

BT5110 Data Management and Warehousing

Tutorial 1: Creating & Populating Tables with Constraints

Mark Meng Huasong

School of Computing
National University of Singapore

23 - 27 Aug 2021



Greeting!

Welcome to BT5110!

Mark MENG Huasong (孟华松)

B.Eng.(Hons), Computer Science, Nanyang Technological University
M.Comp., Infocomm Security, National University of Singapore

Been in cyber-security R&D industry since 2014, came back NUS for PhD in 2019

I will be your TA for the first half of the semester!

Question 1 (a-c)

(a) Download the following files from Luminus “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?

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

Solution 1 (b)

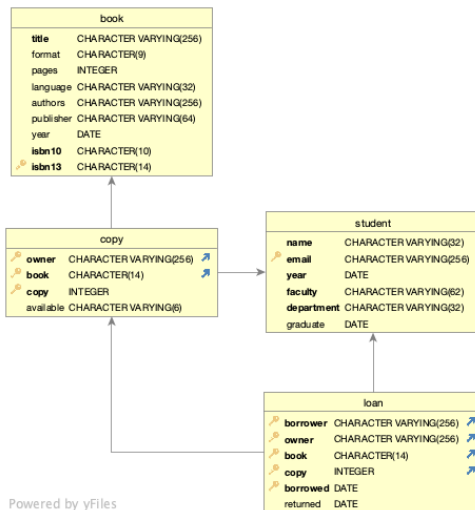


Figure: ER Diagram of NUNStASchema (plotted by DbVisualizer)

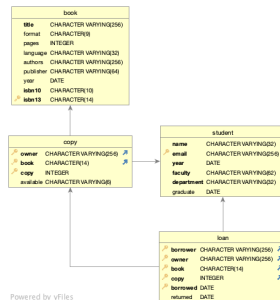
Solution 1 (b) Cont.

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.



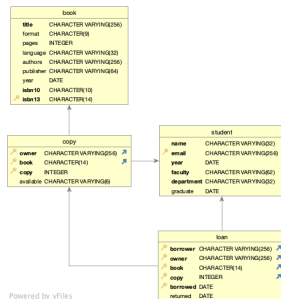
Solution 1 (b) Cont.

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.



Solution 1 (c)

There is a bug in `NUNStASchema.sql`, as the populating order of table `loan` (line 29-39) and `copy` (line 41-47) are wrong.

You can fix it by **swapping** these two code sections.

Then execute all SQL files except `NUNStAClean.sql` through PgAdmin.

Notice

You need to execute all those SQL files except `NUNStAClean.sql` for the sequential questions.

Question & Solution 2 (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      '2003-01-01' ,  
9      '0321197844' ,  
10     '978-0321197849');
```

Solution: Notice the implicit order of the fields.

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

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

Question & Solution 2 (b)

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

Solution: Execute the code bellow:

```
1  INSERT INTO book VALUES (  
2      'An_Introduction_to_Database_Systems',  
3      'paperback',  
4      640,  
5      'English',  
6      'C.J._Date',  
7      'Pearson',  
8      '2003-01-01',  
9      '0321197844',  
10     '978-0201385908');
```

Can you spot the problem in this code?

Solution 2 (b) Cont.

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
```

Remark

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

Question & Solution 2 (c)

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

Solution: Execute the code bellow:

```
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      '2000-01-01',  
9      '0201385902',  
10     '978-0321197849');
```

Will this code work?

Solution 2 (c) Cont.

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
```

Question & Solution 2 (d)

Insert the following new student.

```
1  INSERT INTO student VALUES (  
2    'TIKKI_TAVI' ,  
3    'tikki@gmail.com' ,  
4    '2010-01-01',  
5    'School_of_Computing',  
6    'CS',  
7    NULL);
```

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

Question & Solution 2 (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    '2010-01-01',
6    'School_of_Computing',
7    'CS');
```

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.

Question & Solution 2 (e) Cont.

Extra: How about the following code (insertion).

```
1  INSERT INTO student (name, year, faculty, department)
2  VALUES (
3    'RIKKI_TAVI',
4    '2010-01-01',
5    'School_of_Computing',
6    'CS');
```

Is this code okay?

Question & Solution 2 (e) Cont.

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

```
ERROR:  null value in column "email" violates not-null constraint
DETAIL:  Failing row contains (RIKKI TAVI, null, 2010-01-01,
School of Computing, CS, null).
SQL state: 23502
```


Question & Solution 2 (f)

Change the name of the department 'CS' to 'Computer Science'.

