Database Design and Programming

Tutorial 1: Creating and Populating Tables

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

NUS
National University
of Singapore

NUN Book Exchange System — Overview

At the National University of Ngendipura (NUN), students buy, lend, and borrow books to support their studies. Build an online system that tracks:

- 👩‍🎓 **Students** – Their name, email (as ID), faculty, department, and year of joining
- 📘 **Books** – Title, authors, publisher, year, edition, ISBN-10 & ISBN-13
- 📦 **Copies** – Each physical book owned by a student
- 🔁 **Loans** – Who borrowed what, when it was borrowed, and when it was returned

NUN Book Exchange System — Overview

📌 *NOTE:* Even if a book isn't currently owned by a student, it can still be in the system (e.g., if it was previously owned or recommended for a course).

📌 *NOTE:* For **auditing**, the system keeps records of:

- Book copies and their owners (even if they graduate)
- Graduated students (as long as they were involved in any past loans)

1(a). Run the files

Go to:
Canvas > Files > Cases > Book Exchange

Download the following files:

- `NUNStASchema.sql`
- `NUNStAClean.sql`
- `NUNStAStudent.sql`
- `NUNStABook.sql`
- `NUNStACopy.sql`
- `NUNStALoan.sql`

1(b). What are the SQL files doing?

1(b). What are the SQL files doing?

You're given several SQL scripts

These are:

- `NUNStAClean.sql`:
- `NUNStASchema.sql`:

1(b). What are the SQL files doing?

You're given several SQL scripts

These are:

- `NUNStAClean.sql`: Drops all existing tables and their contents.
- `NUNStASchema.sql`: Builds the structure—tables, constraints etc.
- Data insertion files:
  - `NUNStAStudent.sql`
  - `NUNStABook.sql`
  - `NUNStACopy.sql`
  - `NUNStALoan.sql`

📌 *NOTE:*

- **`DROP TABLE IF EXISTS …`** → safe cleanup (no error if missing).
- **`CREATE TABLE IF NOT EXISTS …`** → safe repeat runs (no error if already exists).

1(c). Use the files to create and populate a database. Any bugs?

Create your database in pgAdmin 4 and open the Query Tool.

STEP 1: Run `NUNStASchema.sql`

```
ERROR:  relation "copy" does not
        exist

        SQL state: 42P01
```

Where is the BUG? 🐞

The loan table references copy table:

```
FOREIGN KEY (owner, book, copy)
REFERENCES copy(owner, book, copy)
```

But copy table is created *after* loan.

Result → ❌ *"relation copy does not exist"* error.

1(c). Use the files to create and populate a database. Any bugs?

Create your database in pgAdmin 4 and open the Query Tool.

STEP 1: Run `NUNStASchema.sql`

🔧 The Fix

Correct Table Creation Order:

Book -> Student -> Copy -> Loan

1(c). Use the files to create and populate a database. Any bugs?

Create your database in pgAdmin 4 and open the Query Tool.

STEP 1: Run `NUNStASchema.sql`

- Creates all tables, domains, keys, and constraints.
- Ensures `student` and `book` tables exist before `copy` and `loan`.

STEP 2: Populate `student` and `book` (in any order):

- `NUNStAStudent.sql` → loads student records
- `NUNStABook.sql` → loads book records

1(c). Use the files to create and populate a database. Any bugs?

- **STEP 3: Populate dependent tables** (strict order due to FOREIGN KEYs):

  a. `NUNStACopy.sql` → loads copies (requires `student` + `book`)

  b. `NUNStALoan.sql` → loads loans (requires `copy`)

1(c). Use the files to create and populate a database. Any bugs?

- **STEP 3: Populate dependent tables** (strict order due to FOREIGN KEYs):
  a. `NUNStACopy.sql` → loads copies (requires `student` + `book`)
  b. `NUNStALoan.sql` → loads loans (requires `copy`)
- **STEP 4: Cleanup script order** in `NUNStAClean.sql` matters for deletions:
  a. Drop `loan` first
  b. Then `copy`
  c. Followed by `student`
  d. Finally `book`

2(a). Insert the following new book. Describe the behavior.

