CENG 316 DATABASE SYSTEMS FINAL

Prepared by Ceren Umay Özten -181180060

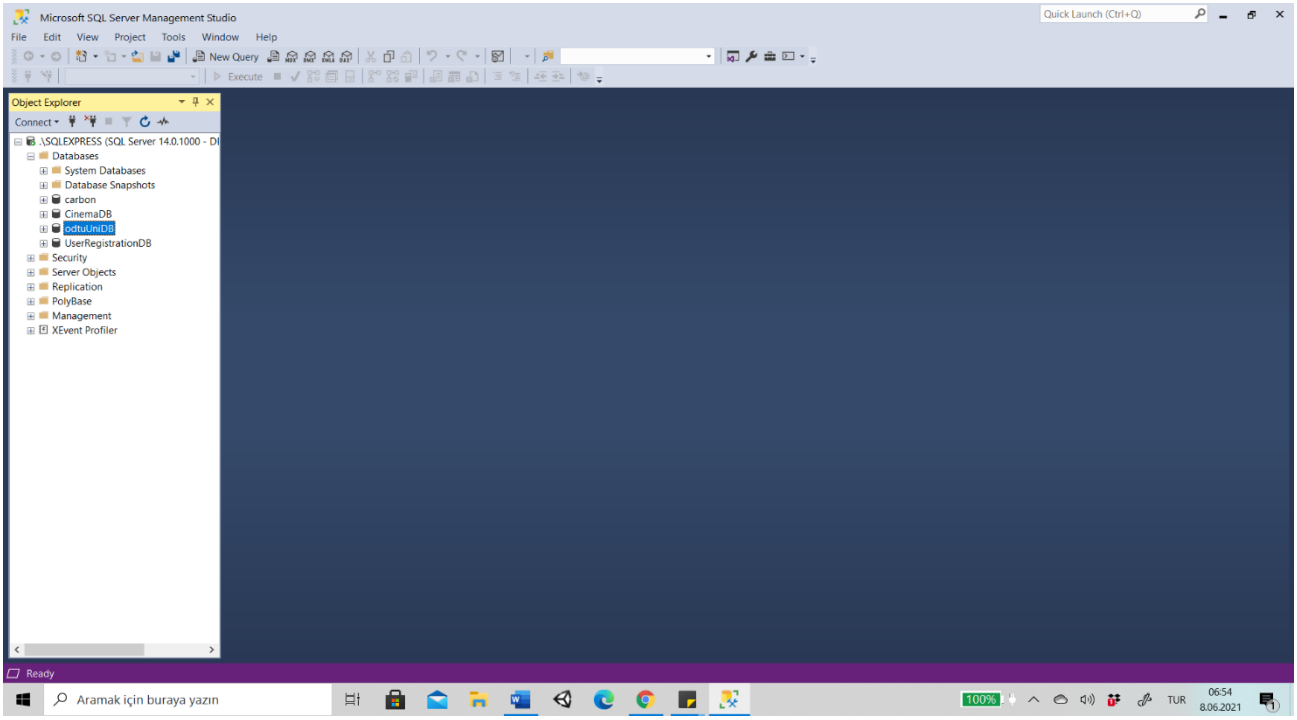
a)

In Microsoft SQL Server Management Studio, we create “odtuUniDB”. And to apply “odtu university schema midterm solution” with Microsoft Server, first we implement reduction to relational schemas. We reduce odtu university schema midterm solution’s E-R diagram to relational schema representation. We simply convert E-R design to relational design and with converting relational design, we turn each strong entity set, each weak entity set and each relationship set into a unique relational schema which can be reality in a database server system.

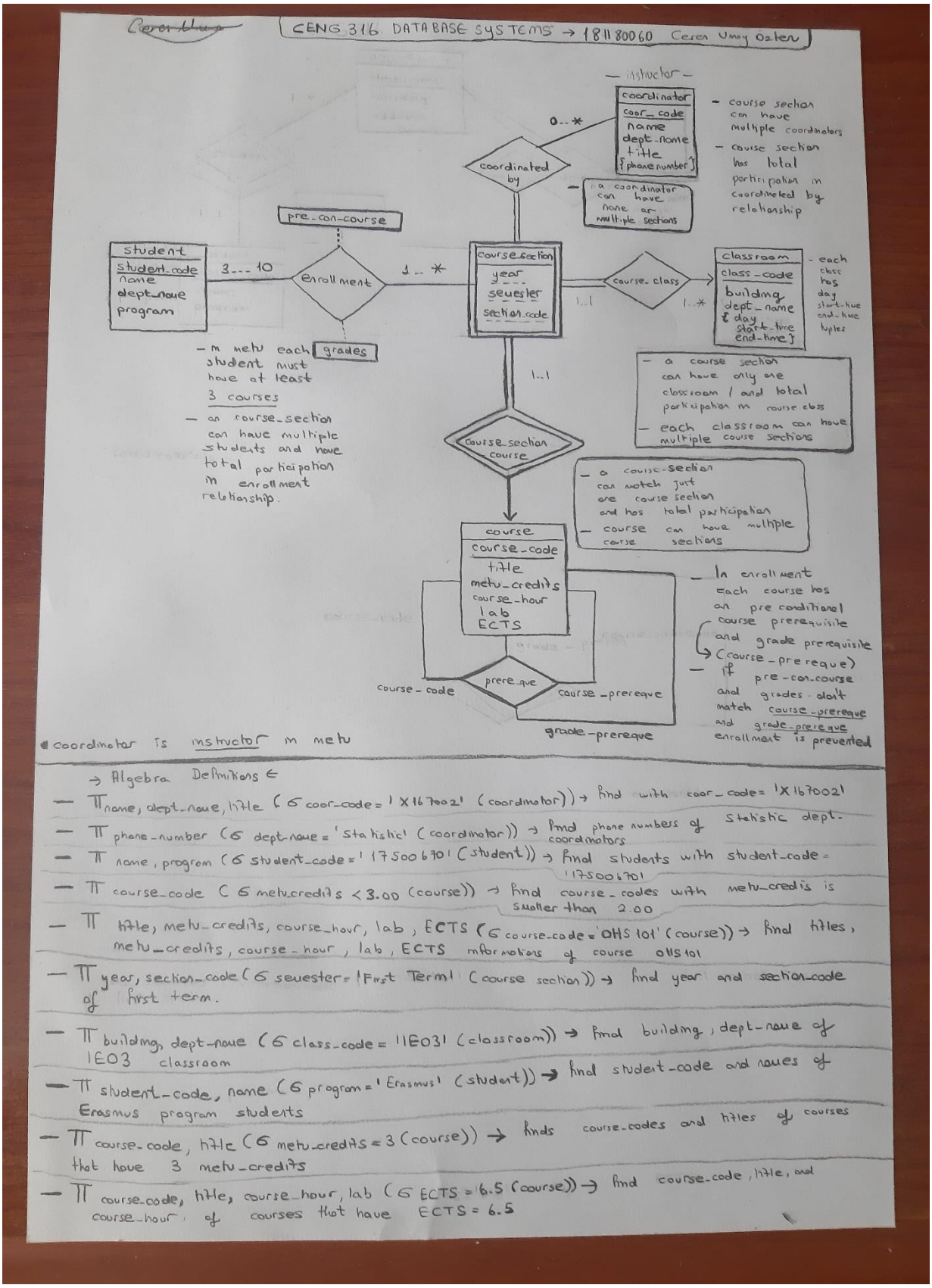
- server type: \. SQLEXPRESS Microsoft SQL server
- server IDE: Microsoft SQL Server Management Studio 18

In odtu university schema midterm solution first we turn each strong entity set -student, coordinator, course, classroom- into a unique relational schema. And implement whether SQL query language reduction or IDE table equivalent to each unique relational schema. SQL query language reduction and IDE table equivalent are recognized by Microsoft SQL server.

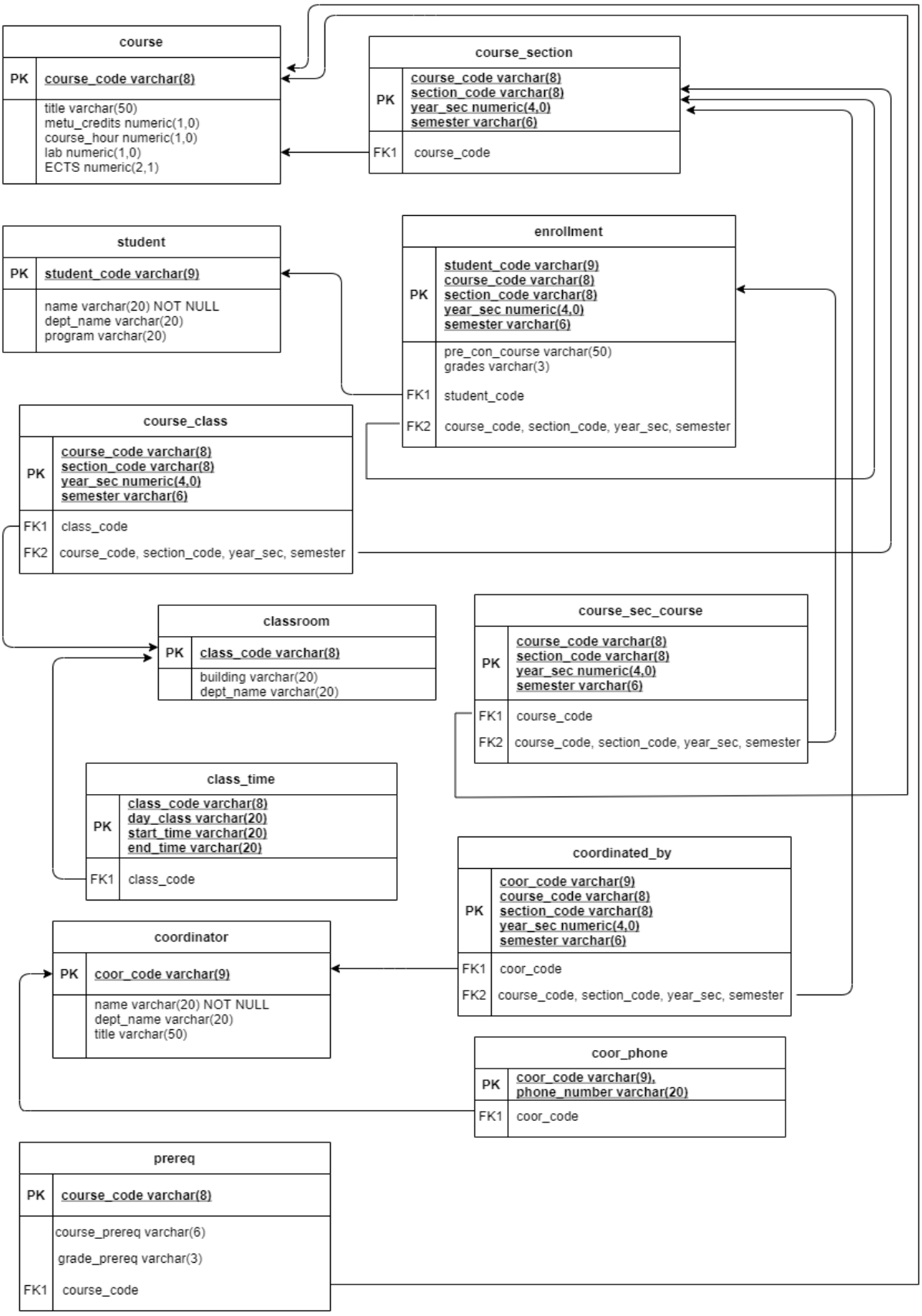
To create “odtuUniDB” database, Microsoft SQL Server Management Studio is opened and in opening window’s Object Explorer menu, “Databases folder link” is clicked. From Databases folder link, a box menu is opened and “new Database…” choice is clicked. Finally in new opening window “New Database”, name of the database is inputted as “odtuUniDB” and “OK” button is clicked.



Microsoft SQL Server Management studio 18 “odtuUniDB” screenshot

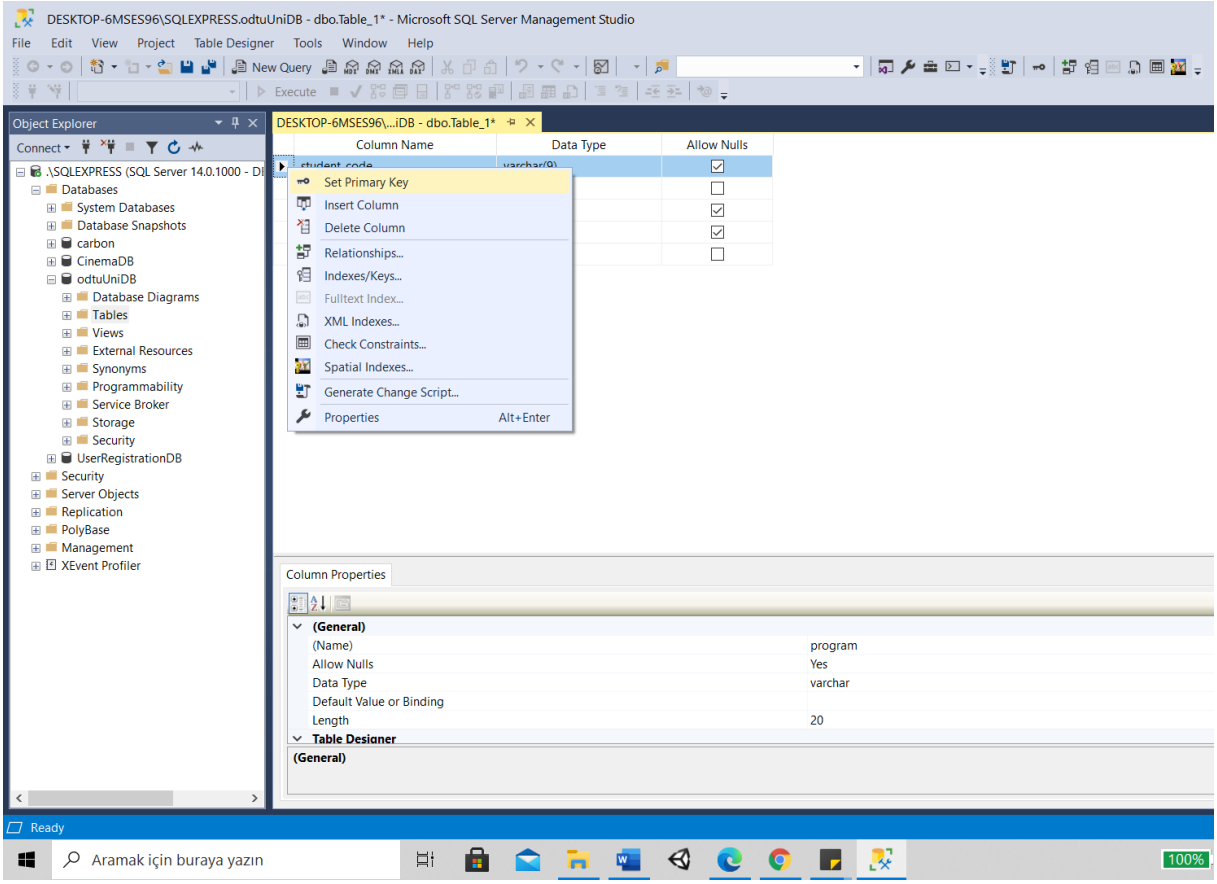


E-R diagram of odtuUniDB table schema



odtuUniDB table schema diagram

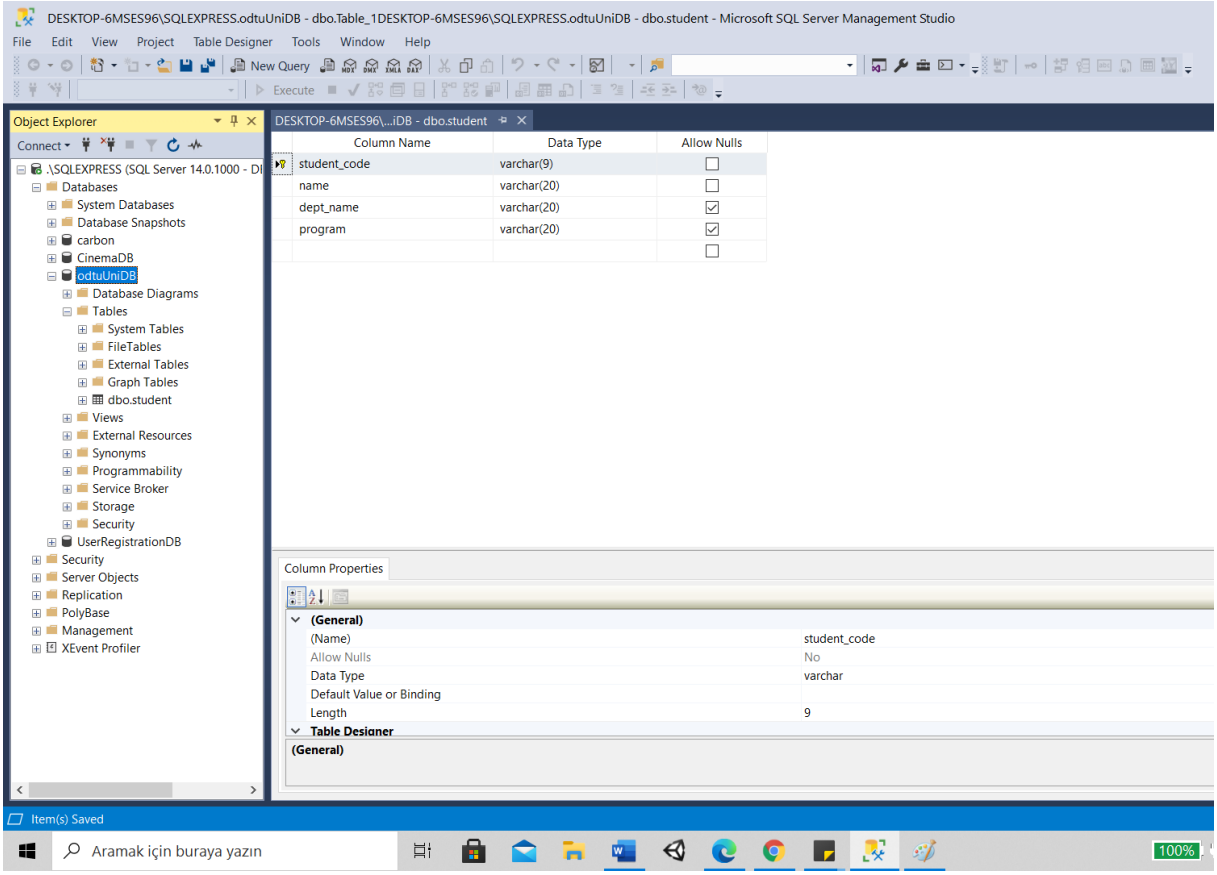
To create student, course, classroom and coordinator schemas' tables, Databases folder link is clicked and in opening new folder link field, odtuUniDB folder link is clicked. Then in opening new link field, Tables folder link is right clicked and from box menu "new" and "Table.." choices is chosen.



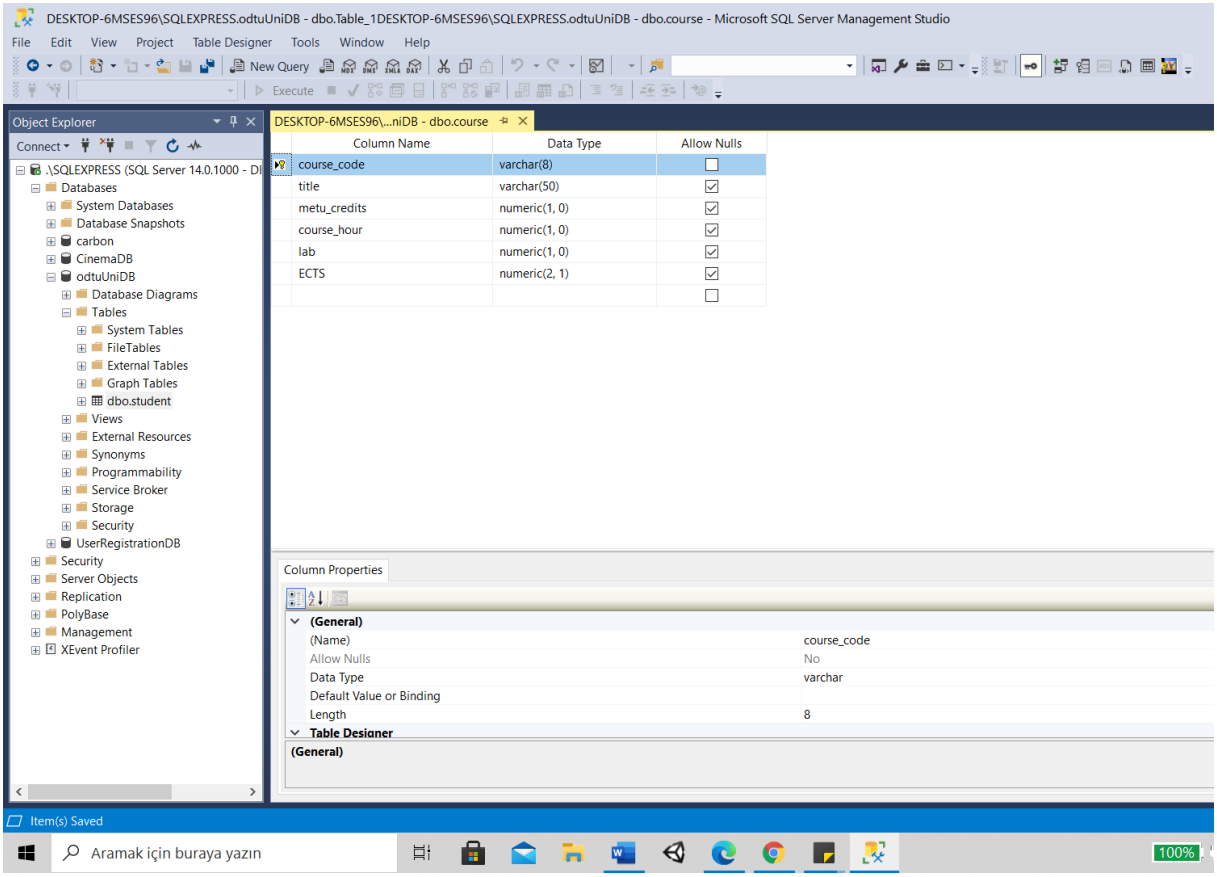
After "Table.." choice is clicked, dbo.table IDE is opened for creating realization of schema attributes and integrity constraints -not null, primary key, foreign key exec.-

In this screenshot student table's primary key is setted as "student_code". Integrity constraints "not null" is setted to name and primary key student_code attributes. Besides student table, course table's primary key is setted as "course_code", classroom table's primary key is setted as "class_code" and coordinator table's primary key is setted as "coor_code".

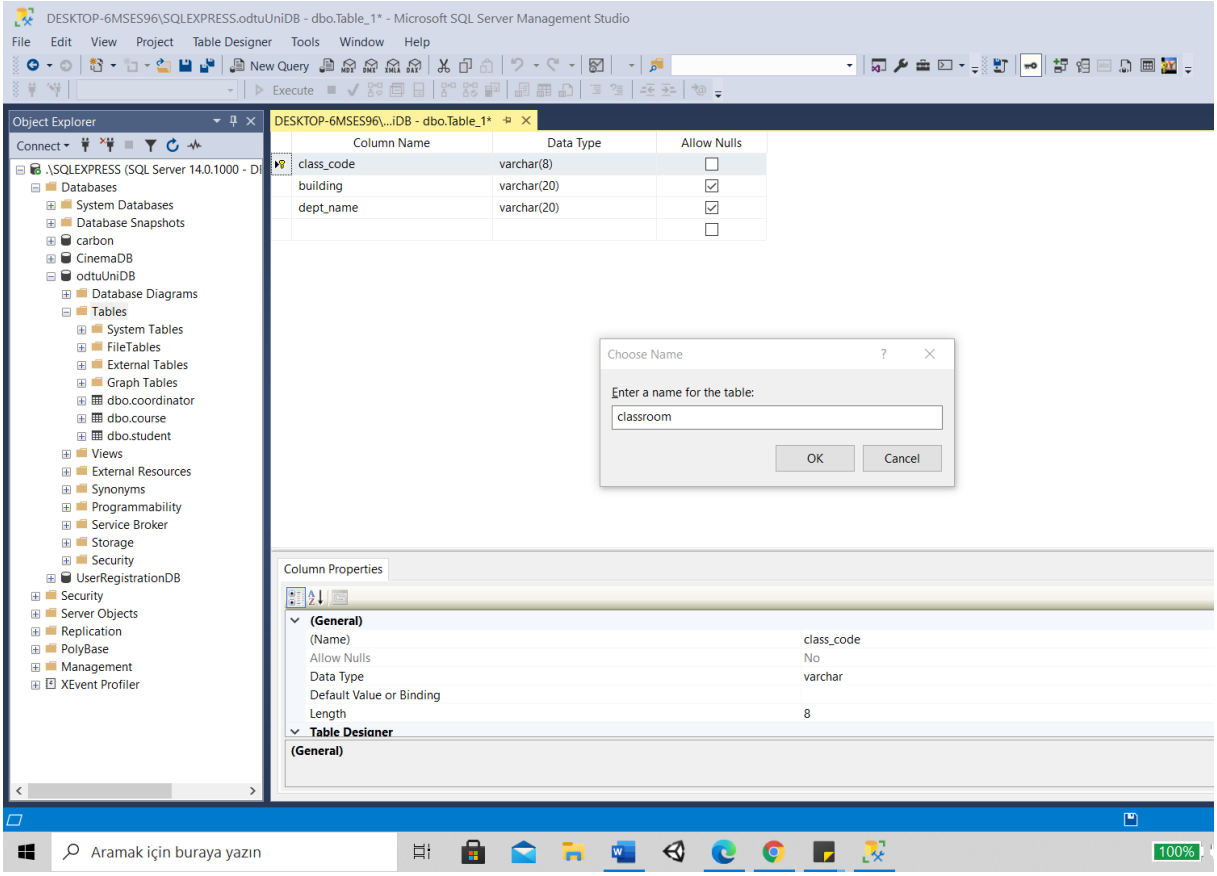
Especially student, course, classroom and coordinator schemas are equivalent to strong entity sets in E-R diagram. Also student, course, classroom and coordinator schemas have no foreign key integrity constraint, just primary key integrity constraint and "not null" integrity constraint.