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

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

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

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

```
1  SELECT * FROM student WHERE department = 'Computer_Science';
```

Question & Solution 2 (g)

Delete the students from the 'chemistry' department.

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

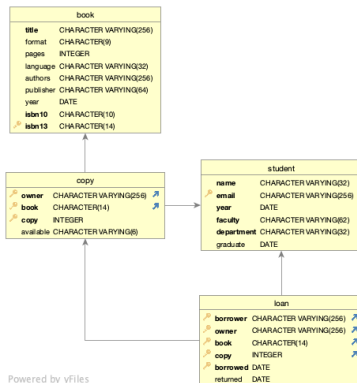
'chemistry' is misspelled with a lower case 'c'. There is no error but nothing is deleted because there is no department 'chemistry'.

Question & Solution 2 (h)

Delete the students from the 'Chemistry' department.

- 1 **DELETE FROM** student
- 2 **WHERE** department = 'Chemistry';

Will this code work?



Powered by yfiles

Question & Solution 2 (h) Cont.

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
```

Question & Solution 3 (a)

Some constraints in PostgreSQL are DEFERRABLE. What does it mean?

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.

Question & Solution 3 (a) Cont.

Very confused right?
DEFERRABLE vs DEFERRED?
Wait! INITIALLY, What?



Picture downloaded from <http://123rf.com/>

Let's do some demo to make this HUMAN-READABLE!

Question & Solution 3 (a) Cont.

Ext. Reading (Optional):

Deferrable SQL Constraints in Depth

<https://begriffs.com/posts/2017-08-27-deferrable-sql-constraints.html>

DEFERRED CONSTRAINTS in PostgreSQL

<http://dbadailystuff.com/deferred-constraints-in-postgresql>

Question & Solution 3 (b)

Insert the following copy of 'An Introduction to Database Systems' owned by Tikki.

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

What is the difference between the following two SQL programs? (cont. on the next page)

Question & Solution 3 (b) Cont.

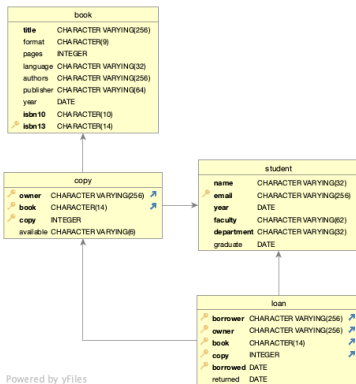
What is the difference between the following two SQL programs?

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

What is the difference between these two SQL programs?

Question & Solution 3 (b) Cont.



Powered by yfiles

```
1 CREATE TABLE IF NOT EXISTS book (...
2     ISBN13 CHAR(14) PRIMARY KEY
3     ...);
4
5 CREATE TABLE IF NOT EXISTS copy (...
6     book CHAR(14) REFERENCES book(ISBN13
7     ) DEFERRABLE,
8     ...);
9
10 CREATE TABLE IF NOT EXISTS loan (...
11     book CHAR(14),
12     ...);
```

Book is a foreign key of copy, but NOT a foreign key of loan.

Question & Solution 3 (b) Cont.

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

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.

```
ERROR: current transaction is aborted, commands ignored until end of  
transaction block  
SQL state: 25P02
```

Question & Solution 3 (b) Cont.

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

```
1 BEGIN TRANSACTION;  
2 DELETE FROM book WHERE ISBN13 = '978-0321197849';  
3 DELETE FROM copy WHERE book = '978-0321197849';  
4 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.

Extra: END is a PostgreSQL extension that provides functionality equivalent to COMMIT, which is specified in the SQL standard.

Question & Solution 3 (b) Cont.

```
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;
```

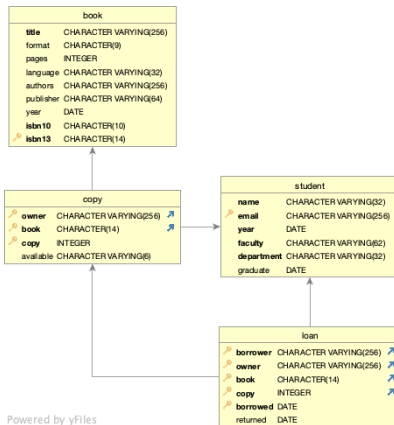
In the second transaction, the constraints are checked at the end of the transaction and are not violated. Namely, **SET CONSTRAINTS ALL DEFERRED** makes the reference from the table copy to the table book checkable at the end of transactions. The combined effect of the two deletions in the transaction 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.

