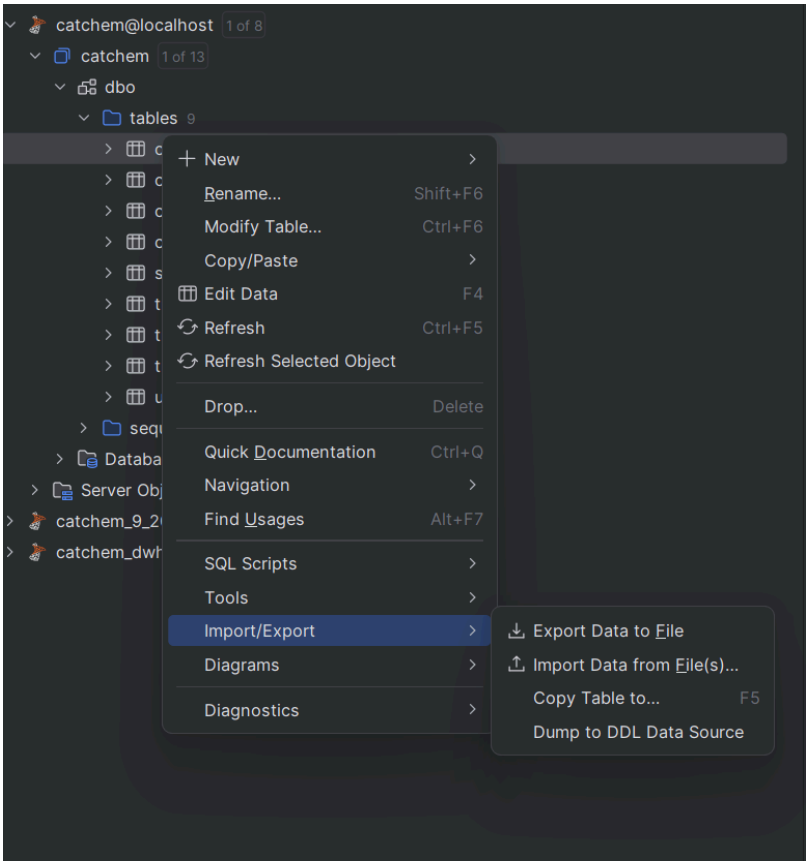
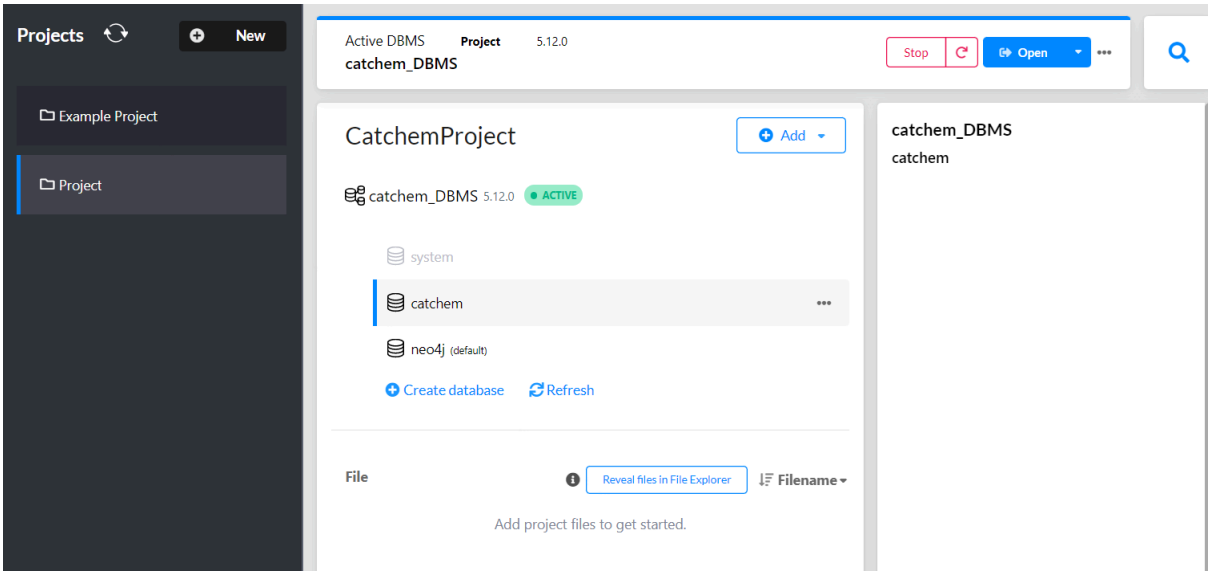
TASK5: No SQL database for Neo4j