Code: INSERT INTO

```
INSERT INTO book VALUES (
  'An Introduction to Database Systems',
  'paperback',
  640,
  'English',
  'C. J. Date',
  'Pearson',
  '2003-01-01',
  '0321197844',
  '978-0321197849'
);
```

	title character varying (256)	format character (9)	pages integer	language character varying (32)	authors character varying (256)
308	Selling Today (11th Edition)	paperback	544	English	Gerald L Manning, Barry L Ree
309	The Accidental Salesperson: How to Take Contro...	paperback	204	English	Chris Lytle
310	Storyselling for Financial Advisors : How Top Pro...	paperback	256	English	Scott West, Mitch Anthony
311	Mastering the Complex Sale: How to Compete an...	paperback	304	English	Jeff Thull
312	An Introduction to Database Systems	paperback	640	English	C. J. Date

2(a). Insert the following new book. Describe the behavior.

Code: INSERT INTO

```
INSERT INTO book VALUES (
  'An Introduction to Database Systems',
  'paperback',
  640,
  'English',
  'C. J. Date',
  'Pearson',
  '2003-01-01',
  '0321197844',
  '978-0321197849'
);
```

📌 NOTE:

- Use single quotes(') for string values, not double quotes (").
- Dates should follow the yyyy-mm-dd format — it's the ISO standard.
- To verify your data insertion, run the following: `SELECT * FROM book;`

2(b) Inserting a Book with a Different ISBN13

Code: INSERT INTO

```sql
INSERT INTO book VALUES (
  'An Introduction to Database Systems',
  'paperback',
  640,
  'English',
  'C. J. Date',
  'Pearson',
  '2003-01-01',
  '0321197844',
  '978-0201385908'
);
```

2(b) Inserting a Book with a Different ISBN13

Code: INSERT INTO

```
INSERT INTO book VALUES (
```

Code: Error Message

```
ERROR: Key (isbn10)=(0321197844) already exists.duplicate key value ...
ERROR: duplicate key value violates unique constraint "book_isbn10_key"
SQL state: 23505
Detail: Key (isbn10)=(0321197844) already exists.
```

```
);
```

2(b) Inserting a Book with a Different ISBN13

Code: INSERT INTO

```
INSERT INTO book VALUES (
  'An Introduction to Database Systems',
  'paperback',
  640,
  'English',
  'C. J. Date',
  'Pearson',
  '2003-01-01',
  '0321197844',
  '978-0201385908'
);
```

🤔 *Why does the Error happen?*

- The isbn10 value '0321197844' already exists in the book table.
- isbn10 is defined as UNIQUE, so duplicates aren't allowed.
- PostgreSQL blocks the insert!

2(b) Inserting a Book with a Different ISBN13

Code: INSERT INTO

```
INSERT INTO book VALUES (
  'An Introduction to Database Systems',
  'paperback',
  640,
  'English',
  'C. J. Date',
  'Pearson',
  '2003-01-01',
  '0321197844',
  '978-0201385908'
);
```

✅ *TAKEAWAYS:*

- Unique constraints prevent duplicate values in key fields like isbn10.
- Always check what fields have UNIQUE or PRIMARY KEY constraints before inserting.

2(c) What happens if we insert the *same book* with a **new** `isbn10` but an already existing `isbn13`?

Code: INSERT INTO

```sql
INSERT INTO book VALUES (
  'An Introduction to Database Systems',
  'paperback',
  640,
  'English',
  'C. J. Date',
  'Pearson',
  '2003-01-01',
  '0201385902',
  '978-0321197849'
);
```

2(c) What happens if we insert the *same book* with a **new** `isbn10` but an already existing `isbn13`?

Code: INSERT INTO

Code: Error Message

```
ERROR: Key (isbn13)=(978-0321197849) already exists.duplicate key ...
ERROR: duplicate key value violates unique constraint "book_pkey"
SQL state: 23505
Detail: Key (isbn13)=(978-0321197849) already exists.
```

```
    '0201385902',
    '978-0321197849'
  );
```

2(c) What happens if we insert the *same book* with a **new** `isbn10` but an already existing `isbn13`?

