# 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.

# 1(a). Different departments (single table)

## SQL

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

## Relational Algebra

$$\pi_{\text{department}}(\ \texttt{department}\ )$$

*Note:* RA projection is set-based (duplicates removed), matching the PK property here.

# 1(b). Departments with enrolled students

## SQL

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

## Relational Algebra

$$\pi_{\text{department}}(\ \texttt{student}\ )$$

*Set semantics already imply DISTINCT.*

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

## SQL

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

## Relational Algebra (extended projection)

$$\pi_{\text{book},\ \text{duration}:=\text{returned}-\text{borrowed}+1}\left( \sigma_{\text{returned}\neq\text{NULL}}( \texttt{loan} ) \right)$$

*Ordering is not part of classical RA.*

# 1(c). Loan duration including unreturned

## SQL (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;
```

## SQL (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;
```

## *Relational Algebra (with NVL/IF as built-ins)*

$$\pi_{book,\ duration:=\mathrm{NVL}(returned,\mathrm{CURRENT\_DATE})-borrowed+1}(\ loan\ )$$

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

## SQL

```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)

## SQL

```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,
     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)

## SQL

```sql
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;
```

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

## SQL

```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;
```

## Relational Algebra (union of roles)

$$\pi_{s.\text{email}}\ \sigma_{l.\text{borrowed}<s.\text{year}}\ (\texttt{loan}\ l\ \bowtie_{l.\text{borrower}=s.\text{email}}\ \texttt{student}\ s)$$
$$\cup\ \pi_{s.\text{email}}\ \sigma_{l.\text{borrowed}<s.\text{year}}\ (\texttt{loan}\ l\ \bowtie_{l.\text{owner}=s.\text{email}}\ \texttt{student}\ s)$$

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

## SQL

```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;
```

## Relational Algebra (union)

$$\pi_{s.\text{email}}\ \sigma_{l.\text{borrowed}=s.\text{year}}\ (\texttt{loan}\ l\ \bowtie_{l.\text{borrower}=s.\text{email}}\ \texttt{student}\ s)$$
$$\cup\ \pi_{s.\text{email}}\ \sigma_{l.\text{borrowed}=s.\text{year}}\ (\texttt{loan}\ l\ \bowtie_{l.\text{owner}=s.\text{email}}\ \texttt{student}\ s)$$

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

## SQL (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;
```

## Relational Algebra (intersection)

$$\pi_{s.\text{email}}\ \sigma_{l.\text{borrowed}=s.\text{year}}\ (\text{loan}\ l\ \bowtie_{l.\text{borrower}=s.\text{email}}\ \text{student}\ s)$$
$$\cap\ \pi_{s.\text{email}}\ \sigma_{l.\text{borrowed}=s.\text{year}}\ (\text{loan}\ l\ \bowtie_{l.\text{owner}=s.\text{email}}\ \text{student}\ s)$$

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

## SQL (EXCEPT)

```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;
```

## Relational Algebra (set difference)

$$\pi_{s.\text{email}}\ \sigma_{l.\text{borrowed}=s.\text{year}}\ (\text{loan}\ l\ \bowtie_{l.\text{borrower}=s.\text{email}}\ \text{student}\ s)$$
$$-\ \pi_{s.\text{email}}\ \sigma_{l.\text{borrowed}=s.\text{year}}\ (\text{loan}\ l\ \bowtie_{l.\text{owner}=s.\text{email}}\ \text{student}\ s)$$

# 2(f). Books *never* borrowed

**SQL**

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

**Relational Algebra (set difference)**

$$\pi_{\texttt{ISBN13}}(\ \texttt{book}\ ) \ - \ \pi_{\texttt{book}}(\ \texttt{loan}\ )$$

**Alternative (outer join + IS NULL)** is not part of classical RA.

# 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