SQL: Creating and Populating Tables

Students at the National University of Ngendipura (NUN) buy books for their studies. They also lend and borrow books to and from other students. Your company, Apasaja Private Limited, is commissioned by NUN Students Association (NUNStA) to implement an online book exchange system that records information about students, books that they own and books that they lend and borrow.

The database records the name, faculty, and department of each student. Each student is identified in the system by her email. The database also records the date at which the student joined the university (year attribute).

The database records the title, authors, publisher, year and edition and the ISBN-10 and ISBN-13 for each book. The International Standard Book Number, ISBN-10 or -13, is an industry standard for the unique identification of books. It is possible that the database records books that are not owned by any students (because the owners of a copy graduated or because the book was advised by a lecturer for a course but not yet purchased by any student.)

The database records the date at which a book copy is borrowed and the date at which it is returned. We refer to this information as a loan record.

For auditing purposes the database records information about the books, the copies and the owners of the copies as long as the owners are students or as there are loan records concerning the copies. For auditing purposes the database records information about graduated students as long as there are loan records concerning books that they owned.

Questions

Not all questions will be discussed during tutorial. You are expected to attempt them before coming to the tutorial. You may be randomly called to present your answer during tutorial. You are encouraged to discuss them on Canvas Discussion.

1. **Data Definition Language**.

   You are provided with the relational schema and with a sample instance of the database as of December 31, 2010. You are given the SQL data definition language code to create the schema and the SQL data manipulation language code to create a instance of the database.

   (a) Download the following files from Canvas `Files` ▶ `Cases` ▶ `Book Exchange`.

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

(b) Read the SQL files. What are they doing?

Comments:

We can categorize the SQL files as follows.

- **Clean Up**: `NUNStAClean.sql` drops all the tables and their contents, if needed.

  Note: We use the keyword `IF EXISTS` to only perform `DROP TABLE` if the table exists. Otherwise, the statement does nothing. Without the keyword, the statement will throw an exception if the table does not exist. This will then skip any subsequent statements.

- **Schema**: `NUNStASchema.sql` creates the tables with the associated domain declarations and constraints.

  Note: We use the keyword `IF NOT EXISTS` to only perform `CREATE TABLE` if the table does not exist. Otherwise, the statement does nothing. Without the keyword, the statement will throw an exception if the table exists. This will then skip any subsequent statements.

- **Data**: `NUNStAStudent.sql`, `NUNStABook.sql`, `NUNStACopy.sql`, as well as `NUNStALoan.sql` populate their respective tables. The order the scripts is executed is important as will be explained in the next subquestion.

  Note: You may generate the data manually, use web crawler to find data online, implement a custom script to generate random data, or Mockaroo [4].

We highly recommend categorizing your work in the same as above. Using a little automation (e.g., bash script or Python script), you may then initialize or restart your code by running the scripts in the order above.

(c) Use the files to create and populate a database. If there is a bug, find it, and fix it.

Comments:

Create a database with a name of your choice in pgAdmin 4. Open the query tool in the database. Load and run the SQL files.

1. The first file to be run is `NUNStASchema.sql`. It creates the different tables. The referential integrity constraints (`FOREIGN KEY`) impose that the table `copy` and the table `loan` are created after the tables `student` and `book` and in that order.
2. The following two tables can be populated in any order.
   - The table `student` is populated using `NUNStAStudent.sql`.
   - The table `book` is populated using `NUNStABook.sql`.
3. The following two tables (`copy` and `loan`) can only be populated after the two tables above (`student` and `book`) are populated. The order must be in the specified order below because of the referential integrity constraints (`FOREIGN KEY`).
   (i) The table `copy` is populated using `NUNStACopy.sql`.
   (ii) The table `loan` is populated using `NUNStALoan.sql`.

The referential integrity constraints (`FOREIGN KEY`) impose that the table `loan` and the table `copy` are deleted before the tables `student` and `book` and in that order in `NUNStAClean.sql`.

2. **Insertion, Deletion, and Update**.

   The following set of questions assume Question 1 has already been completed.

   (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'
);
```

Comments:

Notice the following from the command above.

- A string is enclosed within a single quote `'` instead of double quotes `"`.
- The order of the fields is *implicit*. But it can be made explicit if needed.
- A date is given as a string in the `yyyy-mm-dd` format. This may depend on the locale, but `yyyy-mm-dd` format is the ISO standard [3].

You can check that the insertion was effective with the following query.

Code: SELECT

```
SELECT * FROM book;
```

   (b) Insert the *same book* but with a different `ISBN13` field. For instance, with `ISBN13` field value of `'978-0201385908'`. Describe the outcome of the operation.

Comments:
The `INSERT INTO` command is shown below.

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'
);
```

The command yields an error because `ISBN10` must be unique. PostgreSQL returns the following error message.