Code: INSERT INTO

```
INSERT INTO book VALUES (
  'An Introduction to Database Systems',
  'paperback',
  640,
  'English',
  'C. J. Date',
  'Pearson',
  '2003-01-01',
  '0201385902',
  '978-0321197849'
);
```

🤔 *WHAT HAPPENED?*

- PostgreSQL rejects it because isbn13 is the **primary key**, and it **must be unique**.
- `SQLSTATE 23505` indicates a duplicate key violation.

2(d) Insert the following new student. Describe the behavior.

Code: INSERT INTO

```
INSERT INTO student VALUES (
  'TIKKI TAVI',
  'tikki@gmail.com',
  '2024-08-15',
  'School of Computing',
  'CS',
  NULL
);
```

	name character varying (32)	email [PK] character varying (256)	year date	faculty character varying (62)	department character varying (32)	graduate date
20	LIU JUN	liujun1989@msn.com	2021-08-01	School of Computing	CS	[null]
21	IRIS BROWN	irisbrown1992@hotmail.com	2022-08-01	School of Computing	CS	[null]
22	QIN YUWEI	qinyuwei2011@hotmail.com	2023-08-01	School of Computing	CS	[null]
23	NG YONG MING	ngyongming2011@yahoo.com	2023-08-01	School of Computing	CS	[null]
24	ANG JIA YI	angjiayi1990@hotmail.com	2023-08-01	School of Computing	CS	[null]
25	TIKKI TAVI	tikki@gmail.com	2024-08-15	School of Computing	CS	[null]

📌 NOTE:

Notice that the value of the field year is NULL . This is because the student has not yet graduated.

2(e) Insert the following new student. Describe the behavior.

Code: INSERT INTO

```
INSERT INTO student (email, name, year, faculty, department) VALUES (
  'rikki@gmail.com',
  'RIKKI TAVI',
  '2024-08-15',
  'School of Computing',
  'CS'
);
```

📌 *NOTE:*

- We have to explicitly indicate the order of fields
- This method is often preferred over implicit!
- The code clearer, safer, and easier to maintain.

2(e) Insert the following new student. Describe the behavior.

Code: INSERT INTO

```
INSERT INTO student (email, name, year, faculty, department) VALUES (
```

Code: Error Message

```
ERROR: Failing row contains (RIKKI TAVI, null, 2024-08-15, School ...
ERROR: null value in column "email" of relation "student" violates ...
SQL state: 23502
Detail: Failing row contains (RIKKI TAVI, null, 2024-08-15, School ...
```

📌 *NOTE:*
This does not work because email field is a primary key and therefore cannot be **NULL**

2(f).Change the name of the dept from 'CS' to 'Computer Science' .

Code: UPDATE

```
UPDATE student
SET department = 'Computer Science'
WHERE department = 'CS';
```

NOTE: You can check whether the update was effective using the following query:

Code: SELECT

```
SELECT *
FROM student
WHERE department = 'Computer Science';
```

Prints all students from the dept!

2 (g) Delete all the students from the 'chemistry' dept.

Code: DELETE FROM

```
DELETE FROM student
WHERE department = 'chemistry';
```

🤔 *WHAT HAPPENED?*

No deletion as chemistry is spelled with a lowercase “c”.

But no error! 🤷

It fails to find any matching row 🔍 and hence 0 deletions!

2.(h) Delete all the students from the 'Chemistry' dept.

Code: DELETE FROM

Code: Error Message

```
ERROR: Key (email)=(xiexin2011@gmail.com) is still referenced from ...
ERROR: update or delete on table "student" violates foreign key ...
SQL state: 23503
Detail: Key (email)=(xiexin2011@gmail.com) is still referenced from ...
```

2.(h) Delete all the students from the 'Chemistry' dept.

