Database Design and Programming

Tutorial 4: Aggregate and Nested Queries

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

(1.a) How many loans involve an owner and a borrower from the same department?

```sql
SELECT COUNT(*)
FROM loan l, student s1, student s2
WHERE l.owner = s1.email
AND l.borrower = s2.email
AND s1.department = s2.department;
```

(1.b) Counting loans where the lender and borrower are from the same faculty

```sql
SELECT d1.faculty, COUNT(*)
FROM loan l, student s1, student s2, department d1, department d2
WHERE l.owner = s1.email
AND l.borrower = s2.email
AND s1.department = d1.department
AND s2.department = d2.department
AND d1.faculty = d2.faculty
GROUP BY d1.faculty;
```

(1.b) Counting loans where the lender and borrower are from the same faculty

```
SELECT d1.faculty, COUNT(*)   🎯 count loans per faculty
FROM loan l, student s1, student s2, department d1, department d2
WHERE l.owner = s1.email   📩 Connect the owner's email to the correct student record.
AND l.borrower = s2.email   📩 Connect the borrower's email to another student entry.
AND s1.department = d1.department   🏛️ Associate the lender with their department details.
AND s2.department = d2.department   🏛️ Do the same for the borrower
AND d1.faculty = d2.faculty   🔗 Key constraint: both students must be from the same faculty!
GROUP BY d1.faculty;   📊 Group the final results by faculty so we can count per group.
```

📌 NOTE: Here `GROUP BY d1.faculty` ensures you don't just get "total loans across the whole university" — instead, you get **loan counts faculty by faculty**.

(1.c) What is the average and standard deviation of loan duration in days?

```
SELECT CEIL(AVG((CASE
WHEN l.returned IS NULL
THEN CURRENT_DATE
ELSE l.returned
END) - l.borrowed + 1)),
CEIL(STDDEV_POP((CASE
WHEN l.returned IS NULL
THEN CURRENT_DATE
ELSE l.returned
END) - l.borrowed + 1))
FROM loan l;
```

📌If the book is not yet returned (NULL), assume it was returned today (CURRENT_DATE). Otherwise, use the actual return date.

📅The number of days between `borrowed` and `returned`

📌If the book is not yet returned (NULL), assume it was returned today (CURRENT_DATE). Otherwise, use the actual return date.

📅The number of days between `borrowed` and `returned`

📌 *NOTE:* CEIL(x) always rounds up, returning the smallest integer ≥ x

(1.c) What is the average and standard deviation of loan duration in days?

```
SELECT CEIL(AVG(temp.duration)), CEIL(STDDEV_POP(temp.duration))
FROM (
    SELECT((CASE
        WHEN l.returned ISNULL THEN CURRENT_DATE
        ELSE l.returned
      END ) - l.borrowed + 1) AS duration
    FROM loan l
) AS temp;
```

(2.a) Print the titles of the different books that have never been borrowed. Use a nested query.

```sql
SELECT b.title
FROM book b
WHERE b.ISBN13 NOT IN (
    SELECT l.book
    FROM loan l
);
```

📌 *NOTE:*
NOT IN and <> ALL are logically equivalent, but some database systems prefer <> ALL for performance reasons.

```sql
SELECT b.title
FROM book b
WHERE b.ISBN13 <> ALL (
    SELECT l.book
    FROM loan l
);
```

📌 *NOTE:*
<> ALL checks that the book's ISBN13 is not equal to every single ISBN13 returned from the subquery.

(2.b) Find students who own a copy of a book that they have never lent to anybody.

```
SELECT s.name
FROM student s
WHERE s.email IN (
    SELECT c.owner
    FROM copy c
    WHERE NOT EXISTS (
        SELECT *
        FROM loan l
        WHERE l.owner = c.owner
        AND l.book = c.book
        AND l.copy = c.copy));
);
```

➡️ *Return the **student names** whose **email ∈ that owner list***

📚 *Builds a list of **owners** of never-lent copies*

Only keep copies without loans!