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.
```

All messages emitted by the PostgreSQL server are assigned five-character error codes that follow the SQL standard's conventions for "`SQLSTATE`" codes [5]. Applications that need to know which error condition has occurred should usually inspect the error code, rather than looking at the textual error message.

(c) Insert the *same book* but with a different `ISBN10` field. For instance, with `ISBN10` field value of `'0201385902'`. Describe the outcome of the operation.

Comments:
The `INSERT INTO` command is shown below.

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'
);
```

The command yields an error because `ISBN13` is a primary key and therefore must

be unique. PostgreSQL returns the following error message.

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.
```

(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
);
```

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

(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'
);
```

Comments:
Notice how we explicitly indicate the order of the fields in the insertion command. In this case, if a field is omitted from the column list, the system attempts to insert a `NULL` value for that field. Try the following insertion.

Code: INSERT INTO

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

The command does not work because `email` field is a primary key and therefore cannot be `NULL`.

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 ...
```

(f) Change the name of the department from `'CS'` to `'Computer Science'`. Describe the behavior.

Code: UPDATE

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

Comments:

You can check that the update was effective with the following queries. The first query below has no result.

Code: SELECT

```sql
SELECT *
FROM student
WHERE department = 'CS';
```

The second query below prints the students from the computer science department.

Code: SELECT

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

Note: The symbol `=` may mean different things depending on the context. Unlike typical programming languages, `=` is used for *comparison* in SQL. However, when it appears in an `UPDATE` statement, the `=` symbol is used similar to assignment.

(g) Delete all the students from the `'chemistry'` department. Describe the behavior.

Comments:

The `DELETE FROM` command is shown below.

Code: DELETE FROM

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

There is nothing deleted as `'chemistry'` is misspelled with a lowercase `'c'`.
Note: There is no error but nothing is deleted because there is no department called `'chemistry'`. See next question.

(h) Delete all the students from the `'Chemistry'` department. Describe the behavior.

Comments:
The `DELETE FROM` command is shown below.

Code: DELETE FROM

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

There is nothing deleted because a constraint is violated. It is not a programming error. It is part of the control of the access to the data.

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 ...
```

Extra challenge: What changes can you make to the schema (i.e., the code in `NUNStASchema.sql`) so that the above deletion command is successful?

3. **Integrity Constraints**.

The following set of questions assume Question 2 has already been completed.

(a) Some constraints in PostgreSQL are `DEFERRABLE` [6]. What does it mean?

Comments:
Upon creation, a `UNIQUE`, `PRIMARY KEY`, or `FOREIGN KEY` constraint is given one of three characteristics below.

- `DEFERRABLE INITIALLY IMMEDIATE`
- `DEFERRABLE INITIALLY DEFERRED`
- `NOT DEFERRABLE`

By default the above constraints and all others are `NOT DEFERRABLE`. A constraint that is a `NOT DEFERRABLE` constraint is always `IMMEDIATE`. By default a `DEFERRABLE` constraint is `INITIALLY IMMEDIATE`. Note how in `NUNStASchema.sql`, we have declared certain constraints as `DEFERRABLE`.
These qualifications refers to when the constraint is checked: immediately after each operation (`INSERT`, `DELETE`, `DELETE`), or at the end of the transaction executing the operation.
Although this is not the default setting, it is preferable that all constraints be deferred. Unfortunately, this is only possible for `UNIQUE`, `PRIMARY KEY`, and `FOREIGN KEY` constraints and not for `CHECK` constraints in the current version of PostgreSQL.

(b) Insert the following copy of `'An Introduction to Database Systems'` owned by Tikki.

Code: INSERT INTO

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

What is the difference between the following two SQL programs?

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;
```

Comments:

In the first transaction, the statement `SET CONSTRAINTS ALL IMMEDIATE;` makes the foreign key from the table `copy` to the table `book` checkable after each operation. The first deletion of the transaction violates this constraint.

This immediate behavior is the default, so the code below has the same issue.

Code: Transaction

```
BEGIN TRANSACTION;
  DELETE FROM book b WHERE b.ISBN13 = '978-0321197849';
  DELETE FROM copy c WHERE c.book = '978-0321197849';
END TRANSACTION;
```

When the execution is interrupted, you need to run `END TRANSACTION;` manually. You can also use commands like `BEGIN`, `COMMIT`, `SAVEPOINT my_savepoint`, and `ROLLBACK TO my savepoint;` to control the flow of transactions.

In the second transaction, the constraints are checked at the end of the transaction and are not violated. Namely, `SET CONSTRAINTS ALL DEFERRED;` makes the reference from the table `copy` to the table `book` checkable at the end of transactions. The combined effect of the two deletions in the transaction does not violate the `FOREIGN KEY` constraint. The transaction is committed.

