Winter 19 Assignment 2

Database Systems (Mcgill University)

School of Computer Science, McGill University

COMP-421 Database Systems, Winter 2019

Written Assignment 2: SQL and Indexing

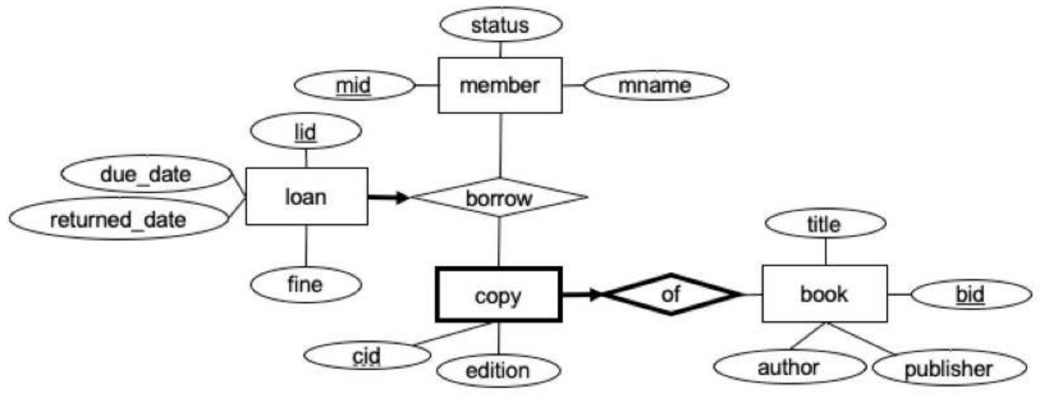
Solution Outline

This is an individual assignment. You are required to work on your own to create the solution. **There is no late submission or extension allowed** for this assignment as we need to post the solutions so that you can be prepared for the midterm.

Ex1 will be graded by an auotmated system. It is very important to read the instructions and follow them exactly. **Ex2** will be graded by the TAs.

Turn in two attachments. One for **Ex1** the tar file for automated system and another attachment for **Ex2** for the TAs to grade. For the later you may turn in a doc/pdf/txt format. Any other format, check with the TAs first.

Ex. 1 — SQL (70 Points)
The following ER model and relational model are designed for a library. Each book has a unique book id with title, author and publisher information stored in the system. A book can have multiple copies or no copies at all. Each copy has a copy id which is unique only for a given book id. Each Member is labeled with a status (i.e. *STUDENT, REGULAR, YOUTH*). For the sake of simplicity we will assume that members do not change their status. Members can borrow books and must return the book before the due date. Overdue fines will be charged if books are returned late. Fine and returned date are initially NULL in the loan table and are updated only on return of the book. Fine is updated to 0 upon return within due date. A member can borrow many copies altogether and return them separately. That is why each loan record only associates with one copy.
A sample schema and few records have been provided as **setup.sh** Please add more records into this as you may need to test various scenarios. Use your individual database accounts to work on the assignment. **DO NOT** use your project groups database account as it is shared between all of your team members.



book(<u>bid</u>, title, author, publisher)
copy(<u>bid, cid</u>, edition)
bid is a foreign key to book.
member(<u>mid</u>, mname, status)
loan(<u>lid</u>, bid, cid, mid, due_date, returned_date, fine)

mid is a foreign key to member and (bid, cid) is a foreign key to copy.

Important !!

All the sql solutions will be evaluated by an automated system, which compares the output data produced by executing your query on our dataset with the expected output result for the correct query. So it is important that you include the correct column names, in the correct order, perform any ordering on output tuples as asked etc.

Double check your SQL for **typos**, for example if you spelt '*STUDNT*' instead of '*STUDENT*', a query might not return the correct records and you will not get any points.
While the columnn and table names are not case sensitive, **the data itself can be case sensitive**. So do not write '*Student*', where it was required to write '*STUDENT*' this can produce no results or wrong results.
For more details read the attached sql formatting guide. If you have questions about this post it in the discussion forum for assignment 2. **Remember you will either get 0 or all points for a given SQL question !!**

For this assignment you will not create views or tables in your solution. You can however use derived tables as we saw in class in your SQL. All your answers should be comprised of only a select query. **Output ONLY the attributes in the question, following the exact order mentioned in the question**. Adding attributes not mentioned can result in a 0 score !

