Database Design and Programming

Tutorial 2: Simple Queries

Biswadeep Sen
School of Computing
National University of Singapore
biswadeep@u.nus.edu

Simple Queries

- **Definition:** Queries that use basic operations like `SELECT`, `FROM`, and `WHERE` to retrieve data from one or more tables.
- **Purpose:** Used to fetch specific rows or columns, filter data, or perform straightforward tasks like sorting or aggregating.

1.(a) Print the different departments

1.(a) Print the different departments

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

1.(a) Print the different departments

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

Q: Why not use DISTINCT?

1.(a) Print the different departments

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

Q: Why not use DISTINCT?

We created the `department` table to have information about the *unique* departments, and the faculties. And `department` was the primary key in this table. So, we do not need `DISTINCT` in this query.

1.(a) Print the different departments

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

Q: Why not use DISTINCT?

We created the `department` table to have information about the *unique* departments, and the faculties. And `department` was the primary key in this table. So, we do not need `DISTINCT` in this query.

To verify:

```sql
SELECT COUNT(DISTINCT department) FROM department;
```

and

```sql
SELECT COUNT(*) FROM department;
```

Return the same value.

1.(b) Departments in which students are enrolled?

```
SELECT DISTINCT s.department

FROM student s;
```

📌**NOTE:**

- Some departments (e.g., *Undecidable Computation*) may have no enrolled students, so check the **`student`** table instead of `department`.
- Use `DISTINCT` on `department` to remove duplicates when listing all departments with students.
- **Alternative question:** “Print the department of the different students that are enrolled”
    - Here the focus is on distinct students, so `DISTINCT` should not be applied to department.

1.(c) Returned Copies — Print ISBN-13 & Loan Duration (Order: ISBN-13 ↑, Duration ↓; Single-Table Query)

```
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;
```

📌NOTE:

- 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.

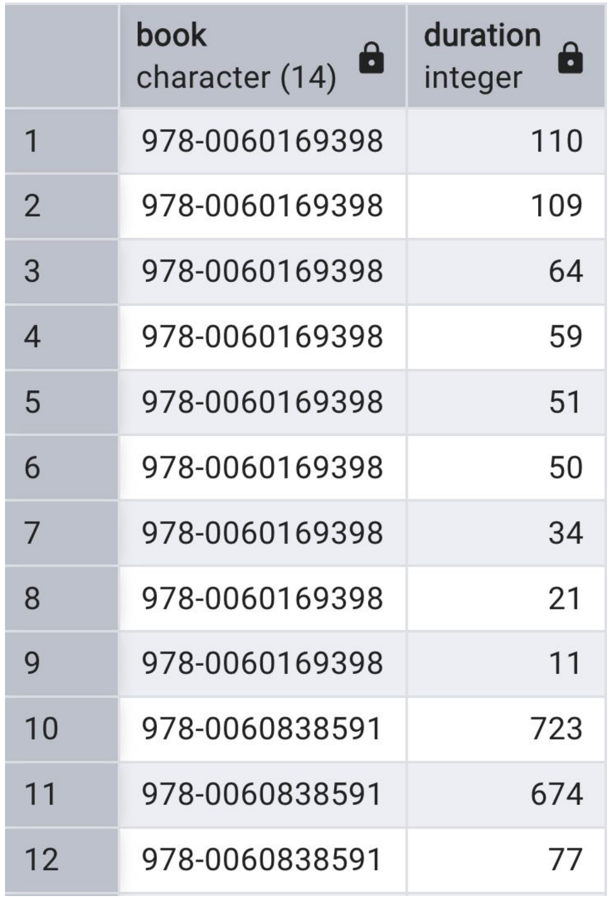
🤔 Can you modify the query to also print the loan duration of copies that have NOT been returned, where the duration is calculated until the current date?

