**Theory:**

## Partitioned Join:

The relational tables that are to be joined gets partitioned on the joining attributes of both tables using same partitioning function to perform Join operation in parallel.

#### How does Partitioned Join work?

Assume that relational tables r and s need to be joined using attributes r.A and s.B. The system partitions both r and s using same partitioning technique into n partitions. In this process A and B attributes (joining attributes) to be used as partitioning attributes as well for r and s respectively. r is partitioned into r0, r1, r2, …, rn-1and s is partitioned into s0, s1, s2, …, sn-1. Then, the system sends partitions ri and si into processor Pi, where the join is performed locally.as Equi-Joins and Natural Joins can be performed using Partitioned join technique .

Equi-Join or Natural Join is done between two tables using an equality condition such as r.A = s.B. The tuples which are satisfying this condition, i.e, same value for both A and B, are joined together. Others are discarded. Hence, if we partition the relations r and s on applying certain partitioning technique on both A and B, then the tuples having same value for A and B will end up in the same partition. Let us analyze this using simple example;

|  |  |  |  |
| --- | --- | --- | --- |
| RegNo | SName | Gen | Phone |
| 1 | Sundar | M | 9898786756 |
| 3 | Karthik | M | 8798987867 |
| 4 | John | M | 7898886756 |
| 2 | Ram | M | 9897786776 |

Table 1 - STUDENT

|  |  |
| --- | --- |
| RegNo | Courses |
| 4 | Database |
| 2 | Database |
| 3 | Data Structures |
| 1 | Multimedia |

Table 2 – COURSES\_REGD

Let us assume the following;

The RegNo attributes of tables STUDENT and COURSES\_REGD are used for joining.

Observe the order of tuples in both tables. They are not in particular order. They are stored in random order on RegNo.

Partition the tables on RegNo attribute using Hash Partition. We have 2 disks and we need to partition the relational tables into two partitions (possibly equal). Hence, n is 2.

The hash function is, h(RegNo) = (RegNo mod n) = (RegNo mod 2). And, if we apply the hash function we shall get the tables STUDENT and COURSES\_REGD partitioned into Disk0 and Disk1 as stated below.

|  |  |
| --- | --- |
| Partition 0 | Partition 1 |
| |  |  |  |  | | --- | --- | --- | --- | | RegNo | SName | Gen | Phone | | 4 | John | M | 7898886756 | | 2 | Ram | M | 9897786776 |   STUDENT\_0 | |  |  |  |  | | --- | --- | --- | --- | | RegNo | SName | Gen | Phone | | 1 | Sundar | M | 9898786756 | | 3 | Karthik | M | 8798987867 |   STUDENT\_1 |
| |  |  | | --- | --- | | RegNo | Courses | | 4 | Database | | 2 | Database |   COURSES\_REGD\_0 | |  |  | | --- | --- | | RegNo | Courses | | 3 | Data Structures | | 1 | Multimedia |   COURSES\_REGD\_1 |

From the above table, it is very clear that the same RegNo values of both tables STUDENT and COURSES\_REGD are sent to same partitions. Now, join can be performed locally at every processor in parallel.

One more interesting fact about this join is, only 4 (2 Student records X 2 Courses\_regd records) comparisons need to be done in every partition for our example. Hence, we need total of 8 comparisons in partitioned join against 16 (4 X 4) in conventional join.

**Program :**

**Join using Hash Partitioning on join Attribute**

import mysql.connector

conn = mysql.connector.connect(host='localhost',

    database='exammanagement',

    user='root',

    password='root')

cur = conn.cursor()

def hashpartition(tablename, numberofpartitions):

        list\_partition = range(numberofpartitions)

        for j in list\_partition:

                cur.execute("DROP TABLE IF EXISTS " +tablename+ str(j))

                cur.execute("CREATE TABLE "+ tablename+ str(j) + " AS SELECT \* FROM " + tablename + " WHERE dept\_id % " + str(numberofpartitions) + " = " + str(j) + ";")