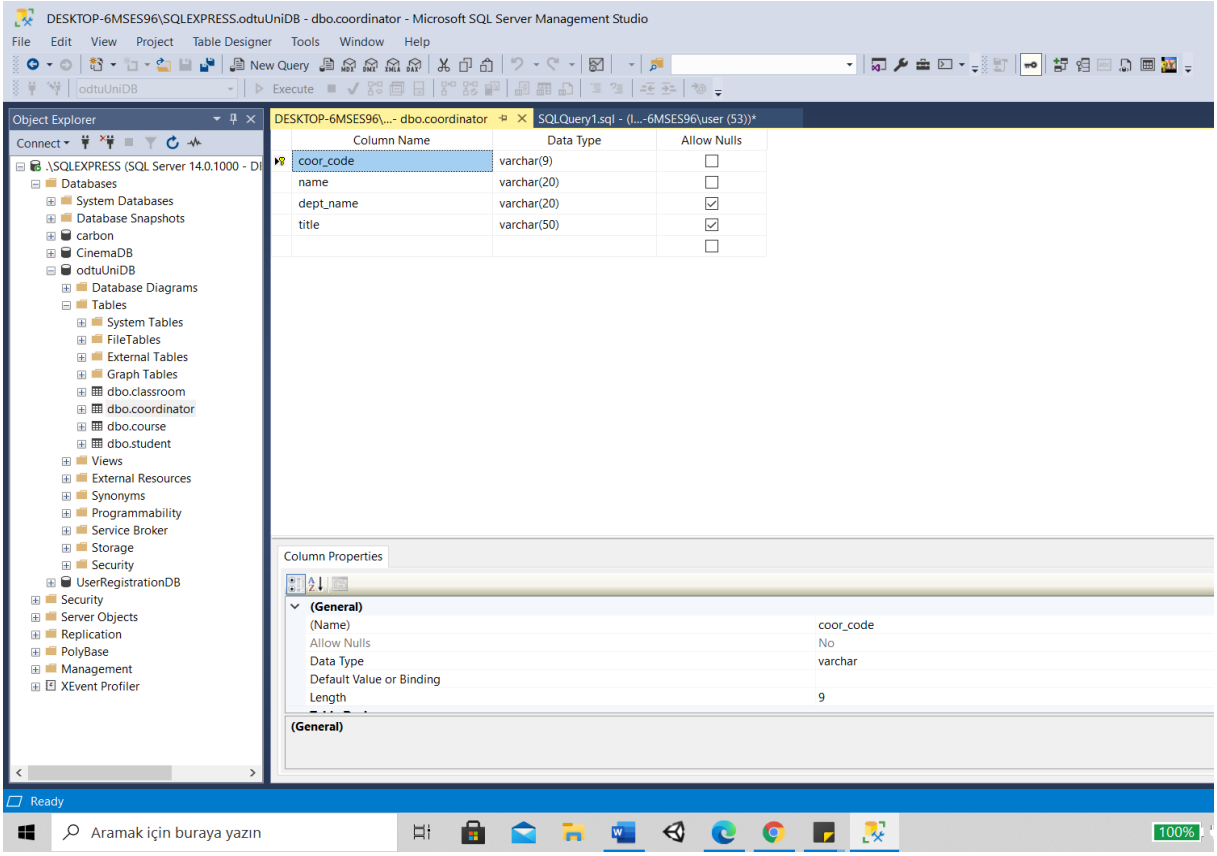
student table



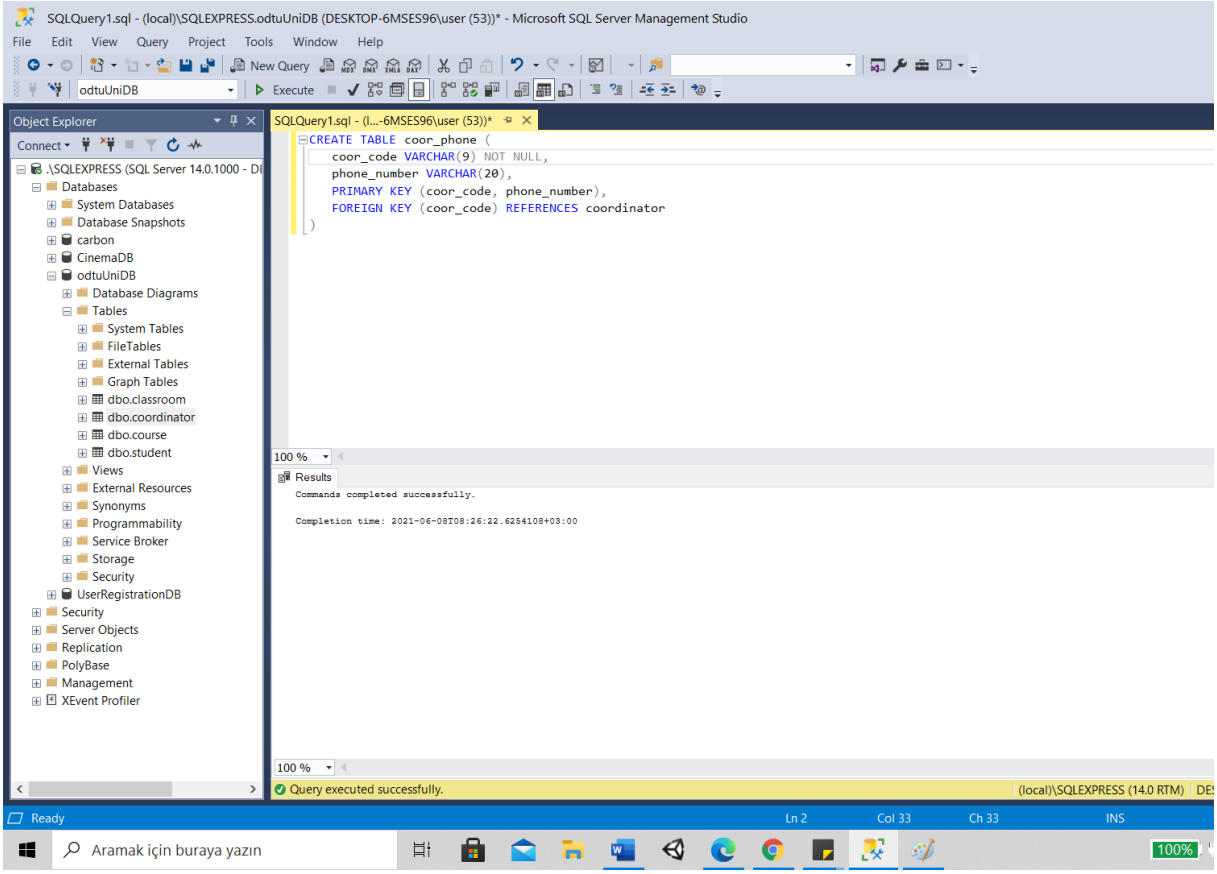
course table



classroom table



coordinator table

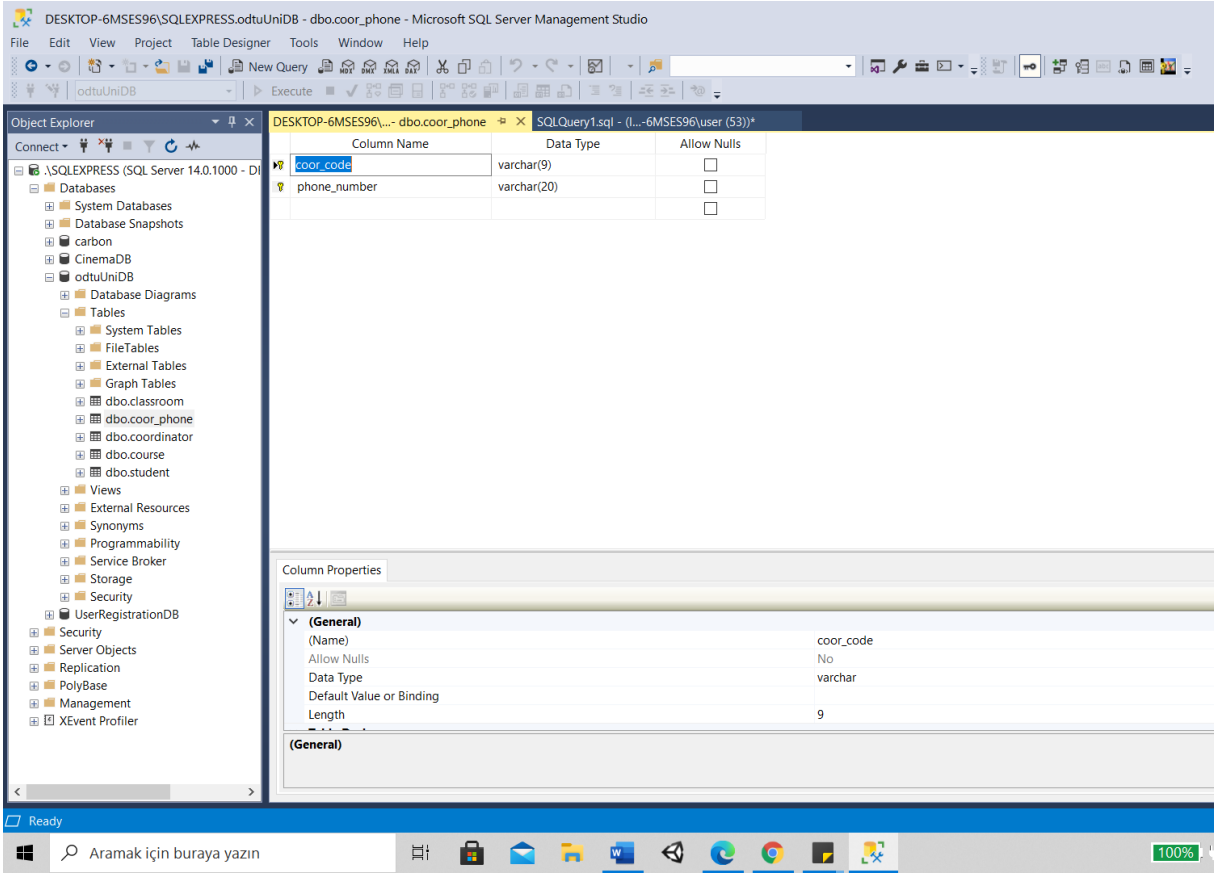


SQL Query language of coor_phone

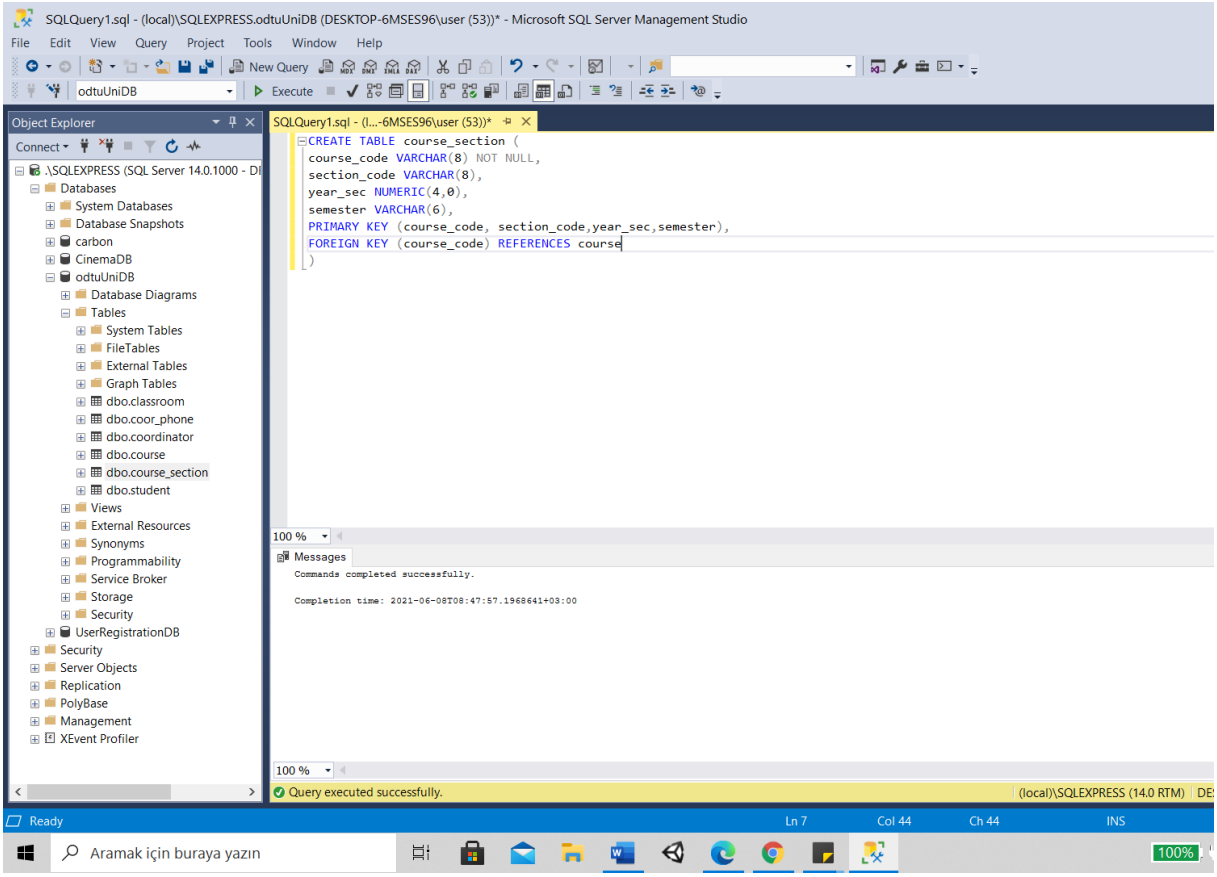
After finishing student table, course table, classroom table and coordinator table in dbo.table IDE, “New Query” button is clicked and SQLQuery coding field is opened and coor_phone schema, course_section schema, class_time schema, enrollment schema, course_class schema, course_sec_course schema, coordinated_by schema and prereq schema are coded in SQLQuery language.

Especially enrollment schema, course_class schema, course_sec_course schema, coordinated_by schema and prereq schema are all derived from relationship sets in E-R diagram. class_time and coor_phone are multivalued attribute equivalent schemas -phone_number, day_class, start_time and end_time-. course_section schema is weak entity set in E-R diagram which has identifying “course” strong entity set.

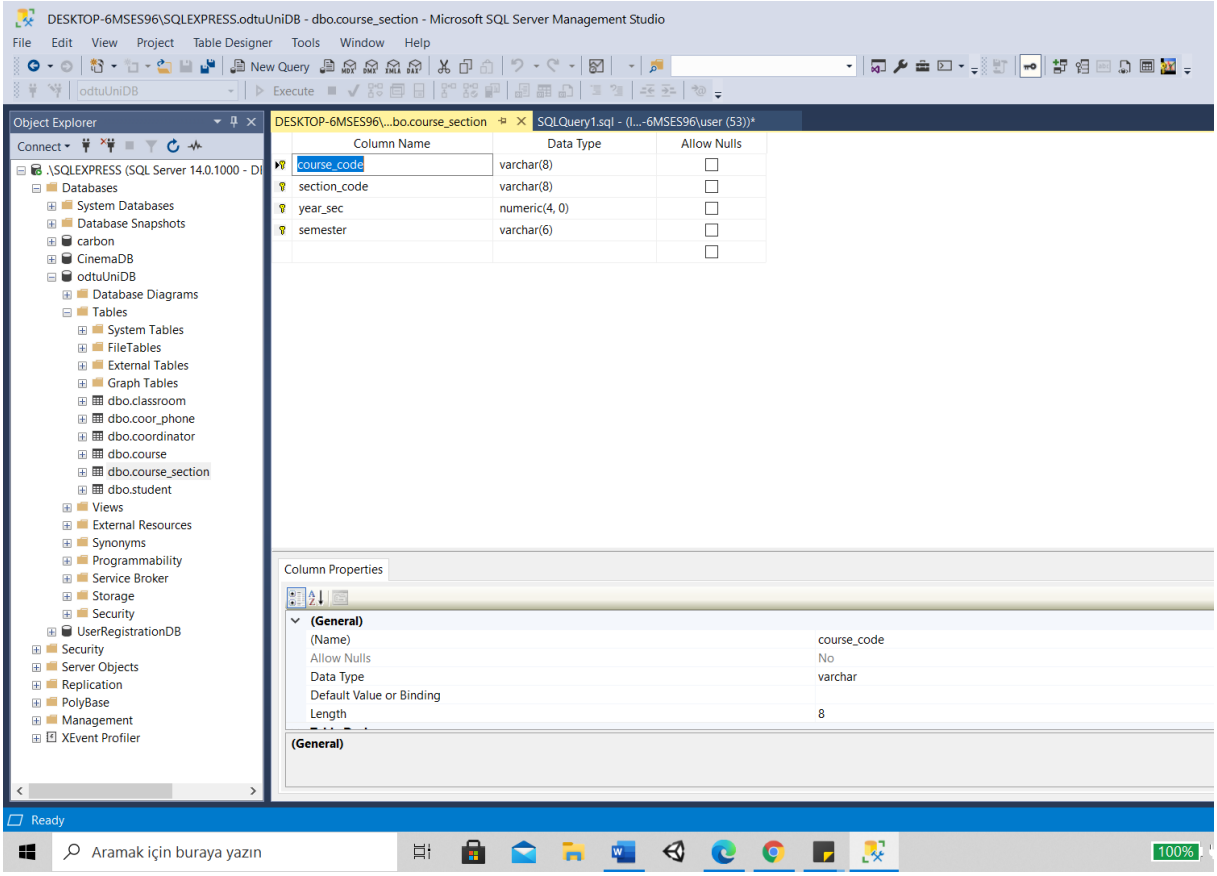
Coor_phone schema, course_section schema, class_time schema, enrollment schema, course_class schema, course_sec_course schema, coordinated_by schema and prereq schema have foreign key integrity constraints and primary key integrity constraints.