1.(c) Returned Copies — Print ISBN-13 & Loan Duration (Order: ISBN-13 ↑, Duration ↓; Single-Table Query)

```sql
SELECT l.book,

(COALESCE(l.returned, CURRENT_DATE) - l.borrowed + 1) AS duration

FROM loan l

ORDER BY l.book ASC, duration DESC;
```

📌NOTE: COALESCE returns the first non-null value

1.(c) Returned Copies — Print ISBN-13 & Loan Duration (Order: ISBN-13 ↑, Duration ↓; Single-Table Query)



	book character (14)	duration integer
1	978-0060169398	110
2	978-0060169398	109
3	978-0060169398	64
4	978-0060169398	59
5	978-0060169398	51
6	978-0060169398	50
7	978-0060169398	34
8	978-0060169398	21
9	978-0060169398	11
10	978-0060838591	723
11	978-0060838591	674
12	978-0060838591	77

OUTPUT

1.(c) Returned Copies — Print ISBN-13 & Loan Duration (Order: ISBN-13 ↑, Duration ↓; Single-Table Query)

```sql
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;
```

Alternative: Use CASE WHEN

2(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*.

2(a). *continued…*

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;
```

2(a). *continued…*

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;
```

Retrieves the book title, owner's name and faculty, and borrower's name and faculty.

Specifies the tables involved in the query to join and fetch data from.

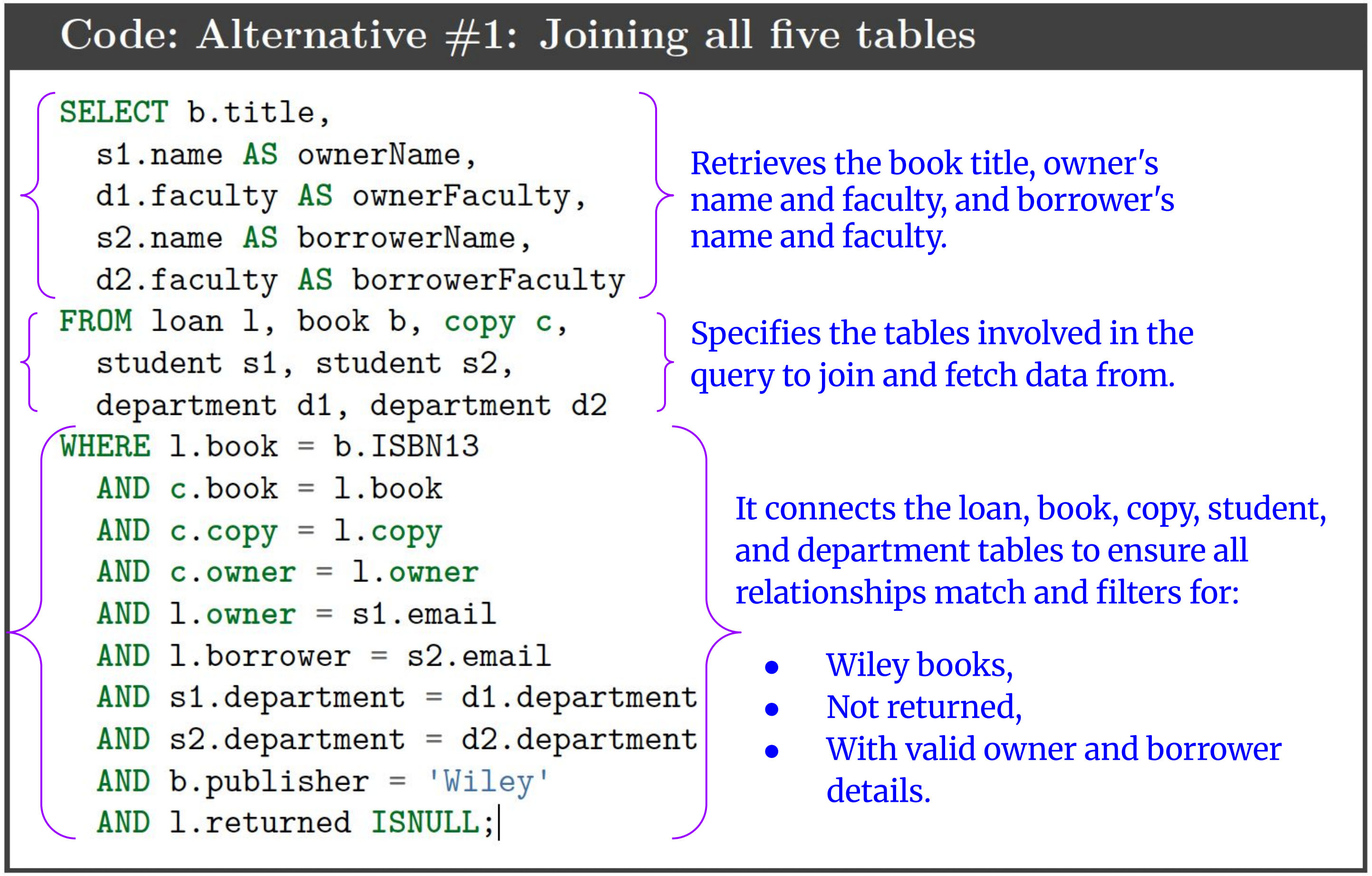
→ Connect each loan to the book that was borrowed