def jointable(numberofpartitions):

        list\_partition = range(numberofpartitions)

        for j in list\_partition:

                cur.execute("DROP TABLE IF EXISTS sub\_dept"+ str(j))

                cur.execute("CREATE TABLE sub\_dept"+ str(j) + " AS SELECT \* FROM subject"+str(j) +" natural join department"+ str(j)+";")

def result(numberofpartitions):

        cur.execute("DROP TABLE IF EXISTS final")

        query = "CREATE TABLE final AS (SELECT \* FROM sub\_dept0"

        for j in range(1,numberofpartitions):

                query+=(" union SELECT \* FROM sub\_dept"+str(j))

        query+= " ) ;"

        cur.execute(query)

def display():

        #cur.execute("select \* from final")

        cur.execute("SELECT \* FROM final")

        # fetch all the matching rows

        result = cur.fetchall()

        print("Before Partitioning")

        # loop through the rows

        for row in result:

                print(row)

n=int(input())

hashpartition('subject',n)

hashpartition('department',n)

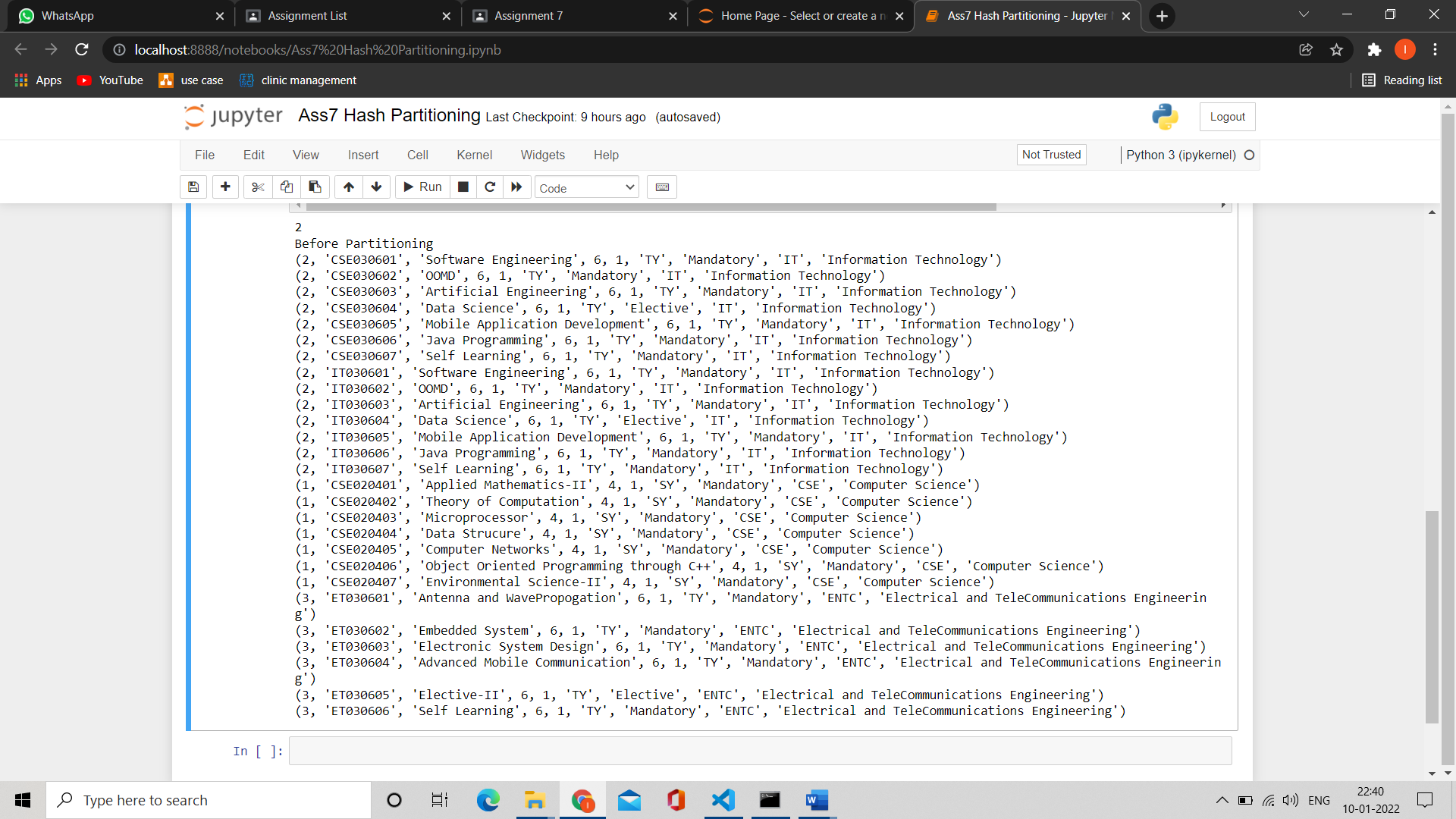
jointable(n)

