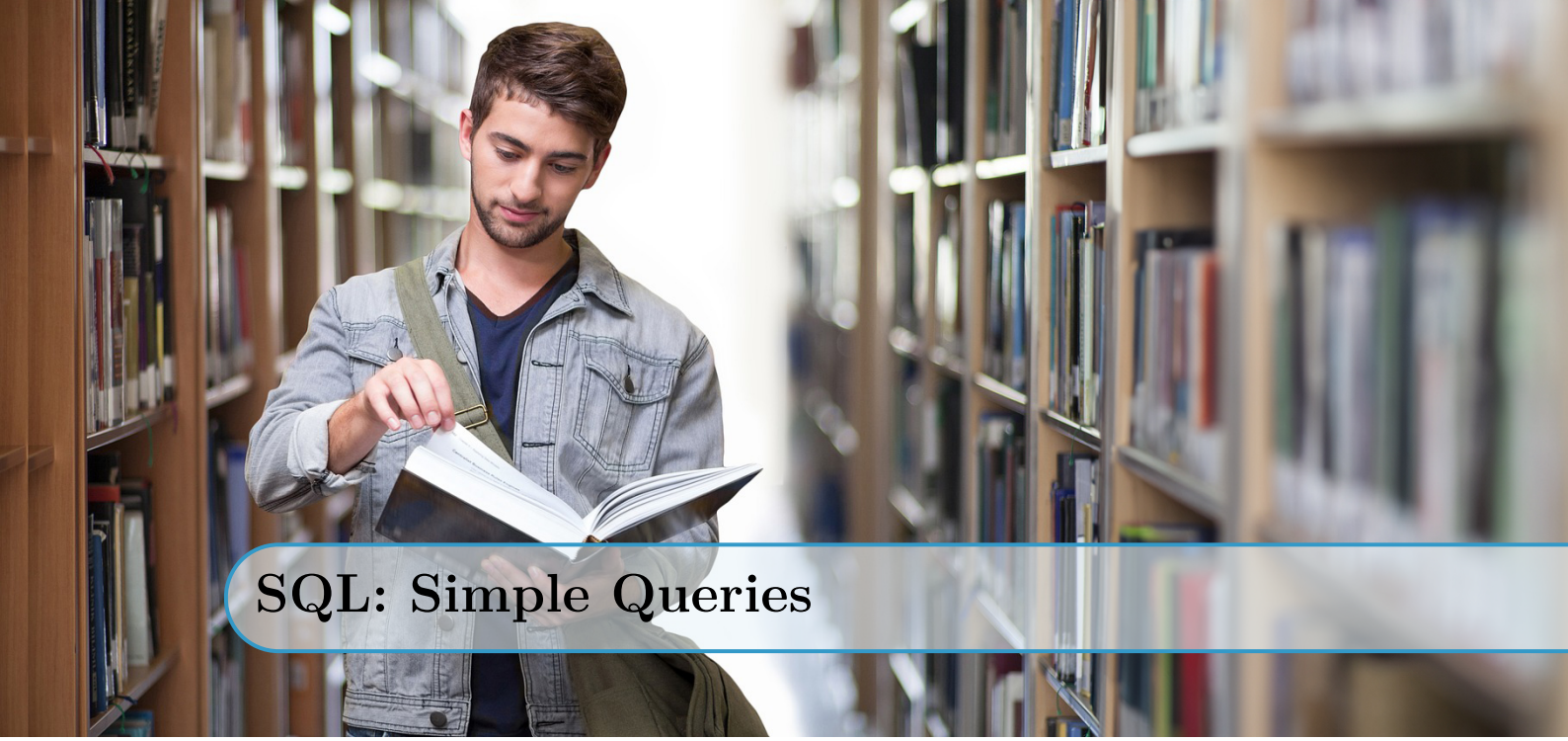

SQL: Simple Queries

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.

Important: This tutorial is designed to be solved using **simple queries only**. This means your answers should not contain nested or aggregate queries.

1. **Single-Table Queries**.

   (a) Print the different departments.

   Comments:
   One alternative is the following.

   Code:

   ```
   SELECT d.department
   FROM department d;
   ```

   Notice that the query does not require `DISTINCT` keyword to eliminate duplicate. Duplicates are guaranteed not to occur because `department` is the `PRIMARY KEY` of the table `department`.

   Further notice that we are *aliasing* the table `department` with the name `d`. Additionally, we use `d.department` to refer to the column and not just `department`. Both are good practices to ensure the code's readability.

   (b) Print the different departments in which students are enrolled.

   Comments:
   There could be departments in which no student is enrolled. This is the case for the department of `"Undecidable Computation"`. We need to look into the `student` table instead.

   Code:

   ```
   SELECT DISTINCT s.department
   FROM student s;
   ```

   Notice that the query requires the `DISTINCT` keyword to eliminate duplicates since it is very likely that there is more than one student in most departments.
   It is also important to understand the wording of the question. We require **different departments**. Hence, there should be no duplicate department. Consider an alternative wording below.

