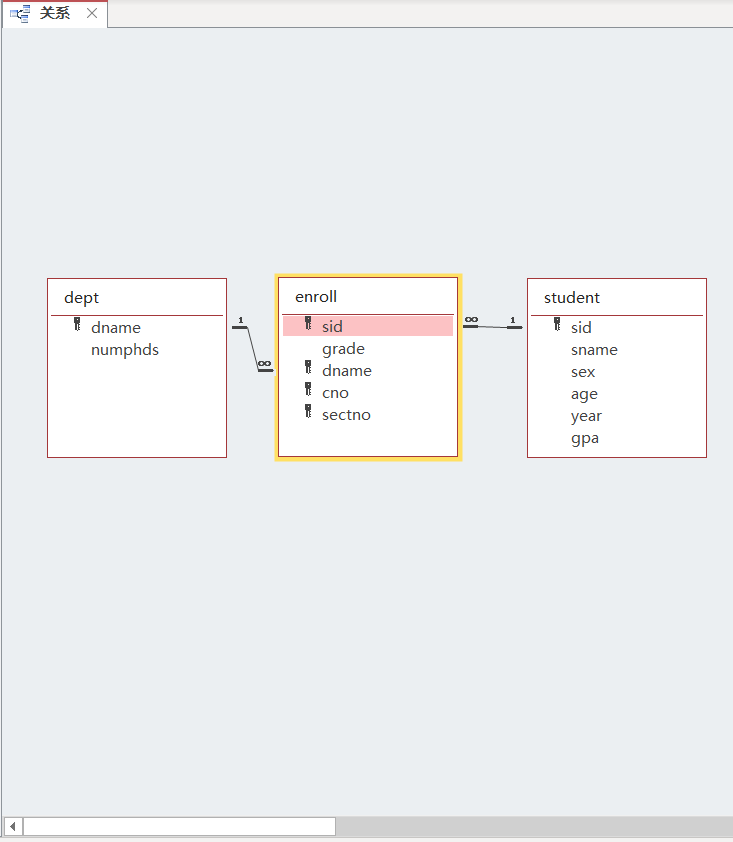
No exception for updating.



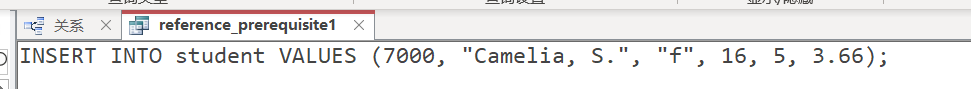
# **Third: Referential Integrity Constraint**

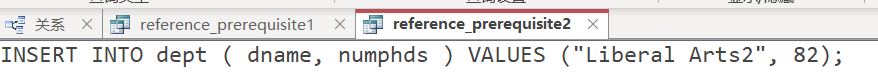
Prerequisites:

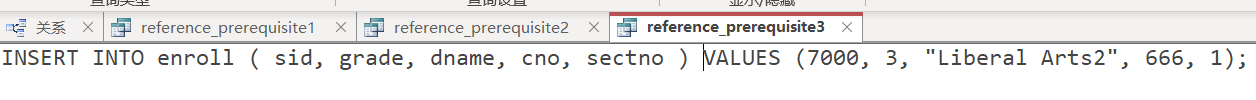
In the “relation” view, three tables were chosen to demonstrate how referential integrity constraint is to help synchronize the change amongst an assortment of tables and offer warnings when the foreign keys can not be linked to the due values. The foreign keys, major focus here, encompass sid and dname, linking enroll to student and dept respectively.



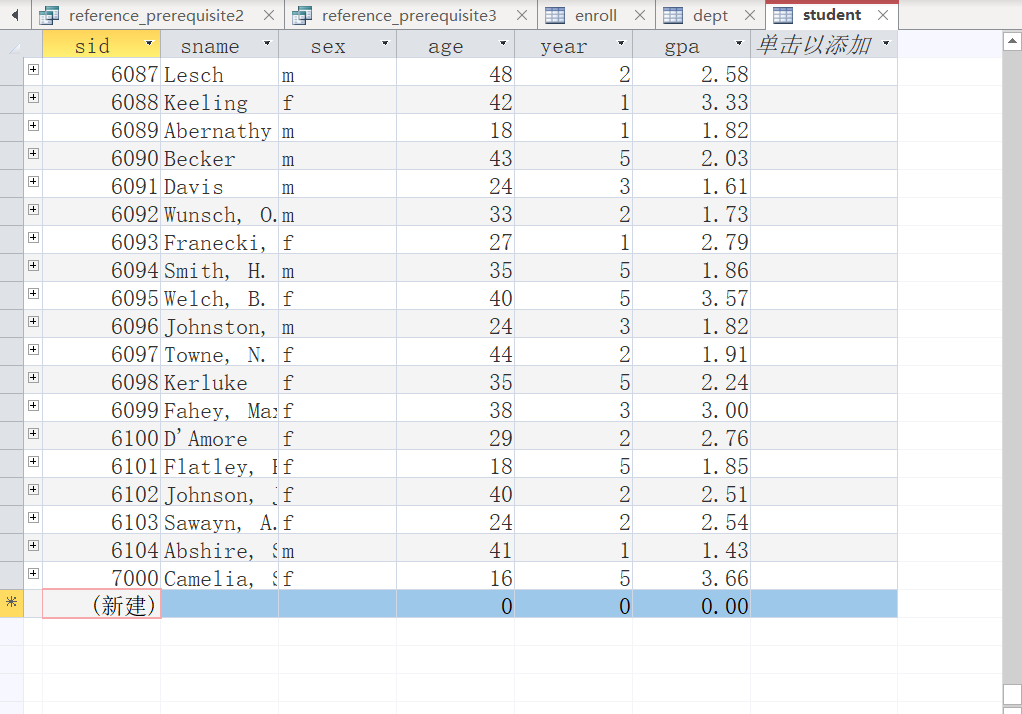
Prepare data to verify the constraint.