To appreciate the effect of deferrable constraints, reset the database until you reach this question again. When executing Transaction #2, copy line-by-line. After line 3, we can check that the database is in an *inconsistent* state where we have a copy without a book.

Code: Checking Book

```
-- Result is empty, because after line 3 the book has been deleted
SELECT *
FROM book b
WHERE b.ISBN13 = '978-0321197849';
```

Code: Checking Copy

```
-- However, this is non-empty, despite having a FOREIGN KEY to book
-- => Intermediate state is inconsistent!
SELECT *
FROM copy c
WHERE c.book = '978-0321197849';
```

4. **Modifying the Schema**.

   The following set of questions assume Question 3 has already been completed.

   (a) Argue that there is no need for the `available` field in the table `copy`. Make the necessary changes.

Comments:

The availability of a copy can be derived from the fact that the copy has not been returned (and we could handle copies that are temporarily unavailable for other reasons in another table, but let us ignore this aspect). The following query finds the copies loaned but not returned and unavailable (there should be none).

Code: Query

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

We can then drop the `available` field in the table `copy`.

Code: ALTER TABLE

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

We could even create a view called `copy_view` with the field restored.

Code: VIEW

```
CREATE OR REPLACE VIEW copy_view (owner, book, copy, available) AS (
  SELECT DISTINCT c.owner, c.book, c.copy,
  CASE
    WHEN EXISTS (
      SELECT * FROM loan l
      WHERE l.owner = c.owner
        AND l.book = c.book
        AND l.copy = c.copy
        AND l.returned ISNULL
    ) THEN 'FALSE'
    ELSE 'TRUE'
  END
  FROM copy c
);

SELECT * FROM copy_view; -- use the view like a table
```

Can the view be updated?

Code: UPDATE

```
UPDATE copy_view
SET owner = 'tikki@google.com'
WHERE owner = 'tikki@gmail.com';
```

In principle, we should be able to update all fields except `available`. In reality, this action is not directly possible. However, it can be programmed using `INSTEAD OF` triggers or unconditional `ON UPDATE DO INSTEAD` rules.

Triggers is a topic for the second half of the semester.

We can also drop the view once it is no longer needed.

Code: DROP VIEW

```
DROP VIEW copy_view;
```

(b) Argue that the table `student` should not contain both the fields `department` and `faculty`. Make the necessary changes.

Comments:

The `department` determines the faculty (i.e., if we know the `department`, then we know the `faculty`). This notion called **functional dependencies** will be formalized in the second half of the semester.

This information can be stored once and for all in a separate table. The `student` table need only to store the `department`.

The changes can be done with the following SQL code.

Code: Modification

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

INSERT INTO department
  SELECT DISTINCT department, faculty
  FROM student;

ALTER TABLE student
DROP COLUMN faculty;

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

Comments:

Table naming convention varies between companies. Please follow the convention set by the company. Our convention is generally the following.

- Use singular noun for table name (e.g., `book` instead of `books`).
- Use snake case for table name for clarity (e.g., `copy_view` instead of `copyView` or `CopyView` or `copyview`).

While you are not forced to adopt our convention, please be **consistent** in your own convention (e.g., do not mix singular and plural such as `book` with `students`). This makes collaborating for project easier.

References

[1] S. Bressan and B. Catania. *Introduction to Database Systems*. McGraw-Hill Education, 2006. ISBN: 9780071246507.

[2] Hector Garcia-Molina, Jeffrey D. Ullman, and Jennifer Widom. *Database Systems: The Complete Book*. 2nd ed. Prentice Hall Press, 2008. ISBN: 9780131873254.

[3] *ISO 8601: Date and Time Format*. `https://www.iso.org/iso-8601-date-and-time-format.html`. [Online; last accessed 2025].

[4] *Mockaroo: Random Data Generator and API Mocking Tool*. `https://mockaroo.com/`. [Online; last accessed 2025].

[5] *PostgreSQL Docs: Appendix A. PostgreSQL Error Codes*. `https://www.postgresql.org/docs/current/errcodes-appendix.html`. [Online; last accessed 2025].

[6] *PostgreSQL Docs: CREATE TABLE (Deferrable)*. `https://www.postgresql.org/docs/17/sql-createtable.html#SQL-CREATETABLE-PARMS-DEFERRABLE`. [Online; last accessed 2025].

[7] Raghu Ramakrishnan and Johannes Gehrke. *Database Management Systems*. 2nd. USA: McGraw-Hill, Inc., 2000. ISBN: 0072440422.

[8] *W3schools Online Web Tutorials*. `https://www.w3schools.com/`. [Online; last accessed 2025].