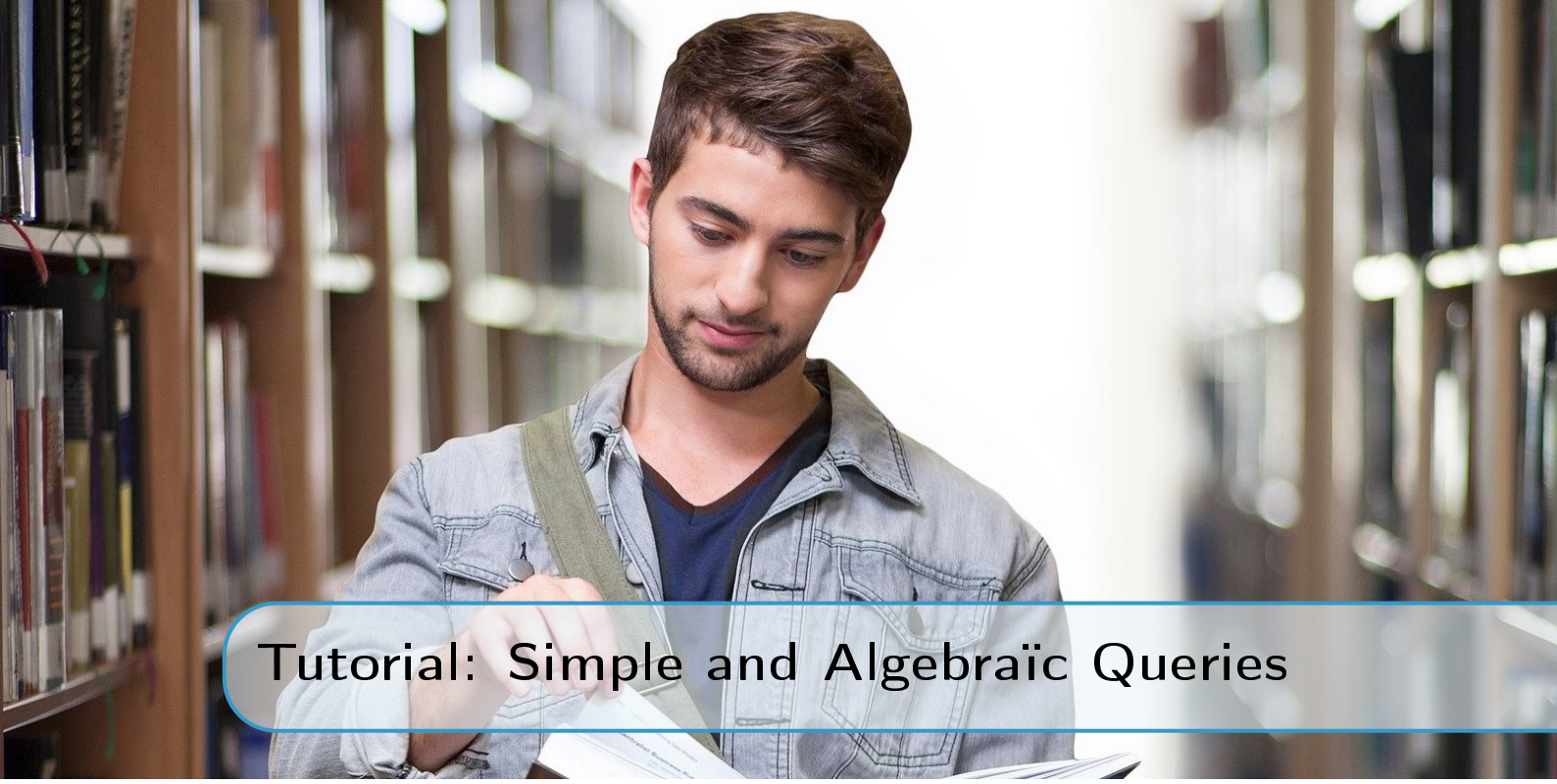

Tutorial: Simple and Algebraïc Queries

This tutorial uses the schema and data of the database created in Tutorial 1.