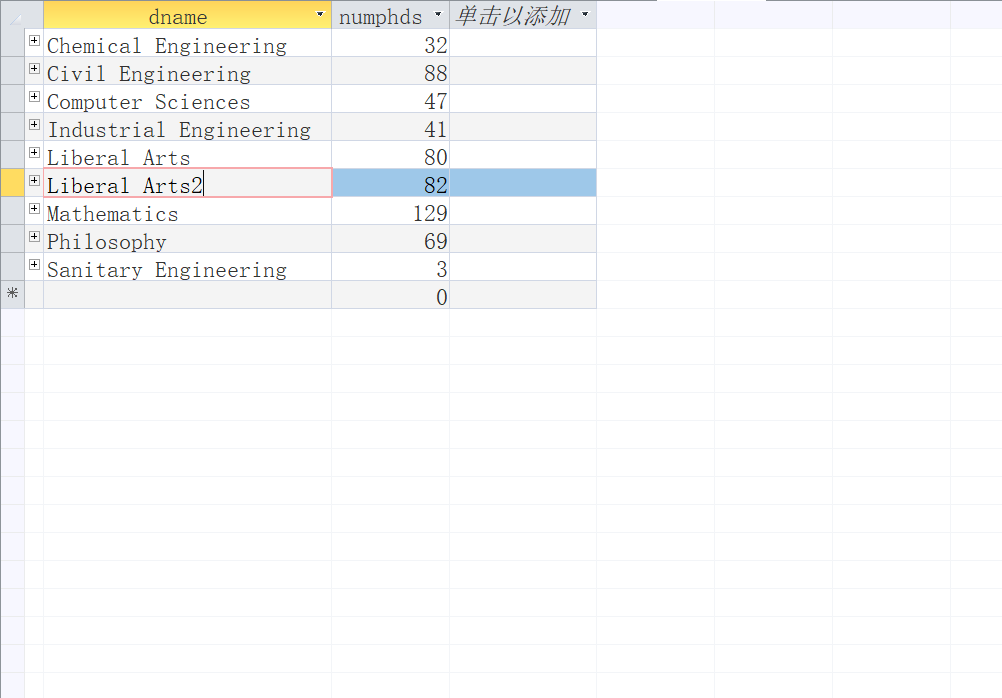


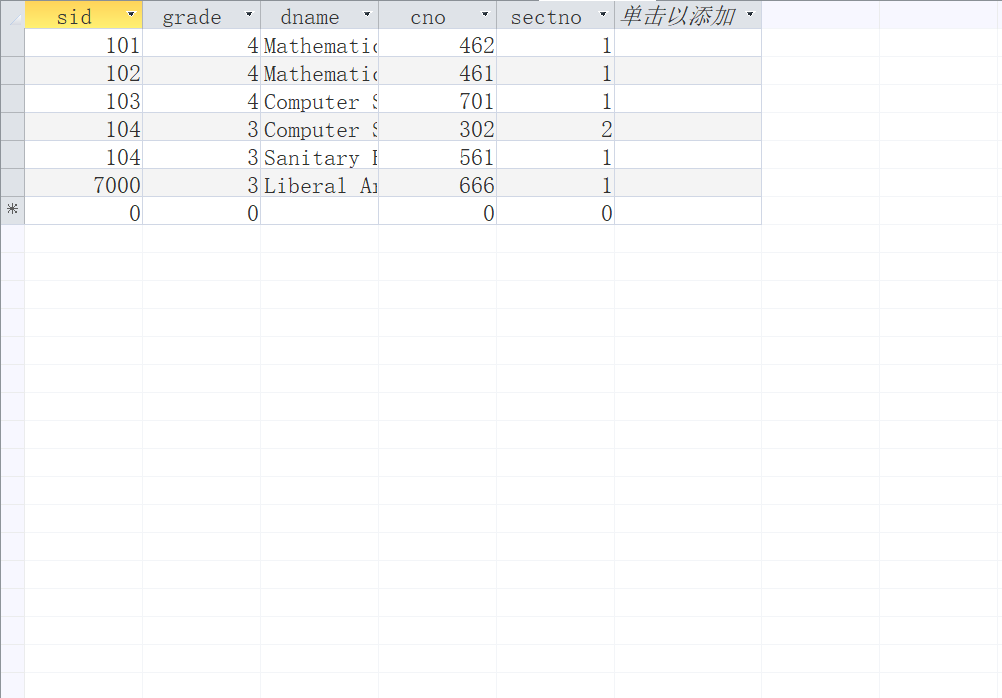




