תרגיל בית מס' 4

נושאי התרגיל: CC, Indexes, Neo4j

- מתרגלת אחראית: מעיין ארנברג.
- ההגשה **בזוגות ומוקלדת** בפורמט PDF **בלבד**, עד לתאריך 6/7/2023.
- שאלות על התרגיל יש לשאול בפורום הייעודי בפיאצה: https://piazza.com/technion.ac.il/spring2023/236363.
- שאלות אדמיניסטרטיביות יש להפנות לאופיר פדר.
- הבהרות שנוספו לאחר פרסום התרגיל מסומנות בצהוב.

פרטי המגישים:

	ת.ז.
1.	313511602
2.	200819456

שאלה 1 – Indexes (35 נק')

מסד הנתונים "Music Maestro" מחזיק מידע על אומנים ואלבומים.

אמנים – מזהה, שם, תאריך לידה, מוצא, וז'אנר אופייני — Artists(**ID, Name, BirthDate, Nationality, Genre**)

שירים – מזהה, שם, תאריך פרסום, ומזהה אומן — Songs(**ID, Title, ReleaseDate, ArtistID**)

עבדו תחת ההנחות הבאות:

- כל בלוק מכיל 1024 בתים (bytes)
- כל רשומה בכל יחס תופסת 100 בתים
- רשומות אינן מפוצלות בין בלוקים
- במערכת רשומים 1000 אמנים ו-100,000 שירים
- אומן מוציא לכל היותר שיר אחד בחודש
- מסד הנתונים מעודכן לתאריך פרסום התרגיל (יוני 2023) ואינו מכיל שירים עם תאריך פרסום עתידי

1. התבוננו בשאילתה הבאה, למציאת השירים החדשים של כל אמני ה-Jazz:

```
SELECT A.ID, S.ID
FROM Artists A, Songs S
WHERE S.ArtistsID = A.ID AND
      A.Genre = 'Jazz' AND
      S.ReleaseDate >= TO_DATE('2023/01/01', 'YYYY/MM/DD');
```

הראו דוגמה לתכנון שאילתת nested loop join, ונתחו את ה-I\O cost בהתאם. הניחו כי במסד הנתונים אין אינדקסים כלל.

```
1)
FOR each row R1 in Artists A
  IF R1.Genre = 'Jazz' THEN
    FOR each row R2 in Songs S
      IF R2.ArtistID = R1.ID AND R2.ReleaseDate >= TO_DATE('2023/01/01', 'YYYY/MM/DD') THEN
        Output (R1.ID, R2.ID)

The I/O cost for this operation is:
B(A) = number of blocks in the Artists = 1,000 / floor(1024/100) = 100
B(S) = number of blocks in the Songs = 100,000 / floor(1024/100) = 10,000
Cost = B(A) * B(S) = 100 * 10,000 = 1,000,000 I/Os.
```

המערכת מתחזקת את הטבלאות הבאות:

בעת נוסיף למסד הנתונים את האינדקס הבא:

```
CREATE INDEX artist_date_idx ON Songs(ArtistID, ReleaseDate);
```

ידוע כי האינדקס ממומש כעץ B+ עם דרגה d=10.

2. לכל אחת מהשאילתות הבאות, תארו במילים כיצד ניתן לבצע את השאילתה באופן יעיל תוך שימוש באינדקס, ונתחו את עלות ה- I\O .

i. החזרת תאריך הפרסום של השיר החדש ביותר של אומן מסוים:

```
SELECT S.ReleaseDate
FROM Songs S
WHERE S.ArtistsID = '236363'
ORDER BY S.ReleaseDate Desc
LIMIT 1;
```

2) i.
In the given query, we're looking for the latest song from an artist with 'ArtistID' 236363. Since the B+ tree index sorts the 'ReleaseDate' in ascending order for each 'ArtistID', the database can traverse the B+ tree index to find the entries for 'ArtistID' 236363 and then quickly find the entry with the latest 'ReleaseDate' (which will be at the end of these entries).

