



Tutorial: 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) and the date of her graduation (**graduate** attribute, which is NULL if the student has not graduated at the time of the database instance.).

The database records the title, authors, publisher, edition, 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.

The database records information about books, copies and owners as long as the owners are students. Loan records of books that have been returned are deleted at the end each semester. The database records information about graduated students as long as there are loan records concerning books that they owned or borrowed but did not return.

1. You are provided with the relational schema and with an instance of the database dated from the end of academic year 2021–2022 (31st July 2022). 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,
NUNStASudent.sql,
NUNStABook.sql,
NUNStACopy.sql,
NUNStALoan.sql,
and NUNStAClean.sql.

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

Solution: NUNStASchema.sql creates the tables with the associated domain declarations and constraints.

NUNStASudent.sql, NUNStABook.sql, NUNStACopy.sql and NUNStALoan.sql populate the tables.

NUNStAClean.sql drops all the tables and their content, if needed.

- (c) Use the files to create and populate a database (they may need some “bug” fixing).

Solution: 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.

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.

The table `student` is populated using `NUNStASudent.sql`.

The table `book` is populated using `NUNStABook.sql`.

These two tables can be populated in any order.

The table `copy` is populated using `NUNStACopy.sql`.

The table `loan` is populated using `NUNStALoan.sql`.

The table `copy` and the table `loan` can only be populated after the tables `student` and `book` are populated and in that order because of the referential integrity constraints (FOREIGN KEY).

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` matters.

2. Insertion, deletion, and update.

- (a) Insert the following new book.

```
1 INSERT INTO book VALUES (  
2 'An Introduction to Database Systems',  
3 'paperback',  
4 640,  
5 'English',  
6 'C. J. Date',  
7 'Pearson',  
8 '0321197844',  
9 '978-0321197849');
```

Solution:

Notice the implicit order of the fields.

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

```
1 SELECT *  
2 FROM book;
```

- (b) Insert the same book with a different ISBN13, for instance '978-0201385908'.

Solution:

```
1 INSERT INTO book VALUES (  
2 'An Introduction to Database Systems',  
3 'paperback',  
4 640,  
5 'English',  
6 'C.J. Date',  
7 'Pearson',  
8 '0321197844',  
9 '978-0201385908');
```

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

ERROR: duplicate key value violates unique constraint "book_isbn10_key"

DETAIL: Key (isbn10)=(0321197844) already exists.

SQL state: 23505

All messages emitted by the PostgreSQL server are assigned five-character error codes that follow the SQL standard's conventions for "SQLSTATE" codes. Applications that need to know which error condition has occurred should usually test the error code, rather than looking at the textual error message. See www.postgresql.org/docs/13/errcodes-appendix.html

- (c) Insert the same book with the original ISBN13 but with a different ISBN10, for instance '0201385902'.

Solution:

```
1 INSERT INTO book VALUES (  
2 'An Introduction to Database Systems',  
3 'hardcover',  
4 938,  
5 'English',  
6 'C.J. Date',  
7 'Addison Wesley Longman',  
8 '0201385902',  
9 '978-0321197849');
```

The command yields an error because ISBN13 is a primary key and therefore unique. PostgreSQL returns the following error message.

ERROR: duplicate key value violates unique constraint "book_pkey"

DETAIL: Key (isbn13)=(978-0321197849) already exists.

SQL state: 23505

- (d) Insert the following new student.

```
1 INSERT INTO student VALUES (  
2 'TIKKI TAVI',  
3 'tikki@gmail.com',  
4 '2021-08-01',  
5 'School of Computing',  
6 'CS',  
7 NULL);
```

Solution: 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.

```
1 INSERT INTO student (email, name, year, faculty, department)  
2 VALUES (  
3 'rikki@gmail.com',  
4 'RIKKI TAVI',  
5 '2021-08-01',  
6 'School of Computing',  
7 'CS');
```

Solution:

Notice how we explicitly indicate the order of the fields in the insertion command. In this case, if a field is omitted, the system attempts to insert a null value.

Try the following insertion.

```
1 INSERT INTO student (name, year, faculty, department)  
2 VALUES (  
3 'RIKKI TAVI',  
4 '2021-08-01',  
5 'School of Computing',  
6 'CS');
```

The command does not work because **email** is a primary key and therefore cannot be null.

ERROR: null value in column "email" of relation "student"
violates not-null constraint

DETAIL: Failing row contains (RIKKI TAVI, null, 2021-08-01, School of Computing, CS, null).

SQL state: 23502

- (f) Change the name of the department 'CS' to 'Computer Science'.

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

Solution:

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

```
1 SELECT *
2 FROM student
3 WHERE department = 'CS';
```

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

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

- (g) Delete the students from the 'chemistry' department.

Solution:

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

'chemistry' is misspelled with a lower case 'c'. There is no error but nothing is deleted because there is no department 'chemistry'.

- (h) Delete the students from the 'Chemistry' department.

Solution:

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

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

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

3. Integrity constraints (to be discussed in class.)

- (a) Some constraints in PostgreSQL are DEFERRABLE. What does it mean?

Solution: Upon creation, a UNIQUE, PRIMARY KEY, or FOREIGN KEY constraint is given one of three characteristics: DEFERRABLE INITIALLY IMMEDIATE, DEFERRABLE INITIALLY DEFERRED, and NOT DEFERRABLE.