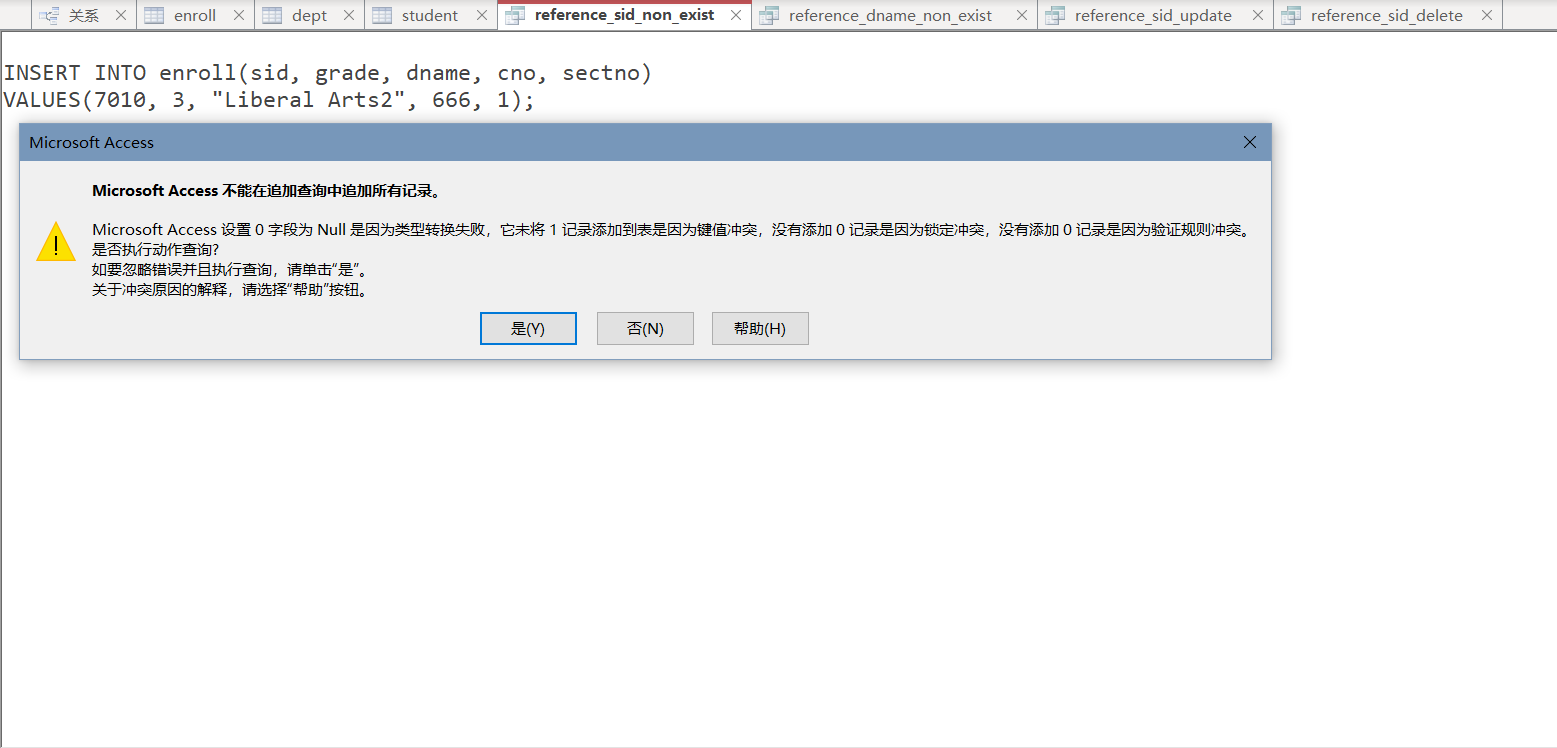
After modification:



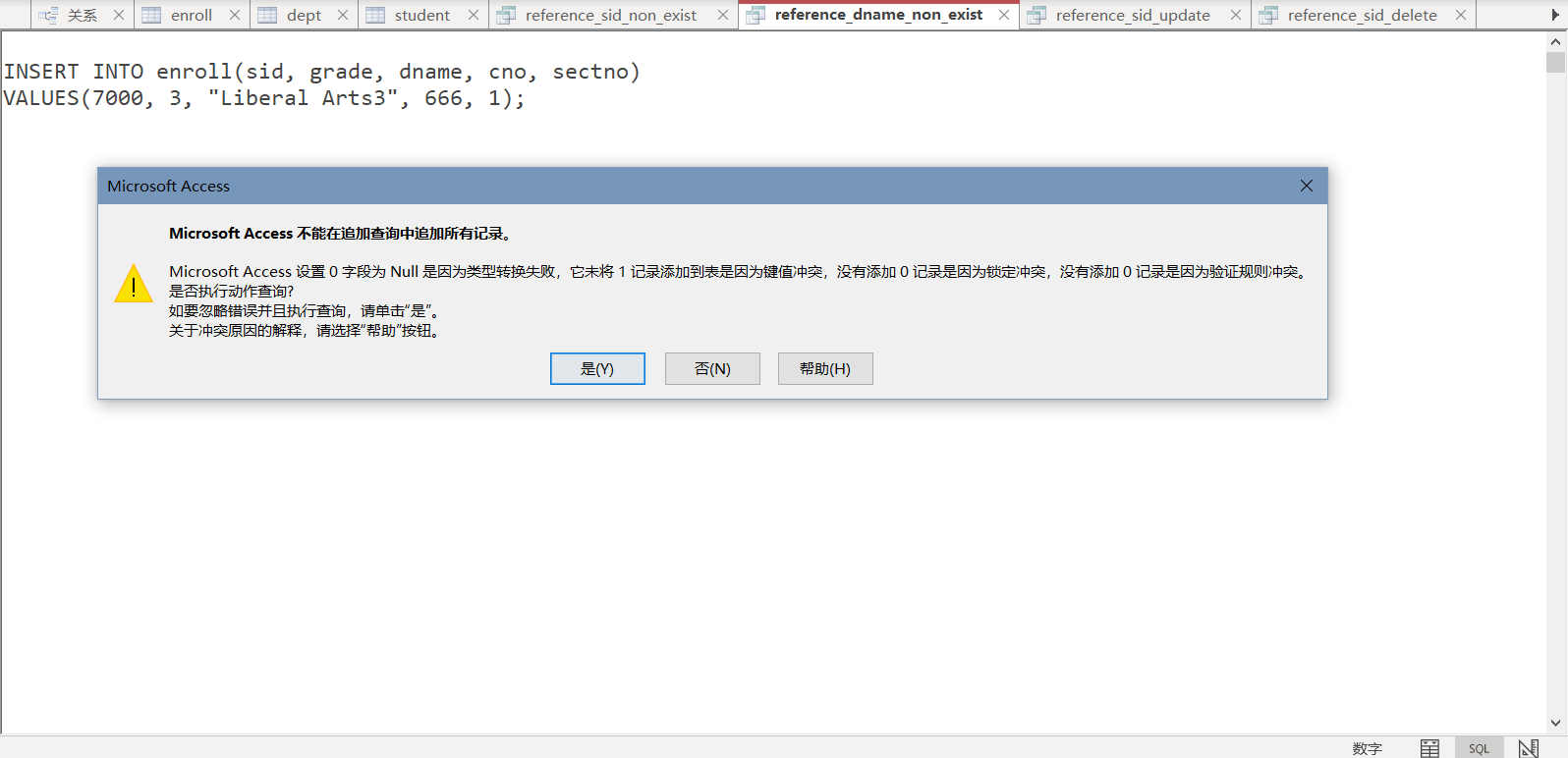




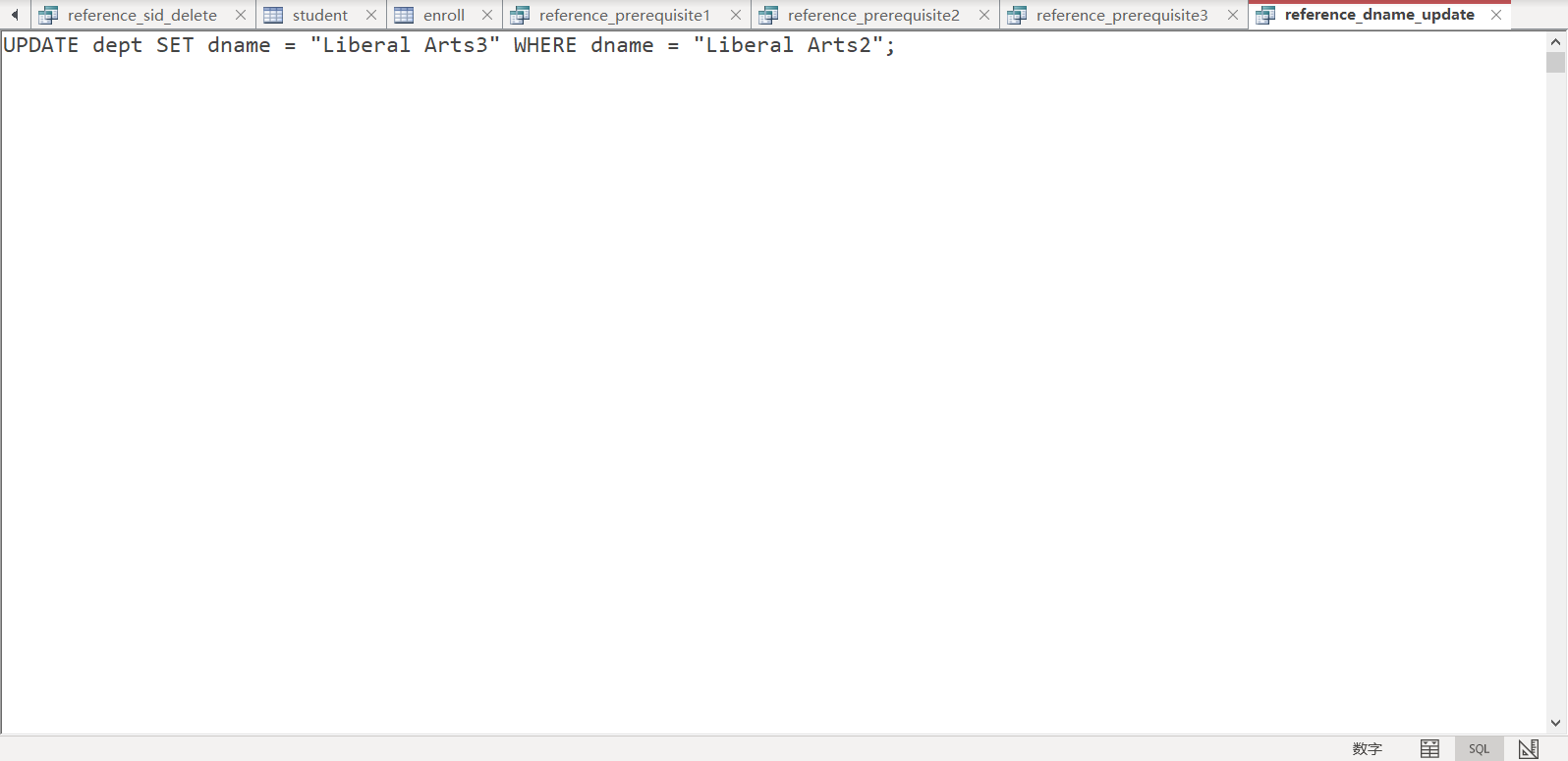
Insert with an invalid sid i.e. one not in the student table.

