SQL: Aggregate and Nested Queries

Students at the National University of Ngendipura (NUN) buy books for their studies. They also lend and borrow books to and from other students. Your company, Apasaja Private Limited, is commissioned by NUN Students Association (NUNStA) to implement an online book exchange system that records information about students, books that they own and books that they lend and borrow.

The database records the name, faculty, and department of each student. Each student is identified in the system by her email. The database also records the date at which the student joined the university (year attribute).

The database records the title, authors, publisher, year and edition and the ISBN-10 and ISBN-13 for each book. The International Standard Book Number, ISBN-10 or -13, is an industry standard for the unique identification of books. It is possible that the database records books that are not owned by any students (because the owners of a copy graduated or because the book was advised by a lecturer for a course but not yet purchased by any student.)

The database records the date at which a book copy is borrowed and the date at which it is returned. We refer to this information as a loan record.

For auditing purposes the database records information about the books, the copies and the owners of the copies as long as the owners are students or as there are loan records concerning the copies. For auditing purposes the database records information about graduated students as long as there are loan records concerning books that they owned.

This tutorial uses the schema and data for the database created in "SQL: Creating and Populating Tables" including all the updates done during the tutorial.

Questions

Not all questions will be discussed during tutorial. You are expected to attempt them before coming to the tutorial. You may be randomly called to present your answer during tutorial. You are encouraged to discuss them on Canvas Discussion.

1. **Aggregate Queries**.

   (a) How many loans involve an owner and a borrower from the same department?

   Comments:
   One alternative is the following.

   Code: Query

```
SELECT COUNT(*)
FROM loan l, student s1, student s2
WHERE l.owner = s1.email
  AND l.borrower = s2.email
  AND s1.department = s2.department;
```

   (b) For each faculty, print the number of loans that involve an owner and a borrower from this faculty?

   Comments:
   One alternative is the following.

   Code: Query

```
SELECT d1.faculty, COUNT(*)
FROM loan l, student s1, student s2, department d1, department d2
WHERE l.owner = s1.email
  AND l.borrower = s2.email
  AND s1.department = d1.department
  AND s2.department = d2.department
  AND d1.faculty = d2.faculty
GROUP by d1.faculty;
```

   (c) What are the average and the standard deviation [3] of the duration of a loan in days?

   Comments:
   One alternative is the following.

   Code: Alternative #1

```
SELECT CEIL(AVG((CASE
      WHEN l.returned ISNULL THEN CURRENT_DATE
      ELSE l.returned
    END) - l.borrowed + 1)),
  CEIL(STDDEV_POP((CASE
      WHEN l.returned ISNULL THEN CURRENT_DATE
      ELSE l.returned
    END ) - l.borrowed + 1))
FROM loan l;
```

   Another is the following.

Code: Alternative #2

```
SELECT CEIL(AVG(temp.duration)), CEIL(STDDEV_POP(temp.duration))
FROM (
    SELECT((CASE
        WHEN l.returned ISNULL THEN CURRENT_DATE
        ELSE l.returned
      END ) - l.borrowed + 1) AS duration
    FROM loan l
) AS temp;
```

There is another alternative using `COALESCE` but it is left as an exercise.

2. **Nested Queries**.

   (a) Print the titles of the different books that have never been borrowed. Use a nested query.

Comments:
There is no such book if you did not inserted one during the previous tutorial. You can create some records if you want a non-empty result.

Code: Alternative #1

```
SELECT b.title
FROM book b
WHERE b.ISBN13 NOT IN (
  SELECT l.book
  FROM loan l);
```

Equivalently by definition of `NOT IN`, we have the following.

Code: Alternative #2

```
SELECT b.title
FROM book b
WHERE b.ISBN13 <> ALL (
  SELECT l.book
  FROM loan l);
-- x NOT IN s  ===  forall y IN S : x <> y
```

Always use one of the quantifiers `ALL` or `ANY` in front of subqueries wherever possible even though some systems may be lenient with this requirement. You should still use the quantifiers `ALL` or `ANY` even if the result is only a single value (i.e., scalar subquery). This may prevent careless mistakes.

Note that it is possible for the result to contain the same title several times (*since there could be different books with the same title*). There is no need to use `DISTINCT` as the query asks for the different books, not for the different titles. In fact, using `DISTINCT` is a mistake here.

(b) Print the name of the different students who own a copy of a book that they have never lent to anybody.

Comments:

One alternative is the following.