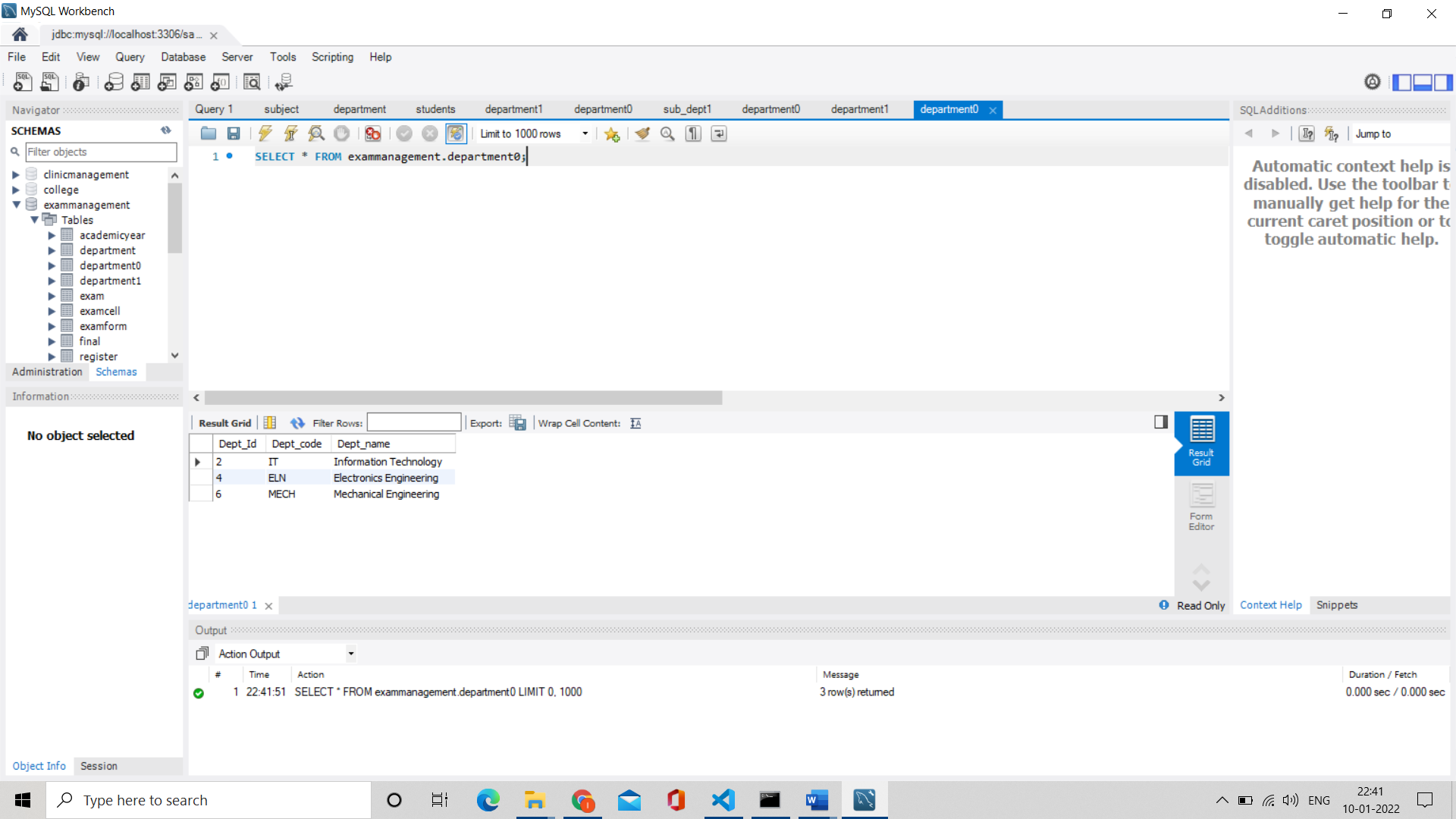
result(n)