1. Simple Queries.

   (a) Print the different departments.

   Solution:

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

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

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

   Solution:

   There could be departments in which no student is enrolled. This is the case of the department of `Undecidable Computations`. We need to look into the `student` table.

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

   Notice that the query requires `DISTINCT` to eliminate duplicates since it is very likely that there is more than one student in most departments.

   (c) Let us check the integrity of the data. Print the emails of the students who borrowed or lent a copy of a book before they joined the university. There should not be any. Use a simple query.

   Solution:

```
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, as follows.

```
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);
```

   Other correct but less preferred solutions use `CROSS JOIN`, `INNER JOIN`, `UNION`, nested queries, etc.

(d) 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.

Solution:

```
SELECT book, returned - borrowed + 1 AS duration
FROM loan
WHERE returned IS NOT NULL
ORDER BY book ASC, duration DESC;
```

Equivalently, as follows.

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

`ASC` is the default, but it is strongly recommended to indicate it for clarity.

Notice that the duration can be null if the book has not been returned yet. For a complete answer, you need to calculate the duration until the current date to include the books that have not been returned yet.

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

Or, as folows.

```
SELECT book,
  ((CASE
  WHEN returned ISNULL
  THEN '2010-12-31'
  ELSE returned
  END) - borrowed + 1) AS duration
FROM loan
ORDER BY book ASC, duration ASC;
```

(e) 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.

Solution:

We join primary keys and foreign keys to stitch tables together properly.

```
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 and by the foreign and primary key constraints.

```
SELECT b.title,
  s1.name AS ownerName,
  d1.faculty AS ownerFaculty,
  s2.name AS borrowerName,
```

```
  d2.faculty AS borrowerFaculty
FROM loan l, book b,
  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;
```

2. Algebraïc 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. Use INNER JOIN.

Solution: Technically, all simple queries are also algebraic since they use one table or the cross or Cartesian product of several tables. However, we shall refer to the explicit use of `CROSS JOIN`, `INNER JOIN`, `LEFT OUTER JOIN`, `RIGHT OUTER JOIN`, `FULL OUTER JOIN`,`JOIN`,`NATURAL JOIN`, `UNION`, `INTERSECT`, and `EXCEPT` as *algebraïc* queries.

```
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 copy c ON c.book = l.book
    AND c.copy = l.copy
    AND c.owner = l.owner
  INNER JOIN student s1 ON l.owner = s1.email
  INNER JOIN student s2 ON l.borrower = 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;
```

You can omit the table `copy` and the `copy` column since the existence of the corresponding rows and values is guaranteed by design and by the foreign and primary key constraints.

```
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.borrower = 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;
```

   (b) Print the emails of the different students who borrowed or lent a copy of a book on the day that they joined the university. Use an algebraïc query.

Solution:

```
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;
```

`DISTINCT` is not needed because `UNION` eliminates duplicates (so do `INTERSECT`, `EXCEPT` and `MINUS`). The corresponding simple query is generally preferable.

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

The simple query requires an explicit `DISTINCT`.

(c) Print the emails of the different students who borrowed and lent a copy of a book on the day that they joined the university. Use an algebraïc query.

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

```
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;
```

Note that the corresponding simple query is more complicated. It needs two `loan` tables.

```
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;
```

(d) Print the emails of the students who borrowed but did not lend a copy of a book on the day that they joined the university. Use an algebraïc query.

Solution:

```
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 corresponding simple query. We would need to use nested or aggregate queries to write alternative answers to this type of question.

(e) Print the ISBN13 of the books that have never been borrowed. Use an algebraïc query.

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

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

or, using an `OUTER JOIN`, which introduces `NULL` values,

```
SELECT  b.ISBN13
FROM book b LEFT OUTER JOIN loan l ON b.isbn13 = l.book
WHERE l.book ISNULL;
```

There is no corresponding simple query. We would need to use nested or aggregate queries for this type of questions.

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.