   > Print the department of the different students that are enrolled.

   Here, the emphasis is on the **different students**. The query will be similar to above *except* that `DISTINCT` should not be used. A student is identified by their `email` and not `department`. So we should not make the `department` distinct.

(c) For each copy that has been borrowed and returned, print the ISBN13 of the book and the duration of the loan. Order the results in ascending order of the ISBN13 and descending order of duration. Remember to use only one single table.

Comments:

One possible answer:

Code: Alternative #1

```
SELECT l.book, l.returned - l.borrowed + 1 AS duration
FROM loan l
WHERE l.returned IS NOT NULL -- equivalently NOT (l.returned ISNULL)
ORDER BY l.book ASC, duration DESC;
```

Since `x IS NOT NULL` is equivalent to `NOT (x ISNULL)`, we have an alternative solution as shown in the comment above. Note that for sorting, `ASC` is the default but we *highly recommend* indicating it for clarity.

Extra challenge: Can you modify the query to also print the loan duration of copies that have **NOT** been returned, where the duration is calculated until the *current date*?

Answer: To handle `NULL` returned dates for unreturned copies, we can use `COALESCE` or `CASE-WHEN`. To get the current date, use the **variable** `CURRENT_DATE`.

Code: Alternative #2: COALESCE

```
SELECT l.book,
  (COALESCE(l.returned, CURRENT_DATE) - l.borrowed + 1) AS duration
FROM loan l
ORDER BY l.book ASC, duration DESC;
```

Code: Alternative #3: CASE-WHEN

```
SELECT l.book,
  ((CASE
    WHEN l.returned ISNULL THEN CURRENT_DATE
    ELSE l.returned
  END) - l.borrowed + 1) AS duration
FROM loan l
ORDER BY l.book ASC, l.duration DESC;
```

2. **Multi-Table Queries**.

   (a) For each loan of a book published by Wiley that has not been returned, print the title of the book, the name and faculty of the owner and the name and faculty of the borrower.

Comments:
We join the `PRIMARY KEY` and the `FOREIGN KEY` to stitch the tables together.

Code: Alternative #1: Joining all five tables

```
SELECT b.title,
  s1.name AS ownerName,
  d1.faculty AS ownerFaculty,
  s2.name AS borrowerName,
  d2.faculty AS borrowerFaculty
FROM loan l, book b, copy c,
  student s1, student s2,
  department d1, department d2
WHERE l.book = b.ISBN13
  AND c.book = l.book
  AND c.copy = l.copy
  AND c.owner = l.owner
  AND l.owner = s1.email
  AND l.borrower = s2.email
  AND s1.department = d1.department
  AND s2.department = d2.department
  AND b.publisher = 'Wiley'
  AND l.returned ISNULL;
```

You can omit the table `copy` and the `copy` column since the existence of the corresponding rows and values is guaranteed by design (i.e., by the schema) and `PRIMARY KEY` constraints.

Code: Alternative #2: Omit table "copy"

```
SELECT b.title,
  s1.name AS ownerName,
  d1.faculty AS ownerFaculty,
  s2.name AS borrowerName,
  d2.faculty AS borrowerFaculty
FROM loan l, book b, -- no more copy table
  student s1, student s2,
  department d1, department d2
WHERE l.book = b.ISBN13
  AND l.owner = s1.email
  AND l.borrower = s2.email
  AND s1.department = d1.department
  AND s2.department = d2.department
  AND b.publisher = 'Wiley'
  AND l.returned ISNULL;
```

The above query is also equivalent to:

Code: Alternative #3: INNER JOIN instead of cross product

```
SELECT b.title,
  s1.name AS ownerName,
  d1.faculty AS ownerFaculty,
  s2.name AS borrowerName,
  d2.faculty AS borrowerFaculty
FROM loan l
  INNER JOIN book b ON l.book = b.ISBN13
  INNER JOIN student s1 ON l.owner = s1.email
  INNER JOIN student s2 ON l.borrowed = s2.email
  INNER JOIN department d1 ON s1.department = d1.department
  INNER JOIN department d2 ON s2.department = d2.department
WHERE b.publisher = 'Wiley' AND l.returned ISNULL;
```

In the last solution, the `ON` clause is used for joining the tables (i.e., connecting `PRIMARY KEY` to `FOREIGN KEY`) while the `WHERE` clause is used for additional filters. This is a good practice to improve code clarity. When using `INNER JOIN`, `ON` clause and `WHERE` clause can be interchangeable as long as the attributes make sense.

This is not the case for `OUTER JOIN`. You will need to think very carefully about the join condition for `OUTER JOIN`.

(b) Let us check the integrity of the data. Print the different emails of the students who borrowed or lent a copy of a book before they joined the University. There should not be any.

Comments:

One alternative is the following.