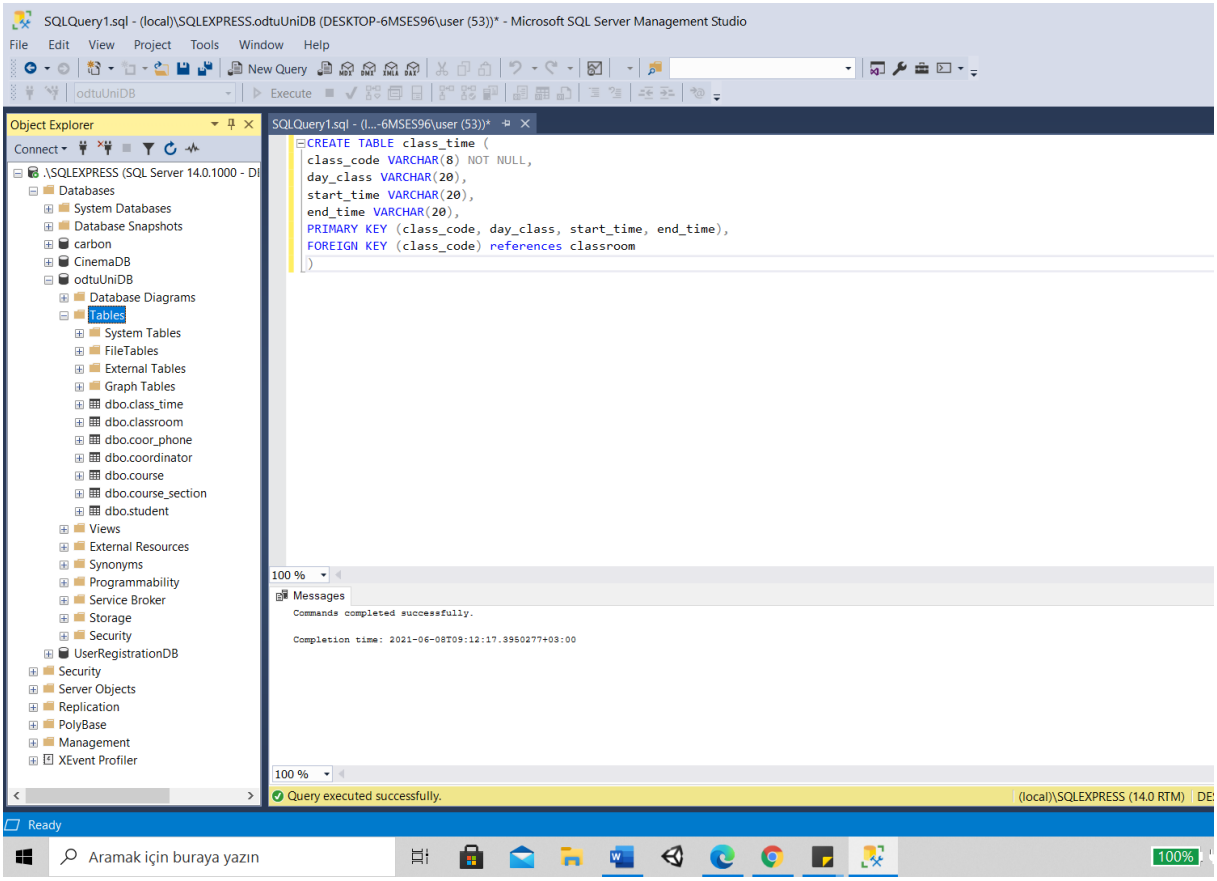
coor_phone table



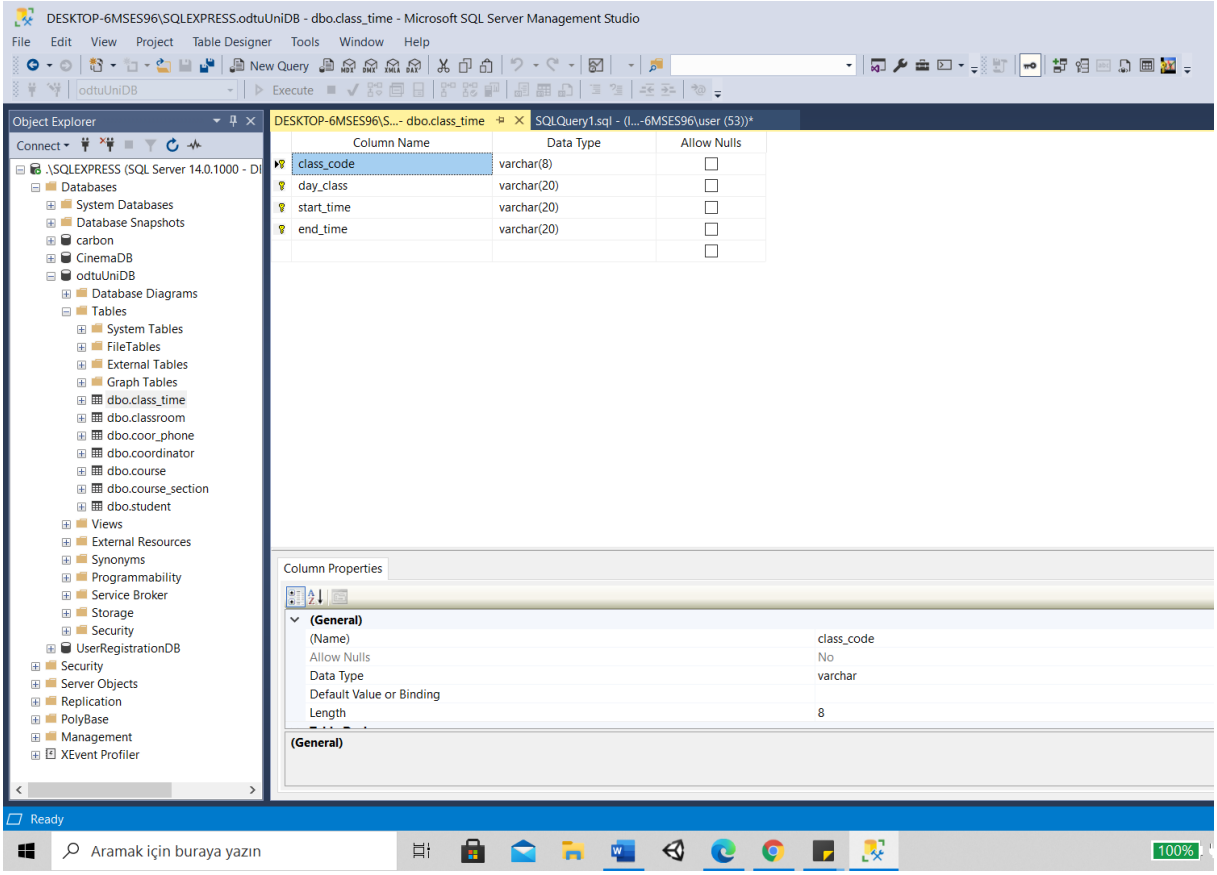
SQLQuery language of course_section



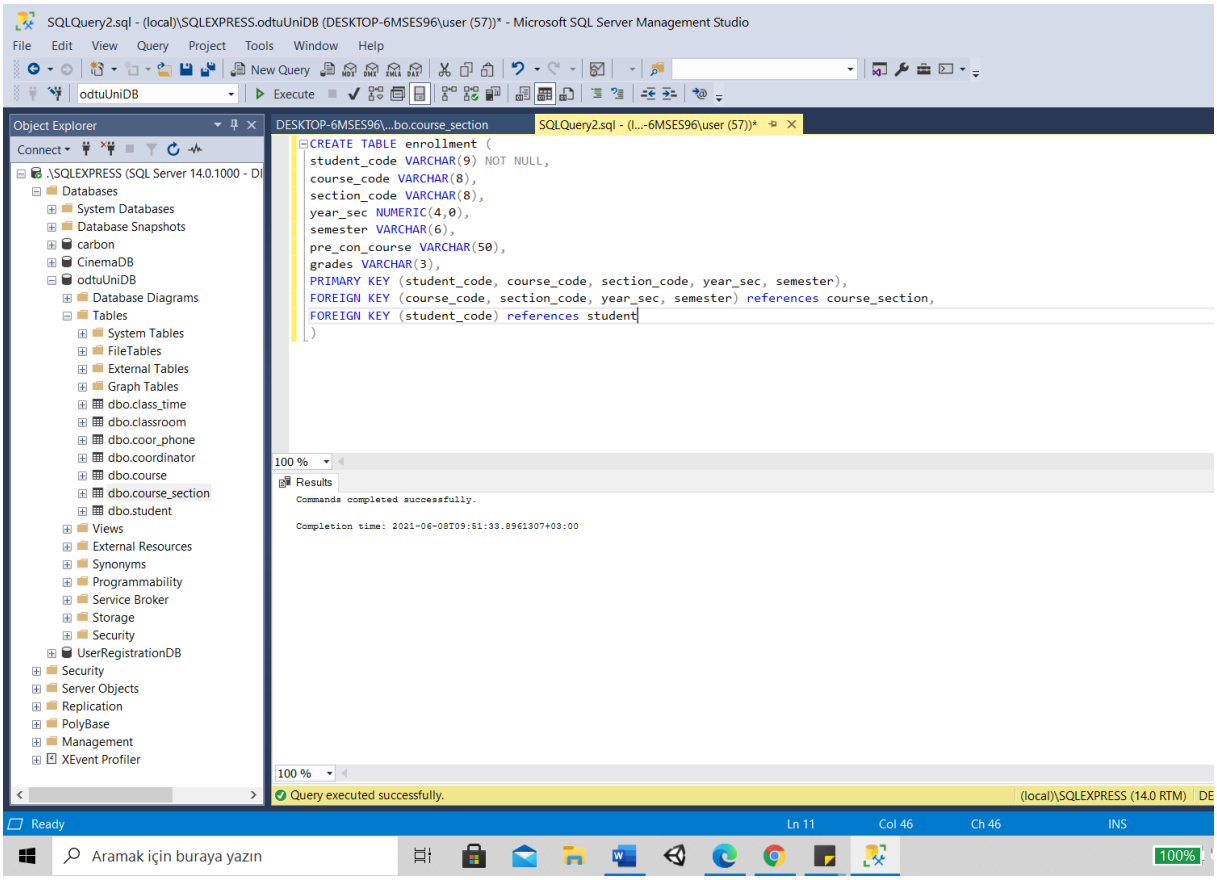
course_section table



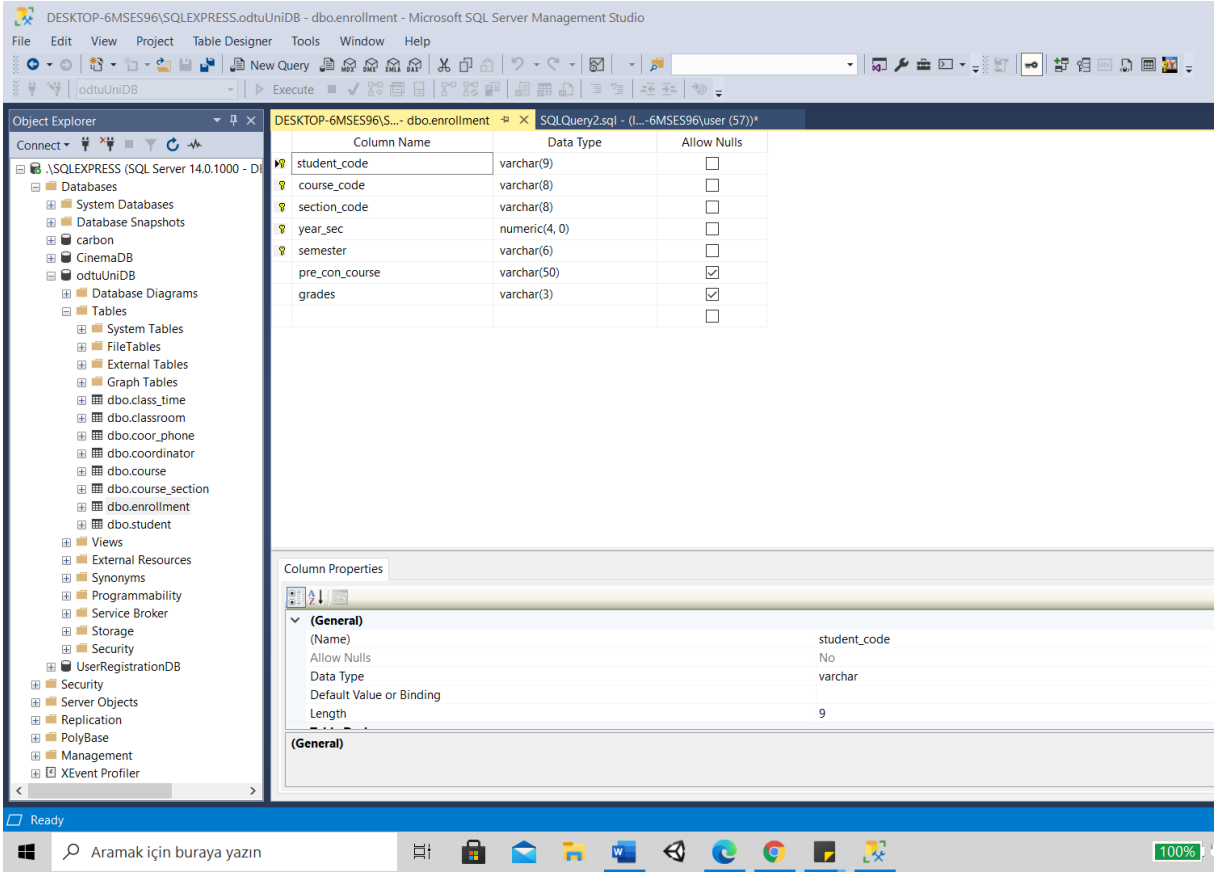
SQLQuery language of class_time



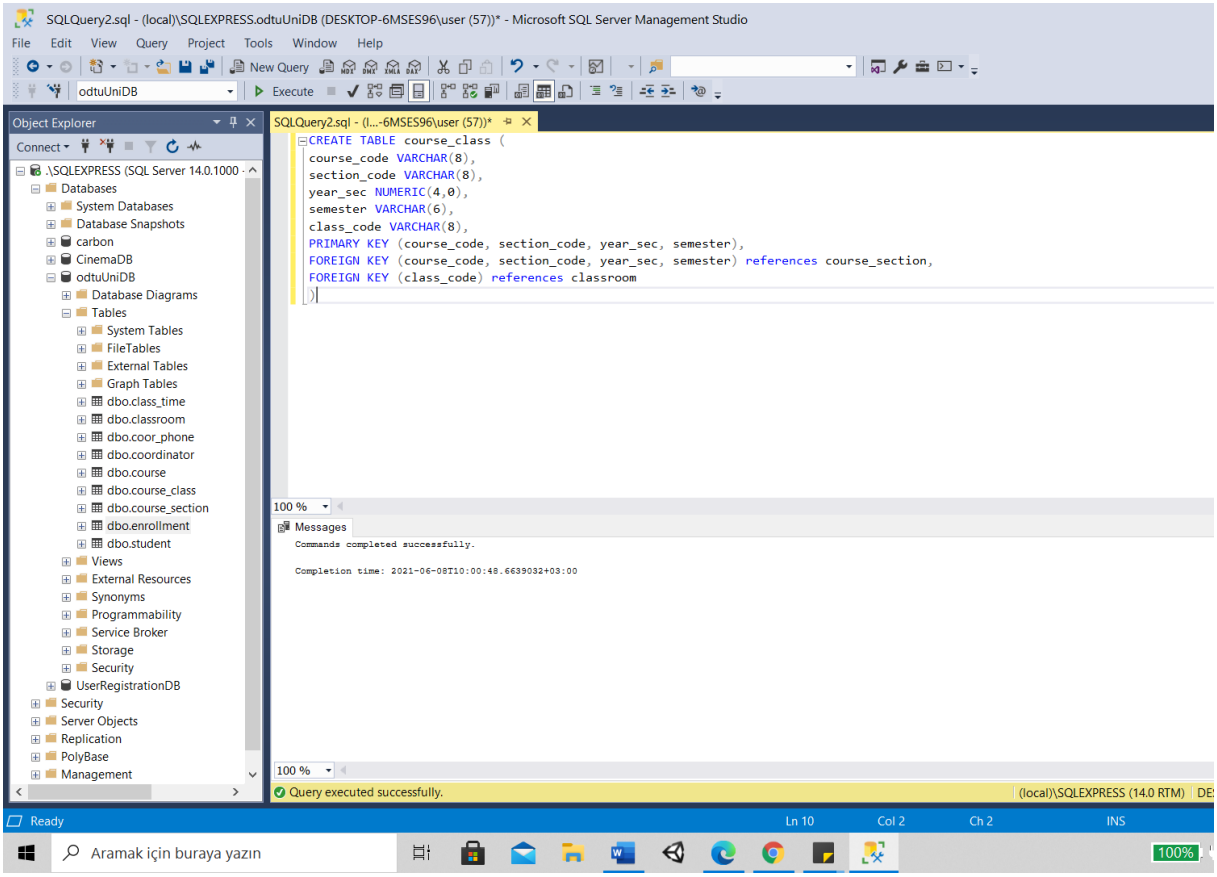
class_time table



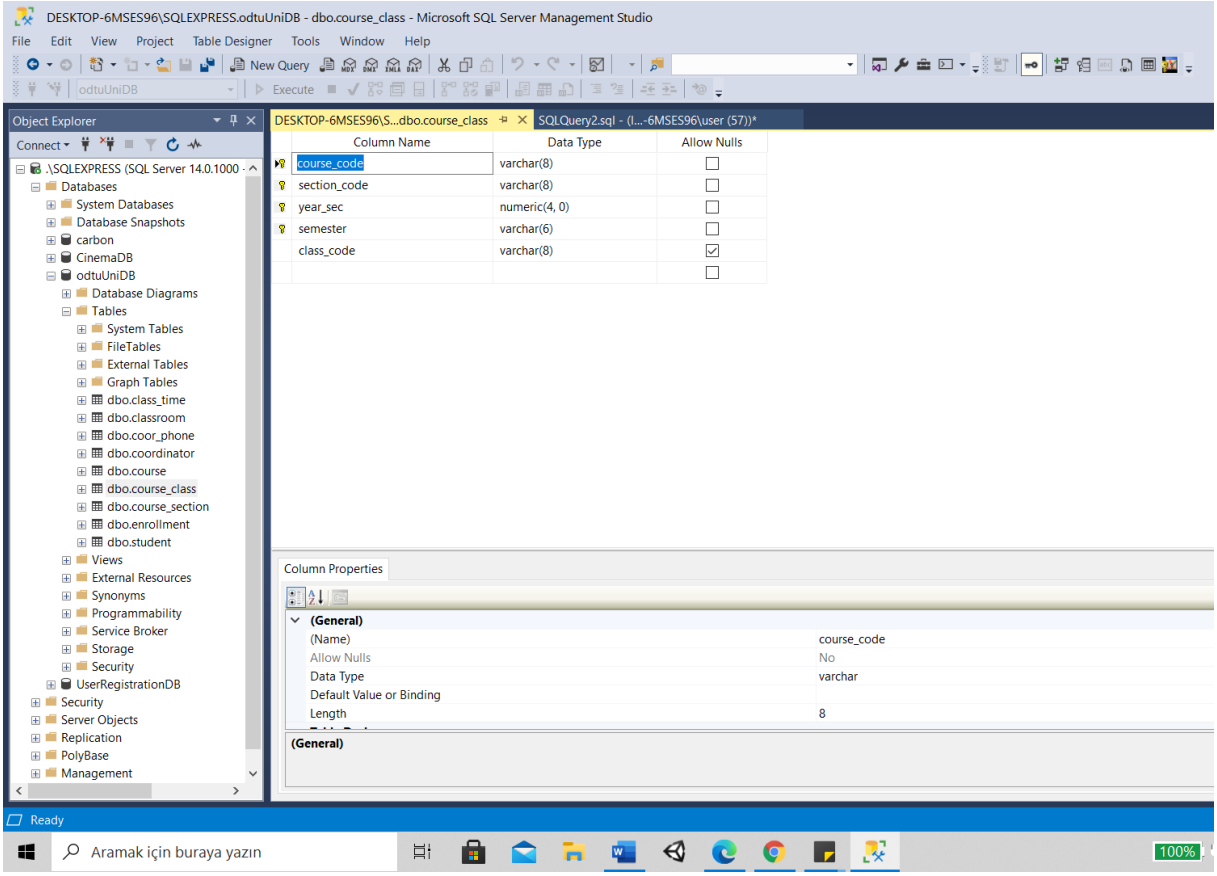
SQLQuery language of enrollment



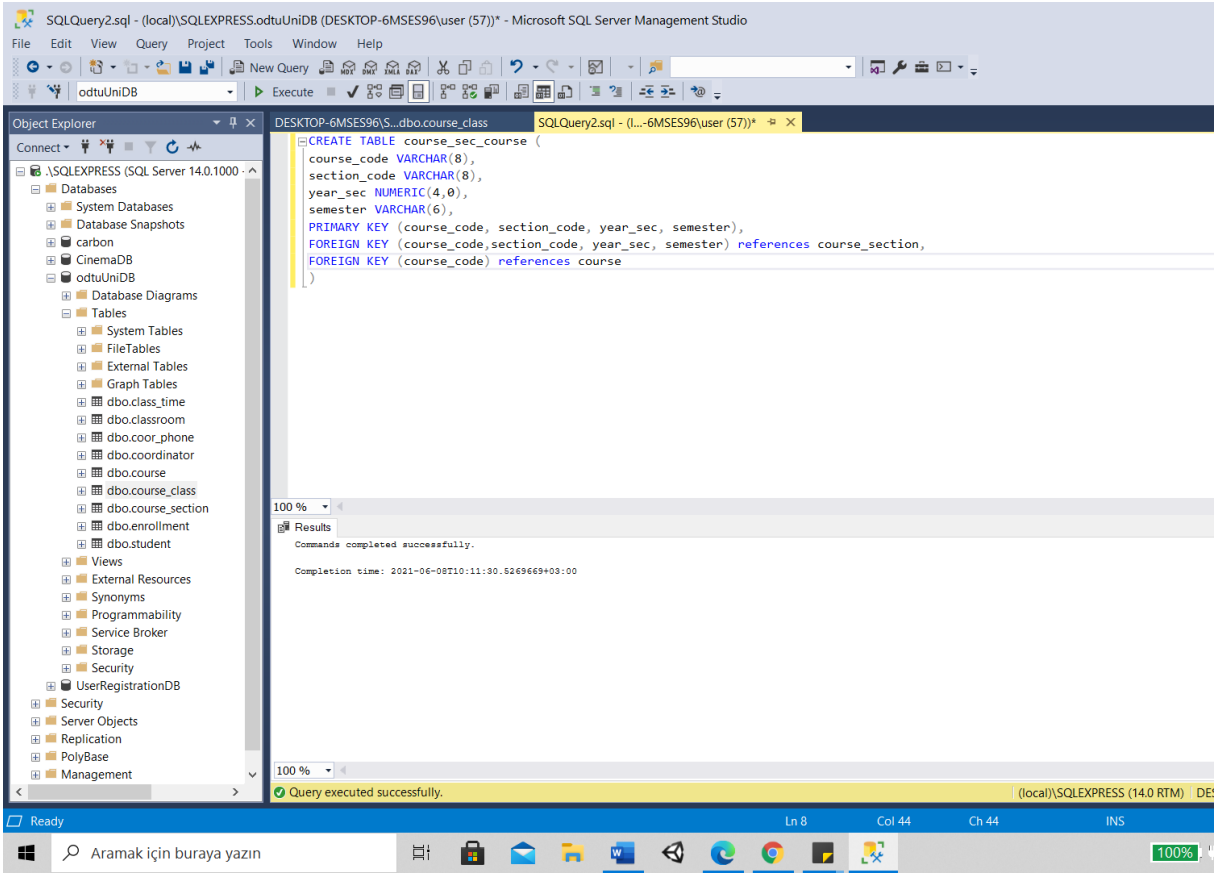
enrollment table



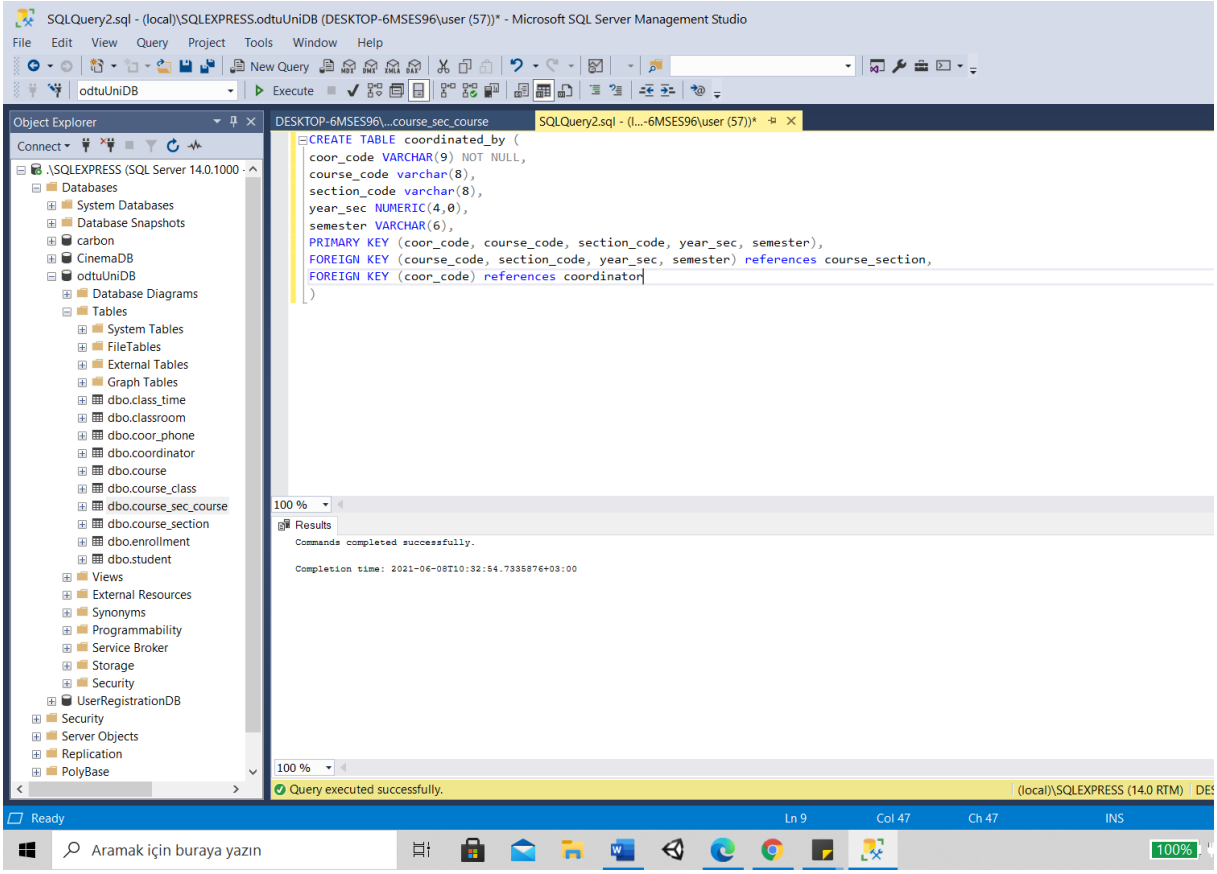
SQLQuery language of course_class



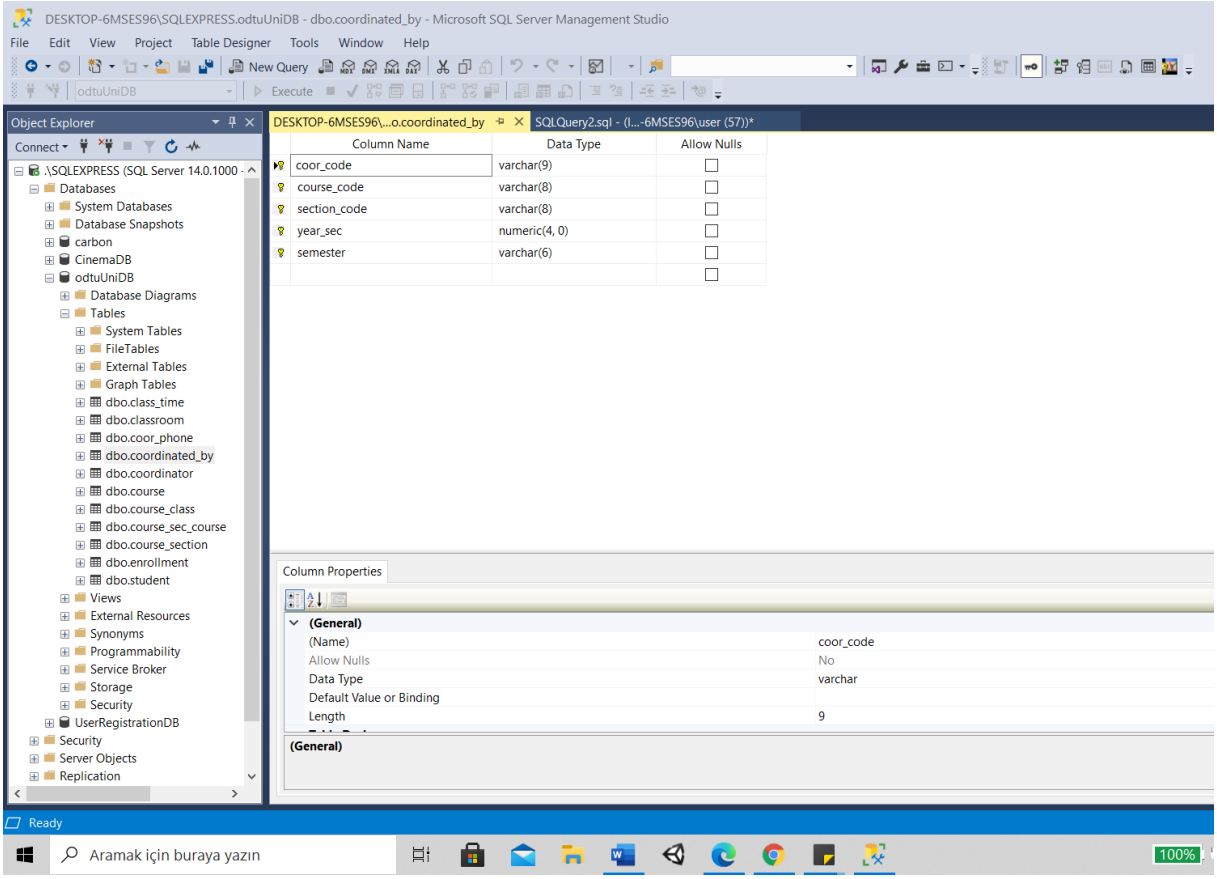
course_class table



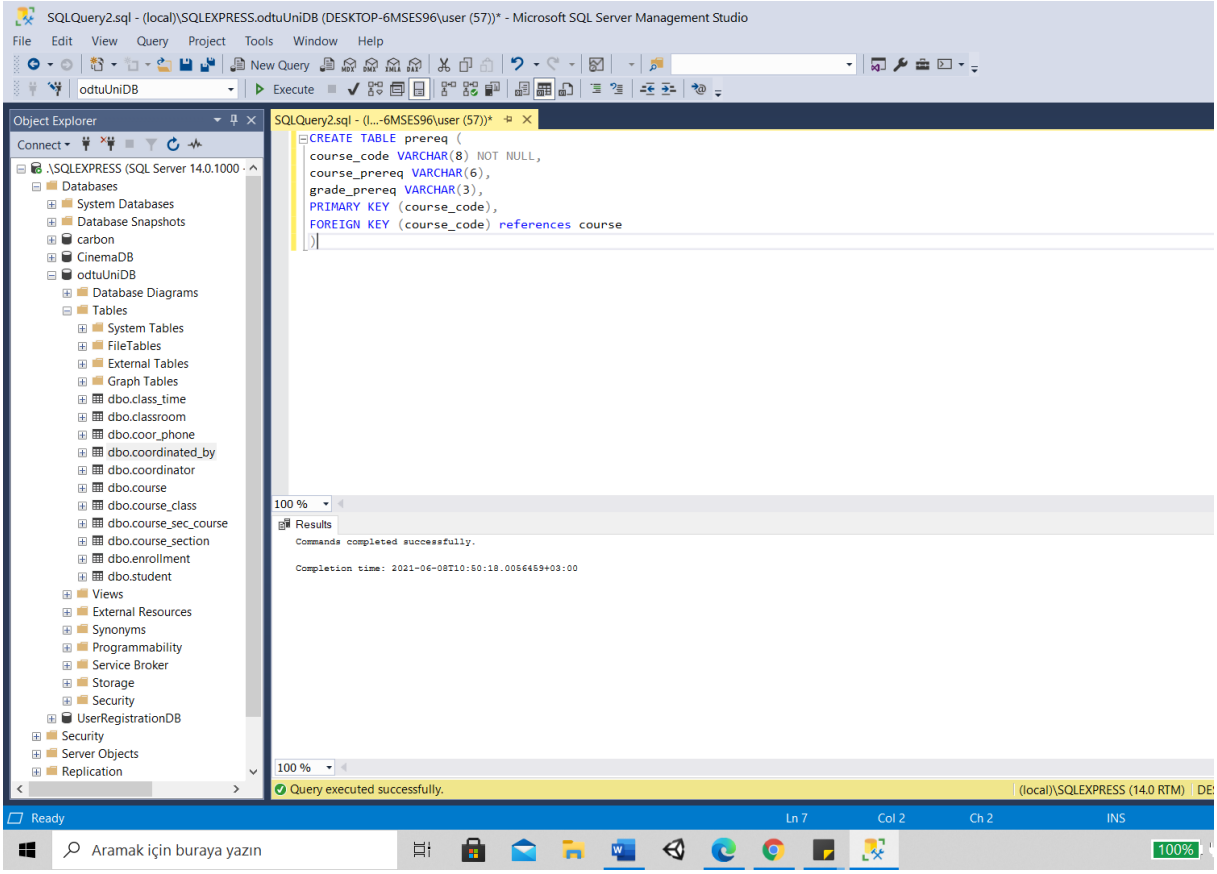
SQLQuery language of course_sec_course



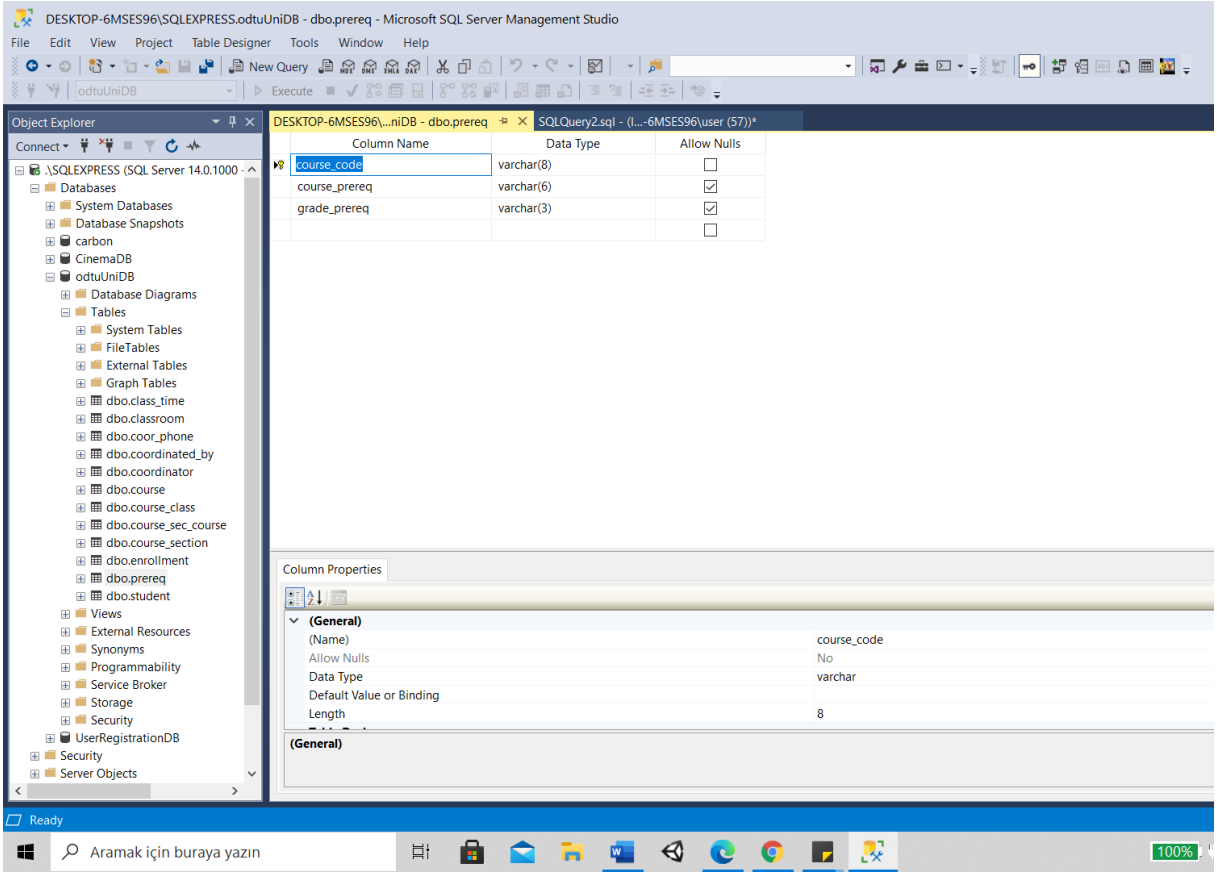
SQLQuery language of coordinated_by



coordinated_by table



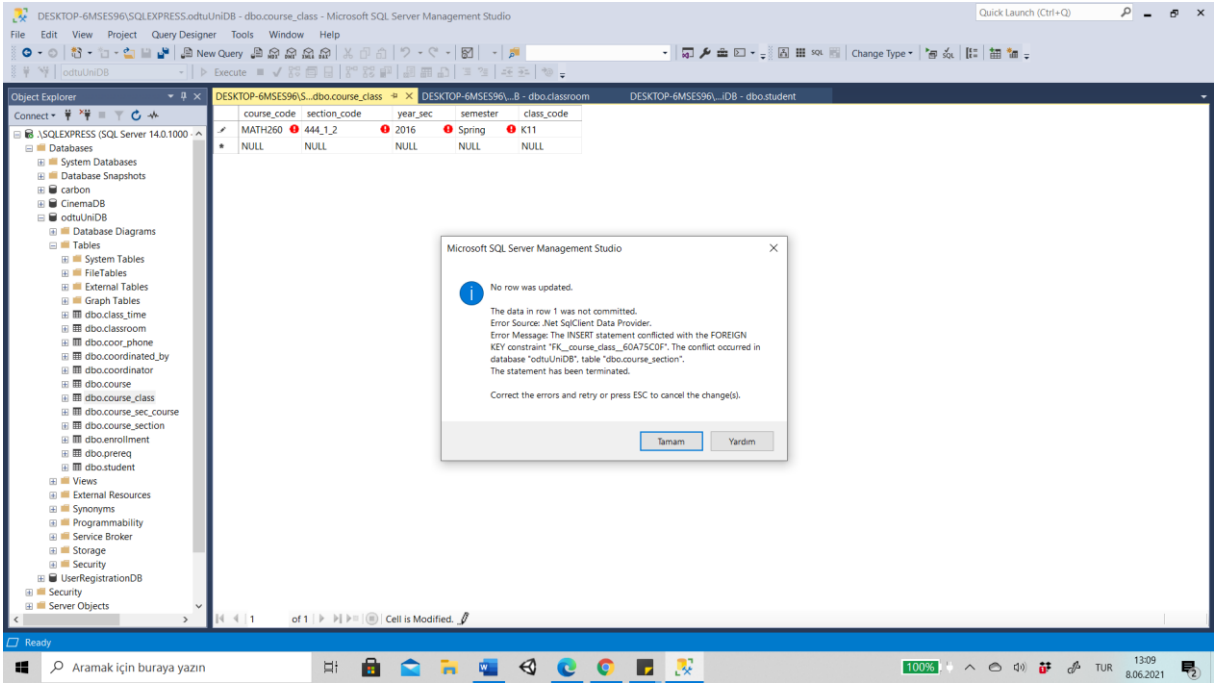
SQLQuery language of prereq



prereq table

Performans metrics

Atomicity	Multivalued attributes -phone_number, start_time, end_time, day_class- are converted to atomic table schema attributes	Very good
Durability	During commit failure course_class table schema's information has not gone	good
Lesser Redundancy Capacity	course_code, section_code, year_sec, semester attributes are overly repeated	Not good
Consistency	course-course_section-course_class queue is obligation during a new course adding. Without adding a new course to course table schema, that new course can not be added to course_section -can not commited- Also that new course's class_code attribute must be exist in classroom table schema, before adding to course_class table schema	Very good
Speed	Commit speed is considerably fast during data commiting	good



course_class schema's foreign key integrity constraint does not let a new data input without course_section reference match

b)

Normalization, is achieved by structuring a relational database in accordance with a series of normal forms in order to reduce data redundancy and improve data integrity There are three type normal forms in basic:

- First Normal Form (NF1)
- Second Normal Form (NF2)
- Third Normal Form (NF3)

In first normal form, each attribute has ‘atomic’ data -no multivalued data types- and each schema table has a unique identifier primary key integrity constraint.

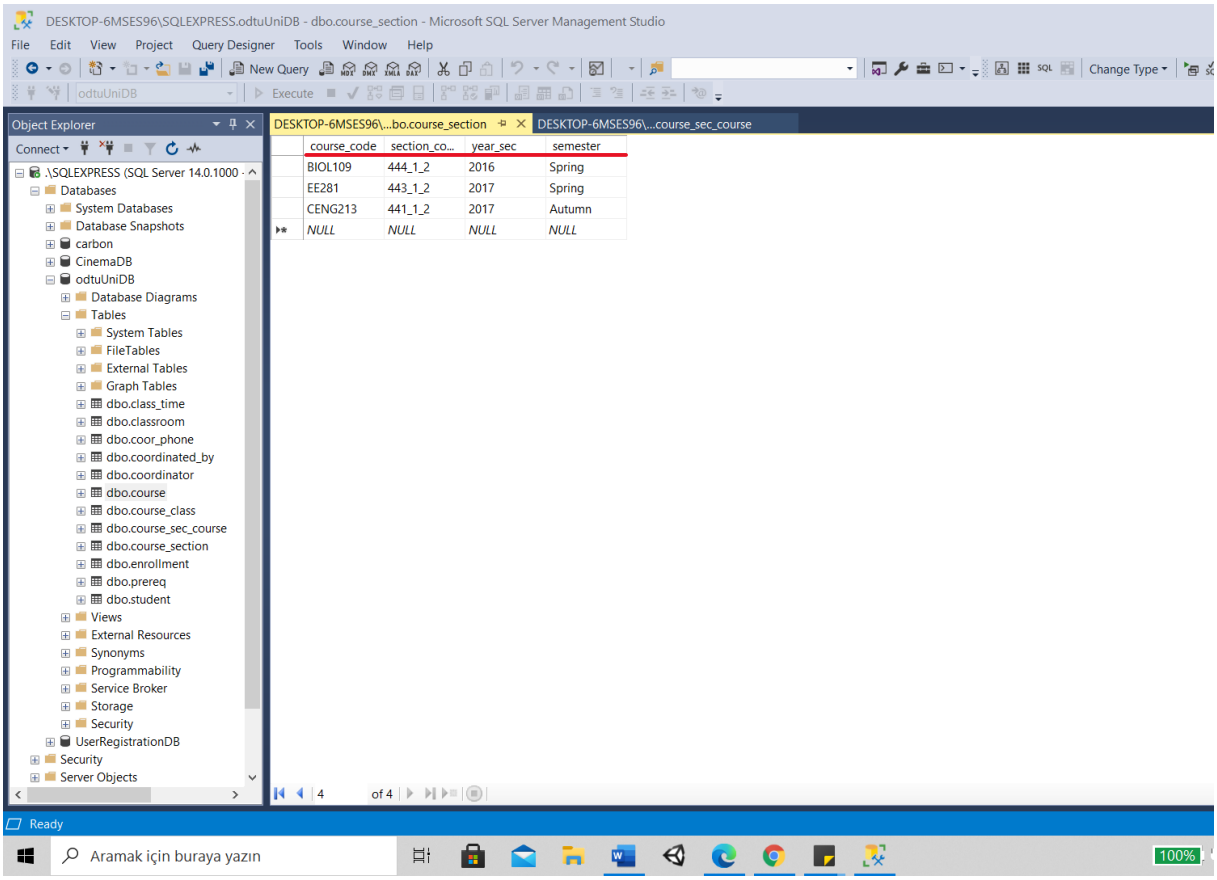
In second normal form, every non-prime attribute is dependent exclusively on the primary key split tables.

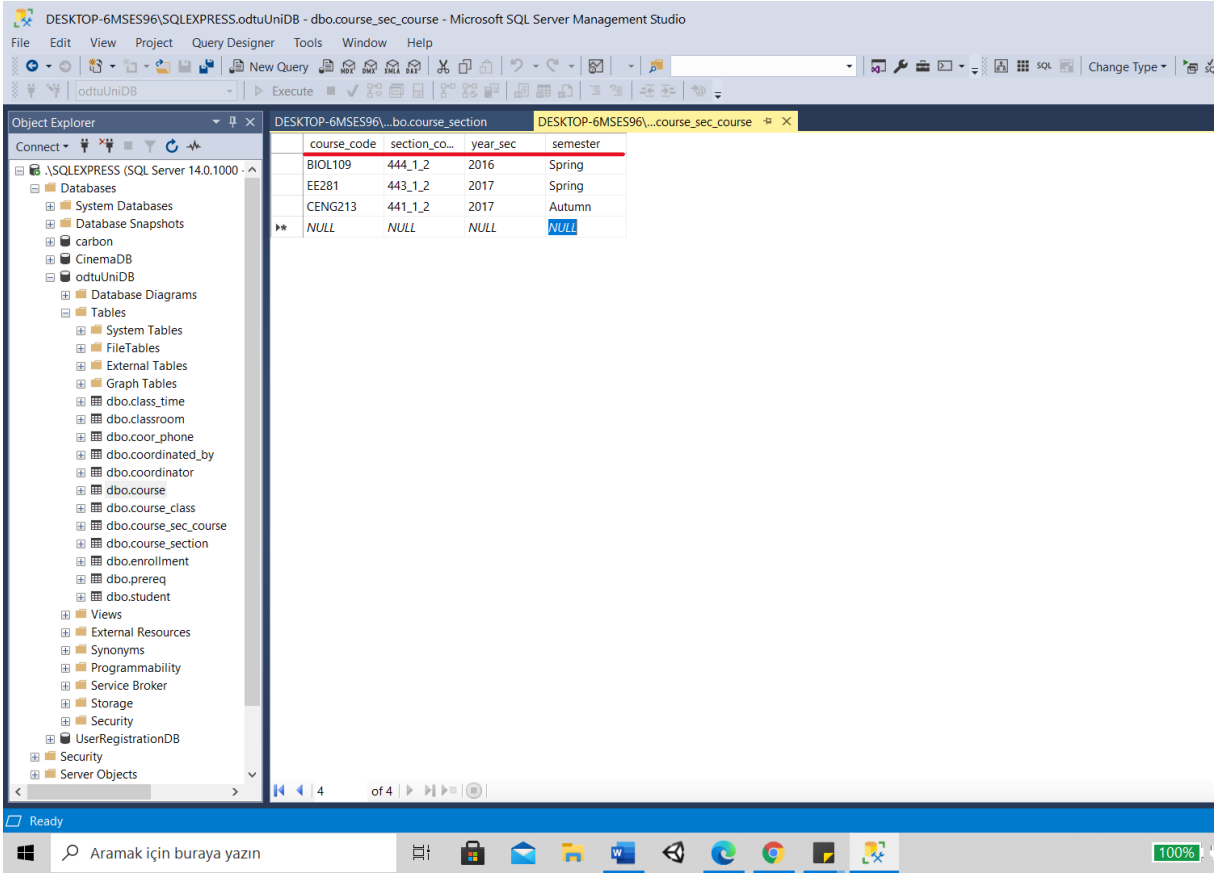
In third normal form, every attribute is determined only by primary key.

In odtuUniDB database schema, all multivalued data types – {day_class, start_time, end_time}, {phone_number}- are converted to table schemas with primary key integrity constraints and foreign key integrity constraints. Consequently, non-atomic data types are removed from database schema. First normal form’s atomic data obligation is served in odtuUniDB.

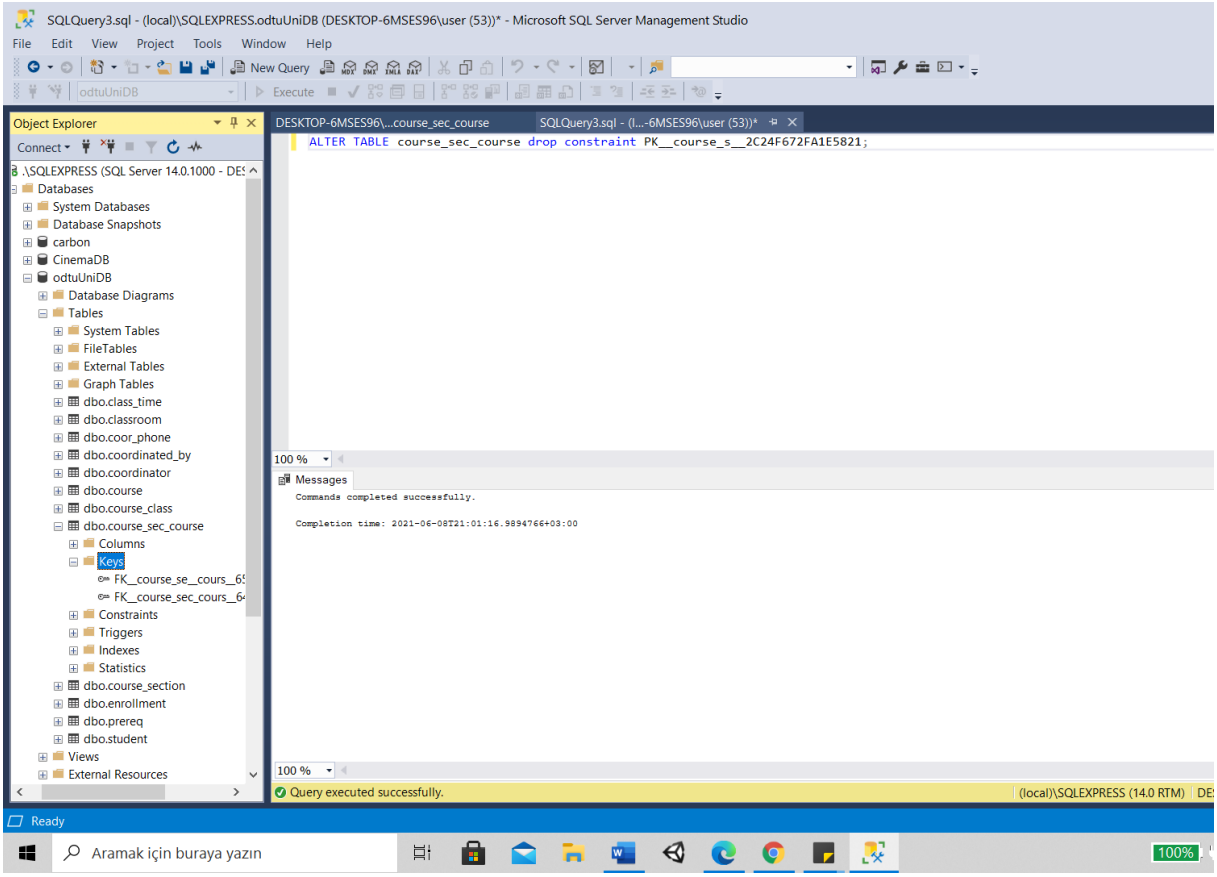
In odtuUniDB every atomic data type has a relationship with a primary key integrity constraint direct or indirect. Also there is no table schema without primary key integrity constraint -or primary key attribute- Therefore second normal form and in undirect way third normal form is achieved by odtuUniDB.

However, odtuUniDB database schema is highly weak on introducing less redundancy and it has highly redundant table schemas and redundant information. For instance derivation of a weak entity set *course_section* table schema has course_code, section_code, year_sec ve semester attributes. While *course_sec_course* table schema has also course_code, section_code, year_sec, and semester attributes. Especially *course_sec_course* is derived from relationship set including weak entity set *course_section* and *course_sec_course* causes information redundancy with additional *course_section* table schema. In order to prevent information redundancy we implement normalization on course_sec_course and drop course_sec_course table schema.