Unless specified, your output query should not produce duplicate results in your output resultset. Use the technique taught in class to eliminate duplicate records from the output.
Where an output ordering is asked for, remember to order the output records. The technique for this was also shown in class.

1. (2 Pts) List the book id, copy id and edition number of all 2nd or 3rd edition copies. Ordering the output by book id and copy id.
2. (2 Pts) List the book id of books with 3rd edition but do not have a 2nd edition. Order the output by book id.
3. (2 Pts) List the book id and title of the books ever borrowed by the member with member id 11111111. Order the output by book id.
4. (2 Pts) List the book id, title and overdue fine for loans that the member with member id 11111111 has ever been charged. Order the output by book id.
5. (3 Pts) List the book id and title of books that both member 11111114 and member 11111118 have borrowed it before. Order the output by book id.
6. (3 Pts) List the book id and title of books that have ever been borrowed by member 11111114 but not by member 11111118. Order the output by book id.
7. (2 Pts) List the book id and title of books published by *Vintage* and ever borrowed by the member with member id 11111111. WITHOUT using joins. Order the output by book id.
8. (3 Pts) List the book id and title of books published by *Vintage* and ever borrowed by the member with member id 11111111. Using joins. Order the output by book id.
9. (3 Pts) List the book id and title of books published by *Vintage* and ever borrowed by the member with member id 11111111. Using a correlated subquery. Order the output by book id.
10. (4 Pts) List the book id and title of books that have at least one copy but have not been borrowed yet. Using a correlated query. Order the output by book id.
11. (2 Pts) Find the total number of members, name the output column `nummembers`.
12. (3 Pts) Find the total number of members who have been charged at least a fine before, name the output column `nummembers`.
13. (3 Pts) List the id and name of members who have more than or equal to 3 overdue charged loan records. Order the output by member id.
14. (4 Pts) List the book id and the number of loans of each book (name this column `numloans`). Order the output by descending order of numloans and then ascending order of book id.
15. (5 Pts) List the average number of loans of a member. Name the average column `avgloans`.
16. (6 Pts) List `status` and average number of overdue loans of members each `status`. Name the average column `avgloans`. Include members with no overdue loans when computing the average. Order the output by status.

17. (5 Pts) List each book id and how many times the copies of that book have been returned by members (name this column `numloans`). If a book was never loaned, `numloans` must be 0. Ignore the books that do not have a copy in the library. Write this query without using any outer joins. Order the output by book id.
18. (6 Pts) Redo the above question with an outer join. **Hint:-** use COALESCE
19. (4 Pts) Find the book that has the highest number of loan records associated with *STUDENT* members. List the book id and the number (name this column `numloans`). Order the output by book id.
20. (6 Pts) List the member id and total overdue fine of *REGULAR* members whose total overdue fine is larger than average overdue among all members. Name the this column `totalfine`. Order the output by decreasing order of overdue fine. When calculating the average, consider for the members with the total overdue fine of 0 (include) and ignore those with only NULL values in the table.

Answer (Ex. 1) —

See the last five pages of this document.

Ex. 2 — Indexing (30 Points)

Consider the `loan` table from the previous question.

`loan(lid INTEGER, bid INTEGER, cid INTEGER, mid INTEGER, due_date DATE, returned_date DATE, fine NUMERIC(6,2))`

Here is some additional information.

- An INTEGER has 64 bits, DATE has 4 bytes and the size of NUMERIC is 8 bytes.
- There are 30,000 books in the library. Each book has 5 copies on average. For simplicity, assume each copy has an equal probability of being borrowed.
- There are 50,000 members of the library.
- 5% of the members are inactive (have never borrowed books). Each active member has 60 loan histories on average.
- Each rid has 10 Bytes, each pointer (of internal index pages) has 6 Bytes.
- Leaf pages are filled on average 60% and page size is 4000 bytes.
- Intermediate pages can have a fill factor in the range 50 - 100%. The root might have any fill factor.

Now assume there exists an indirect, clustered type II B+-tree index on lid of loan and an unclustered type II B+-tree indirect index on `(bid, cid)` columns of loan. A single data entry may NOT spread over more than one leaf page.