Code: Alternative #1

```
SELECT s.name
FROM student s
WHERE s.email IN (
  SELECT c.owner
  FROM copy c
  WHERE NOT EXISTS (
    SELECT *
    FROM loan l
    WHERE l.owner = c.owner
      AND l.book = c.book
      AND l.copy = c.copy));
```

Equivalently by definition of `IN`, we have the following.

Code: Alternative #2

```
SELECT s.name
FROM student s
WHERE s.email = ANY (
  SELECT c.owner
  FROM copy c
  WHERE NOT EXISTS (
    SELECT *
    FROM loan l
    WHERE l.owner = c.owner
      AND l.book = c.book
      AND l.copy = c.copy));
```

The query can also be written as follows but the highlighted tuple construction does not always work on other systems than PostgreSQL.

Code: Alternative #3: Tuple Construction

```
SELECT s.name
FROM student s
WHERE s.email IN (
  SELECT c.owner
  FROM copy c
  WHERE (c.owner, c.book, c.copy) NOT IN ( -- this line may not work
    SELECT l.owner, l.book, l.copy
    FROM loan l));
```

The following **incorrect** query would print several times the names of students who own several copies that have never been borrowed if such cases occurred. We would not be able to differentiate the repeated names of students who own several copies that have never been borrowed from the repeated names of different students

with the same name.

Code: Incorrect Query

```
-- INCORRECT
SELECT s.name
FROM student s, copy c
WHERE s.email = c.owner
AND NOT EXISTS (
  SELECT *
  FROM loan l
  WHERE l.owner = c.owner
    AND l.book = c.book
    AND l.copy = c.copy);
```

(c) For each department, print the names of the students who lent the most.

Comments:

One alternative is the following.

Code: Query

```
SELECT s.department, s.name, COUNT(*)
FROM student s, loan l
WHERE l.owner = s.email
GROUP BY s.department, s.email, s.name
HAVING COUNT(*) >= ALL (
  SELECT COUNT(*)
  FROM student s1, loan l1
  WHERE l1.owner = s1.email
    AND s.department = s1.department
  GROUP BY s1.email);
```

Notice that there are three such students in the department of geography (that is why one should almost never use *TOP N* queries, for instance, using `LIMIT` [4]).

If we create a new department called Undecidable Computations with some students who never borrowed any book, what would happen? If there were students in the department of Undecidable Computations, should we print all of them or none of them? They would all have borrowed zero book, which would be the maximum in the department... We should print them all (using `OUTER JOIN`, `CASE-WHEN`, and `ISNULL` to consider the cases of 0 loan). Are there students who never borrowed a book?

Note that we need to group by `s.name` in order to print the name although there is no ambiguity. Some systems, like PostgreSQL, relax this rule. It is recommended not to use this relaxation for the sake of portability.

(d) Print the emails and the names of the different students who borrowed all the books authored by Adam Smith.

Comments:

There is no such student. We can create one student with the corresponding records as follows.

Code: Test Case

```
INSERT INTO loan VALUES
  ('nihanran1989@msn.com', 'choyweixiang2011@gmail.com',
   '978-0553585971', 1, '2024-03-10', NULL);
```

Code: Query

```
SELECT s.email, s.name
FROM student s
WHERE NOT EXISTS (
  SELECT *
  FROM book b
  WHERE authors = 'Adam Smith'
    AND NOT EXISTS (
      SELECT *
      FROM loan l
      WHERE l.book = b.ISBN13
      AND l.borrower = s.email));
```

This is the typical query for *universal quantification*.

Comments:

As a good practice, try to make your query easily readable and understandable. One way is to use *indentation* to "group" the nested queries in such a way that queries in the same nesting have the same indentation level.

References

[1] S. Bressan and B. Catania. *Introduction to Database Systems*. McGraw-Hill Education, 2006. ISBN: 9780071246507.

[2] Hector Garcia-Molina, Jeffrey D. Ullman, and Jennifer Widom. *Database Systems: The Complete Book*. 2nd ed. Prentice Hall Press, 2008. ISBN: 9780131873254.

[3] *PostgreSQL Docs: Aggregate Functions*. `https://www.postgresql.org/docs/current/functions-aggregate.html`. [Online; last accessed 2025].

[4] *PostgreSQL Docs: SELECT (LIMIT)*. `https://www.postgresql.org/docs/current/sql-select.html#SQL-LIMIT`. [Online; last accessed 2025].

[5] Raghu Ramakrishnan and Johannes Gehrke. *Database Management Systems*. 2nd. USA: McGraw-Hill, Inc., 2000. ISBN: 0072440422.

[6] *W3schools Online Web Tutorials*. `https://www.w3schools.com/`. [Online; last accessed 2025].