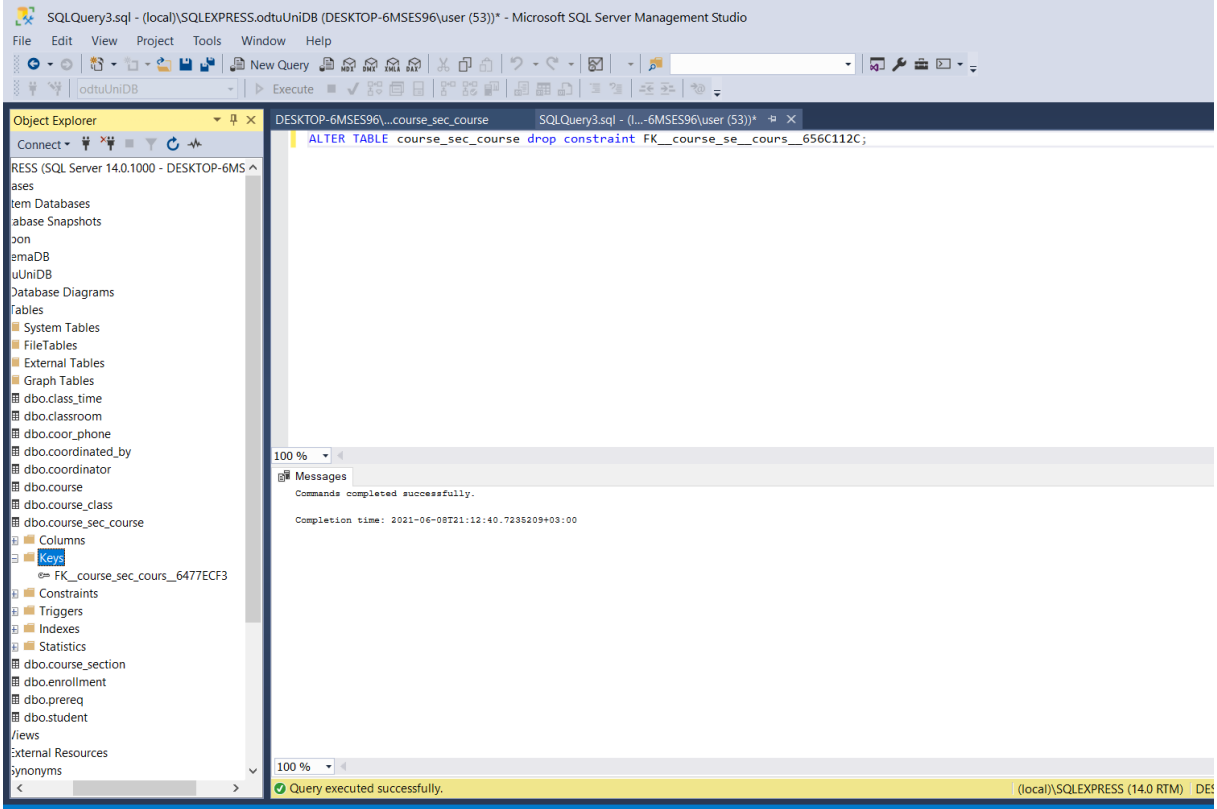


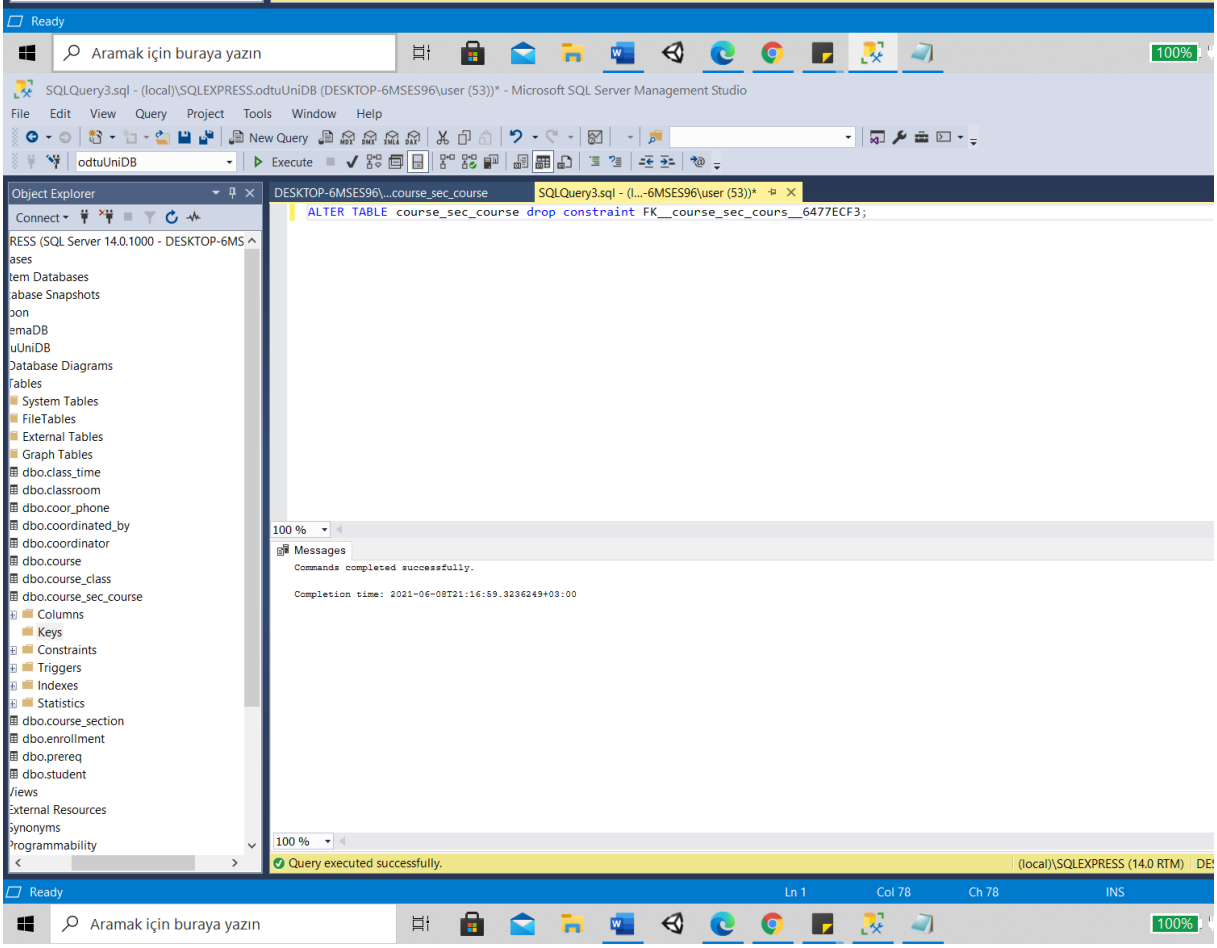
course_section table schema attributes vs course_sec_course table schema attributes

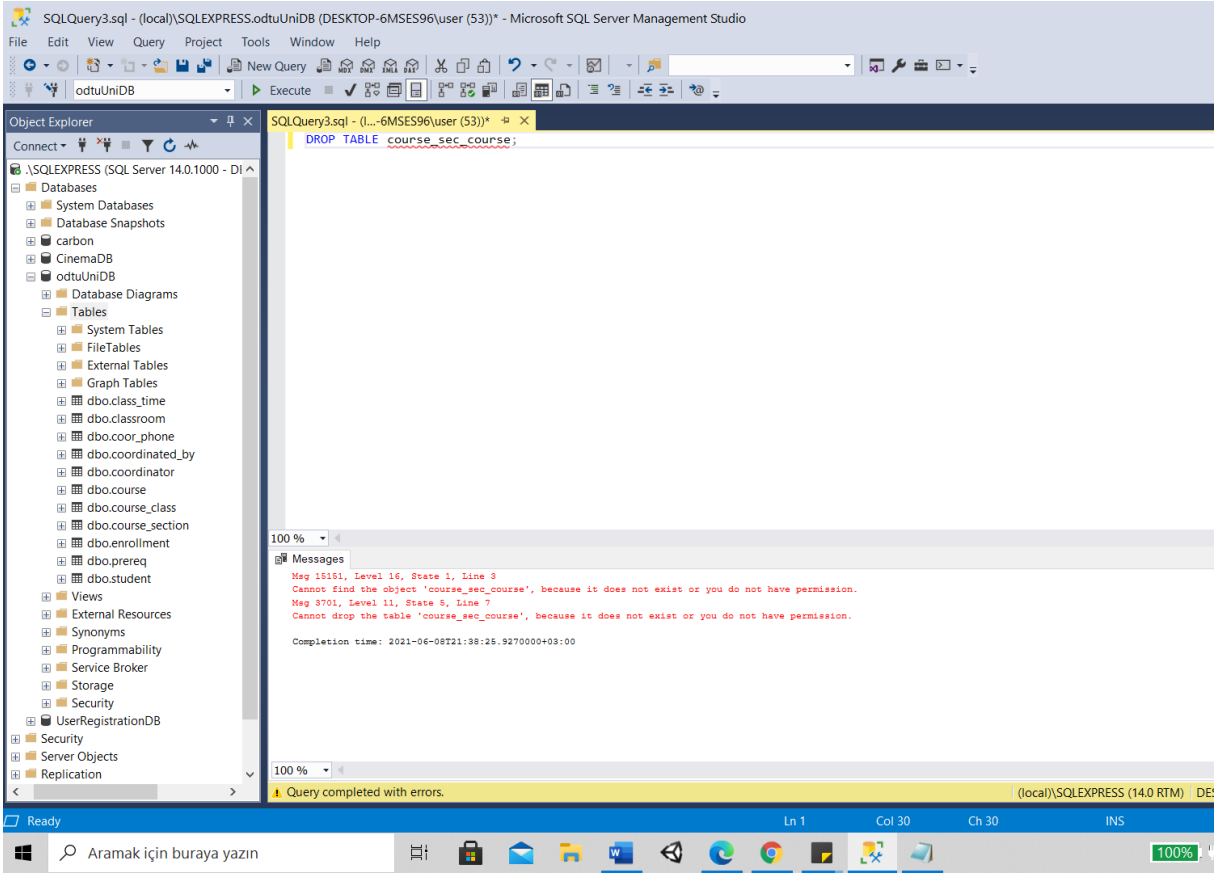


SQLQuery language for dropping primary key integrity constraint

In order to drop course_sec_course schema firstly we drop primary key and foreign key integrity key constraints.



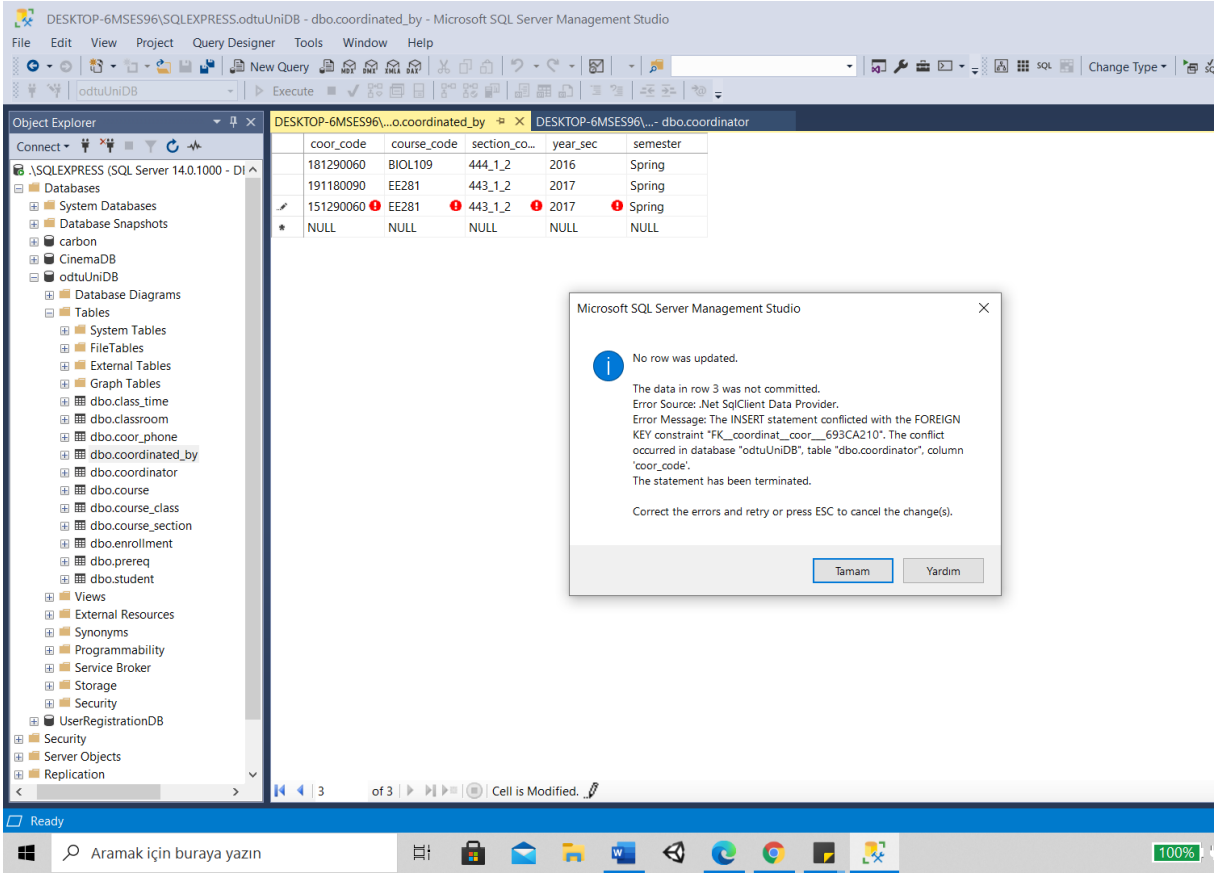




SQLQuery language for dropping the table -we grant ourself control on course_sec_course-

Performans metrics

Atomicity	multivalued attributes -phone_number, start_time, end_time, day_class are converted to atomic table schema attributes earlier. And good condition of atomicity has not been changed.	Very good
Durability	During foreign key matching failure, coordinated_by table schema's information has not gone.	good
Lesser Redundancy Capacity	redundant course_sec_course table schema is removed from odtuUniDB, therefore information redundancy in odtuUniDB database schema is decreased	good
Consistency	after redundant course_sec_course table schema is removed consistency of odtuUniDB database schema is persistent. Therefore when we try to add a new record which does not exist in coordinator table schema to coordinated_by table schema. coordinated_by table schema's foreign key integrity constraint prevents addition commit.	Very good
Speed	Commit speed is considerably fast during data commiting.	good



coordinated_by table schema's coor_code foreign key reference does not let commiting of data

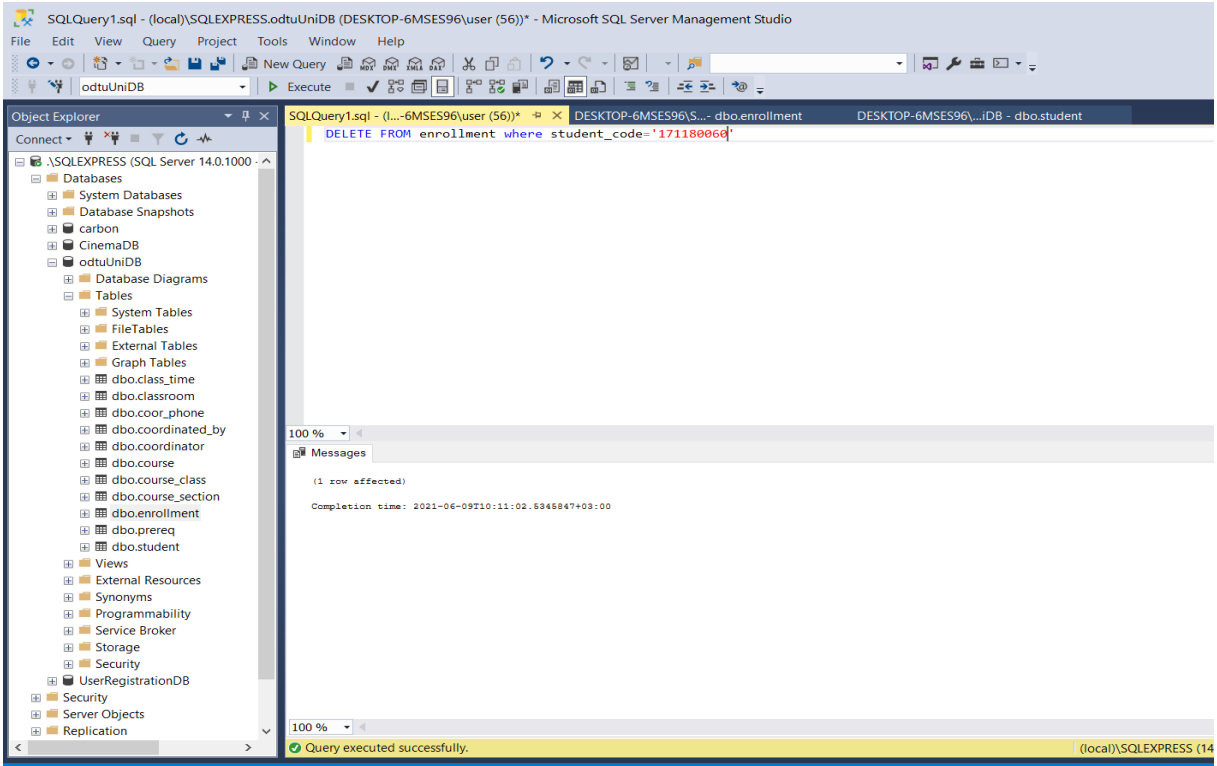
c)

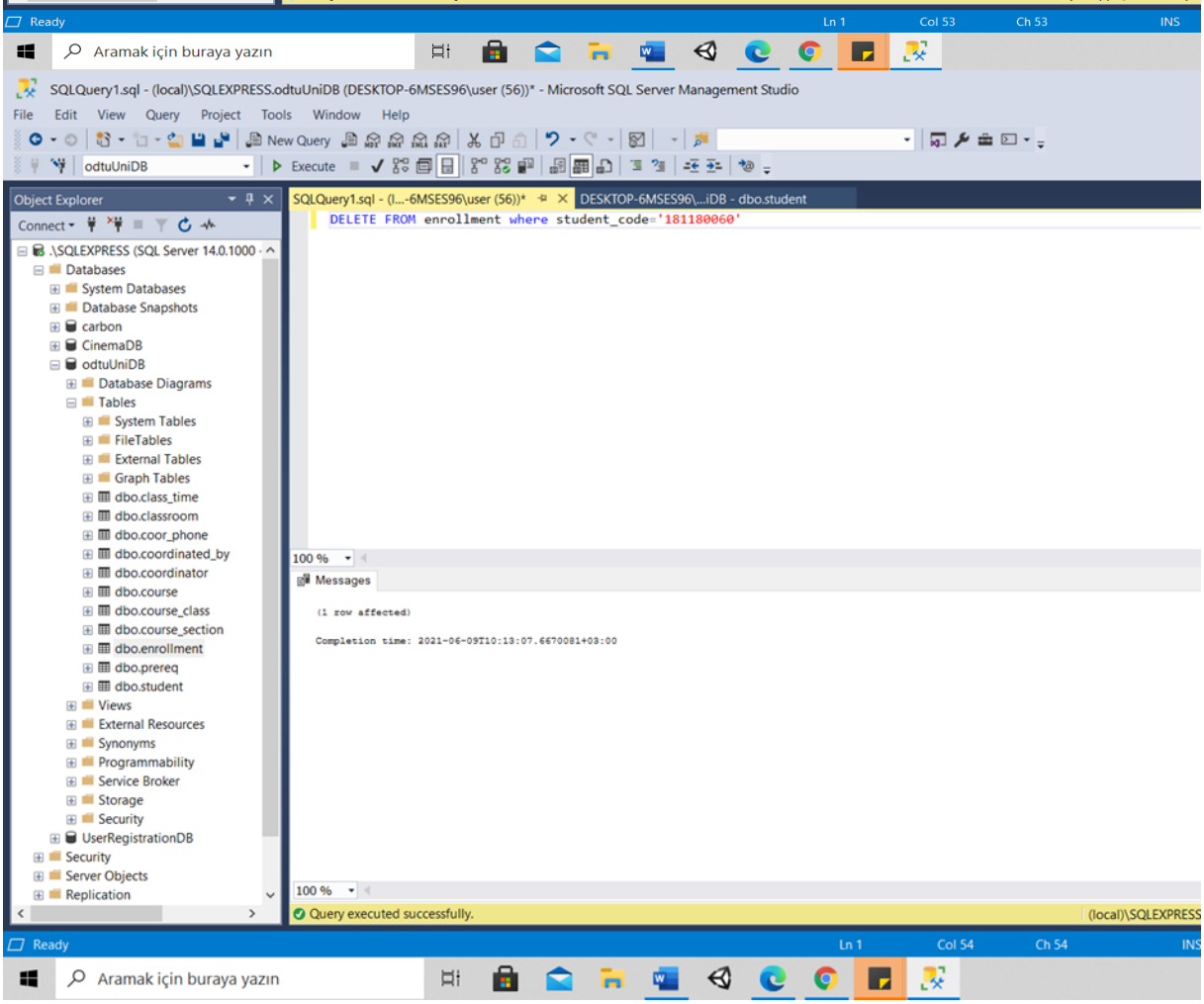
Indexing is a additional data structure which increases transaction operations especially data retrieval. In a database table, indexing removes obligation of linear search -table scanning- and provides quickly enough search of wanted, particular data attribute [1]. Logical searching steps are decreased dramatically by indexing.

There are several index types. The basic ones are clustered and non-clustered index types. Non-clustered index type, does not include the data itself and just points the location of the data itself. Non-clustered index is a separate entity that has data location points -kind of pointers-. Clustered index type, actually rearranges the real data and makes it easier to searching transactions. Clustered index is into the data while non-clustered index is a separate entity that points the data.

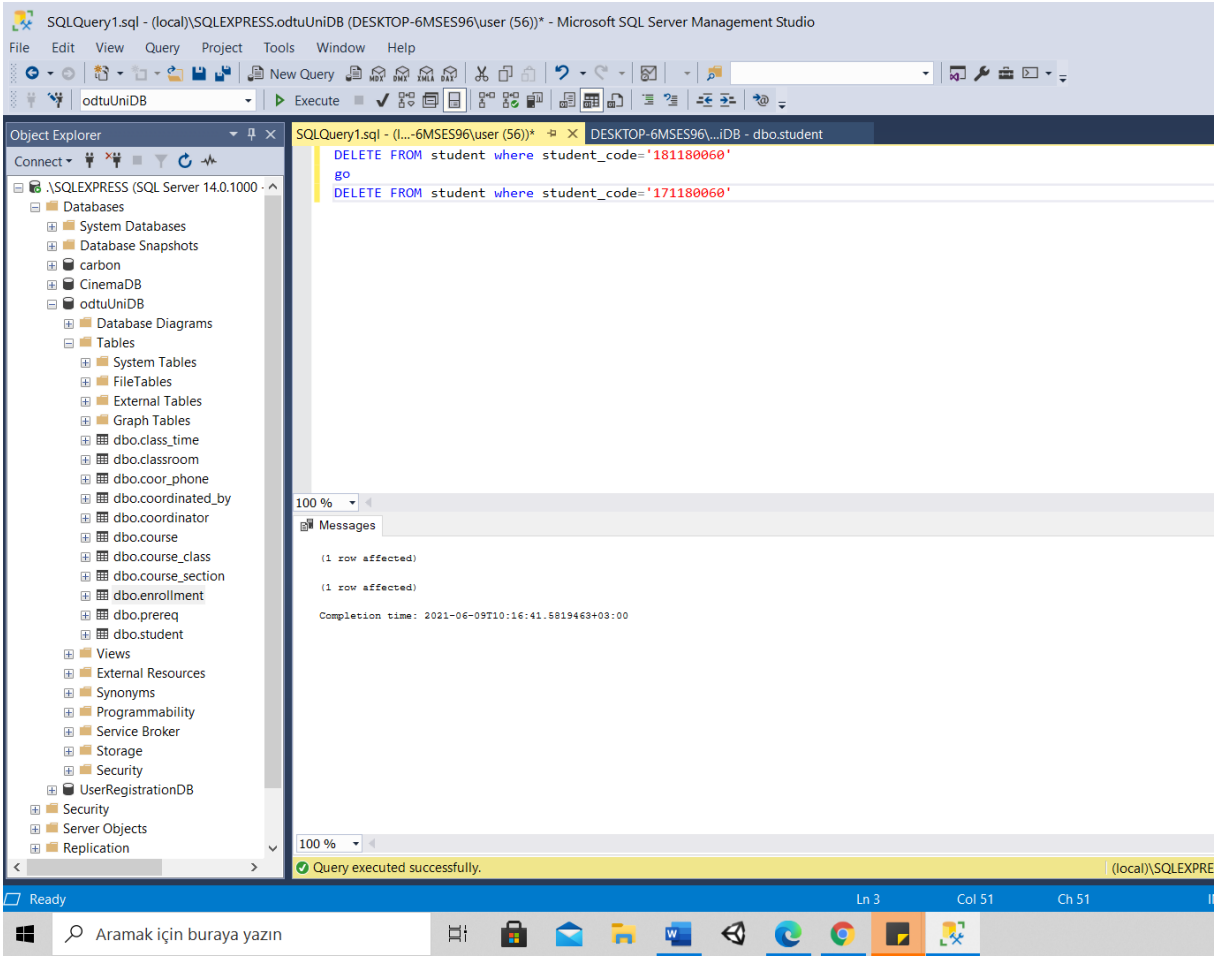
Integrity constraints primary key and foreign key can be behavior like clustered index in Microsoft SQL Server Management Studio. In odtuUniDB database schema, primary key and foreign key constraints can behave as clustered index. To demonstrate this qualification:

First of all for adding 100000 new records to "student" table schema, we delete "enrollment" table's all records referencing "student" table schema's *student_code* attribute. Then delete student table schema's existing records.