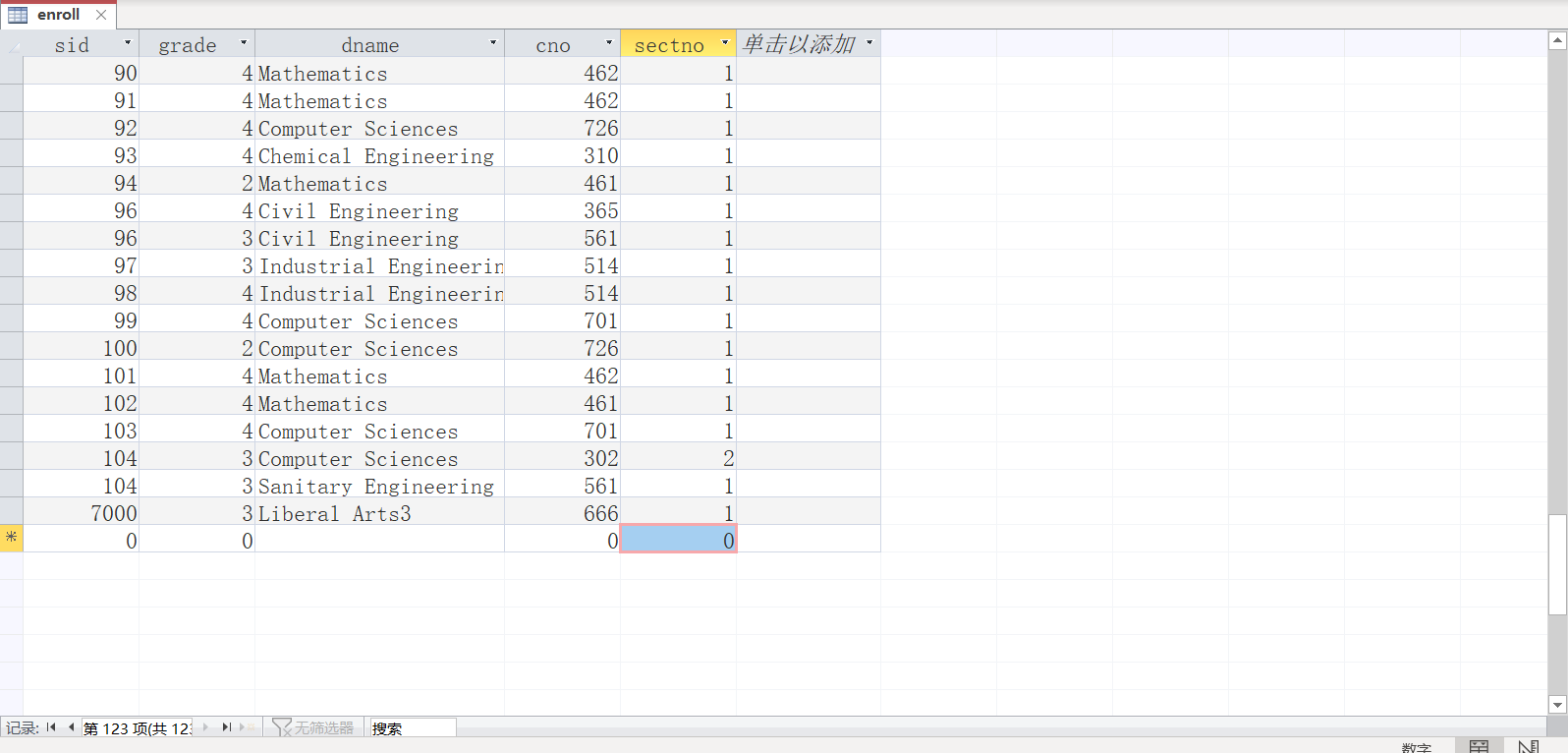


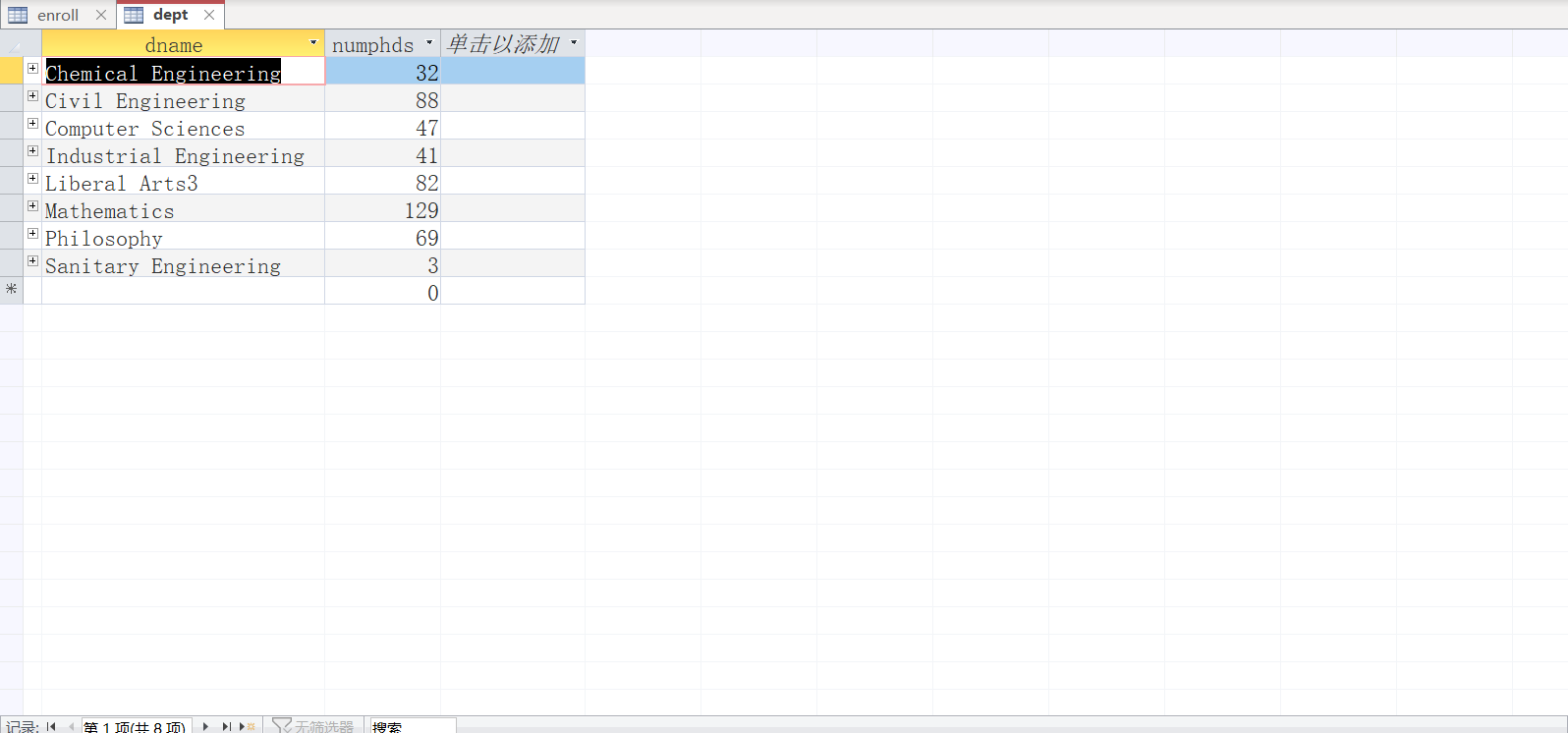
Another scenario, with invalid dname.



Updating dname in dept leads to a following update in enroll.