🔍 *Return any loan rows for this **exact copy** (`owner`, `book`, `copy`).*

(2.b) Find students who own a copy of a book that they have never lent to anybody.

```
SELECT s.name
FROM student s
WHERE s.email ANY (
    SELECT c.owner
    FROM copy c
    WHERE NOT EXISTS (
        SELECT *
        FROM loan l
        WHERE l.owner = c.owner
        AND l.book = c.book
        AND l.copy = c.copy));
```

📌NOTE:

ANY is functionally the same as IN

Use **DISTINCT** if you want to avoid listing the same student multiple times when they have multiple never-lent copies

(2.c) For each department, print the names of the students who lent the most.

```
SELECT s.department, s.name, COUNT(*)
FROM student s, loan l
WHERE l.owner = s.email
GROUP BY s.department, s.email, s.name
HAVING COUNT(*) >= ALL
  (SELECT COUNT(*)
   FROM student s1, loan l1
   WHERE l1.owner = s1.email
     AND s.department = s1.department
   GROUP BY s1.email);
```

- HAVING `COUNT(*) >= ALL(...)` -keep students whose count is ≥ **every count** in the dept's list → i.e., the **department maximum** (ties included).
- **Main query**: groups by **department** and **student**; **counts** each student's loans.

- **Inner query**: for the **same department**, returns the **list of loan counts** for all students.

(2.c) For each department, print the names of the students who lent the most.

```sql
SELECT s.department, s.name, COUNT(*)
FROM student s, loan l
WHERE l.owner = s.email
GROUP BY s.department, s.email, s.name
HAVING COUNT(*) >= ALL
  (SELECT COUNT(*)
   FROM student s1, loan l1
   WHERE l1.owner = s1.email
     AND s.department = s1.department
   GROUP BY s1.email);
```

Data (loan counts per student)

- **CS**
  - Alice — **alice@u.edu → 3**
  - Bob — **bob@u.edu → 1**
- **Math**
  - Dave — **dave@u.edu → 2**
  - Eve — **eve@u.edu → 2**

(2.c) For each department, print the names of the students who lent the most.

```
SELECT s.department, s.name, COUNT(*)
FROM student s, loan l
WHERE l.owner = s.email
GROUP BY s.department, s.email, s.name
HAVING COUNT(*) >= ALL
  (SELECT COUNT(*)
   FROM student s1, loan l1
   WHERE l1.owner = s1.email
     AND s.department = s1.department
   GROUP BY s1.email);
```

Inner lists (recomputed per department)

- **CS:** `[3, 1]`
- **Math:** `[2, 2]`

(2.c) For each department, print the names of the students who lent the most.

```sql
SELECT s.department, s.name, COUNT(*)
FROM student s, loan l
WHERE l.owner = s.email
GROUP BY s.department, s.email, s.name
HAVING COUNT(*) >= ALL
  (SELECT COUNT(*)
   FROM student s1, loan l1
   WHERE l1.owner = s1.email
     AND s.department = s1.department
   GROUP BY s1.email);
```

Outer check `COUNT(*) >= ALL(list)` (per student)

- **Alice (alice@u.edu)**: `3 >= ALL([3,1])` → **keep** ✅
- **Bob (bob@u.edu)**: `1 >= ALL([3,1])` → **drop** ❌
- **Dave (dave@u.edu)**: `2 >= ALL([2,2])` → **keep** ✅
- **Eve (eve@u.edu)**: `2 >= ALL([2,2])` → **keep** ✅

(2.c) For each department, print the names of the students who lent the most.

```
SELECT s.department, s.name, COUNT(*)
FROM student s, loan l
WHERE l.owner = s.email
GROUP BY s.department, s.email, s.name
HAVING COUNT(*) >= ALL
  (SELECT COUNT(*)
   FROM student s1, loan l1
   WHERE l1.owner = s1.email
     AND s.department = s1.department
   GROUP BY s1.email);
```