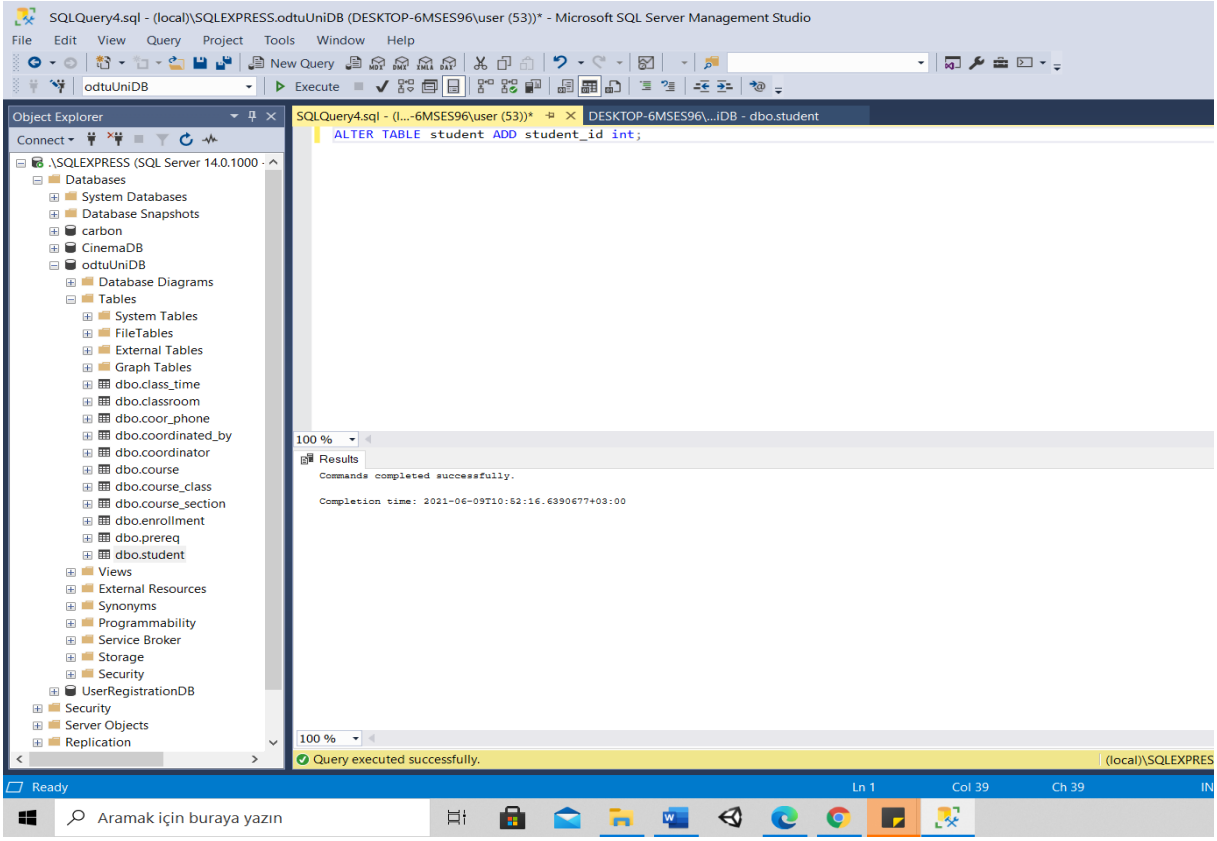
SQLQuery language of enrollment table record deletion

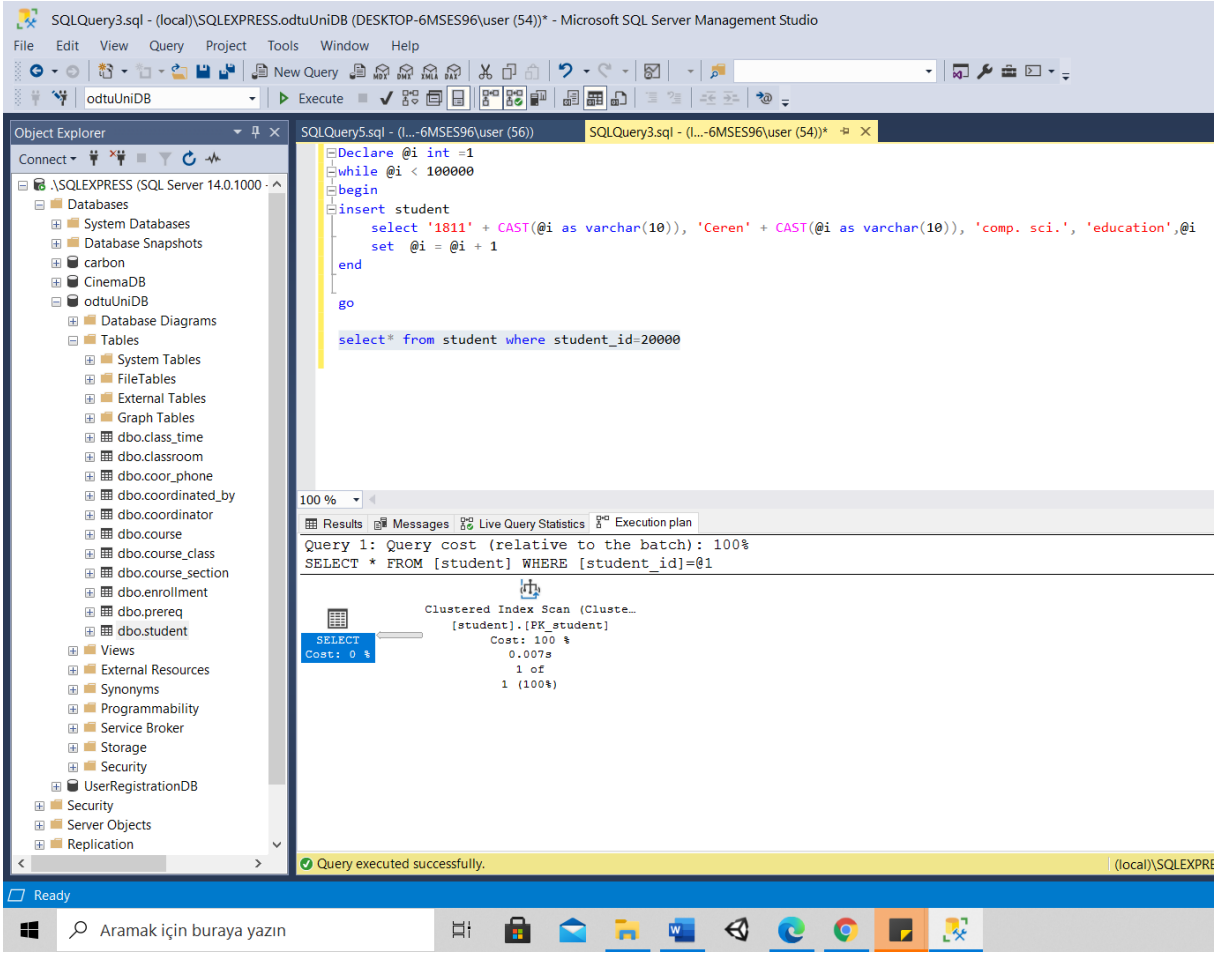


SQLQuery language of student table record deletion

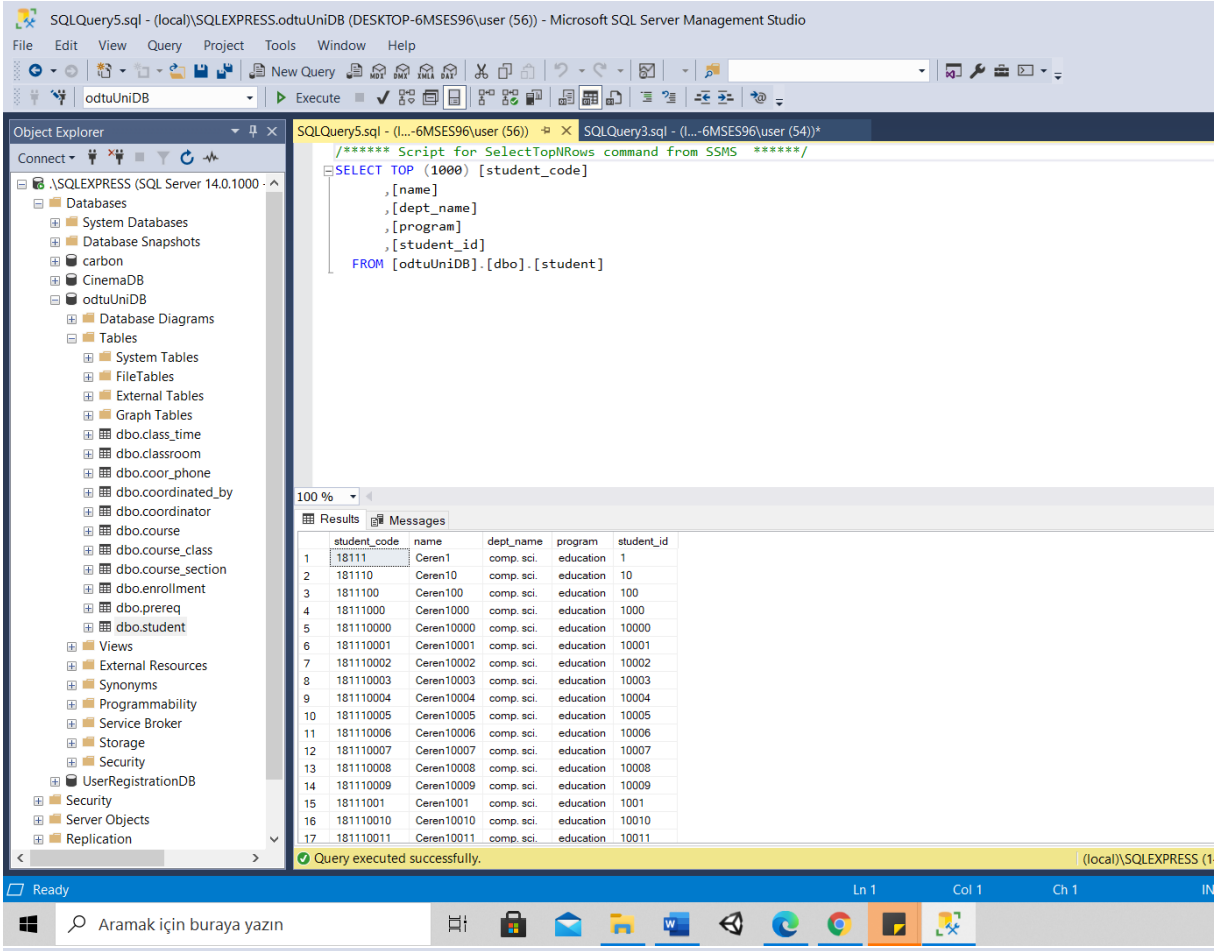
Secondly for implementing clustered index we add “student_id” column attribute to student table schema. student_id must be int domain type. student_id column attribute is important factor for adding student table schema a sequence order. student table schema’s sequence order contributes clustered index in logical manner. Also int domain of student_id, gains us ability of adding 100000 records to student table schema in looping way -while loop-. In our SQLQuery script, we create @i that maintains while loop’s numeric repetition until 100000 threshold. And in while loop’s every repeat attempt, student table schema is inserted by select clause statement’s record identification with increased student_id attribute and @i is setted with one incremental itself. After while loop SQLQuery script is executed, we add a new SQLQuery script that selects a specific record with student_id=20000 and before execution of select SQLQuery script, “Include Life Query Statistics”, “Include Actual Execution Plan” buttons are clicked -white boxes beside Execute and tick shaped button-. Then select SQLQuery script is executed, record with student_id=20000 is brought and in “Execution plan” bottom window, searching mechanism is showed as ‘Clustered Index Scan [Student]. [PK_student]’.

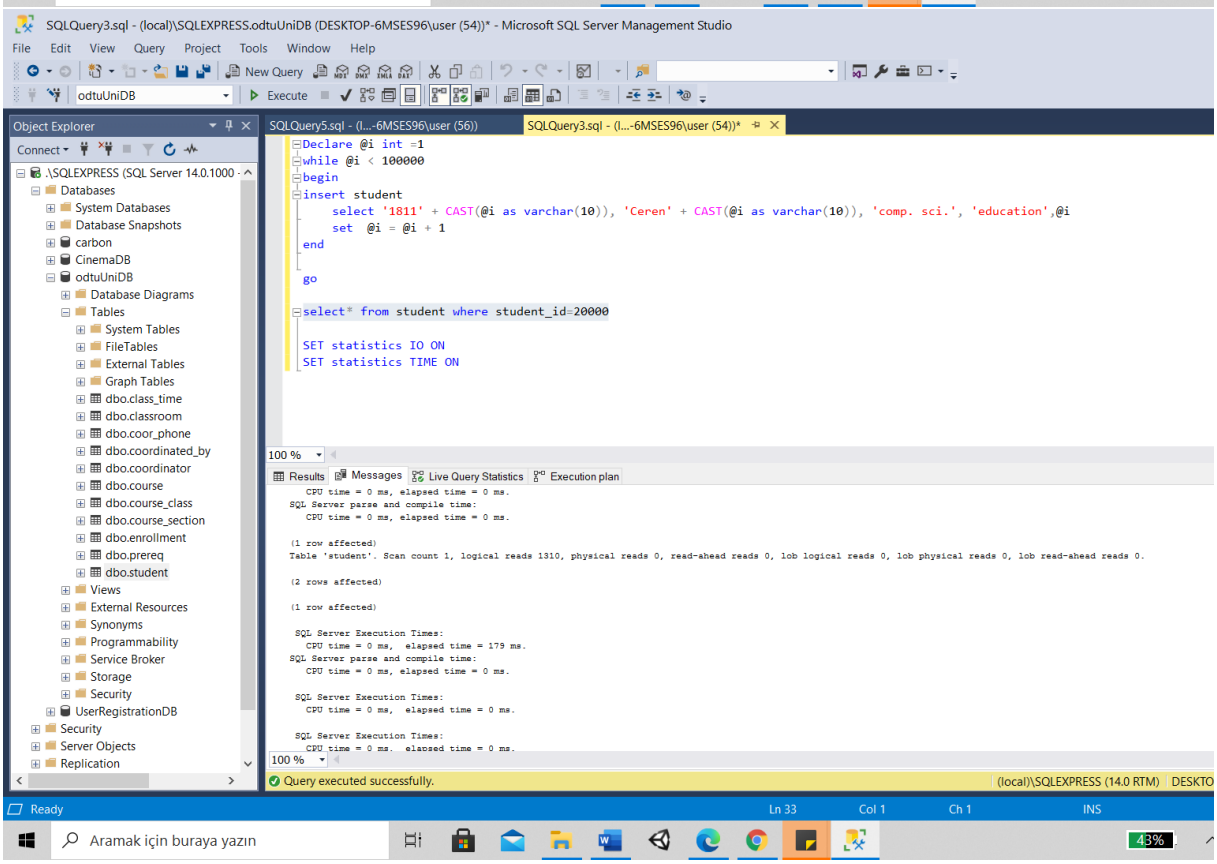
This situation shows us, student table schema’s integrity constraint primary key -student_code- is used for clustered indexing in Microsoft SQL Server Management Studio 18.





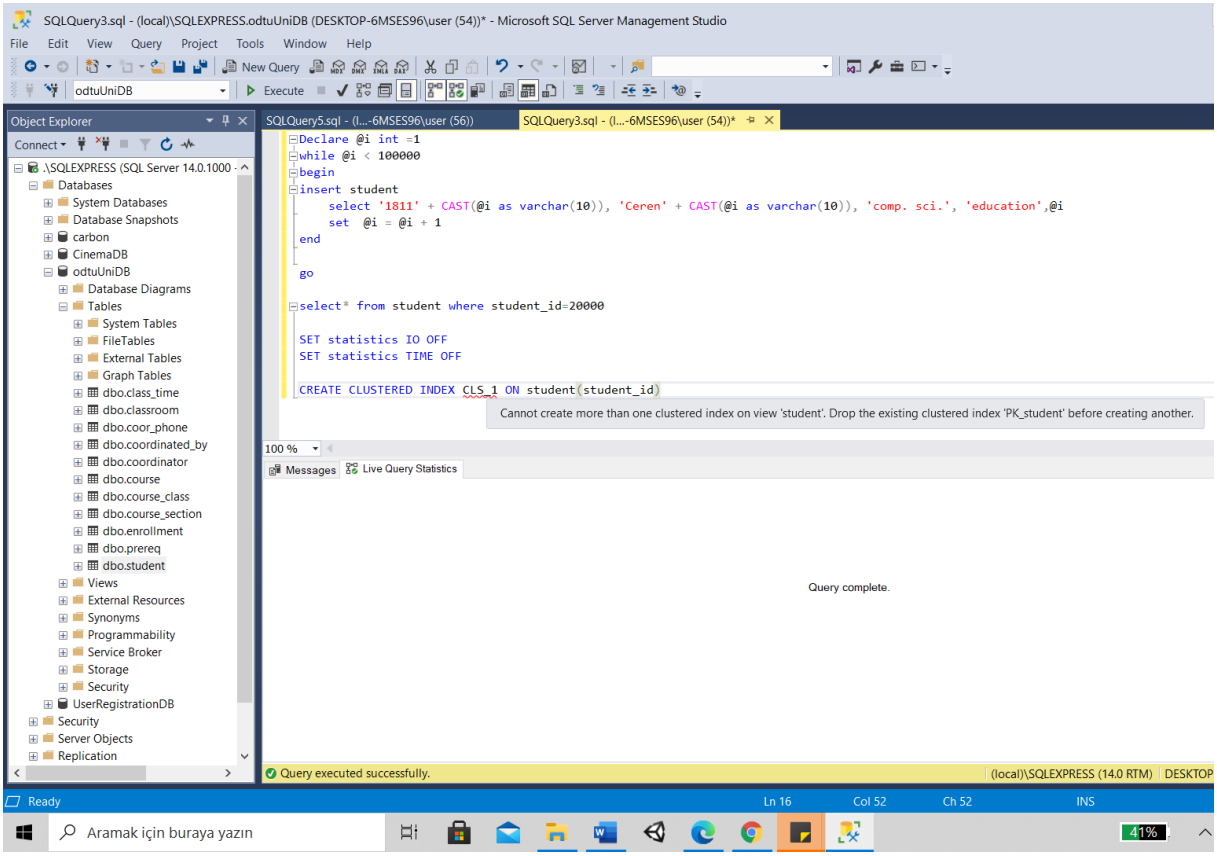
SQLQuery languages of altering table with student_id and adding 100000 records





result of while loop SQLQuery and ‘SET STATISTICS’ SQLQuery

We execute select SQLQuery script again but this time before select SQLQuery script execution, we execute ‘SET statistics IO ON’ and ‘SET statistics TIME ON’ SQLQuery scripts. With execution of select SQLQuery script -beside SET STATISTICS SQLQuery scripts- “Messages” bottom window is opened and statistical information is demonstrated. According to Messages bottom window report, table student select SQLQuery script has 1310 logical reads, 0 physical reads, 0 lob physical reads and 0 lob read-ahead reads. Clustered Index mechanism can find one record among 100000 records with 1310 logical reads.

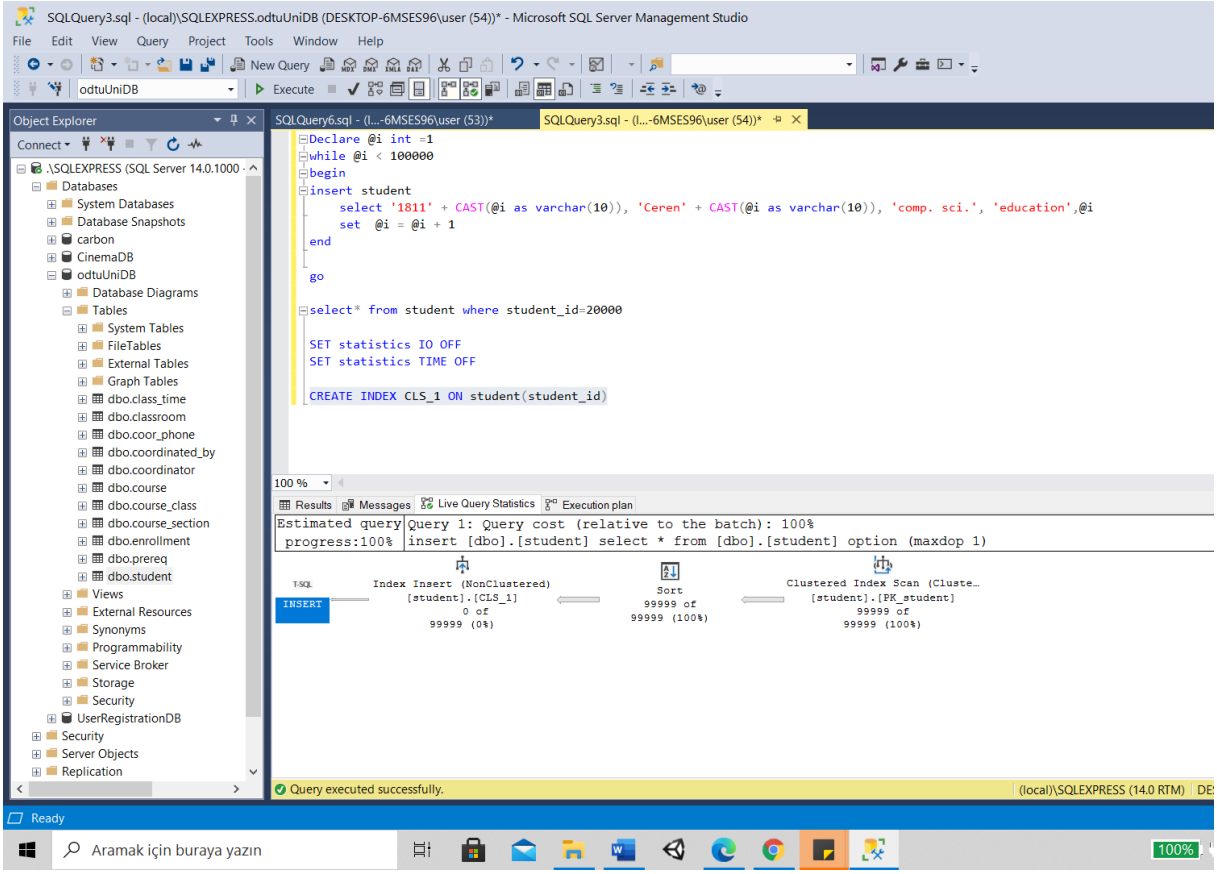


adding of ‘CREATE CLUSTERED INDEX’ SQLQuery script and error message

We go back to new created student_id attribute and try to create another clustered index in addition to primary key integrity constraint. For his purpose ‘CREATE CLUSTERED INDEX’ SQLQuery script is built upon student table schema’s student_id attribute. But a error message is shown, the error message tells us that on student table schema, an additional clustered index student_id can not be created before dropping of existing PK_student -primary key-.

In odtuUniDB database schema, all table schemas -*student*, *course_section*, *course*, *coordinator*, *classroom*, *coor_phone*, *class_time*, *enrollment*, *course_class*, *coordinated_by*, *prereq*- have primary key integrity constraint For this reason additional clustered index can not be created unless primary keys of each table schema is dropped.

Our inability to create clustered index, makes us choose non-clustered index mechanism and ‘CREATE INDEX’ SQLQuery script. We build ‘CREATE INDEX’ SQLQuery script upon student_id attribute again and execute ‘CREATE INDEX’ SQLQuery.