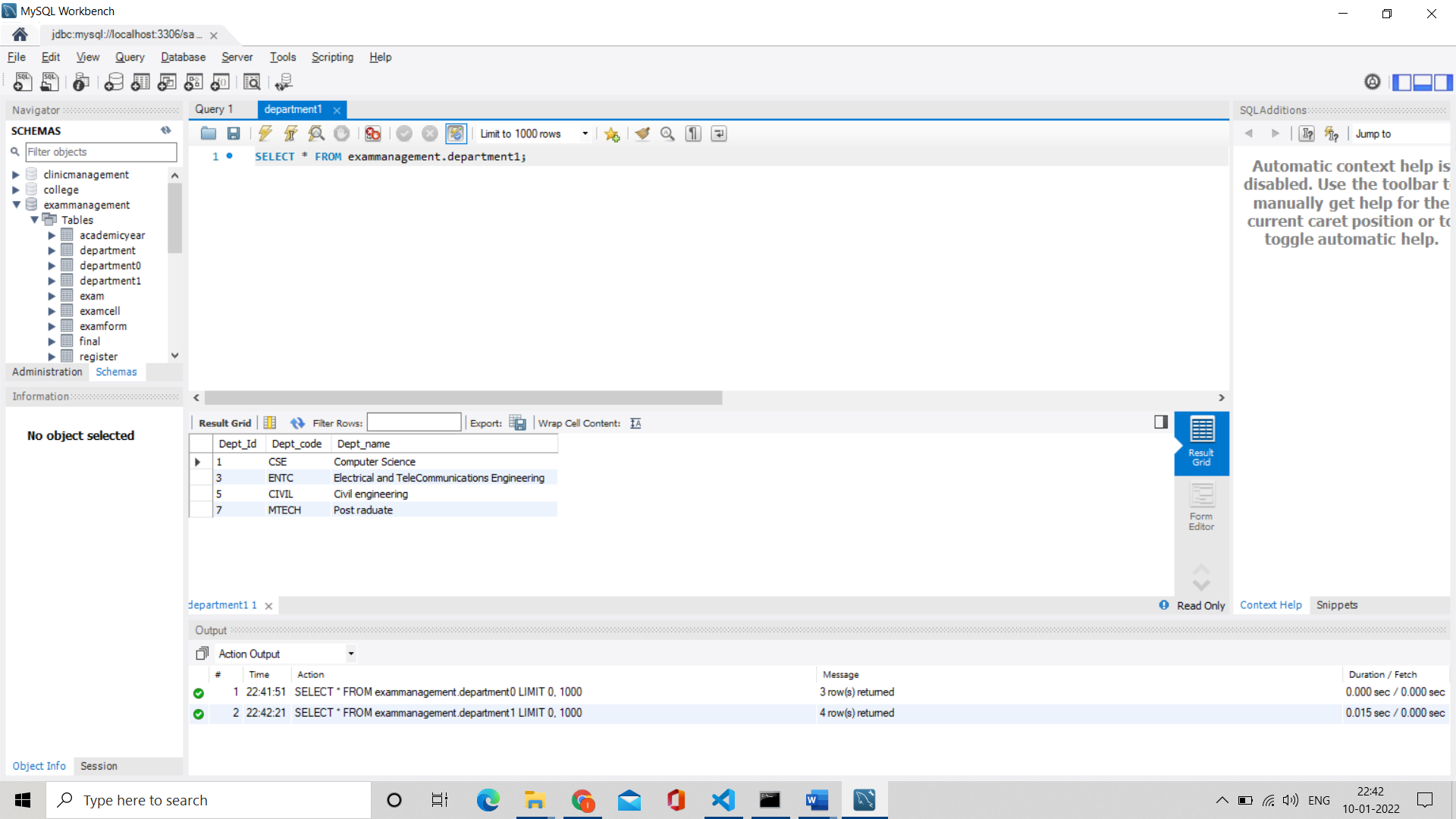
display()