Code: Alternative #1

```
SELECT DISTINCT s.email
FROM loan l, student s
WHERE (s.email = l.borrower OR s.email = l.owner)
  AND l.borrowed < s.year;
```

Equivalently, we have the following solution by distributing `l.borrowed < s.year` into the disjunction.

Code: Alternative #2

```
SELECT DISTINCT s.email
FROM loan l, student s
WHERE (s.email = l.borrower AND l.borrowed < s.year)
   OR (s.email = l.owner AND l.borrowed < s.year);
   -- (x OR y) AND z  ===  (x AND z) OR (x AND y)
```

Other correct solutions can use `CROSS JOIN`, `INNER JOIN`, or `UNION`.

(c) Print the emails of the different students who borrowed or lent a copy of a book on the day that they joined the university.

Comments:
The following query is generally preferrable **BUT** it requires an explicit `DISTINCT` keyword. Additionally, it is not easily extensible to the next 2 questions.

Code:

```
SELECT DISTINCT s.email
FROM loan l, student s
WHERE (s.email = l.borrower OR s.email = l.owner)
  AND l.borrowed = s.year;
  -- Can you rewrite this with INNER JOIN?
```

Alternatively, we can use `UNION`. The `DISTINCT` keyword is not needed because `UNION` (as well as `INTERSECT` and `EXCEPT`) already eliminates duplicates.

Code: using UNION

```
SELECT s.email
FROM loan l, student s
WHERE s.email = l.borrower AND l.borrowed = s.year
UNION
SELECT s.email
FROM loan l, student s
WHERE s.email = l.owner AND l.borrowed = s.year;
```

(d) Print the emails of the different students who borrowed and lent a copy of a book on the day that they joined the university.

Comments:
We could also make use of set operators like the previous part, using `INTERSECT` here.

Code: using INTERSECT

```
SELECT s.email
FROM loan l, student s
WHERE s.email = l.borrower AND l.borrowed = s.year
INTERSECT
SELECT s.email
FROM loan l, student s
WHERE s.email = l.owner AND l.borrowed = s.year;
```

An equivalent query without `INTERSECT` is more complicated as it requires two `loan` tables.

Code: no INTERSECT

```
SELECT DISTINCT s.email
FROM loan l1, loan l2, student s
WHERE s.email = l1.borrower AND l1.borrowed = s.year
  AND s.email = l2.owner AND l2.borrowed = s.year;
  -- Can you rewrite this with INNER JOIN?
```

We cannot simply use a single `loan` table (let's alias that table as `l`) because the following condition has a different meaning.

```
s.email = l.borrowed AND s.email = l.owner
```

The above condition implies `l.borrowed = l.owner` by *transitivity* of equality. This means, the student is borrowing their own books which is only a subset of the solution.

(e) Print the emails of the different students who borrowed but did not lend a copy of a book on the day that they joined the university.

Comments:

Again we can use a set operator, `EXCEPT`.

Code: using EXCEPT

```
SELECT s.email
FROM loan l, student s
WHERE s.email = l.borrower AND l.borrowed = s.year
EXCEPT
SELECT s.email
FROM loan l, student s
WHERE s.email = l.owner AND l.borrowed = s.year;
```

There is no alternative simple query without using `EXCEPT`. We need to use *nested* or *aggregate* queries to write alternative answers to this type of question.

(f) Print the different ISBN13 of the books that have never been borrowed.

Comments:

There is no such book. You may create some records yourself if you want a non-empty result.

Code: Alternative #1: EXCEPT

```
SELECT b.ISBN13
FROM book b
EXCEPT
SELECT l.book
FROM loan l;
```

We can also use `OUTER JOIN`. Note that this introduces `NULL` values. We can then check for those `NULL` values using `ISNULL`.

Code: ALternative #2: OUTER JOIN

```
SELECT b.ISBN13
FROM book b LEFT OUTER JOIN loan l ON b.ISBN13 = l.book
WHERE l.book ISNULL;
```

Comments:

Queries should solve the corresponding question regardless of the current data set being used. This means your code should still work with other data sets. We will test with other data sets satisfying the schema.

This means *hardcoding* is strictly not allowed. Use of constants is prohibited *unless* they are explicitly mentioned in the question. For example, if "Changi Airport" is mentioned, you may use it (e.g., `'Changi Airport'`). However, other related constants (e.g., `'Singapore'` because Changi Airport is in Singapore or `'SIN'` as the IATA code for Changi Airport) are not allowed.

Note that SQL queries tend to be short with small changes affecting the entire output. This means, it is difficult to argue about the "logic" of the query especially when there are partial marks. For partial marks to be awarded (*if any*), (i) the query must be executable, (ii) the columns must be correct (i.e., correct name after aliasing, correct data type, correct order, etc), and (iii) the difference in the rows must be minimal.

Codes that cannot be executed may receive immediate 0 marks.

Finally, although SQL queries are case-insensitive, we highly recommend using uppercase for keywords. This eases reading the query. However, in the industry, please follow the convention set by the company.

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

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