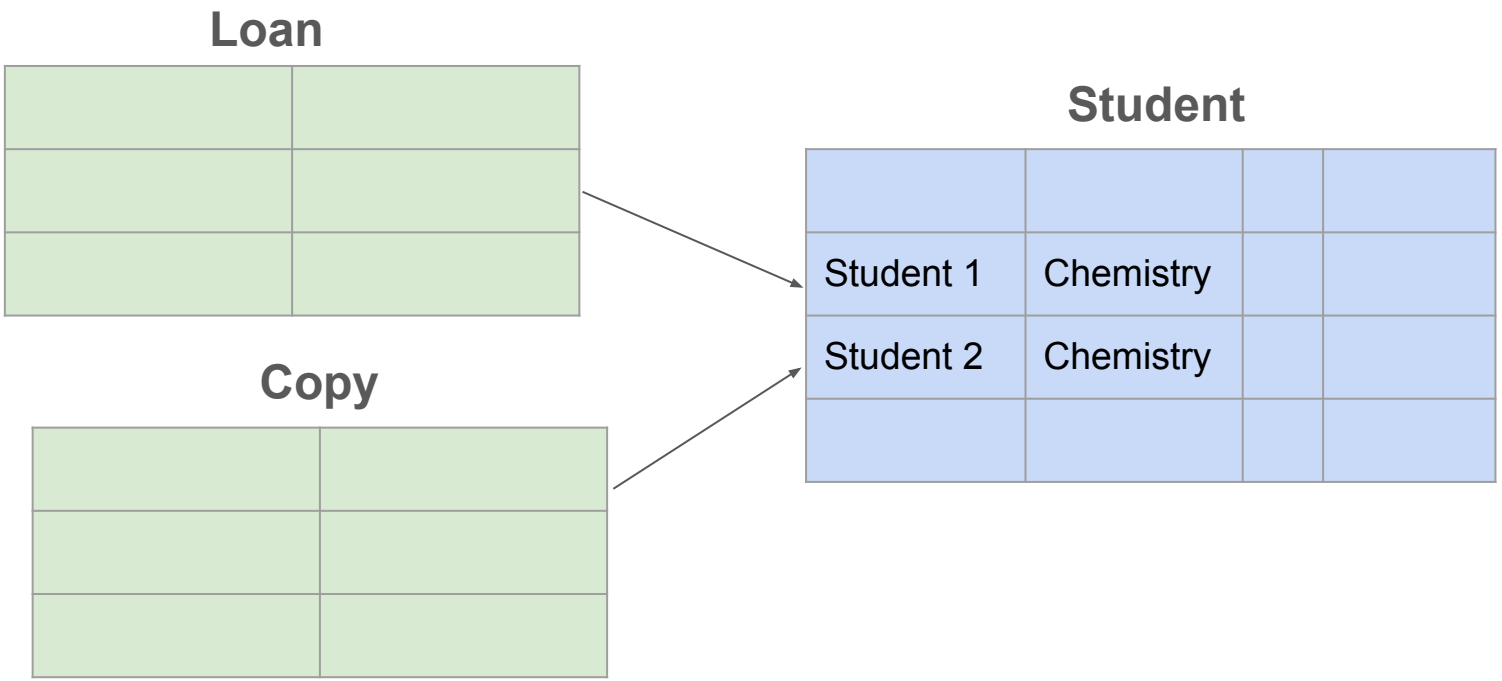
Code: DELETE FROM

```
DELETE FROM student
WHERE department = 'Chemistry';
```

