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).

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.

1. (This part is to be completed in preparation for the tutorial.)

   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`,
   `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.

   `NUNStAStudent.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 (there is a bug, find it and fix it).

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 `NUNStAStudent.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. (This part is to be completed in preparation for the tutorial.) Insertion, deletion, and update.

(a) Insert the following new book.

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

Solution:

Notice the implicit order of the fields.

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

```
SELECT *
FROM book;
```

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

Solution:

```
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.

```
ERROR:  Key (isbn10)=(0321197844) already exists.duplicate key value violates unique constrain

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. 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/16/errcodes-appendix.html`

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

Solution:

```
INSERT INTO book VALUES (
'An Introduction to Database Systems',
'hardcover',
938,
'English',
'C.J. Date',
'Addison Wesley Longman',
'2000-01-01',
'0201385902',
'978-0321197849');
```

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

```
ERROR:  Key (isbn13)=(978-0321197849) already exists.duplicate key value violates unique const

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.

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

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

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

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

```
ERROR:  Failing row contains (RIKKI TAVI, null, 2024-08-15, School of Computing, CS, null).nul

ERROR:  null value in column "email" of relation "student" violates not-null constraint
SQL state: 23502
Detail: Failing row contains (RIKKI TAVI, null, 2024-08-15, School of Computing, CS, null).
```

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

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

Solution:

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

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

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

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

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

Solution:

```
DELETE FROM student
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:

```
DELETE FROM student
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:  Key (email)=(xiexin2011@gmail.com) is still referenced from table "loan".update or del

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

3. (This part is to be done during the tutorial.) Integrity constraints.

(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 INITIALLYIMMEDIATE`, `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.

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

What is the difference between the following two SQL programs?

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

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

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

```
BEGIN TRANSACTION;
DELETE FROM book
WHERE ISBN13 = '978-0321197849';
DELETE FROM copy
WHERE 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` 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. (This part is to be done during the tutorial.) Modifying the Schema.

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

Solution: 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).

```
SELECT owner, book, copy, returned
FROM loan
WHERE returned IS NULL;
```

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

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

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

```
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;
```

Can the view be updated?

```
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`. It is however not directly possible. However, it can be programmed using `INSTEAD OF UPDATE` triggers or unconditional `ON UPDATE DO INSTEAD` rules.

```
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 department determines the faculty. 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.

```
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);
```

References

[1] W3schools online web tutorials. `www.w3schools.com`. Visited on 26 July 2021.

[2] S. Bressan and B. Catania. *Introduction to Database Systems*. McGraw-Hill Education, 2006.

[3] H. Garcia-Molina, J.D. Ullman, and J. Widom. *Database Systems: The Complete Book*. Pearson international edition. Pearson Prentice Hall, 2009.

[4] R. Ramakrishnan and J. Gehrke. *Database Management Systems*. McGraw-Hill, 2002.