- **Dept - Name - Count**
- CS - Alice - 3
- Math - Dave - 2
- Math - Eve - 2

(2.c) For each department, print the names of the students who lent the most.

```sql
SELECT s.department, s.name, COUNT(*)
FROM student s, loan l
WHERE l.owner = s.email
GROUP BY s.department, s.email, s.name
HAVING COUNT(*) >= ALL
  (SELECT COUNT(*)
   FROM student s1, loan l1
   WHERE l1.owner = s1.email
     AND s.department = s1.department
   GROUP BY s1.email);
```

📌NOTE:
Multiple top lenders appear together if there's a tie, and departments with no loans will not appear unless handled with an outer join

(2.d) Return students who have borrowed every book written by Adam Smith

```
SELECT s.email, s.name
FROM student s
WHERE NOT EXISTS (
    SELECT *
    FROM book b
    WHERE authors = 'Adam Smith'
      AND NOT EXISTS (
          SELECT *
          FROM loan l
          WHERE l.book = b.isbn13
            AND l.borrower = s.email  ));
```

Collects **all Adam Smith books that student s missed**.

- If the student missed some books → this returns a non-empty set ({Book 1, Book 2,...}).
- If the student missed none → this returns empty.

is TRUE if student s never borrowed b.

is FALSE if student s has borrowed b.

(2.d) Return students who have borrowed every book written by Adam Smith

```
SELECT s.email, s.name
FROM student s
WHERE NOT EXISTS (
    SELECT *
    FROM book b
    WHERE authors = 'Adam Smith'
      AND NOT EXISTS (
          SELECT *
          FROM loan l
          WHERE l.book = b.isbn13
            AND l.borrower = s.email  ));
```

Returns set of missed books

- Now we wrap the above in `NOT EXISTS`.
- If the set of "missed books" is non-empty → `EXISTS` is true → `NOT EXISTS` is false → student is excluded.
- If the set of "missed books" is empty → `EXISTS` is false → `NOT EXISTS` is true → student is included.

(2.d) Return students who have borrowed every book written by Adam Smith

```sql
SELECT s.email, s.name
FROM student s
WHERE NOT EXISTS (
    SELECT *
    FROM book b
    WHERE authors = 'Adam Smith'
      AND NOT EXISTS (
          SELECT *
          FROM loan l
          WHERE l.book = b.isbn13
            AND l.borrower = s.email  ));
```

📌 NOTE:

🟠 **Inner NOT EXISTS**
Checks that this specific Adam-Smith title does not appear in the student's loan history.

🔴 **Outer NOT EXISTS**
Ensures no Adam-Smith book exists for which the inner check is true—i.e., there are zero titles the student hasn't borrowed.

✅ Passes only if **every Adam-Smith book appears in the student's loans!**

(2.d) Return students who have borrowed every book written by Adam Smith

```sql
SELECT s.email, s.name
FROM student s
WHERE NOT EXISTS (
    SELECT *
    FROM book b
    WHERE authors = 'Adam Smith'
      AND NOT EXISTS (
          SELECT *
          FROM loan l
          WHERE l.book = b.isbn13
            AND l.borrower = s.email  ));
```

Adam-Smith books: A1, A2
Loans:
Alice → A1, A2
Bob → A1

`NOT EXISTS ( … loan … )` is **TRUE** if the student has *no loan* for that book.👉 This effectively **adds that book to the student's "missed set."**

Alice: missed set = {} → outer `NOT EXISTS` = TRUE → **keep** ✅
Bob: missed set = {A2} → outer `NOT EXISTS` = FALSE → **drop** ❌

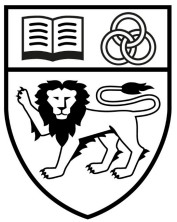
Thank you for joining!

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

(2.a) Print the titles of the different books that have never been borrowed. Use a nested query.

Why Not Use `DISTINCT`?

- The question asks for **different book titles**, **not unique books**.
- The `title` column may contain duplicate values (if the library has multiple copies of the same book).
- Since we are checking books that **do not appear in `loan`**, `DISTINCT` is unnecessary.