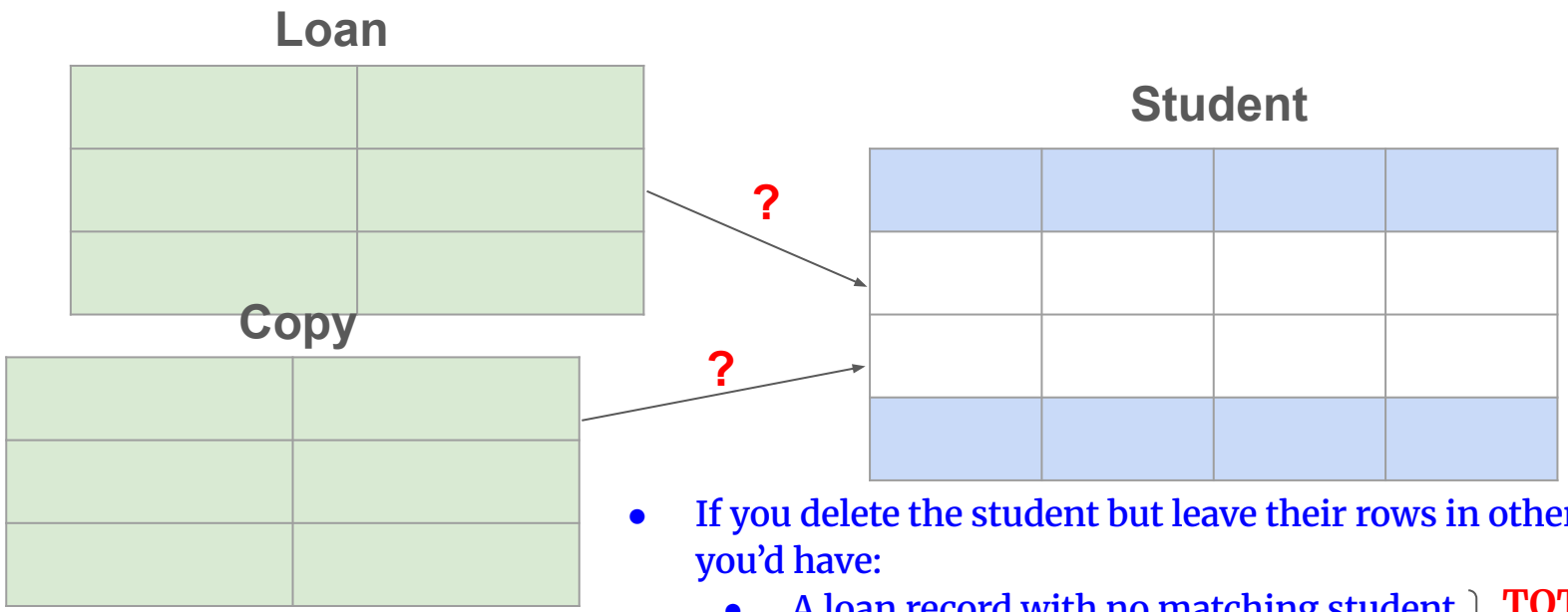
🤔WHY the DELETE fails?

- **Foreign key constraint:** student.email is referenced in other tables (e.g., loan, copy).
- A Chemistry student's email still exists in related tables.

2.(h) Delete all the students from the 'Chemistry' dept.



2.(h) Delete all the students from the 'Chemistry' dept.



- If you delete the student but leave their rows in other tables, you'd have:
  - A loan record with no matching student
  - A copy record with no matching owner

  } TOTAL CHAOS!
- Not an error in code — it's data integrity control! 😊

3(a). DEFERRABLE Constraints- What does it mean?

Deferring = temporarily allow violations in a transaction, as long as they're fixed before commit.

Upon creation, constraints (UNIQUE, PRIMARY KEY, FOREIGN KEY) can be:

- **NOT DEFERRABLE** – 🚫Always checked immediately; can't delay.
- **DEFERRABLE INITIALLY IMMEDIATE** – ⚡Checked immediately by default; can delay until commit.
- **DEFERRABLE INITIALLY DEFERRED** – ⏳Checked at commit by default; can switch to immediate.

📌 *NOTE:* CHECK constraints are always immediate (e.g., salary > 0 fails right away if set to -100).

3(a). DEFERRABLE Constraints- What does it mean?

```
-- NOT DEFERRABLE
BEGIN
  op1   -- causes violation
  -- check runs NOW → FAILS
  op2   -- never runs!
ROLLBACK
```

```
-- DEFERRABLE, INITIALLY DEFERRED
(default)
BEGIN
  op1   -- causes violation
(temporarily OK)
  op2   -- fixes violation
  COMMIT  -- checks ON COMMIT → PASS
```

3(a). DEFERRABLE Constraints- What does it mean?

```
-- DEFERRABLE, INITIALLY
IMMEDIATE (default)

BEGIN

  op1    -- causes violation

  -- check runs NOW → FAILS

  op2    -- never runs!

ROLLBACK
```

```
-- DEFERRABLE, INITIALLY IMMEDIATE
(defer this txn)

BEGIN
  SET CONSTRAINTS ... DEFERRED

  op1    -- causes violation
(temporarily OK)

  op2    -- fixes violation

  COMMIT  -- checks ON COMMIT → PASS
```

3(b). What is the difference?

