Tutorial: Aggregate and Nested Queries

This tutorial uses the schema and data of the database created in Tutorial 2.

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

Solution:

```
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?

Solution:

```
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 of the duration of a loan in days?

Solution:

```
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;
```

or

```
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;
```

2. Nested Queries

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

Solution: There is no such book if you have not inserted one during last tutorial. You may create some with the corresponding records if you want a non-empty result.

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

or, equivalently,

```
SELECT  b.title
FROM book b
WHERE b.ISBN13 <> ALL (
  SELECT  l.book
  FROM loan l);
```

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.

Note that there could be several time the same title (since there could be different books with the same title) but not the same book. There is no need to use `DISTINCT` since the query ask for the different books but not for the different titles.

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

Solution:

```
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));
```

or, equivalently,

```
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.

```
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
      (SELECT l.owner, l.book, l.copy
      FROM loan l));
```

The following 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.

```
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.

Solution:

```
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).

If we create a new department calle dUndecidable 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` and `ISNULL` to consider the cases of 0 loan). Are there students who never borrowed a book?

Note that we need to group by department in order to print the department 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.

Solution:

There is no such student. You may create some with the corresponding records if you want a non-empty result.

```
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
```

```
11          AND l.borrower = s.email));
```

References

[1] W3schools online web tutorials. `www.w3schools.com`. Visited on 26 July 2021.

[2] S. Bressan and B. Catania. *Introduction to Database Systems*. McGraw-Hill Education, 2006.

[3] H. Garcia-Molina, J.D. Ullman, and J. Widom. *Database Systems: The Complete Book*. Pearson international edition. Pearson Prentice Hall, 2009.

[4] R. Ramakrishnan and J. Gehrke. *Database Management Systems*. McGraw-Hill, 2002.