Question & Solution 4 (a)

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

Do you agree with that? Why?
Any potential issue if we keep it?



Powered by yFiles

Question & Solution 4 (a) Cont.

Solution: 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 available field in the table copy

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

Question & Solution 4 (a) Cont.

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

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

Question & Solution 4 (a) Cont.

Can the view be updated?

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

In principle we should be able to update all fields except available.

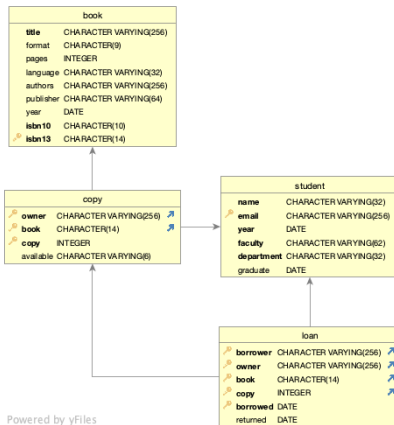
It is however not directly possible but it can be programmed using
INSTEAD OF UPDATE triggers or unconditional ON UPDATE DO INSTEAD
rules.

```
1  DROP VIEW copy_view;
```

Question & Solution 4 (b)

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

Do you agree with that? Why?



Powered by yFiles

Question & Solution 4 (b) Cont.

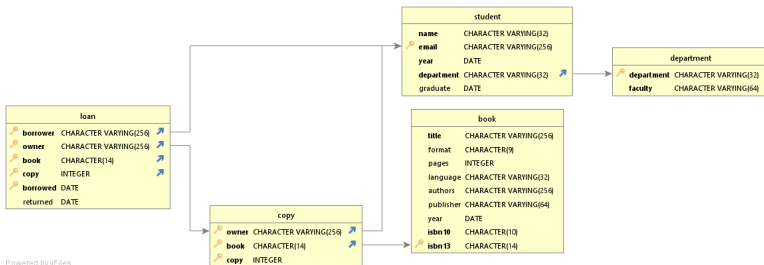
Solution: The faculty is determined by the department. This information can be stored once and for all in a separate table. The student table needs only to store the department.

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 FROM student;  
7  
8  ALTER TABLE student DROP COLUMN faculty;  
9  
10 ALTER TABLE student  
11 ADD FOREIGN KEY (department) REFERENCES department(department);
```

End of Tutorial 1

Below is what your database (should) be like by the end of this tutorial:



(plotted by DbVisualizer)

Submission due this week

(a) Your project 1 (Generating fake but realistic data) is due on this Friday 5:00PM (SGT)

(not Anytime on Earth, not 11:59PM, not sure the LumiNus and Internet work well at the last minute).

(b) Double check your code! Never lose any mark due to **typo** and **missing semicolons** (;)

(c) Creativity and “code cleanness” are always appreciated.

(extremely important since you are business students :D).

Submission due this week: Code Cleaness

Just an example! There are lots of free tools online for you to take advantage...



The screenshot shows the Code Beautify website's SQL Formatter tool. The interface includes a header with the site name and navigation links, a main title 'SQL Formatter' with a star icon, and a 'Save & Share' button. The tool is divided into three main sections: 'SQL Input', a central control panel, and 'Result : Beautify SQL'.

SQL Input: Contains a text area with the following SQL query:

```
1 select sr_customer_sk as ctr_customer_sk ,sr_store_sk
   as ctr_store_sk ,sum(SR_REVERSED_CHARGE) as
   ctr_total_return from store_returns ,date_dim
   where sr_returned_date_sk = d_date_sk and d_year
   =1999 group by sr_customer_sk ,sr_store_sk limit
   10;
```

Control Panel: Features buttons for 'Load Url', 'Browse', 'Beautify', 'Minify', 'Remove Comments', and 'Download'.

Result : Beautify SQL: Displays the formatted SQL query:

```
1 select
2   sr_customer_sk as ctr_customer_sk,
3   sr_store_sk as ctr_store_sk,
4   sum(SR_REVERSED_CHARGE) as ctr_total_return
5 from
6   store_returns,
7   date_dim
8 where
9   sr_returned_date_sk = d_date_sk
10  and d_year = 1999
11 group by
12   sr_customer_sk,
13   sr_store_sk
14 limit
15   10;
16
```

For any further question, please feel free to email me:

huasong.meng@u.nus.edu

See you next week!