For a B+ tree of order d, the height of the tree is $log_d(N)$, where N is the number of keys. In our case, d is given as 10, and N is the number of songs, 100,000.

The cost of traversing the B+ tree is proportional to its height, so we'll have:
$Traverse\ cost = log_d(N) = log_{10}(100{,}000) = 5$ I/Os

ii. החזרת כל השירים של אומן מסוים החל משנת 2023:

```sql
SELECT S.ID
FROM Songs S
WHERE S.ArtistsID = '236363'
AND S.ReleaseDate >= TO_DATE('2023/01/01', 'YYYY/MM/DD');
```

ii .

In this query, we are looking for all songs from the artist with 'ArtistID' 236363 that were released starting in 2023. The database can traverse the B+ tree index to find the entries for 'ArtistID' 236363, and then continue to traverse the entries to find songs with 'ReleaseDate' starting from '2023/01/01'. It will return the IDs of all matching songs.

In terms of I/O cost:

1 .Traversing the B+ tree: As with the previous query, the cost of traversing the B+ tree to find the entries for 'ArtistID' 236363 will be 5 I/Os.

2 .Fetching the records: Assuming that the artist has been releasing songs all 2023, and considering that an artist releases at most one song a month, the artist would have at most 6 months (of 2023), so we have 6 songs.

As each block contains 10 records. We will go for the extreme case that the songs split into 2 blocks.

So, the total cost will be the cost of traversing the B+ tree plus the cost of fetching the records, which is 5 + 2 = 7 I/O operations.

3. תארו ביצוע של השאילתה מסעיף 1, תוך שימוש באלגוריתם index nested loop join עם האינדקס הנתון:

```sql
SELECT A.ID, S.ID
FROM Artists A, Songs S
WHERE S.ArtistsID = A.ID
AND A.Genre = 'Jazz'
AND S.ReleaseDate >= TO_DATE('2023/01/01', 'YYYY/MM/DD');
```

3)

1. Scan each row R1 in the Artists table (A) where A.Genre = Jazz. This is the outer loop.

2. For each Jazz artist (A.ID = R1.ID), use the artist_date_idx index to find matching entries in the Songs table (S). This is an indexed lookup, and we are looking for entries where S.ArtistsID = R1.ID and S.ReleaseDate >= TO_DATE(2023/01/01, YYYY/MM/DD). This is the inner loop.

3. For each matching song, output the artist ID and song ID (R1.ID, R2.ID).

Calculate the I/O cost for this operation:

1. Scanning the Artists table: We need to read all blocks in the Artists table to find Jazz artists. Assuming the size of the Artists table remains the same (approximately 100 blocks), this will take 100 I/Os.

2. Indexed lookup on Songs: For each Jazz artist, we perform an indexed lookup using the artist_date_idx index. As we estimated earlier, assuming that there are about 100 Jazz artists, and considering that traversing the B+ tree index for each artist takes about 5 I/Os, this part will take approximately 5 * 100 = 500 I/Os.

3. Fetching the matching songs: As each Jazz artist releases at most one song per month, we can assume that from January 2023 to June 2023, each Jazz artist would have released about 6 songs. So, for 100 Jazz artists, this would be about 6 * 100 = 600 songs. Each block can contain about 10 songs, so fetching these songs would take about 600 / 10 = 60 I/Os.

The total I/O cost of this operation would be the sum of the above costs: 100 (for scanning the Artists table) + 500 (for the indexed lookup on Songs) + 60 (for fetching the matching songs) = 660 I/Os. This is better than the cost in section 1.

שאלה 2 – Currency Control (35 נק')

1. לכל אחד מהתזמונים הבאים, בדקו והסבירו:
- האם התזמון בר סידור מבט?
- האם התזמון בר סידור קונפליקט?

I. $R_2(x)\ W_1(x)\ W_2(y)\ R_3(y)\ R_1(y)\ W_2(x)$

For View Serializability, T1's write on x need to be after T2's initial read but before T2's final write. This cannot be preserved in a serial schedule, so the schedule is not view serializable.