Connect the copy table to the loan — same book, same copy

Join the loan's borrower (owner) email to the borrower's (owner) student record

Get their departments

2(a). *continued…*



Code: Alternative #1: Joining all five tables

```sql
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;
```

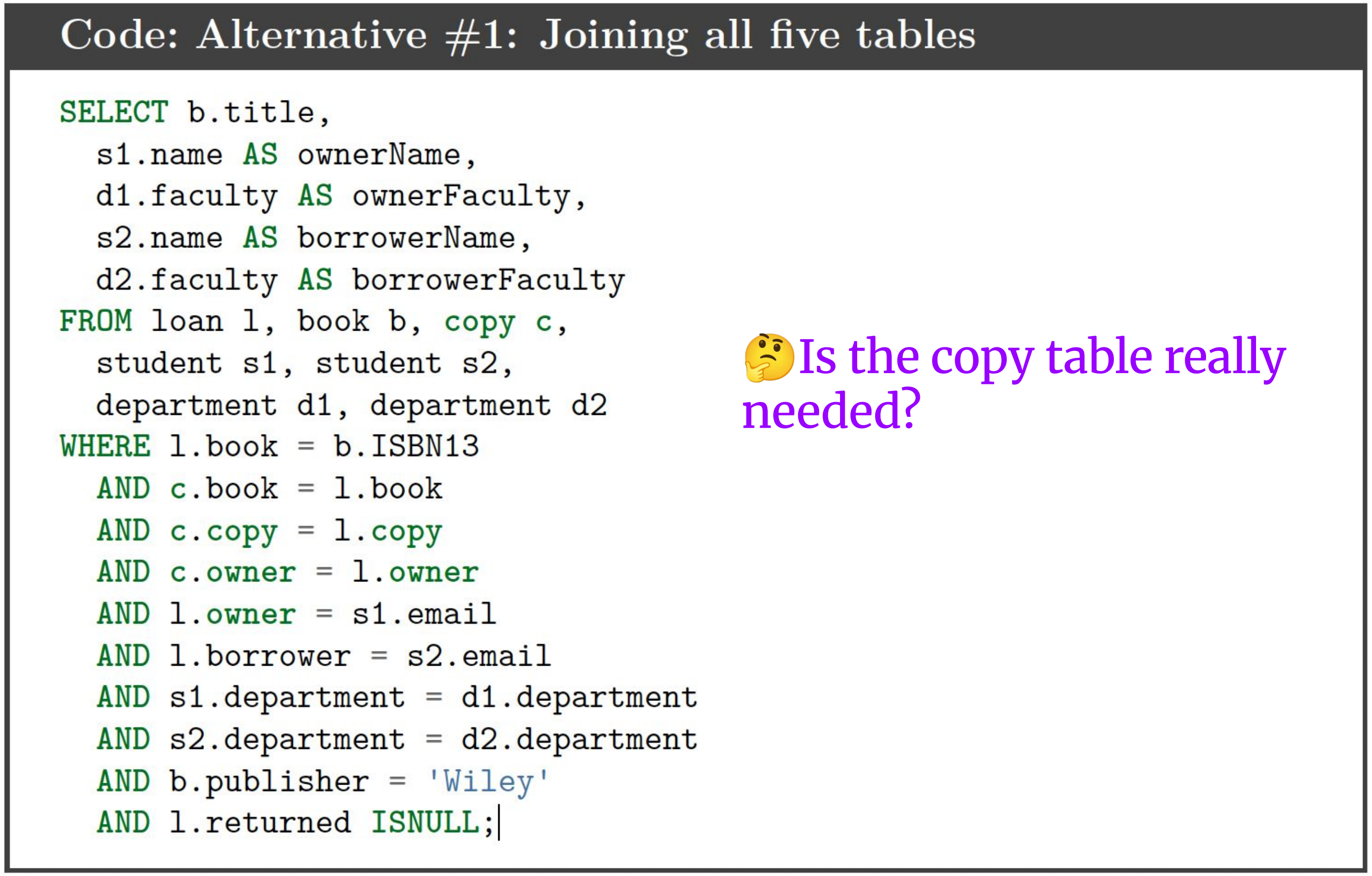
Retrieves the book title, owner's name and faculty, and borrower's name and faculty.