1. **Export catchem database to csv.file**
   Connect your IDE (Pycharm) to SSMS, then choose the table to export as csv file.



2. **Create new Project and inside of it, you create new DBMS.**
   I created new DBMS called “catchem”, where I am going to import all csv file from SSMS.



3. How to import csv file to Neo4j DBMS

```
catchem$ CREATE INDEX city_id_index FOR (ci:City) ON (ci.city_id)
```

Create an index when it takes too much time to load CSV to database so that it can load data faster.

```
catchem$
```

```
:auto LOAD CSV WITH HEADERS FROM 'file:///city2.csv' AS row
CALL {
    WITH row
    MERGE (ci:City {city_id: row.city_id})
    SET ci.city_name = row.city_name,
        ci.latitude = row.latitude,
        ci.longitude = row.longitude,
        ci.postal_code = row.postal_code,
        ci.country_code = row.country_code
} IN TRANSACTIONS OF 1000 ROWS;
```

Table

Added 1229589 labels, created 1229589 nodes, set 7552534 properties, completed after 49331 ms.

```
:auto LOAD CSV WITH HEADERS FROM 'file:///stage2.csv' AS row
CALL {
    WITH row
    MERGE (s:Stage {id: row.id})
    SET s.container_size = toInteger(row.container_size),
        s.description = row.description,
        s.latitude = toFloat(row.latitude),
        s.longitude = toFloat(row.longitude),
        s.sequence_number = toInteger(row.sequence_number),
        s.type = toInteger(row.type),
        s.visibility = toInteger(row.visibility)
} IN TRANSACTIONS OF 1000 ROWS;
```

Make relationship between city and country

```
:auto LOAD CSV WITH HEADERS FROM 'file:///city2.csv' AS row
CALL {
    WITH row
    MATCH (c:City {city_id: row.city_id})
    MATCH (co:Country {code: row.country_code})
    MERGE (c)-[:LOCATED_IN]→(co)
} IN TRANSACTIONS OF 1000 ROWS;
```

Table

Code

Created 1264588 relationships, completed after 204250 ms.

Make relationship between treasure and city

```
:auto LOAD CSV WITH HEADERS FROM 'file:///treasure2.csv' AS row
CALL {
    WITH row
    MATCH (t:Treasure {treasure_id: row.id})
    MATCH (c:City {city_id: row.city_city_id})
    MERGE (t)-[:LOCATED_IN]→(c)
} IN TRANSACTIONS OF 1000 ROWS;
```

Created 13913 relationships, completed after 203494 ms.

Make relationship between hunter and treasure found

```
1  :auto LOAD CSV WITH HEADERS FROM 'file:///treasure_log2.csv' AS row
2  CALL {
3      WITH row
4      MATCH (h:Hunter {id: row.hunter_id})
5      MATCH (t:Treasure {treasure_id: row.treasure_id})
6      MERGE (h)-[:FOUND]→(t)
7  } IN TRANSACTIONS OF 1000 ROWS;
```

Table

Created 14762 relationships, completed after 445662 ms.