By default the above constraints and all others are NOT DEFERRABLE. A NOT DEFERRABLE constraint is always IMMEDIATE. By default a DEFERRABLE constraint is INITIALLY IMMEDIATE.

These qualifications refers to when the constraint is checked: immediately after each operation (INSERT, DELETE, UPDATE), 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.

```
1 INSERT INTO copy
2 VALUES (
3 'tikki@gmail.com',
4 '978-0321197849',
5 1);
```

What is the difference between the following two SQL programs?

```

1 BEGIN TRANSACTION;
2 SET CONSTRAINTS ALL IMMEDIATE;
3 DELETE FROM book
4 WHERE ISBN13 = '978-0321197849' ;
5 DELETE FROM copy
6 WHERE book = '978-0321197849' ;
7 END TRANSACTION;

```

```

1 BEGIN TRANSACTION;
2 SET CONSTRAINTS ALL DEFERRED;
3 DELETE FROM book WHERE ISBN13 = '978-0321197849' ;
4 DELETE FROM copy WHERE book = '978-0321197849' ;
5 END TRANSACTION;

```

Solution: In the first transaction, the statement `SET CONSTRAINTS ALL IMMEDIATE;` checks the reference from the table `copy` to the table `book` checkable after each operation. The first deletion of the transaction violates the constraint.

```

1 BEGIN TRANSACTION;
2 DELETE FROM book
3 WHERE ISBN13 = '978-0321197849';
4 DELETE FROM copy
5 WHERE book = '978-0321197849';
6 END TRANSACTION;

```

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

```

1 BEGIN TRANSACTION;
2 DELETE FROM book
3 WHERE ISBN13 = '978-0321197849';
4 DELETE FROM copy
5 WHERE book = '978-0321197849';
6 END TRANSACTION;

```

When the execution is interrupted, you need to rollback the transaction explicitly and manually `ROLLBACK;` manually.

You can also use commands like `BEGIN`, `COMMIT` as well as `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.

Note that SQLite has a pragma called `defer_foreign_keys` to control deferred foreign keys.

4. Modifying the Schema (to be discussed in class.)

- (a) Add a field `available` to the table `copy` that indicates whether the copy is currently borrowed, `'false'`, or is available, `true'`. Make the necessary changes. Is it needed?

Solution: Add the field with a default value set to true.

```

1 ALTER TABLE copy
2 ADD COLUMN available BOOLEAN DEFAULT 'true';

```

```

1 UPDATE copy
2 SET available = 'false'
3 WHERE (owner, book, copy) IN
4 (SELECT owner, book, copy
5 FROM loan
6 WHERE returned ISNULL);

```

The availability of a copy can be derived. The following query finds the copies loaned but not returned and therefore not available.

```

1 SELECT owner, book, copy, returned
2 FROM loan
3 WHERE returned ISNULL;

```

We drop the field `available` in the table `copy`

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

We create a view `copy_view` with the field restored.

```
1 CREATE OR REPLACE VIEW copy_view (owner, book, copy, available)
2 AS
3 (SELECT DISTINCT c.owner, c.book, c.copy,
4     CASE
5         WHEN EXISTS (SELECT * FROM loan l
6                     WHERE l.owner = c.owner
7                           AND l.book = c.book
8                           AND l.copy = c.copy AND l.returned ISNULL) THEN 'FALSE'
9         ELSE 'TRUE'
10        END
11 FROM copy c);
12
13 SELECT * FROM copy_view;
```

Can the view be updated?

```
1 UPDATE copy_view
2 SET owner = 'rikki@gmail'
3 WHERE owner = 'tikki@gmail.com'
```

In principle we should be able to update all fields except `available`. It is however not directly possible but it can be programmed using `INSTEAD OF UPDATE` triggers or unconditional `ON UPDATE DO INSTEAD` rules. In addition, in this case, all fields are foreign keys and should be updated accordingly.

```
1 DROP VIEW copy_view;
```

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

Solution: The faculty is determined by the department. This information can be stored once and for all in a separate table. The student table needs only to store the department. The changes can be done with the following SQL code.

```
1 CREATE TABLE department (
2 department VARCHAR(32) PRIMARY KEY,
3 faculty VARCHAR(62) NOT NULL);
4
5 INSERT INTO department
6 SELECT DISTINCT department, faculty
7 FROM student;
8
9 ALTER TABLE student
10 DROP COLUMN faculty;
11
12 ALTER TABLE student
13 ADD FOREIGN KEY (department) REFERENCES department(department);
```

References

- [1] W3schools online web tutorials. www.w3schools.com. Visited on 21 July 2022.
- [2] S. Bressan and B. Catania. *Introduction to Database Systems*. McGraw-Hill Education, 2006.
- [3] R. Elmasri and S. B. Navathe. *Fundamentals of Database Systems*. Pearson, 7th edition, 2015.
- [4] H. Garcia-Molina, J.D. Ullman, and J. Widom. *Database Systems: The Complete Book*. Pearson international edition. Pearson Prentice Hall, 2009.
- [5] R. Ramakrishnan and J. Gehrke. *Database Management Systems*. McGraw-Hill, 2002.