Specifies the tables involved in the query to join and fetch data from.

It connects the loan, book, copy, student, and department tables to ensure all relationships match and filters for:

- Wiley books,
- Not returned,
- With valid owner and borrower details.

2(a). *continued…*

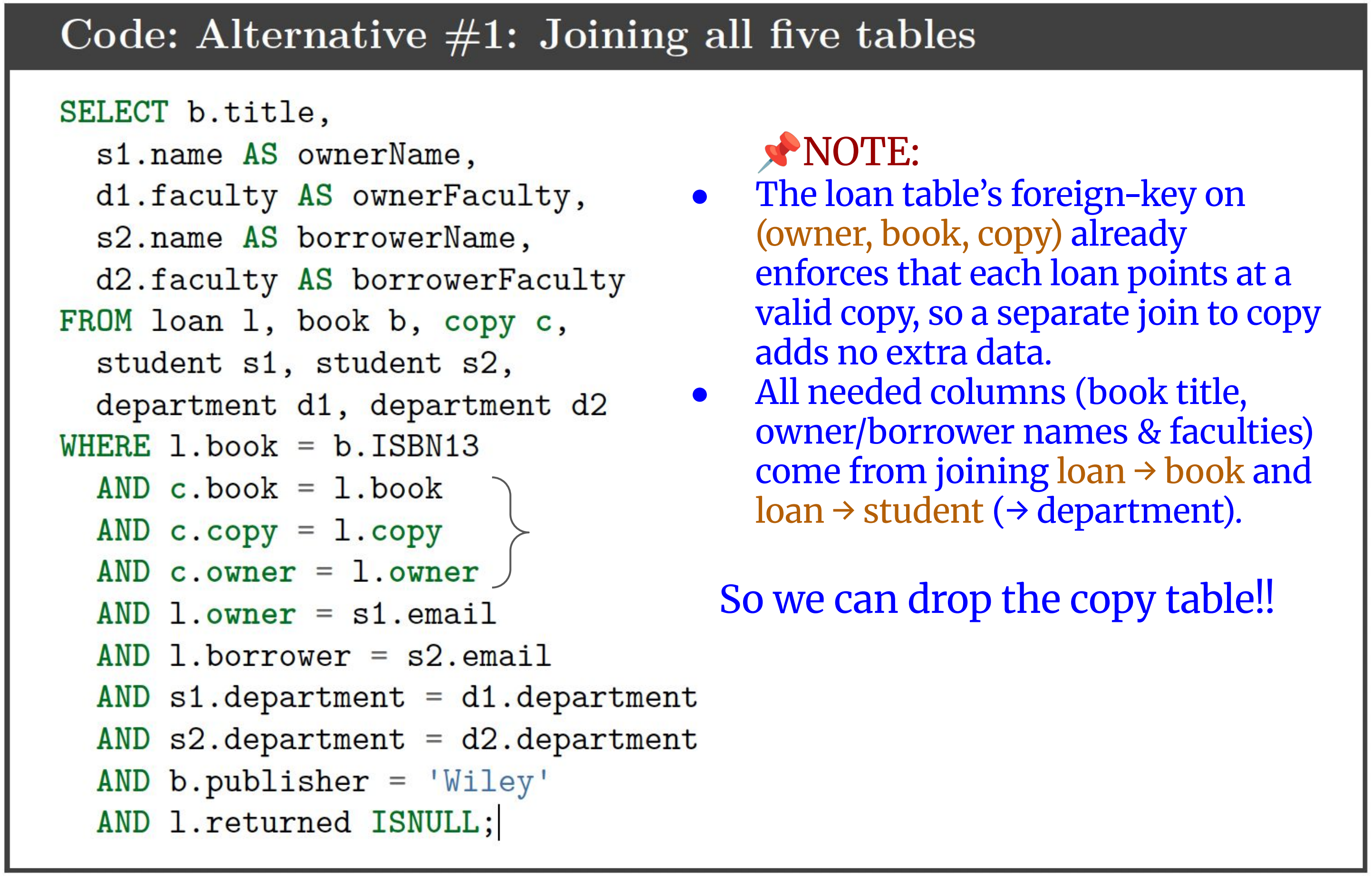


Code: Alternative #1: Joining all five tables

```sql
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;
```

🤔Is the copy table really needed?

2(a). *continued…*



Code: Alternative #1: Joining all five tables

```sql
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;
```

📌NOTE:

- The loan table's foreign-key on (owner, book, copy) already enforces that each loan points at a valid copy, so a separate join to copy adds no extra data.
- All needed columns (book title, owner/borrower names & faculties) come from joining loan → book and loan → student (→ department).

So we can drop the copy table!!

2(a). *continued…*

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,
  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;
```

📌NOTE:

- The loan table's foreign-key on (owner, book, copy) already enforces that each loan points at a valid copy, so a separate join to copy adds no extra data.
- All needed columns (book title, owner/borrower names & faculties) come from joining loan → book and loan → student (→ department).

So we can drop the copy table!!

2(a). *continued…*

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

```sql
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;
```

👉 Think of **INNER JOIN** as:

- **Match-making between two tables.**
- You only keep the rows where both tables “agree” on a value (the ON condition).
- If one side doesn’t have a match, that row disappears from the result.

2(b). Print the different emails of the students who borrowed or lent a copy of a book before they joined the University.

2(b). Print the different emails of the students who borrowed or lent a copy of a book before they joined the University.

```sql
SELECT DISTINCT s.email

FROM loan l, student s

WHERE (s.email = l.borrower OR s.email = l.owner)

AND l.borrowed < s.year;
```

2(b). Print the different emails of the students who borrowed or lent a copy of a book before they joined the University.