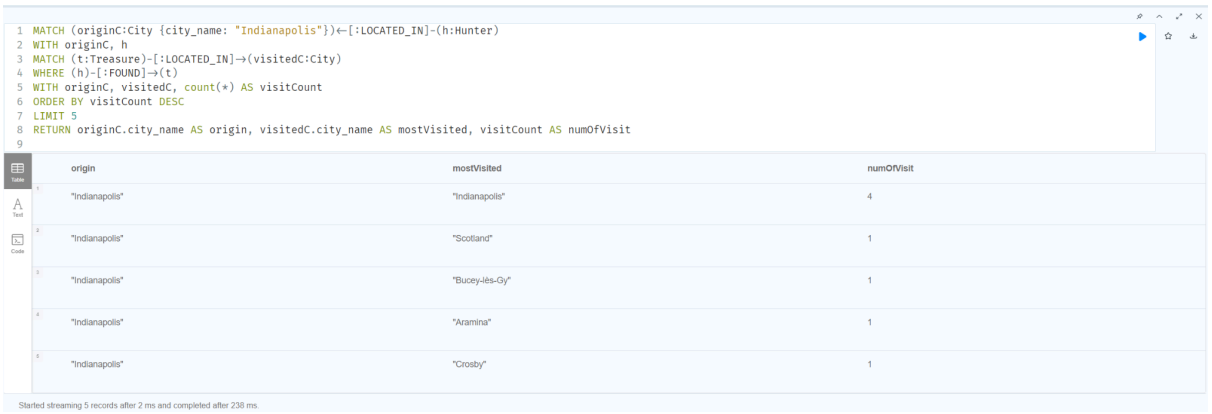
Make relationship between treasure and owner

```
:auto LOAD CSV WITH HEADERS FROM 'file:///treasure2.csv' AS row
CALL {
    WITH row
    MATCH (h:Hunter {id: row.owner_id})
    MATCH (t:Treasure {treasure_id: row.id})
    MERGE (h)-[:OWNS]→(t)
} IN TRANSACTIONS OF 1000 ROWS;
```

Created 13913 relationships, completed after 340399 ms.

Research Question

For a given city, identify which other city is strongly linked to it. You do this by checking which other cities the hunters in that city also visit.



```
MATCH (originC:City {city_name: "Indianapolis"})←[:LOCATED_IN]-(h:Hunter)
WITH originC, h
MATCH (t:Treasure)-[:LOCATED_IN]→(visitedC:City)
WHERE (h)-[:FOUND]→(t)
WITH originC, visitedC, count(*) AS visitCount
ORDER BY visitCount DESC
LIMIT 5
RETURN originC.city_name AS origin, visitedC.city_name AS mostVisited, visitCount AS numOfVisit
```

	origin	mostVisited	numOfVisit
1	"Indianapolis"	"Indianapolis"	4
2	"Indianapolis"	"Scotland"	1
3	"Indianapolis"	"Bucey-lès-Gy"	1
4	"Indianapolis"	"Aramina"	1
5	"Indianapolis"	"Crosby"	1

Started streaming 5 records after 2 ms and completed after 238 ms.

Create a query to find "fellow hunters" who do similar hunts to yourself. These are hunters who often sought the same treasures.

```
MATCH (h1:Hunter {last_name: "Brown"})-[:FOUND]→(t:Treasure)←[:FOUND]-(h2:Hunter)
WHERE h1 <> h2
RETURN DISTINCT h2.last_name AS fellow_hunter, COUNT(t) AS common_treasures
ORDER BY common_treasures DESC
```

	fellow_hunter	common_treasures
1	"Vaca"	1
2	"Crooks"	1
3	"Doyle"	1
4	"Ferry"	1

Started streaming 4 records after 9 ms and completed after 18 ms.

Top 10 Dedicators

```
MATCH (o:Hunter)-[:OWNS]→(t:Treasure)
WITH o, count(*) AS treasureCount
RETURN o.last_name AS OwnerName, treasureCount
ORDER BY treasureCount DESC
LIMIT 10
```

	OwnerName	treasureCount
1	"Nolan"	12
2	"Jones"	11
3	"Bernier"	10
4	"Weber"	10
5	"Dicki"	9
6	"Moya"	9