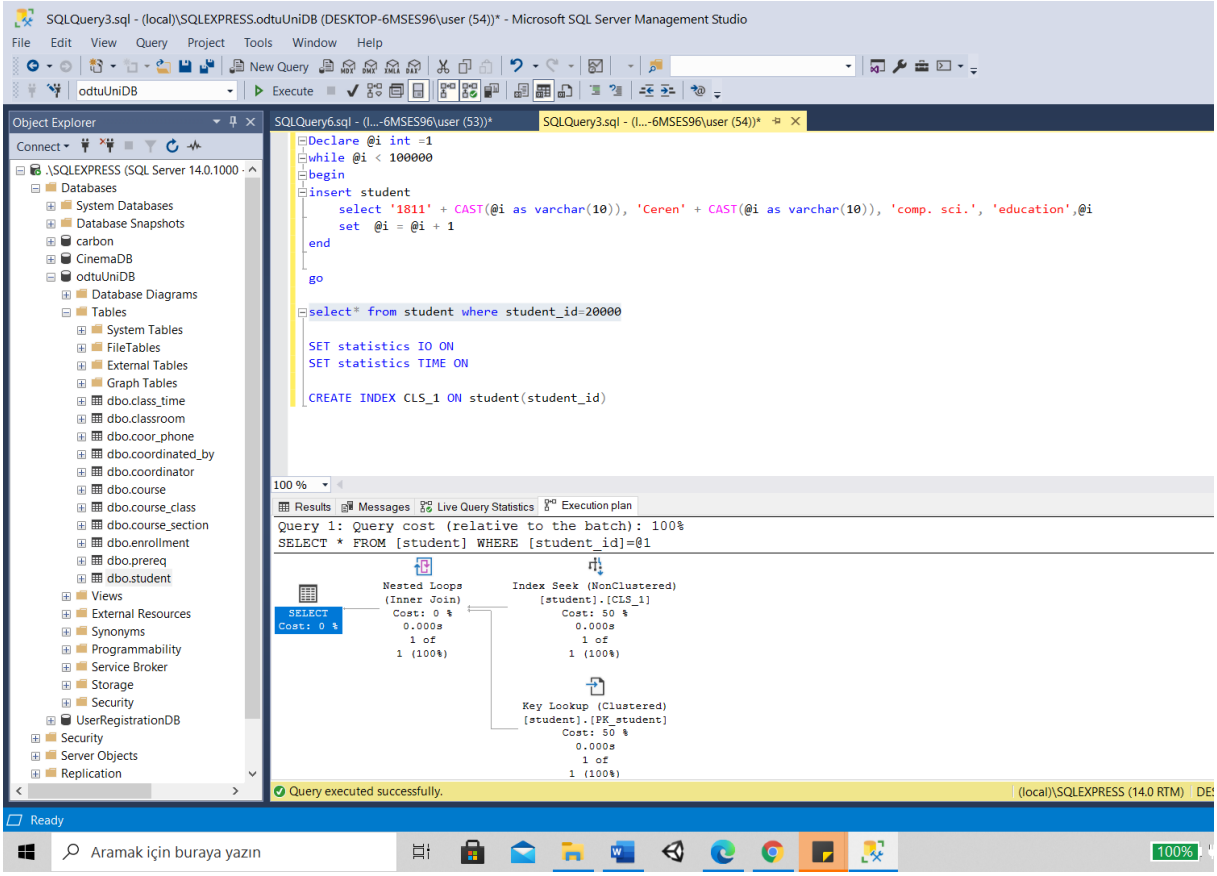


execution of ‘CREATE INDEX’ SQLQuery script

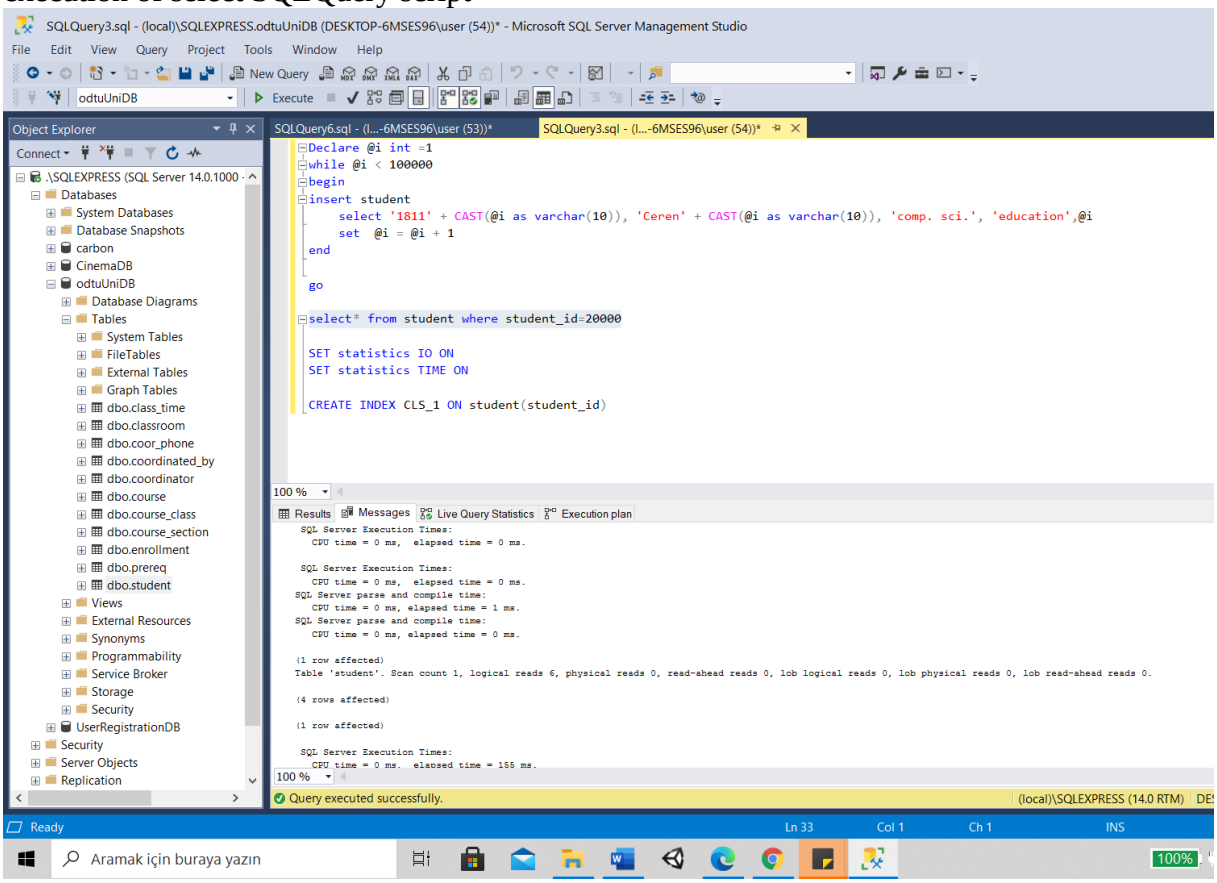
After execution of ‘CREATE INDEX’ SQLQuery, ‘Life Query Statistics’ bottom window shows creation of Non-clustered [student].[CLS_1].

Again we apply ‘SET STATISTICS’ SQLQuery script -turn them ON and execute them- and execute select SQLQuery script. ‘Execution plan’ bottom window shows relationship of Non-clustered [student].[CLS_1] index and Clustered [student].[PK_student] index in Nested Loops (inner join)

From now, everything seems to same except one big difference: statistical information that is showed by ‘Messages’ bottom window. According to Messages bottom window report -this time-, table student select SQLQuery script has just **6 logical reads**, 0 physical reads, 0 lob physical reads and 0 lob read-ahead reads. Before implementing non-clustered index mechanism there is 1310 logical reads, right now after implementing non-clustered index mechanism there is just 6 logical reads. Combination of non-clustered [student].[CLS_1] index and clustered [student].[PK_student] index in Nested Loops (inner join) can find one record among 100000 records with just 6 logical reads -much better than just primary key clustered index-



execution of select SQLQuery script



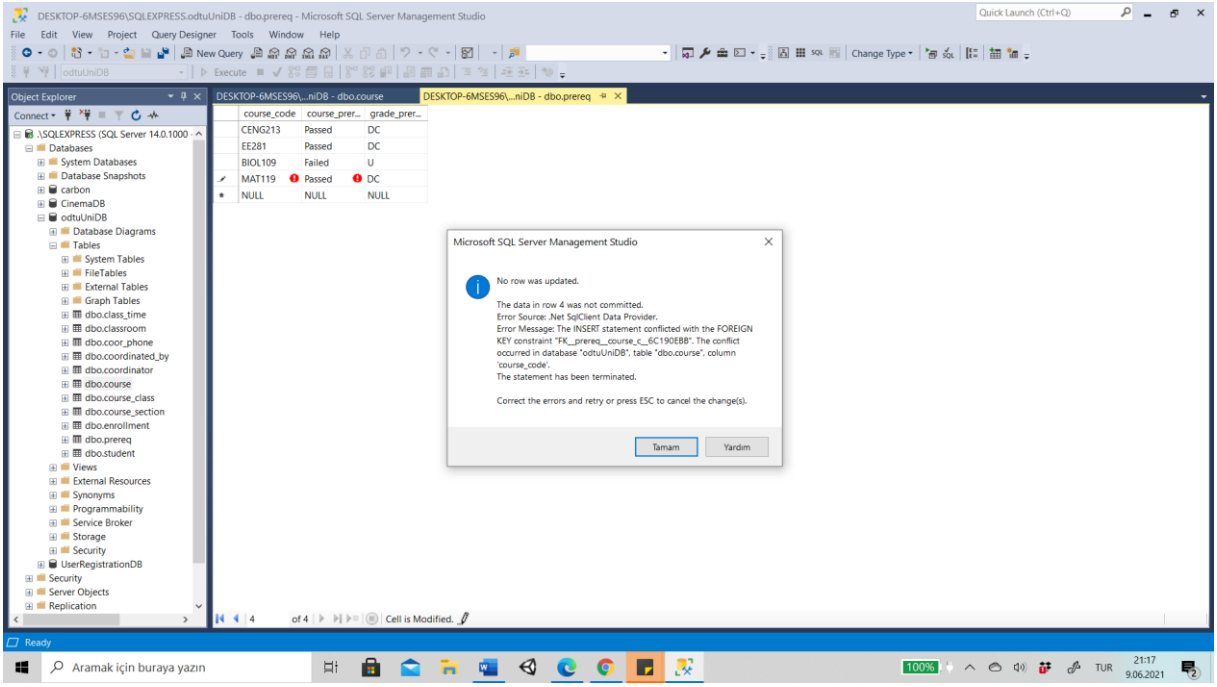
result of ‘SET STATISTICS’ SQLQuery script in non-clustered index mechanism

Decreasing of logical reads number, contributes data retrieval transaction speed considerably.

Combination of Non-clustered index and clustered index elapsed time (ms)	**Just clustered index elapsed time (ms)**
155	179

Performans metrics

Atomicity	All multivalued relational design attributes -day_class, start_time, end_time, phone_number into separate table schemas. Every data piece has a unique attribute column itself -indexing does no change atomicity performans metric	Very good
Durability	During course_code foreign key constraint failure, data loss is not happened.	good
Lesser Redundancy Capacity	Since dropping of course_sec_course redundant table schema, lesser redundancy capacity is improved and odtuUniDB database schema's relationship set information redundancy is removed.	good
Consistency	prereq table schema includes course_prereq -passed,failed- and grade_prereq -AA,DC, S, U exec.- column attributes referencing course table schema. In case of adding a new record with course_code that does not exist in course table schema, foreign key integrity constraint immediately prevents that new record addition and warns us. (consistency between course and prereq table schemas is secured)	Very good
Speed	Data retrieval transaction speed is increased consireably by decreasing of logical reads number and elapsed time	Very good



prereq table schema's course_code foreign key constraint does not let commiting

d)

Particular transactions -retrieving data, deleting data, manipulating data, inserting data, grouping data- are introduced database server engine through sql query requests. And there is possibility of different execution ways that reach to the most fruitful one in a given query request. Query optimization, searches the most fruitful execution way among other differing execution ways in a given sql query request. Especially query optimizer in database server engine determines to explore best execution plan for added sql query request [2]. However query optimizer is not reachable directly from user base in most time. Exploring of best execution plan leads to minimum time consume during execution of relative sql query request and query optimizer try to provide minimum elapsed time for about-to-execute- sql query request in database server engine -Microsoft server database engine, OracleDB engine exec.-. Despite automation of query optimization by query optimizer, a database user -or database executive- can examine several execution plans, rearrange the sql query requests -SQLQuery script- and can measure overall performance of sql query request execution. Also query optimization is a part of general sql performance tuning which prioritizes results and performance for better transaction quality.

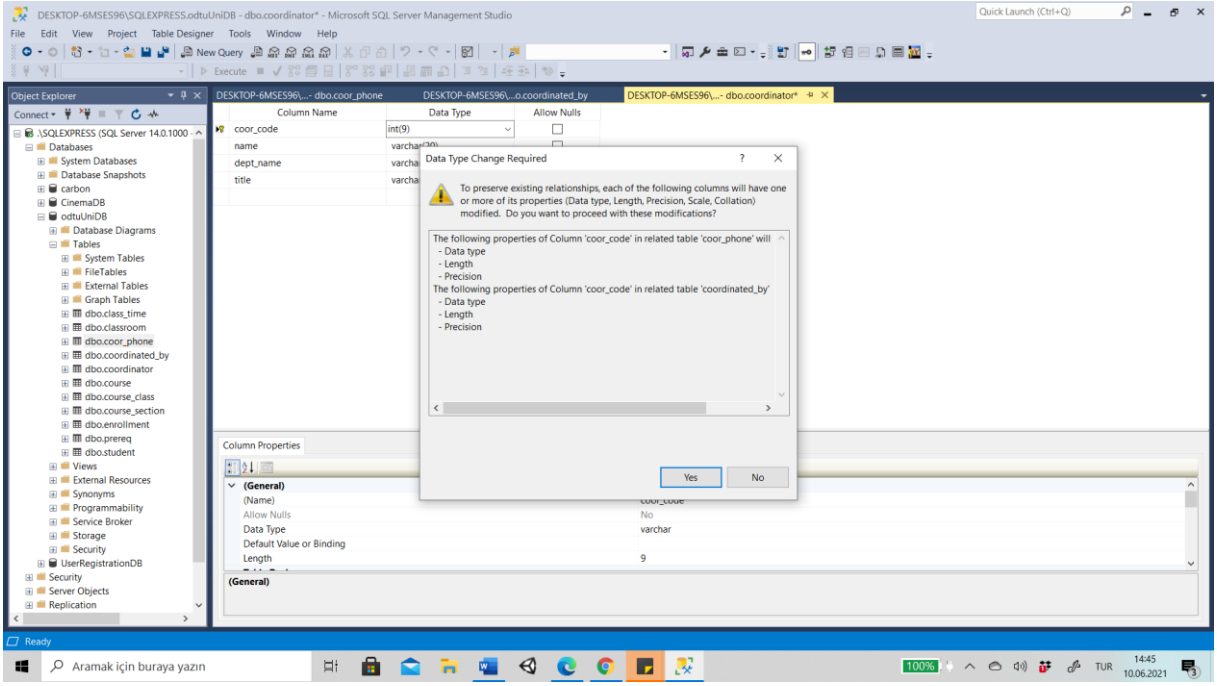
From manual user base there is several ways to accomplish sql query optimization:

- Usage of varchar/nchar instead of char/nchar whenever possible -when character type domain is needed for table schema attribute- Usage of char/nchar character type domain restrict sql query performance.

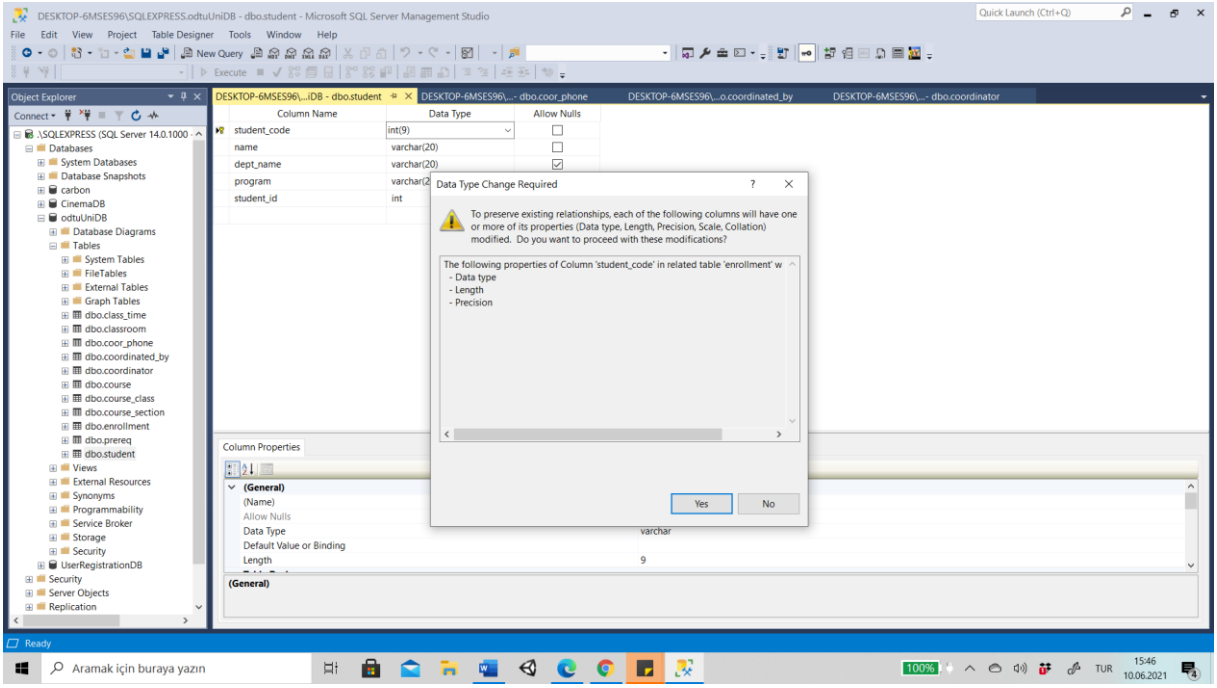
In odtuUniDB database table schema every character type domain is initialized as varchar -varchar(3), varchar(6), varchar(8), varchar(9), varchar(20), varchar(50)-. Therefore there is no performance issue about character type domain -char/nchar-.

- Usage of numeric type domain -int- as much as possible is critic in sql optimization. Besides numeric type domain, character type domain -varchar/nchar exec.- reduces performance of query transactions.

In odtuUniDB database table schema attributes student_code and coor_code had been initialized as character type domain -varchar- in contrast to numeric type domain -int-. For providing usage of numeric type domain in these attributes, we must alter coordinator table schema and student table schema. However altering coordinator and student table schemas is complicated and risky. Table schema coordinator is referenced by both *coor_phone* and *coordinated_by* table schemas and altering coordinator table schema, affects these two important table schemas. Table student is referenced by *enrollment* table schema. Also in the beginning odtuUniDB database table schema is designed as mainly character type domain -varchar- style. We can not alter the any critic table schema that will damage integrity of odtuUniDB relational design.



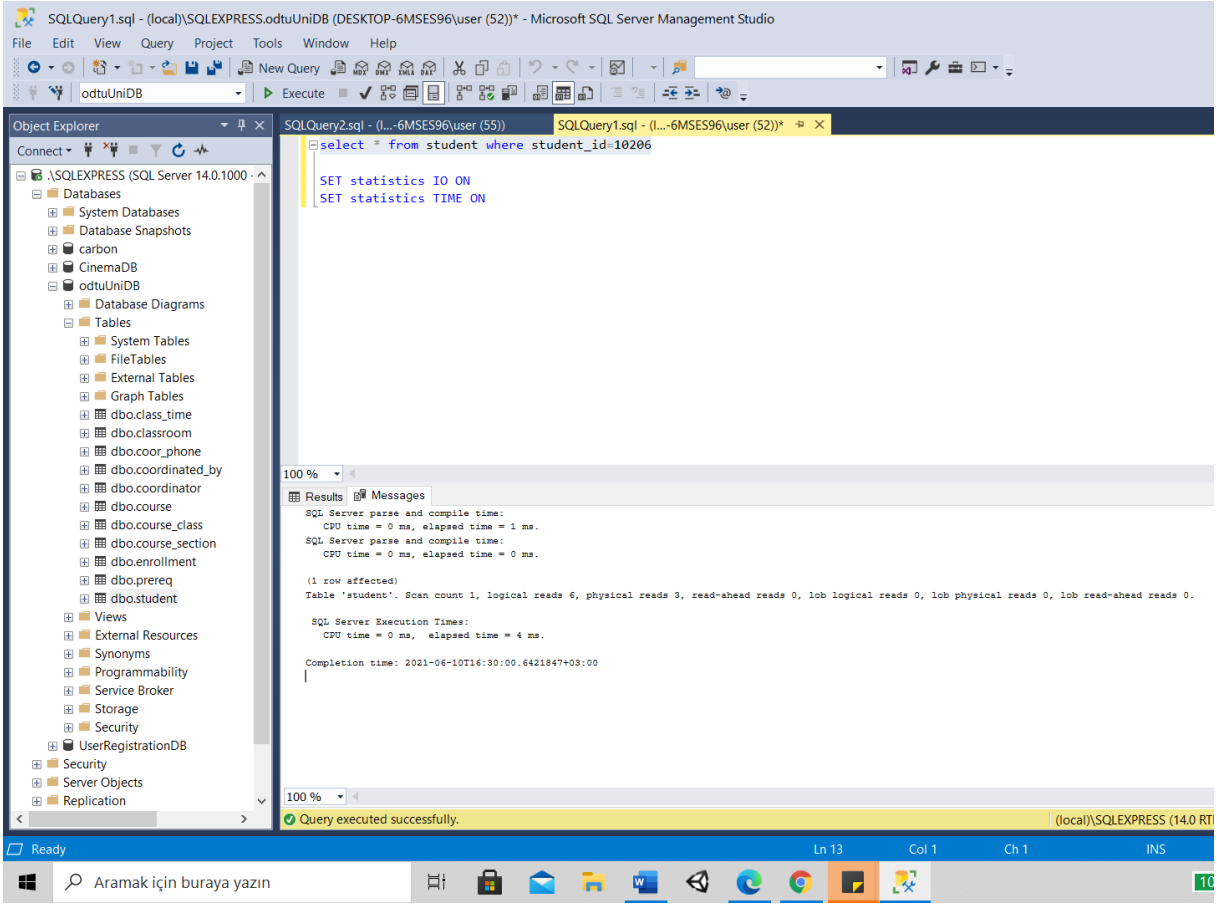
warning message of coordinator table schema altering attempt



warning message of student table schema altering attempt

- In sql query request -SQLQuery script- ‘select * …’ -select every attribute column- SQLQuery script piece is not prefered in case of performance downgrading issue. Instead ‘select id,name …’ -select specific fields- SQLQuery script piece is prefered in order to prevent performance loss. Usage of ‘select specific fields’ SQLQuery script, reduces consume of resources and work stifle besides usage of ‘select * …’ SQLQuery script piece.

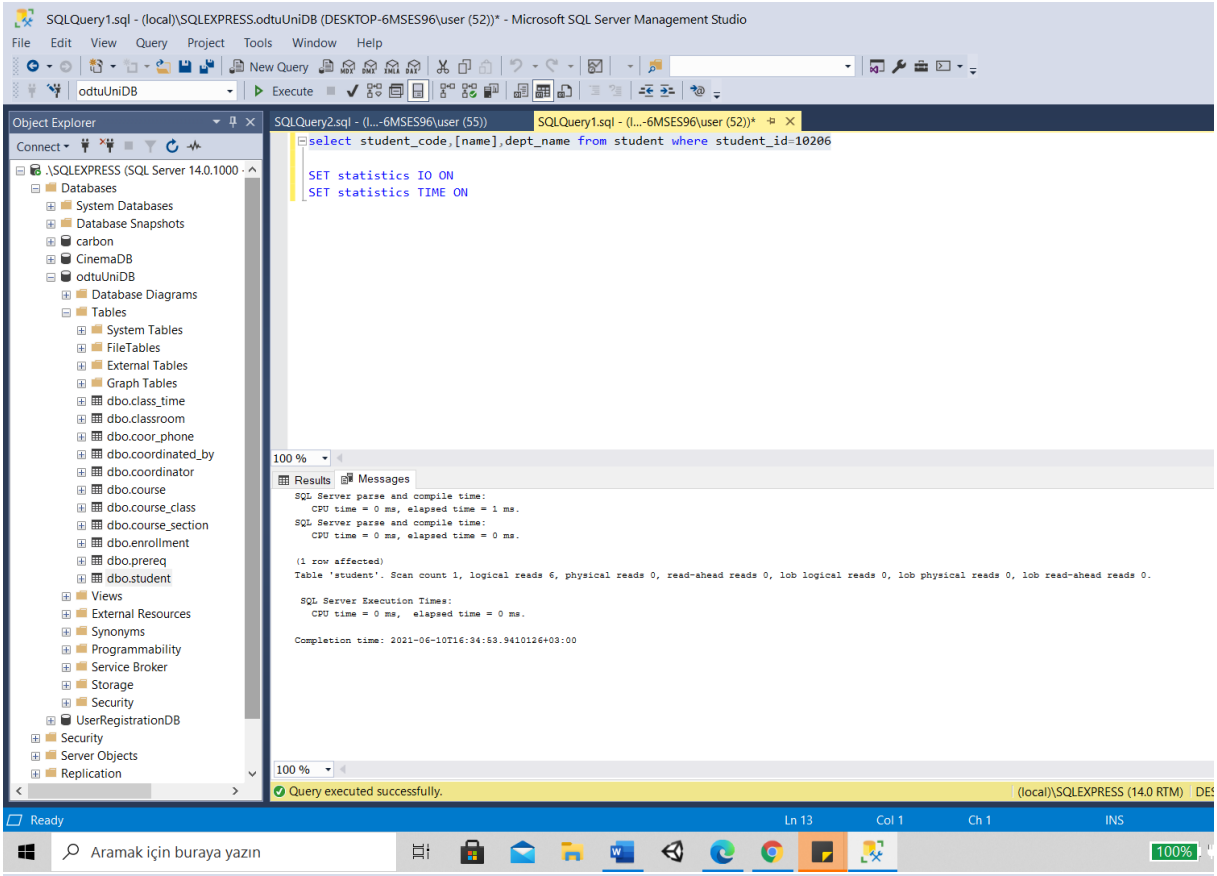
In odtuUniDB database schema first we try to select all attributes of student table record with student_id=10206 -before executing ‘select * from student where student_id=10206’ script, we execute ‘SET STATISTICS’ SQLQuery script- and execute its ‘select * …’ SQLQuery script piece. We get selected record with all student table schema attributes and in ‘Messages’ bottom window, get ‘elapsed time = 4 ms logical reads 6’ text.