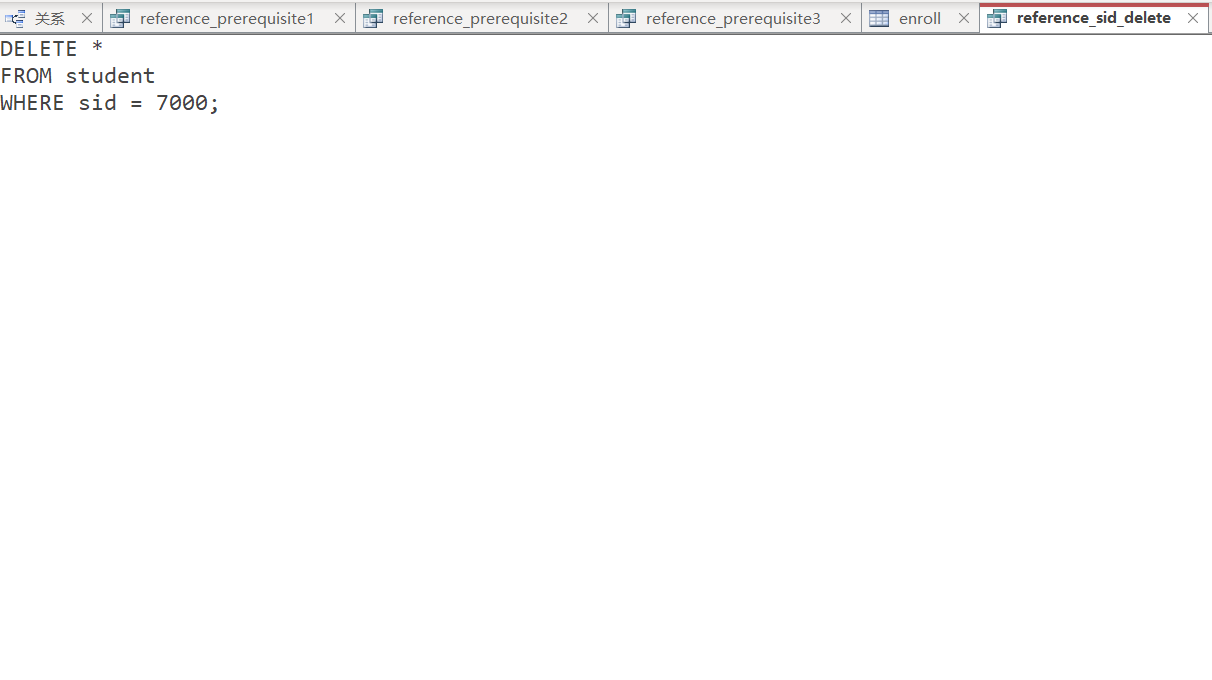




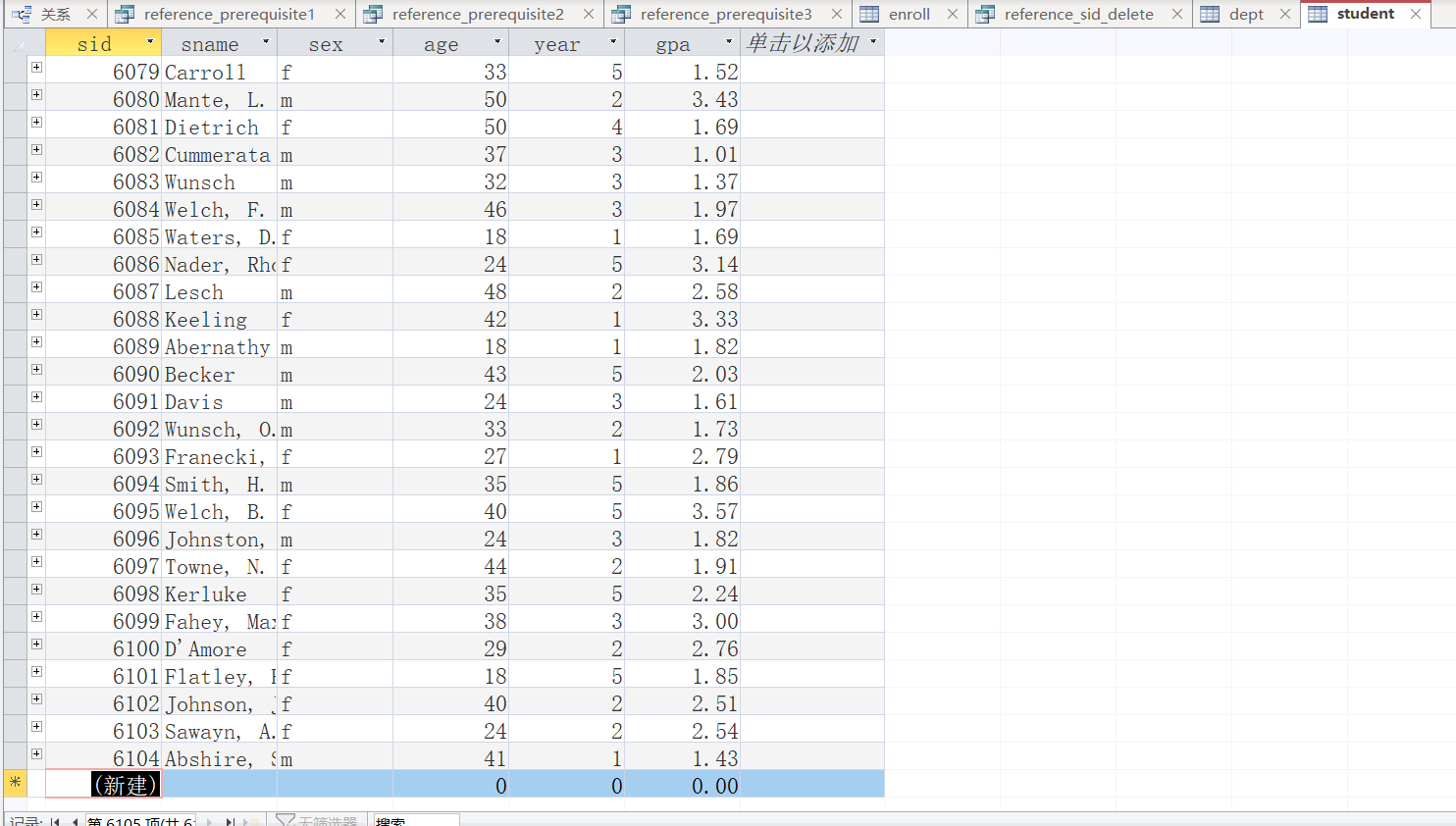


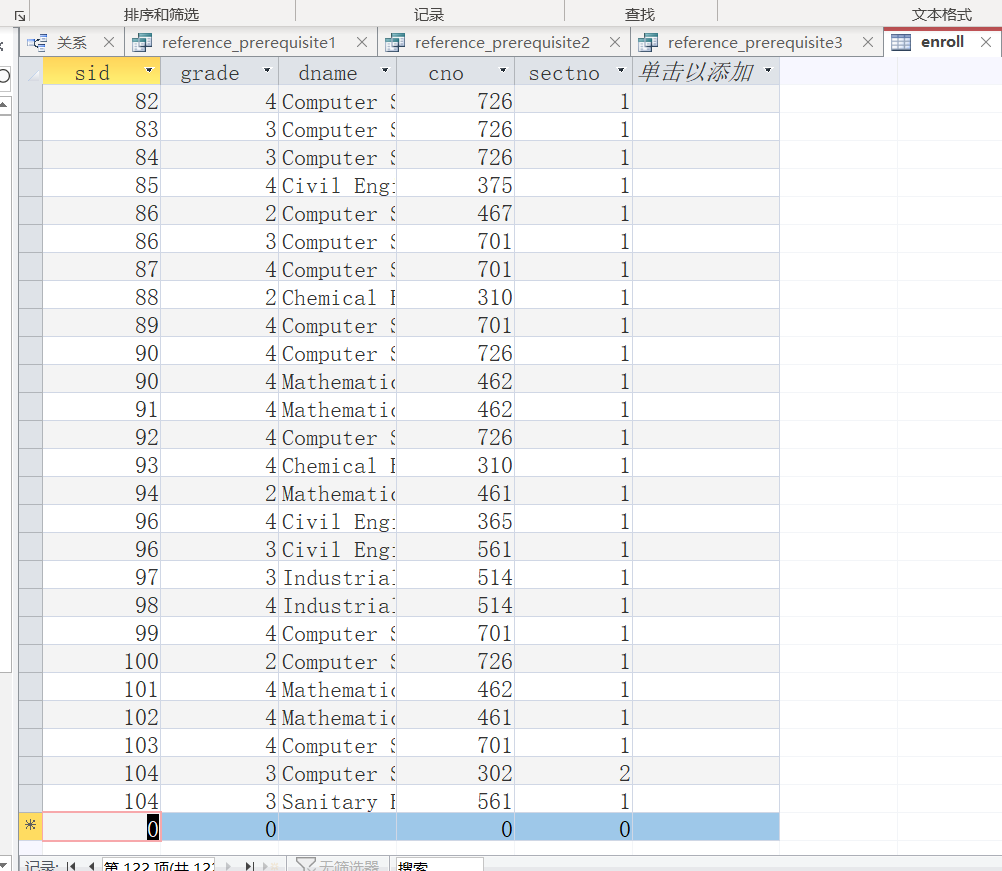
Abort the change above, and repeat the three prerequisites.

This time, the delete sql is tested.



The removal of the student whose sid is 7000 in student table comes hand in hand with that of enroll table, while dept table remains the same.





# 

Plus, if the table is open while the sql being excuted, special marks annotated “deleted” are displayed.

# **Conclusion**

Constraints in database benefits us tremendously in reasonable and plausible schema design, making relational database even more comfortable to work with, and under certain circumstances, can work like a charm.