1. For both indices calculate:
   (a) (10 Points) *The avg. number of rids per data entry, the size of the data entry and the total number of data entries and the number of leaves.*
   (b) (5 Points) *The maximum and minimum possible number of intermediate nodes in the index (for the given possible fill factor range of 50-100%) and the height of the tree in each case.*

Answer (Ex. 2) —

1.

   Note:- minor rounding errors are fine.

   For the clustered type II B+-tree index on `lid`:

   (a) Since lid is unique, there is only 1 rid per data entry.
   Size of the data entry = sizeof(rid) + sizeof(lid) = 10 + 8 = 18 bytes
   Total number of data entries = total number of loan records = 50,000 × 95% × 60 = 2,850,000 data

entries.
Total index size in bytes = 18 bytes per data entry × 2,850,000 data entries = 51,300,000 bytes.
The number of leaf pages required to store them, given a page size of 4000 bytes and 60% usage. = 51,300,000 bytes / (4000 × 0.60) = 21375 leaves.

If you instead started by computing the number of data entries per page, it will go something like this.

4000 × 0.6/18 = 133 data entries per page.
Number of leaves = 2850000/133 = 21429

(b) The number of page pointers an intermediate nodes can hold at 50% fill rate factor will be:
4000 × 0.5 / (sizeof(lid) + sizeof(page pointer)) = 2000/(8+6) = 143 page pointers.
The number of intermediate nodes above leaves= 21375 / 143 = 149
Then a root node to hold the 149 entries.
Height of the tree of this case is 2.

The number of page pointers an intermediate nodes can hold at 100% fill rate factor will be 4000 / (sizeof(lid) + sizeof(page pointer)) = 4000/(8+6) = 285 page pointers.
The number of intermediate nodes above leaves= 21375 / 285 = 75
75 intermediate nodes can be handled with one root node.
The height of the tree of this case is 2.

For the unclustered type II B+-tree indirect index on `(bid, cid)`:

(a) The average number of rids per data entry = 50,000 × 95% × 60 records / (30,000 × 5) = 19 rids per data entry.
Size of a data entry =19 × sizeof(rid) + sizeof(bid) + sizeof(cid) = 190+16 = 206 bytes.
Total number of data entries = total number of individual copies = 30,000 × 5 = 150,000 data entries.
Total index size in bytes = 206 bytes per data entry × 150,000 data entries = 30,900,000 bytes.
The number of leaf pages required to store them, given a page size of 4000 bytes and 60% usage. = 30,900,000 bytes / (4000 × 0.6) = 12875 leaves.

If you instead started by computing the number of data entries per page, it will go something like this.

4000 × 0.6/206 = 11 data entries / page.
150,000/11 = 13,637 leaves.

(b) The number of page pointers an intermediate nodes can hold a 50% fill rate factor will be 4000 × 0.5 /(sizeof(bid) + sizeof(cid) + sizeof(page pointer)) = 2000/22 = 91 page pointers.
The number of intermediate nodes above leaves= 12875 / 91. = 142.
And a root node to hold the 142 intermediate node entries.
The height of the tree of this case is 2.