```
SELECT DISTINCT s.email

FROM loan l, student s

WHERE (s.email = l.borrower OR s.email = l.owner)

AND l.borrowed < s.year;
```

📌NOTE: An alternative solution can be the following

```
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)
```

2.(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.

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

📌NOTE: DISTINCT should be explicitly mentioned!

2.(c) Alternative 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;
```

2.(c) Alternative 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;
```

$\left.\right\}$ Borrowed!

$\left.\right\}$ Lent!

2.(c) Alternative solution

```sql
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;
```

Borrowed!

Lent!

🤔 Do we need DISTINCT here?
🚫No – because UNION automatically eliminates duplicates.

2.(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.

```
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;
```

2(d). Alternative without INTERSECT

```sql
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?
```

2(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.

2.(e) Can we somehow modify this code?

```sql
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;
```

2.(e) Can we somehow modify this code?

```
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;
```

2.(e) Use EXCEPT

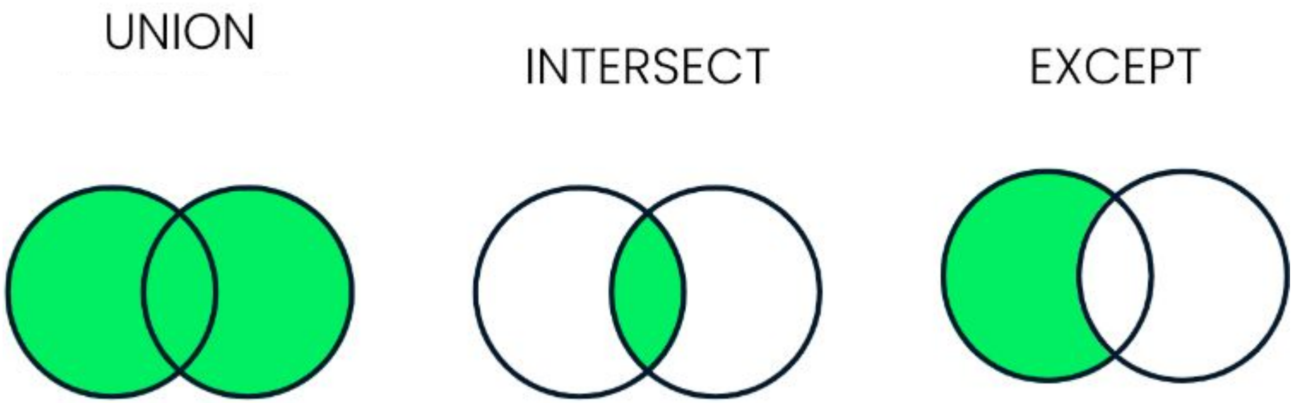
```sql
SELECT s.email
FROM loan l, student s
WHERE s.email = l.borrower AND l.borrowed = s.year
EXCEPT
SELECT s.email
FROM loan l, student s
WHERE s.email = l.owner AND l.borrowed = s.year;
```

2.(e) Use 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;
```

📌NOTE:

- 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.


UNION
INTERSECT
EXCEPT

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

```sql
SELECT b.ISBN13

FROM book b

EXCEPT

SELECT l.book

FROM loan l;
```

2. (f) Alternative using OUTER JOIN

```
SELECT b.ISBN13

FROM book b LEFT OUTER JOIN loan l ON b.ISBN13 = l.book

WHERE l.book ISNULL;
```

⬅️ **LEFT OUTER JOIN:**
Return **all rows from the left table**; if no match on the right, the right-side columns are **NULL**.

2. (f) Alternative using OUTER JOIN

```sql
SELECT b.ISBN13

FROM book b LEFT OUTER JOIN loan l ON b.ISBN13 = l.book

WHERE l.book ISNULL;
```

b.ISBN13	l.book	…
B1	B1	..
B2	B2	…
B3	NULL	NULL
B4	NULL	NULL

Matched!

No match in Loan

Thank you for joining!

Got questions? Post them on the forum or email me:

biswadeep@u.nus.edu

(I reply **within 2 working days** — *faster if coffee is strong* ☕)

Because your learning matters to me! 😊

NUS
National University
of Singapore