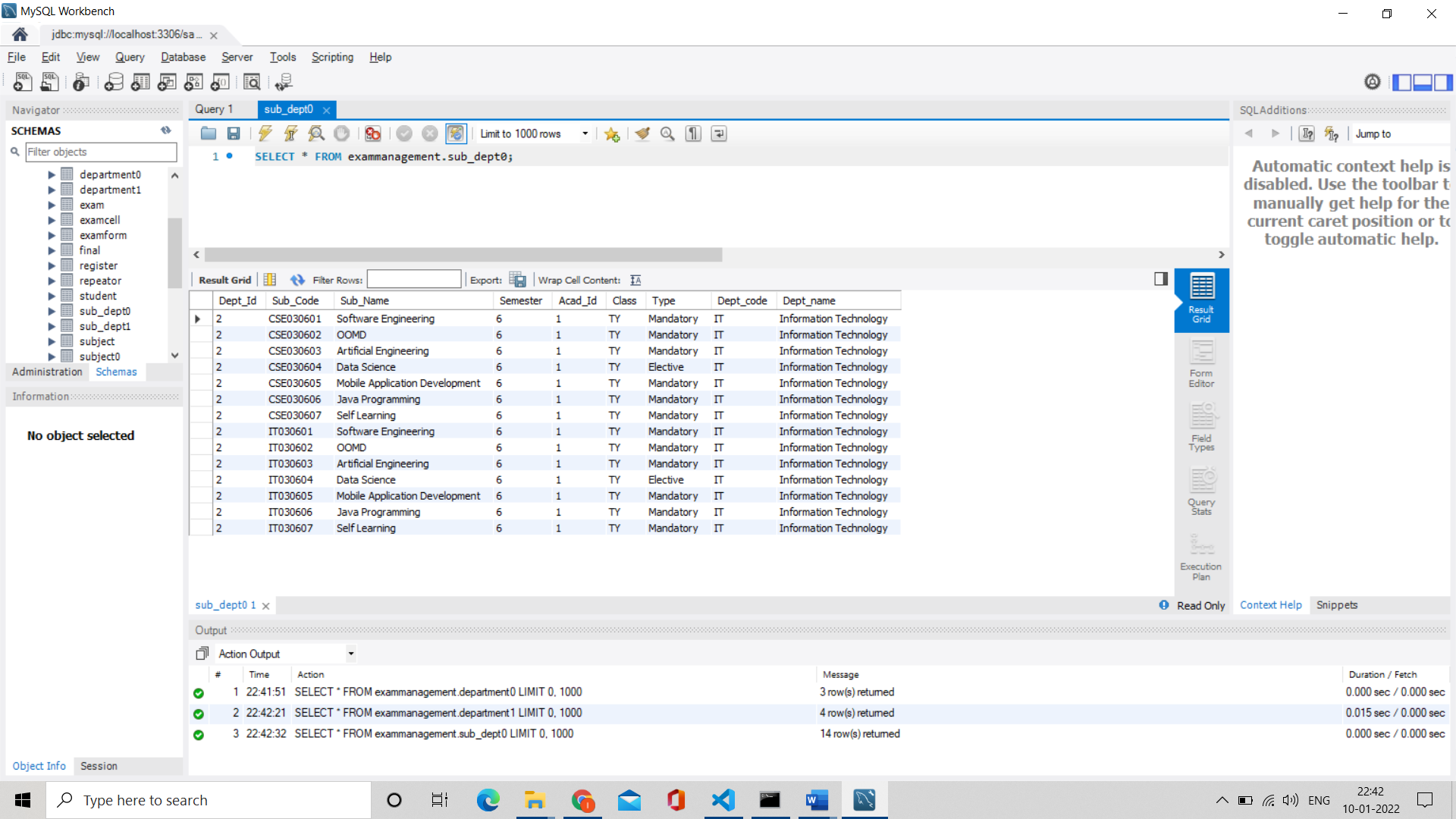
**Screenshots:**

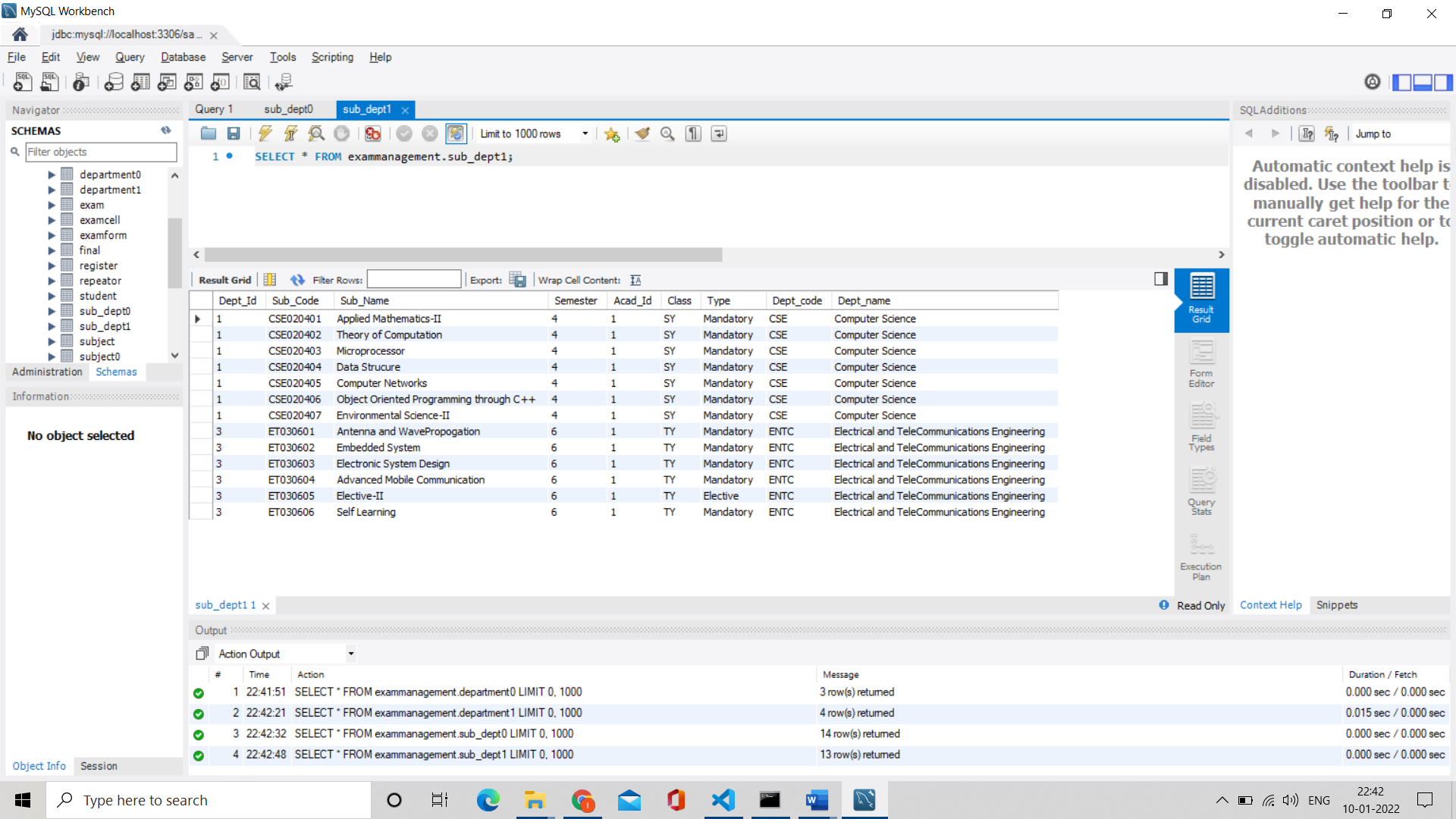
**Join using Hash Partitioning on join Attribute**