The number of page pointers an intermediate nodes can hold a 100% fill rate factor will be 4000 /(sizeof(bid) + sizeof(cid) + sizeof(page pointer)) = 4000/22 = 181 page pointers.
The number of intermediate nodes above leaves= 12875 / 181 = 72.
A root to hold these intermediate nodes.
The height of the tree of this case is 2.

```
--Question 1
SELECT bid, cid, edition
FROM copy
WHERE edition IN (2, 3)
ORDER BY bid, cid;

--Question 2
SELECT DISTINCT bid
FROM copy
WHERE edition = 3 AND edition NOT IN
  (SELECT bid FROM copy WHERE edition = 2)
ORDER BY bid;

--Question 3
SELECT DISTINCT b.bid, b.title
FROM book b, loan l
WHERE b.bid = l.bid AND l.mid = 11111111
ORDER BY b.bid;

--Question 4
SELECT b.bid, b.title, l.fine
FROM book b, loan l
WHERE b.bid = l.bid
  AND l.mid = 11111111 AND fine > 0
ORDER BY b.bid;

--Question 5
SELECT bid, title
FROM book
WHERE bid IN
(
  SELECT bid
  FROM loan
  WHERE mid = 11111114
            INTERSECT
  SELECT bid
  FROM loan
  WHERE mid = 11111118
)
ORDER BY bid;

--Question 6
```

```
SELECT bid, title
FROM book
WHERE bid IN
(
  SELECT bid
  FROM loan
  WHERE mid = 11111114
            EXCEPT
  SELECT bid
  FROM loan
  WHERE mid = 11111118
)
ORDER BY bid;

--Question 7
SELECT bid, title
FROM book
WHERE publisher = 'Vintage'
  AND bid in (SELECT bid FROM loan WHERE mid = 11111111)
ORDER BY bid;

--Question 8
SELECT DISTINCT b.bid, b.title
FROM book b, loan l
WHERE b.bid = l.bid
  AND b.publisher = 'Vintage'
  AND l.mid = 11111111
ORDER BY bid;

--Question 9
SELECT b.bid, b.title
FROM book b
WHERE publisher = 'Vintage'
  AND EXISTS (SELECT * FROM loan l WHERE l.mid = 11111111 AND l.bid = b.bid)
ORDER BY b.bid;

--Question 10
SELECT b.bid, b.title
FROM book b
WHERE EXISTS (SELECT * FROM copy c WHERE b.bid = c.bid)
     AND NOT EXISTS (SELECT * FROM loan l WHERE b.bid = l.bid)
```

```
ORDER BY b.bid;

--Question 11
SELECT COUNT(*) as nummembers FROM member;

--Question 12
SELECT COUNT(*) nummembers
FROM member
WHERE mid in (
    SELECT mid
    FROM loan
    WHERE fine > 0);

--Question 13
SELECT mid, mname
FROM member
WHERE mid IN
(
  SELECT mid
          FROM loan
          WHERE fine > 0
          GROUP BY mid
          HAVING COUNT(*) >= 3
)
ORDER BY mid;

--Question 14
SELECT bid, COUNT(*) as numloans
FROM loan
GROUP BY bid
ORDER BY numloans DESC, bid ASC;

--Question 15
SELECT AVG(cnt) AS avgloans
FROM
(
  SELECT count(*) cnt
  FROM loan
  GROUP BY mid
)t

--Question 16
```

```
SELECT status, AVG(cnt) AS avgloans
FROM
(
   SELECT mid, count(*) AS cnt
   FROM loan
   WHERE fine>0
   GROUP BY mid
   UNION
   SELECT mid, 0 AS cnt
   FROM member
   WHERE mid NOT IN (SELECT mid
   FROM loan
   WHERE fine>0)
)t, member m
WHERE t.mid = m.mid
GROUP BY status
ORDER BY status

--Question 17
SELECT bid, COUNT(*) as numloans
FROM loan
WHERE returned_date IS NOT NULL
GROUP BY bid
UNION
SELECT bid, 0 as numloans
FROM copy
WHERE bid NOT IN ( SELECT bid
              FROM loan
              WHERE returned_date IS NOT NULL )
ORDER BY bid

--Question 18
SELECT DISTINCT c.bid, COALESCE(cnt, 0) as numloans
FROM
(
  SELECT bid, COUNT(*) as cnt
  FROM loan
  WHERE returned_date IS NOT NULL
  GROUP BY bid
)l RIGHT OUTER JOIN copy c
  ON l.bid = c.bid
ORDER BY c.bid;
```

```
--Question 19
SELECT bid, COUNT(*) as numloans
FROM loan l
WHERE mid in (SELECT mid
              FROM member
              WHERE status = 'STUDENT')
GROUP BY bid
HAVING COUNT(*) >= ALL
(
  SELECT COUNT(*) cnt
  FROM loan
  WHERE mid in (SELECT mid
                FROM member
                WHERE status = 'STUDENT')
  GROUP BY bid
)
ORDER BY bid

--Question 20
SELECT m.mid, SUM(fine) as totalfine
FROM member m, loan l
WHERE m.mid = l.mid AND m.status = 'REGULAR'
GROUP BY m.mid
HAVING SUM(fine) > (SELECT AVG(totalfine)
                    FROM (SELECT mid, SUM(fine) as totalfine
                    FROM loan
                    GROUP BY mid)t)
```