BT5110: Tutorial 6 — Simple Queries

Pratik Karmakar

School of Computing,
National University of Singapore

AY25/26 S1

Scenario

Students at the **National University of Ngendipura (NUN)** buy, lend, and borrow books.
NUNStA commissions *Apasaja Private Limited* to implement an online book exchange that records:

- Student info: name, faculty, department, **email** (identifier), join date (`year`).
- Book info: title, authors, publisher, year, edition, **ISBN10**, **ISBN13** (industry IDs; *unique*).
- Loans: `borrowed` date, `returned` date (may be `NULL`).

Auditing keeps data for (i) copies with loans and (ii) graduated students with loaned books.
This tutorial uses the schema/data created in "Creating and Populating Tables".

Important Constraints for This Tutorial

Use *simple queries only*

No **nested** or **aggregate** queries in answers.
Focus on **single-table** and **multi-table** joins and set operators.

We'll present **equivalent** formulations (e.g., CROSS JOIN vs INNER JOIN, `UNION/INTERSECT/EXCEPT`) and discuss readability best practices.

Questions — Single-table

1. Single-Table Queries

(a) Print the different departments.

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

(c) For each copy that has been borrowed and returned, print the `ISBN13` and the loan **duration**. Order by `ISBN13` (ASC) then duration (DESC). Use a **single** table.

Questions — Multi-table

2. Multi-Table Queries

(a) For each unreturned loan of a book published by `'Wiley'`, print: book title, owner name+faculty, borrower name+faculty.

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

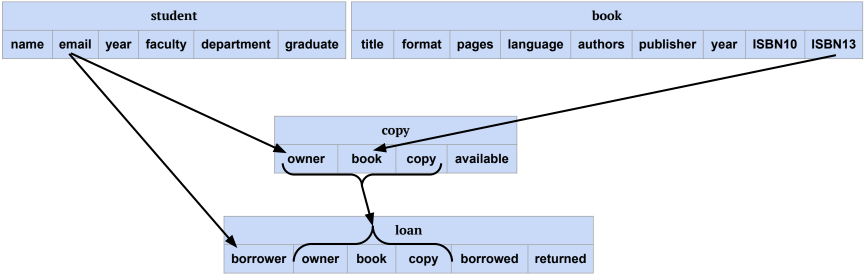
(c) Print emails of students who *borrowed or lent* a copy **on the day** they joined.

(d) Print emails of students who *borrowed and lent* a copy **on the day** they joined.

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

(f) Print `ISBN13` of books that have **never** been borrowed.

Creating the Database



student					
name	email	year	faculty	department	graduate

book								
title	format	pages	language	authors	publisher	year	ISBN10	ISBN13

copy			
owner	book	copy	available

loan					
borrower	owner	book	copy	borrowed	returned

Figure 1: student and book are independent tables (relations). copy relies on student for its owner attribute and on book for its book attribute. loan relies on copy for (owner, book, copy) attributes and on student for its borrower attribute.

Creating the Database

From Figure 1 it is clear that we need to populate either the `student` table or the `book` table first. Then we should populate `copy` and `loan` should be the last.

[1] The order of `NUNStAStudent.sql` and `NUNStABook.sql` are interchangeable.

[2] You need to be in the same directory of these files for these to run. Otherwise provide the entire file paths after `\i`.

Creating the Database

From Figure 1 it is clear that we need to populate either the student table or the book table first. Then we should populate copy and loan should be the last.

How would you find the order when the number of tables (relations) is much higher and have complex set of dependencies?

[1] The order of NUNStAStudent.sql and NUNStABook.sql are interchangeable.

[2] You need to be in the same directory of these files for these to run. Otherwise provide the entire file paths after \i.

Creating the Database

From Figure 1 it is clear that we need to populate either the `student` table or the `book` table first. Then we should populate `copy` and `loan` should be the last.

How would you find the order when the number of tables (relations) is much higher and have complex set of dependencies? Use Topological Sorting

[1] The order of `NUNStAStudent.sql` and `NUNStABook.sql` are interchangeable.