Code: INSERT INTO

```
INSERT INTO copy VALUES (
  'tikki@gmail.com',
  '978 -0321197849',
  1,
  'TRUE'
);
```

Code: Transaction #1

```
BEGIN TRANSACTION;
  SET CONSTRAINTS ALL IMMEDIATE;
  DELETE FROM book WHERE ISBN13 = '978-0321197849';
  DELETE FROM copy WHERE book = '978-0321197849';
END TRANSACTION;
```

Code: Transaction #2

```
BEGIN TRANSACTION;
  SET CONSTRAINTS ALL DEFERRED;
  DELETE FROM book WHERE ISBN13 = '978-0321197849';
  DELETE FROM copy WHERE book = '978-0321197849';
END TRANSACTION;
```

3(b). What is the difference?

Code: Transaction #1

```
BEGIN TRANSACTION;
  SET CONSTRAINTS ALL IMMEDIATE;
  DELETE FROM book WHERE ISBN13 = '978-0321197849';
  DELETE FROM copy WHERE book = '978-0321197849';
END TRANSACTION;
```

SET CONSTRAINTS ALL IMMEDIATE; → checks foreign keys after each operation

Steps:

1. 🗑️Delete book 978-0321197849 from book → ❌ FK violation (a copy still points to it)
2. Second DELETE never runs.

FAILED!

3(b). What is the difference?

Code: Transaction #2

```
BEGIN TRANSACTION;
  SET CONSTRAINTS ALL DEFERRED;
  DELETE FROM book WHERE ISBN13 = '978-0321197849';
  DELETE FROM copy WHERE book = '978-0321197849';
END TRANSACTION;
```

SET CONSTRAINTS ALL DEFERRED; → checks foreign keys at end of transaction

Steps:

1. 🗑️Delete book from book (⚠️ Temporary inconsistency)
2. 🗑️Delete matching copies from copy (⏳ Constraint checked only after both operations finish!)

SUCCESSFUL DELETION!

Database may be temporarily inconsistent, but as long as it's fixed before commit, it's acceptable 😃.

4(a). Is there need for the available field in the table copy?

NO! Availability can be derived from loan:

- No ongoing loan (returned IS NULL) → available ✅
- Ongoing loan → unavailable ❌

📌 Storing it in copy is redundant and risks inconsistent data.

4(a). Is there need for the available field in the table copy?

Code: Query

```
SELECT owner, book, copy, returned
FROM loan
WHERE returned ISNULL;
```

📌Availability can be derived!

Code: ALTER TABLE

```
ALTER TABLE copy
DROP COLUMN available;
```

📌Remove the available column

4(b). Should student contain both dept and faculty?

NOPE!

- Each **department** → **belongs to exactly one faculty**
- So if we know the **department**, we already know the **faculty**
- Storing both in `student` = **duplicate data** → risk of contradictions ⚠️

4(b). Should student contain both dept and faculty?

STUDENT

		Faculty	Dept

📌 *NOTE:* Every time we insert a student, we have to retype both department and faculty.
→ Relies on 100% correct data entry
→ Small mistakes → inconsistencies!
→ Risk of garbage values → dept–faculty mismatch (e.g., CS with Faculty of Arts)

4(b). Should student contain both dept and faculty?

STUDENT

		Faculty	Dept

DEPARTMENT

Dept	Faculty

4(b). Should student contain both dept and faculty?

STUDENT

			Dept

DEPARTMENT

Dept	Faculty

ADVANTAGES:

1. 💾 Saves storage space
2. ✅ Lower risk of inconsistencies (no duplicate entries!)
3. 🔒 Strong data integrity — no ghost/garbage values 👻🚫

4(b). Should student contain both dept and faculty?

Code: Modification

```
CREATE TABLE department (
  department VARCHAR (32) PRIMARY KEY,
  faculty VARCHAR (62) NOT NULL
);
```

Create a lookup table!
One row per department, storing its (single) faculty.

```
INSERT INTO department
  SELECT DISTINCT department, faculty
  FROM student;
```

Populate it from existing data!
Copies unique (department, faculty) pairs out of student

```
ALTER TABLE student
DROP COLUMN faculty;
```

faculty no longer needs to live in student

```
ALTER TABLE student
ADD FOREIGN KEY (department) REFERENCES department (department);
```

— Enforces that each student's department must exist in dept

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