**Program :**

**Join using Range Partitioning on join Attribute**

import mysql.connector

conn = mysql.connector.connect(host='localhost',

                       database='miniexammanagement',

                       user='root',

                       password='root')

cur = conn.cursor()

def jointable(numberofpartitions):

    list\_partition = range(numberofpartitions)

    for j in list\_partition:

        cur.execute("DROP TABLE IF EXISTS sub\_dept"+ str(j))

        cur.execute("CREATE TABLE sub\_dept"+ str(j) + " AS SELECT \* FROM subject"+str(j) +" natural join department"+ str(j)+";")

n=int(input("Enter Partitions: "))

l=list(map(int,input("Enter Ranges : ").split()[:n]))

for i in range(n):

    if i == 0:

        high = l[0]

        low = 0

        cur.execute("DROP TABLE IF EXISTS subject" + str(i))

        cur.execute("DROP TABLE IF EXISTS department" + str(i))

        sql = "CREATE TABLE subject0 AS (SELECT \* FROM subject WHERE dept\_id >" + str(low) + " and dept\_id <=" + str(

            high)

        sql += " ) ; "

        cur.execute(sql)

        sql = "CREATE TABLE department0 AS (SELECT \* FROM department WHERE dept\_id >" + str(low) + " and dept\_id <=" + str(

            high)

        sql += " ) ; "

        cur.execute(sql)

    elif i == (n - 1):

        high = 9999999

        low = l[n-2]

        cur.execute("DROP TABLE IF EXISTS subject" + str(i))

        cur.execute("DROP TABLE IF EXISTS department" + str(i))

        sql = "CREATE TABLE subject" + str(i) + " AS (SELECT \* FROM subject WHERE dept\_id >" + str(low) + " and dept\_id <=" + str(high)

        sql += " ) ; "

        cur.execute(sql)

        sql = "CREATE TABLE department" + str(i) + " AS (SELECT \* FROM department WHERE dept\_id >" + str(

            low) + " and dept\_id <=" + str(high)

        sql += " ) ; "

        cur.execute(sql)

    else:

        low= l[i-1]

        high=l[i]

        cur.execute("DROP TABLE IF EXISTS subject" + str(i))

        cur.execute("DROP TABLE IF EXISTS department" + str(i))

        sql = "CREATE TABLE subject" + str(i) + " AS (SELECT \* FROM subject WHERE dept\_id >" + str(

            low) + " and dept\_id <=" + str(high)

        sql += " ) ; "

        cur.execute(sql)

        sql = "CREATE TABLE department" + str(i) + " AS (SELECT \* FROM department WHERE dept\_id >" + str(

            low) + " and dept\_id <=" + str(high)

        sql += " ) ; "

        cur.execute(sql)

jointable(n)

cur.execute("DROP TABLE IF EXISTS result")

sql = "CREATE TABLE result AS SELECT \* FROM sub\_dept0 "

for k in range(1, n):

    sql += (" union SELECT \* FROM sub\_dept" + str(k))

sql += " ; "

#print(sql)

cur.execute(sql)

**Screenshots :**

**Join using Range Partitioning on join Attribute**