According to the theorem Conflict serializability causes View serializability and since View serializability does not exist then Conflict serializability does not exist either.

In conclusion, the given schedule is neither Conflict Serializable nor View Serializable.

II. $R_2(z)\ W_3(z)\ W_4(z)\ R_2(y)\ W_2(x)\ R_1(z)\ R_4(y)$

We'll first evaluate this schedule for Conflict Serializability. As in the previous analysis, we'll construct a precedence graph.

1 .R_2(z) and W_3(z) conflict on z, with T3 writing after T2 reads, hence there's an edge from T2 to T3.

2 .W_3(z) and W_4(z) conflict on z, with T4 writing after T3 writes, hence there's an edge from T3 to T4.

3 .W_4(z) and R_1(z) conflict on z, with T1 reading after T4 writes, hence there's an edge from T4 to T1.

There are no cycles in this graph, so the schedule is conflict serializable. The equivalent serial schedule based on the graph is T2-> T3-> T4-> T1. We get:

$$R_2(z)R_2(y)\ W_2(x)\ \ W_3(z)\ W_4(z)\ R_4(y)R_1(z)$$

According to the theorem Conflict serializability causes View serializability.

In conclusion, the given schedule is both Conflict Serializable and View Serializable.

.III $R_1(x)\ R_2(x)\ R_1(y)\ W_2(x)\ W_3(y)\ W_1(y)\ W_2(y)$

(iii).

To check for Conflict Serializability, we need to check a precedence graph:

Between R_1(y) and W_3(y), a conflict exists as T1 reads y before T3 writes to y, leading to an edge from T1 to T3.

A conflict exists between W_3(y) and W_1(y) as T3 writes to y before T1, leading to an edge from T3 to T1.

The resultant precedence graph indicates a cycle (T1-> T3-> T1), implying the schedule is not Conflict Serializable.

For View Serializability, we analyze the operations related to each data item by each transaction:

Initial Read Property: T1 is the first to read data item x and data item y. In any view serializable schedule, T1 has to be the first to read x and y.

Final Write Property: T2 is the last to write data item x and data item y. In any view serializable schedule, T2 has to be the last to write x and y.

Direct Read Property: In the given schedule, there are no instances where a transaction reads a data item that was last written by another transaction. This means that the direct read property is satisfied for the given schedule.

Since the given schedule satisfies all the three properties required for view serializability, we can conclude that the schedule is view serializable. We get:

$$R_1(x)\ R_1(y)\ W_1(y)\ W_3(y)\ R_2(x)\ W_2(x)\ W_2(y)$$

In conclusion, the given schedule is not Conflict serializable but is View serializable.

2. בבל סעיף, הביאו דוגמה לתזמון מתאים של שתי טרנזקציות אשר להן פעולות על משתנה משותף אחד לפחות.

I. תזמון אשר יכול להתקבל ע"י פרוטוקול Strict 2PL וגם ע"י Conservative 2PL, ואינו סדרתי.

$$RL_1(x), WL_1(y), R_1(x), RL_2(x), R_2(x), RU_2(x), W_1(y), WU_1(y), RU_1(x)$$

- Strict 2PL: **locks are never released, but rather freed when the txn terminates**
- Conservative 2PL: **all locks are acquired at startup of txn.**

II. תזמון בר סידור מבט, שאינו יכול להתקבל ע"י 2PL.

```
R1(x), R2(x), W1(x), W2(x)
```

He is not Conflict serializable, therefore by theorem he is not 2PL.

But is View serializable by `R1(x), W1(x), R2(x), W2(x)`

שאלה 3 – Neo4j (30 נק')

מסד הנתונים PawsomeDB הוא מסד מסוג Neo4j, המחזיק מידע על גזעי חתולים ועל חתולי מחמד ובעליהם.