[2] You need to be in the same directory of these files for these to run. Otherwise provide the entire file paths after `\i`.

Creating the Database

From Figure 1 it is clear that we need to populate either the student table or the book table first. Then we should populate copy and loan should be the last.

How would you find the order when the number of tables (relations) is much higher and have complex set of dependencies? Use Topological Sorting

In psql run[1]:

```
        \i NUNStAStudent.sql;
        \i NUNStABook.sql;
        \i NUNStACopy.sql;
        \i NUNStALoan.sql;

```

Your database is now ready[2].

[1] The order of NUNStAStudent.sql and NUNStABook.sql are interchangeable.

[2] You need to be in the same directory of these files for these to run. Otherwise provide the entire file paths after \i.

1(a). Different departments (single table)

Good practice: alias tables; refer to columns via the alias.

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

Why no DISTINCT? `department` is the **PRIMARY KEY** of table `department`, so duplicates cannot occur.

1(b). Departments with enrolled students

Query the `student` table; many students share the same department, so we need `DISTINCT`.

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

Wording matters. If we asked "*department of the different students*", then `DISTINCT` should **not** be on `department` (students are identified by `email`, not department).

1(c). Loan duration from a single table (returned only)

```sql
FROM loan AS l
WHERE l.returned IS NOT NULL
ORDER BY l.book ASC, duration DESC;
```

1(c). Loan duration including unreturned

Using COALESCE

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

Using CASE

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

2(a). Unreturned Wiley loans (with COPY join)

```sql
SELECT b.title,
       s1.name AS ownerName,  d1.faculty AS ownerFaculty,
       s2.name AS borrowerName, d2.faculty AS borrowerFaculty
FROM loan AS l, book AS b, copy AS c,
     student AS s1, student AS s2,
     department AS d1, department AS 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 IS NULL;
```

2(a). Unreturned Wiley loans (omit COPY via PK–FK)

```
SELECT b.title,
       s1.name AS ownerName,  d1.faculty AS ownerFaculty,
       s2.name AS borrowerName, d2.faculty AS borrowerFaculty
FROM loan AS l, book AS b,
     student AS s1, student AS s2,
     department AS d1, department AS 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 IS NULL;
```

2(a). Using INNER JOIN (clear ON vs WHERE)

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

Convention: ON for PK–FK join predicates; WHERE for additional filters.

2(b). Borrowed or lent *before* joining

Either role (owner or borrower) and **date check**:

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

Distributed condition (equivalent):

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

2(c). Borrowed or lent *on* joining day

Direct form (needs DISTINCT):

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

Using `UNION` (deduplicates by default):

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

2(d). Borrowed *and* lent *on* joining day

Using `INTERSECT`:

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

Without `INTERSECT` (two aliases of `loan`):

```sql
SELECT DISTINCT s.email
FROM loan AS l1, loan AS l2, student AS s
WHERE s.email = l1.borrower AND l1.borrowed = s.year
  AND s.email = l2.owner    AND l2.borrowed = s.year;
```

2(e). Borrowed *but did not lend on* joining day

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

2(f). Books *never* borrowed

Using `EXCEPT`:

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

Using `LEFT OUTER JOIN` + `IS NULL`:

```sql
SELECT b.ISBN13
FROM book AS b
LEFT OUTER JOIN loan AS l
  ON b.ISBN13 = l.book
WHERE l.book IS NULL;
```

Guidelines & Marking Tips

- **No hardcoding.** Queries must work on any dataset consistent with the schema.
- **Constants only if stated.** If the question names a constant (e.g., 'Wiley'), you may use it; otherwise avoid.
- **Readable style.** Use table aliases, qualify columns, and uppercase SQL keywords.
- **Set operators** (`UNION/INTERSECT/EXCEPT`) inherently deduplicate; `DISTINCT` is redundant with them.
- **Partial marks:** (i) query executes, (ii) correct columns (names, types, order), (iii) minimal row differences.

Questions?
Drop a mail at: pratik.karmakar@u.nus.edu