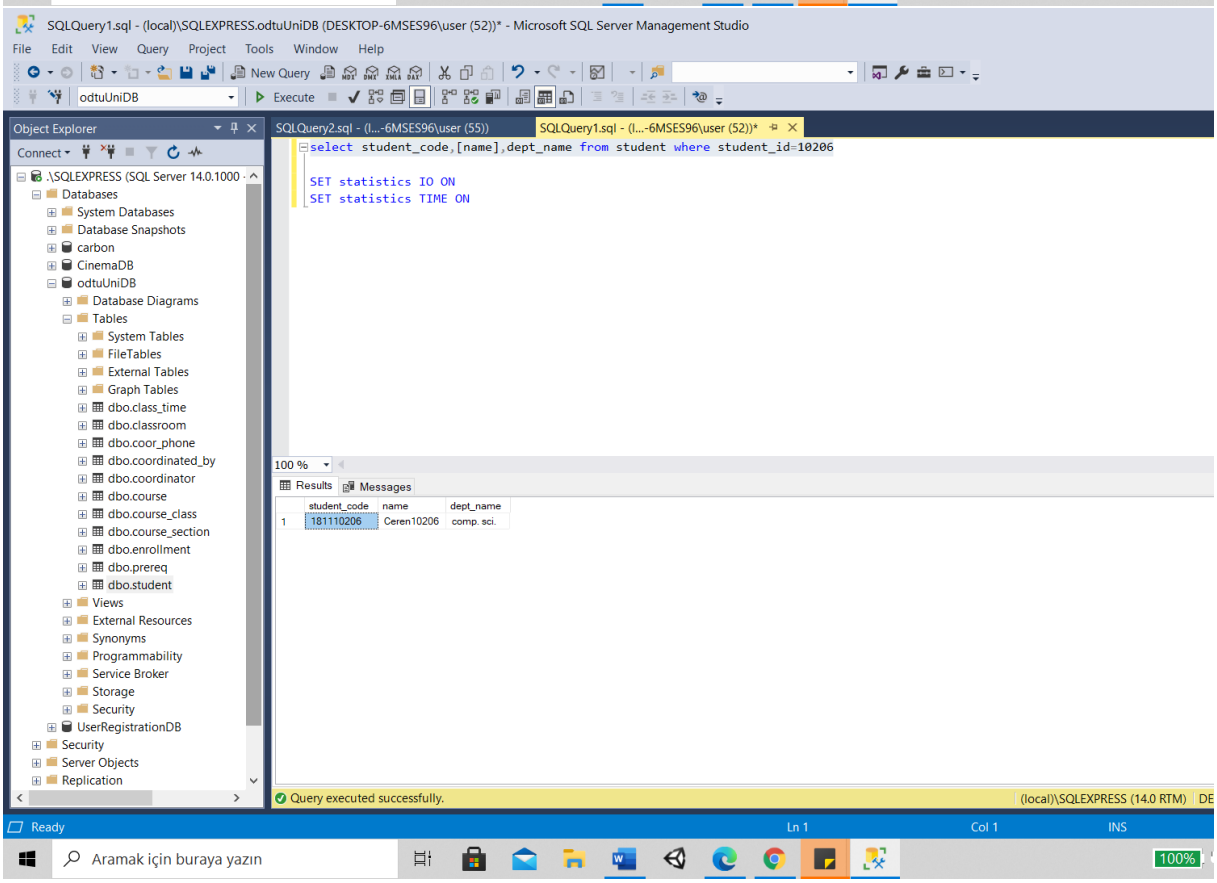


execution of ‘select * …’ SQLQuery script piece

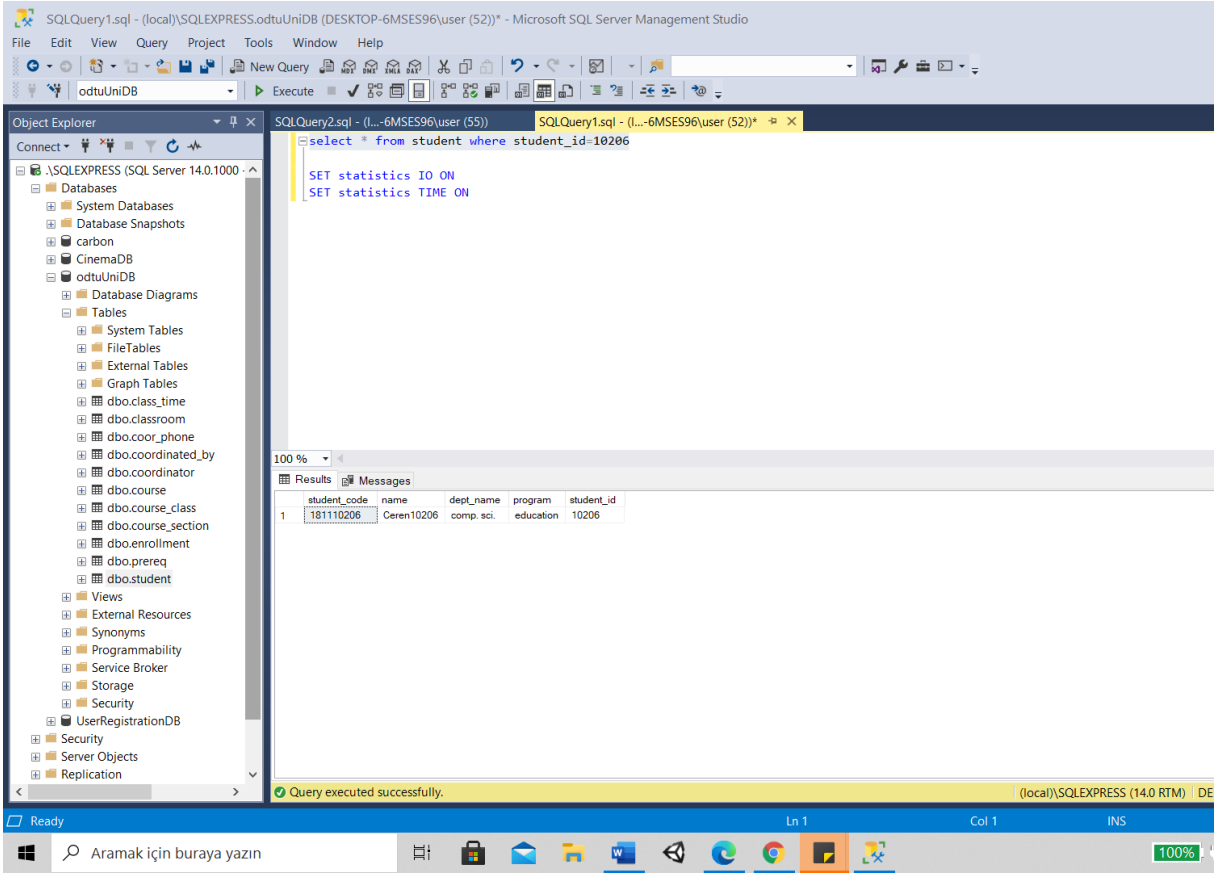
After execution of ‘select * …’ SQLQuery script piece, we try to select student_code, name and dept_name attributes of student table record with student_id=10206 and execute its ‘select student_code, [name], dept_name…’ SQLQuery script piece. We get selected record with student_code, name and dept_name student table schema attributes and in ‘Messages’ bottom window, get ‘elapsed time = **0** ms logical reads 6’ text.

In conclusion, between ‘select * …’ SQLQuery script piece and ‘select stduent_code, [name], dept_name…’ SQLQuery script piece, there is a difference which result of sql optimization performance boost. Consume of resources is decreased, while time efficiency is increased.





execution of ‘select student_code, [name], dept_name’ SQLQuery script piece and result



execution of ‘select * …’ SQLQuery script piece result

- Usage of fuzzy keyword queries, is reminded us ‘like’ SQLQuery script keyword, however usage of ‘like’ keyword brings risk of invalid index situation. For instance from odtuUniDB, between:

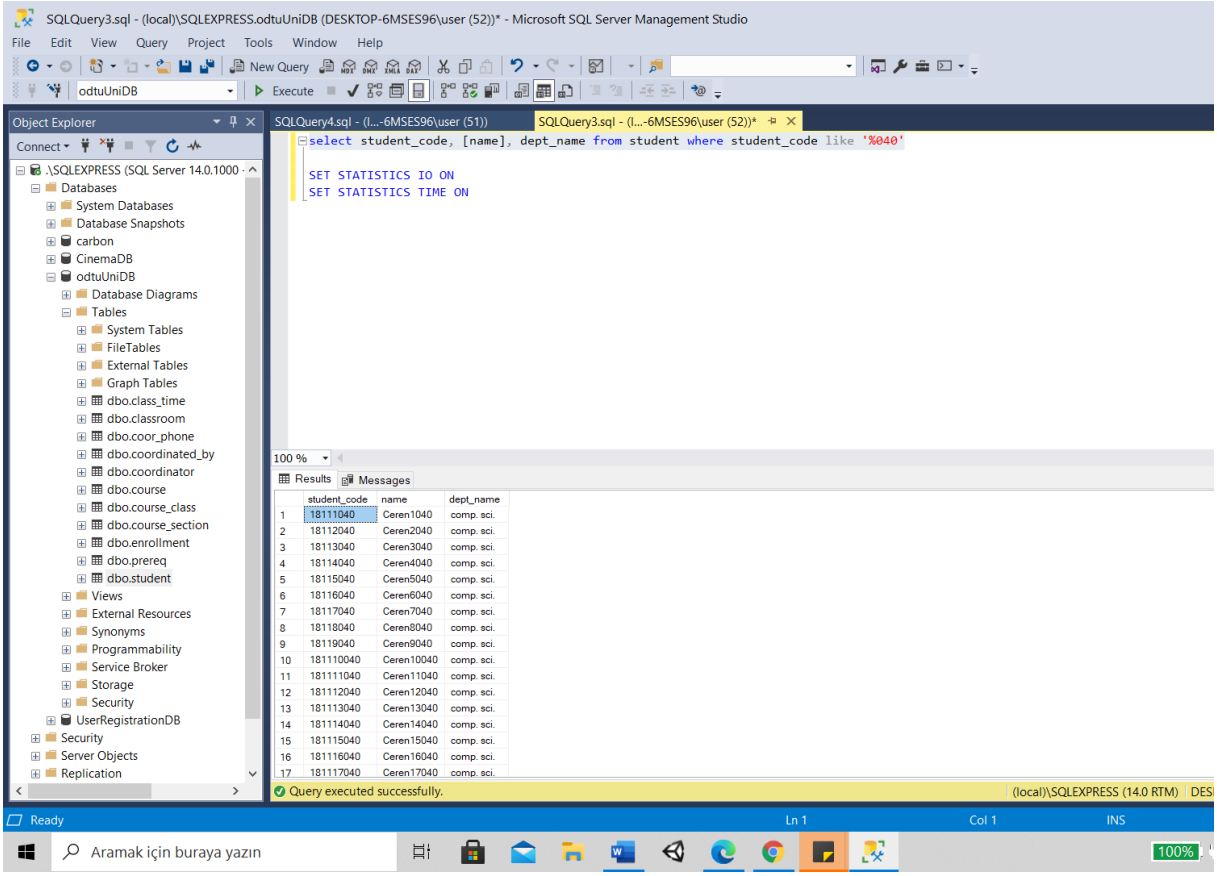
‘select student_code, [name], dept_name from student where student_code like *‘%040’* and

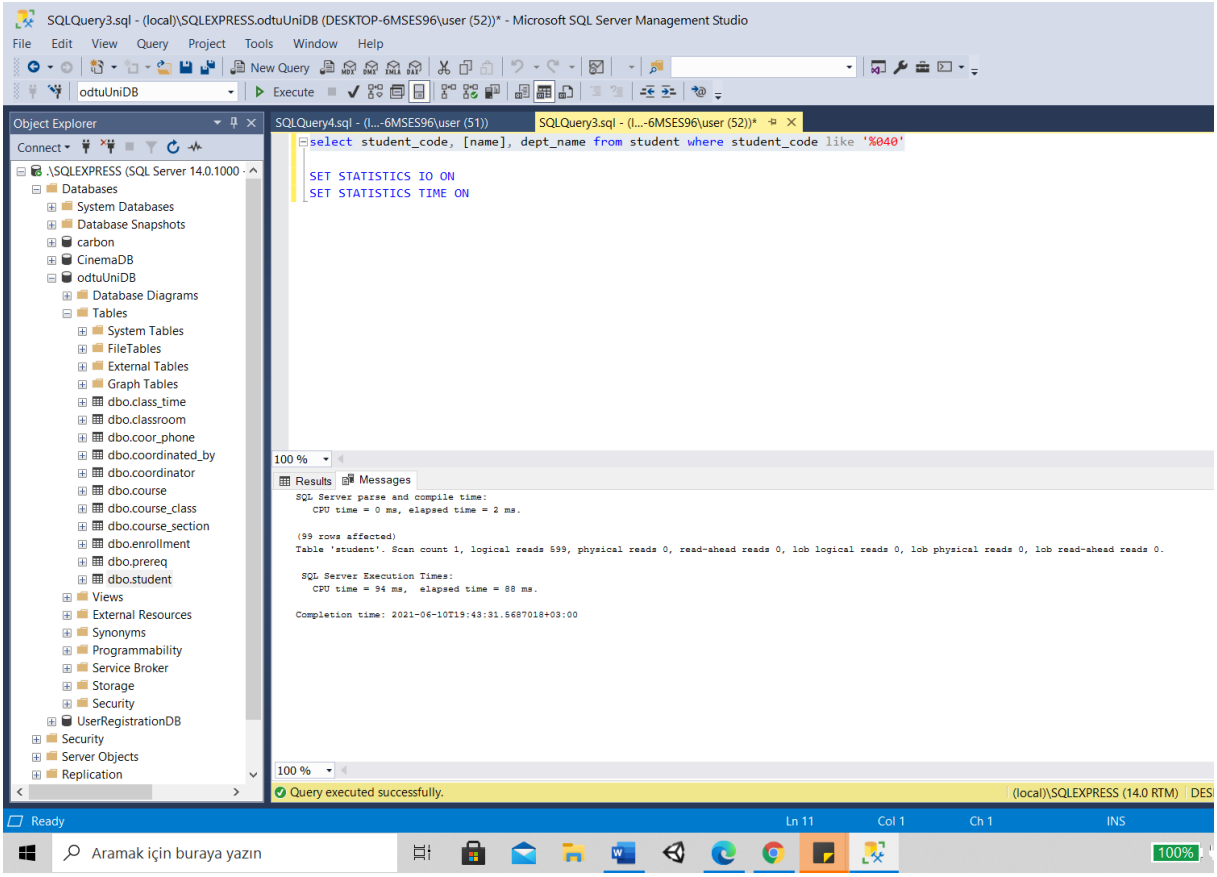
‘select student_code, [name], dept_name from student where student_code like *‘040%’*” SQLQuery scripts there is a performance, elapsed time and result difference.

When we execute ‘…like %040’ SQLQuery script piece -before execution we open ‘SET STATISTICS’ sql query ON- ‘Messages’ bottom window gives us ‘elapsed time = 88 ms’ and ‘logical reads 599’ information. ‘Results’ bottom window shows us all student table schema records with student_code including ‘%040’ section.

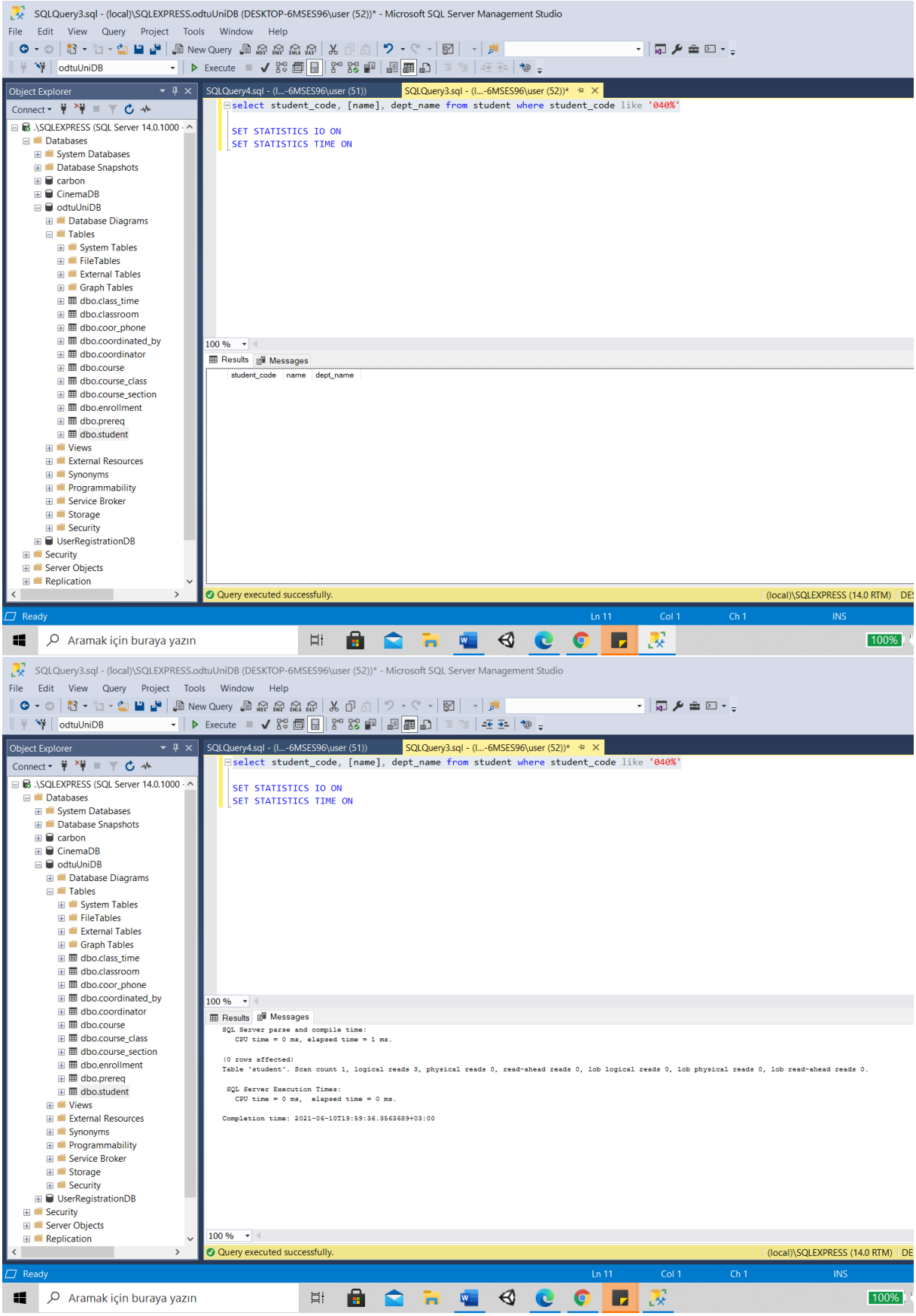
When we execute ‘…like 040%’ SQLQuery script piece, ‘Messages’ bottom window gives us ‘elapsed time = **0**’ and ‘logical reads **3**’ information. ‘Results’ bottom window shows us a blank student table schema with student_code

‘…like 040%’ SQLQuery script piece performs much lower logical reads and elapsed time than ‘…like %040’ SQLQuery script piece. For this reason, usage of like keyword in ‘040%’ manner is more preferable than usage of like keyword in ‘%040’ manner.





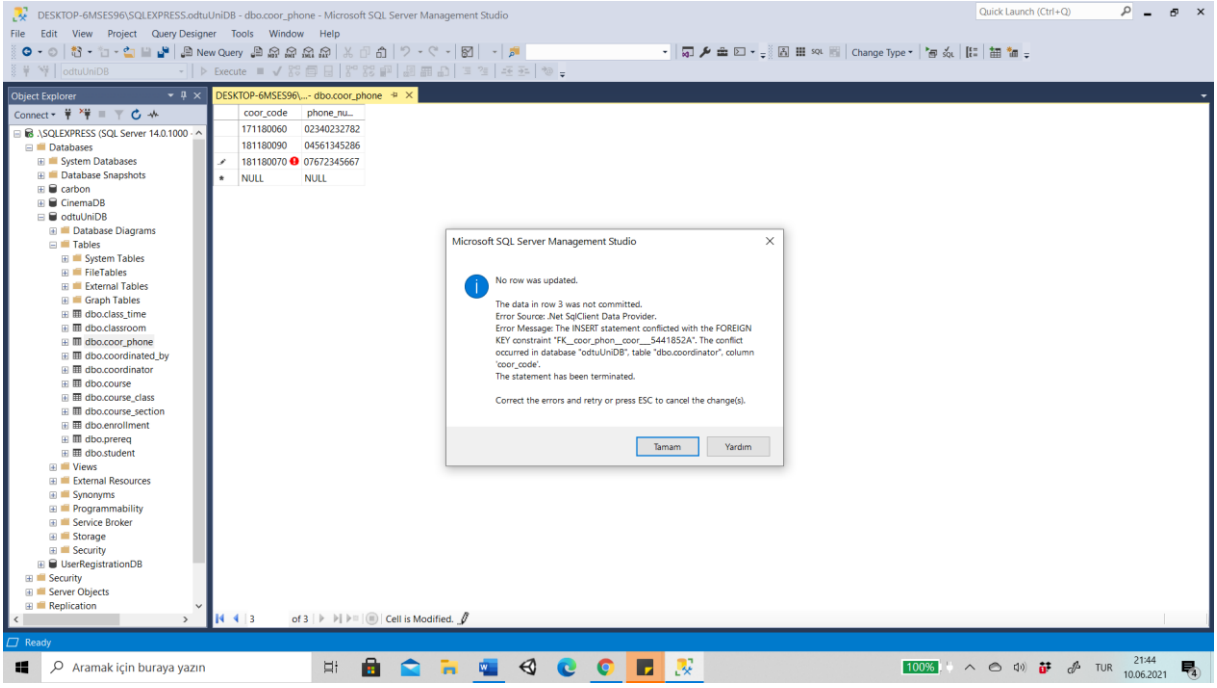
execution of ‘like...%040’ SQLQuery script piece and result



execution of ‘like…040%’ SQLQuery script piece and result

Performans metrics

Atomicity	During query optimization, atomicity of multivalued attributes -separate table schemas of {phone_number} and {day_class, start_time, end_time}	Very good
Durability	During commiting failure coor_phone table schema's does not lose any data.	Good
Lesser Redundancy Capacity	Since dropping of redundant course_sec_course table schema information redundancy that causes by weak relationship set is removed. Query optimization is mostly about query rearrangements and table schema design.	Good
Consistency	During addition of a coordinator phone number which doesn not exist in coordinator table schema, foreign key constraint coor_code in coor_phone table schema reacts and prevents the addition.	Very good
Speed	With query optimization, transaction speed of sql query requests are increased.	Excellent



coor_code foreign key constraint does not let addition of the record

Analysis

In the beginning of a section less redundancy capacity is inadequate situation and there is information redundancy caused by *course_sec_course* relationship set derived table schema. After b section normalization work, information redundancy is removed from odtuUniDB database table schema with dropping of course_sec_course table schema. In c section, clustered index primary key and and new addition *student_id* non-clustered index are combined and a-b section's overall speed is increased. Finally in d section rearrangement of transaction's sql query request -SQLQuery script- is added new speed to overall c section speed.

a, b, c, d comparison table

	a	b	c	d
Atomicity	Very good	Very good	Very good	Very good
Durability	good	good	good	good
Lesser Redundancy Capacity	Not good	good	good	good
Consistency	Very good	Very good	Very good	Very good
Speed	good	good	Very good	Excellent

b, c, d speed table

	b	c	d
Elapsed Time (ms)	179	155	0
Logical Reads	1310	6	6

*In odtuUniDB we add information to table schema's attribute columns. For instance in course_code attribute column we use 'EE281, CENG213, BIOL109'. These information is referenced from

- https://ceng.metu.edu.tr/tr/ogretim-programi
- https://oidb.metu.edu.tr/tr/derslerin-kredi-sistemi

REFERENCES

1. Database Index. https://en.wikipedia.org/wiki/Database_index (2021).

2. Query Optimization. https://en.wikipedia.org/wiki/Query_optimization (2021).