סוגי הצמתים:

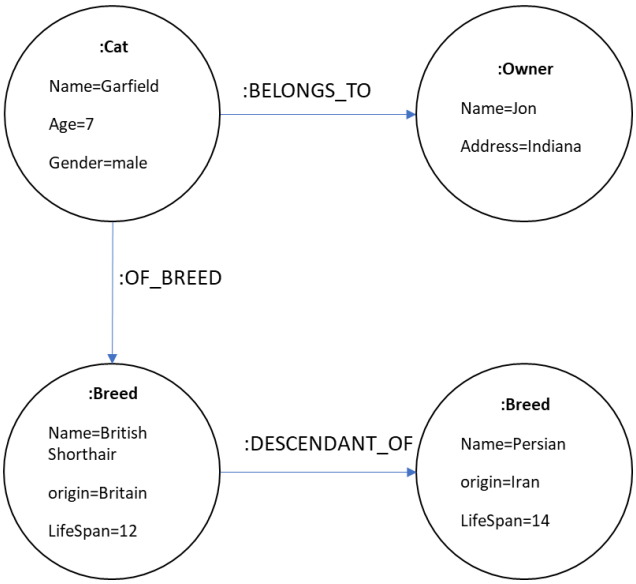
- **Label**: Breed **Properties**: name, origin, lifespan.
- **Label**: Cat **Properties**: name, age, gender.
- **Label**: Owner **Properties**: name, address.

סוגי היחסים:

- **Type**: BELONGS_TO **Properties**: relationship_length.
- **Type**: DESCENDANT_OF.
- **Type**: OF_BREED.

היחס BELONGS_TO מתקיים תמיד בין Cat ל-Owner, היחס DESCENDANT_OF בין Breed ל-Breed, והיחס OF_BREED בין Cat ל- Breed.

להלן דוגמה לתוכן ה-DB:



ענו על כל אחד מהסעיפים הבאים באמצעות שאילתת cypher.

1. לכל גזע חתולים במסד הנתונים, החזירו את שם הגזע ואת מספר החתולים המשתייכים אליו באופן ישיר (OF_BREED). אין צורך לכלול גזעים ללא חתולים.

מיינו את התוצאה בסדר יורד לפי מספר החתולים.

```
MATCH (b:Breed)<-[:OF_BREED]-(c:Cat)

RETURN b.name AS BreedName, COUNT(c) AS NumberOfCats

ORDER BY NumberOfCats DESC
```

2. חזרו על סעיף 1, והפעם לכל גזע חתולים חשבו את מספר החתולים המשתייכים אליו באופן ישיר או משתייכים לגזע צאצא. למשל, בדוגמה, Garfield משתייך באופן ישיר לגזע British Shorthair ובאופן עקיף לגזע Persian.

```
MATCH (b:Breed)<-[:OF_BREED]-(:Cat)
WITH b
OPTIONAL MATCH (b)<-[:DESCENDANT_OF*]-(descendantBreed:Breed)<-
[:OF_BREED]-(:Cat)
RETURN b.name AS BreedName, COUNT(descendantBreed) AS
NumberOfCatsIncludingDescendants
ORDER BY NumberOfCatsIncludingDescendants DESC
```

3. החזירו את שמות האנשים שכל החתולים בבעלותם מאותו מין (gender). אין צורך להחזיר אנשים שאין בבעלותם חתולים.

```
MATCH (o:Owner)<-[:BELONGS_TO]-(c:Cat)
WITH o, collect(c.gender) AS CatGenders
WHERE size(filter(g in CatGenders WHERE g = CatGenders[0])) =
size(CatGenders)
RETURN o.name AS OwnerName
```

4. החזירו את שמות האנשים שלהם חתול מכל גזע במסד הנתונים. בסעיף זה, כמו בסעיף 1, הכוונה להשתייכות ישירה לגזע (OF_BREED).

```
MATCH (o:Owner)<-[:BELONGS_TO]-(c:Cat)-[:OF_BREED]->(b:Breed)
RETURN DISTINCT o.name AS OwnerName
```