Constraints

Garbage in, garbage out

GIGO

On two occasions I have been asked, "Pray, Mr. Babbage, if you put into the machine wrong figures, will the right answers come out?" ... I am not able rightly to apprehend the kind of confusion of ideas that could provoke such a question.

-- Charles Babbage, Passages from the Life of a Philosopher

A question of sanitation

Let’s pretend we have a table for storing data on persons:

```
sqlite> .schema persons
CREATE TABLE persons (last_name TEXT, first_name TEXT, born TEXT, sex TEXT);

sqlite> SELECT * from persons;
last_name        first_name       born        sex
---------------  ---------------  ----------  -----
Nilsson          Tommy            1960-03-11  Man
Norum            John             1964-02-23  Man
Jett             Joan             1958-09-22  Woman
Wilson           Ann              1950-06-19  Woman
sqlite>
```

Types too blunt

As you might have noticed, all the columns were of type text. While this is convenient, it is not a great choice for in particular the Born and Sex columns.

Why?

Garbage in

Suppose we add persons via either a web interface or a command line interface. The clerk who enters new persons into the system forgets what the valid values for Sex and Born were.

The following person is entered into the table:

“Doe”, “Jane”, “11/09/14”,”Female”

Inconsistent data

The state of the table is now:

```
sqlite> SELECT * from persons;
last_name        first_name       born        sex
---------------  ---------------  ----------  -------
Nilsson          Tommy            1960-03-11  Man
Norum            John             1964-02-23  Man
Jett             Joan             1958-09-22  Woman
Wilson           Ann              1950-06-19  Woman
Doe              Jane             11/09/14    Female
```

Ordering by birth date

Let’s order all persons by Born:

```
sqlite> SELECT * FROM persons ORDER BY born;
last_name        first_name       born        sex
---------------  ---------------  ----------  -------
Doe              Jane             11/09/14    Female
Wilson           Ann              1950-06-19  Woman
Jett             Joan             1958-09-22  Woman
Nilsson          Tommy            1960-03-11  Man
Norum            John             1964-02-23  Man
```

The very young Jane now is listed as the oldest person.

Selecting all women

```
sqlite> SELECT * FROM persons WHERE sex='Woman';
last_name        first_name       born        sex
---------------  ---------------  ----------  -------
Jett             Joan             1958-09-22  Woman
Wilson           Ann              1950-06-19  Woman
```

No Jane...

How do we fix this?

What about:

```
sqlite> SELECT * FROM persons WHERE sex='Woman' OR sex='Female';
```

What’s the problem with this?

Relax

I'll need some information first

Just the basic facts

Can you show me where it hurts?

Look for inconsistencies

We could list all variations of Sex:

```
sqlite> SELECT DISTINCT sex FROM persons;
sex
---------------
Man
Woman
Female
sqlite>
```

Two variations for Woman. Not good.

Guess what the DISTINCT keyword does? It is not part of the exam.

Statistics, please

Which version is in majority?

```
sqlite> SELECT sex, COUNT(*) FROM persons GROUP BY sex;
sex               COUNT(*)
---------------   ---------------
Female            1
Man               2
Woman             2
```

Only one “Female”

COUNT combined with GROUP BY is sometimes quite useful. They are not part of the exam, and just used here for reference.

Fixing the problem

Wouldn’t it be better if we only accepted valid input into the persons table?

This is where constraints come into play.

Let’s start with the values for Sex.

Warning! The syntax for constraints used in this lecture are specific to SQLite3. If you are using a different dbms, check the manual for equivalent constraints.

Using an enumeration for the Sex column

```
CREATE TABLE persons2(last_name TEXT, first_name TEXT, born TEXT,
  sex TEXT CHECK( sex IN ('Woman','Man')));
```

Let’s try to violate that constraint!

```
sqlite> INSERT INTO persons2 VALUES('Joplin', 'Janis', '1943-01-19',
  'Female');
Error: CHECK constraint failed: persons2
sqlite>
```

Using a function to check dates

We want to force dates to have the format ‘YYYY-mm-dd HH:MM:SS’:

```
CREATE TABLE persons2(last_name TEXT, first_name TEXT,
 born DATETIME CHECK(born IS datetime(Born)),
 Sex text      CHECK(Sex IN ('Woman','Man'))
);
/* datetime converts a string to a date of the format above if it can */
```

Violation:

```
sqlite> INSERT INTO persons2 VALUES('Joplin', 'Janis', '01/19/43','Woman');
Error: CHECK constraint failed: persons2
sqlite>
```

What about foreign keys?

If we go back to our books example where we normalised the Publisher to its own table, wouldn't it be great to only accept a valid publisher_id for a book?

We don't want a book to have an invalid publisherid, i.e. one that is not an ID in the publishers table.

```
sqlite> select * from publishers;
1|Bonnier
2|Books R us
sqlite>
```

What would it mean to store a book with PublisherID 666? Inconsistency.

Creating a foreign key constraint

Let’s make sure that PublisherID in the books table really references a valid publisher_id in the publishers table:

```
sqlite> CREATE TABLE "books2"( author TEXT, title TEXT,
  isbn TEXT PRIMARY KEY,
  publisherid INTEGER,
  FOREIGN KEY(publisherid) REFERENCES publishers(publisher_id)
);
```

Test: Insert a book with an invalid PublisherID:

```
sqlite> PRAGMA foreign_keys=true;
sqlite> INSERT INTO books2 VALUES ('Tage Danielsson', 'Tankar från roten',
'0-0-0-0-0-6',666);
Error: FOREIGN KEY constraint failed
```

Primary keys are enforced to be unique

In the books table, ISBN was selected as the PRIMARY KEY with only unique values.

Test: Insert a book with an already existing ISBN:

```
sqlite> INSERT INTO books2 VALUES ('Tage Danielsson', 'Tankar från roten',
'0-0-0-0-0-1',2);
Error: UNIQUE constraint failed: books2.ISBN
sqlite>
```

But... we can sanitize input at the application layer

Yes. But can you guarantee that nobody enters data into the database using some other means?

Does it hurt to sanitize input in more than one place?

As a tester, it is your job to find potential future problems, at all layers.

Discussion

What different strategy could we have used for correctness of the values for Sex? Hint: there are only two legal sexes in Sweden. What datatype is typically used for modelling one of two values?

What if Sweden implements a third legal sex. Which strategy seems better for the future?

The persons table could actually be used to represent Authors. What changes would we need in order to implement that in the books table?

Read

http://zetcode.com/db/sqlite/constraints/

https://www.sqlite.org/lang_createtable.html#constraints

https://www.sqlite.org/lang_datefunc.html

https://www.sqlite.org/pragma.html#pragma_foreign_keys

The persons table with constraints

```
PRAGMA foreign_keys=OFF;
BEGIN TRANSACTION;
CREATE TABLE persons (last_name TEXT, first_name TEXT, born TEXT, sex TEXT);
INSERT INTO "persons" VALUES('Nilsson','Tommy','1960-03-11','Man');
INSERT INTO "persons" VALUES('Norum','John','1964-02-23','Man');
INSERT INTO "persons" VALUES('Jett','Joan','1958-09-22','Woman');
INSERT INTO "persons" VALUES('Wilson','Ann','1950-06-19','Woman');
INSERT INTO "persons" VALUES('Doe','Jane','11/09/14','Female');
CREATE TABLE persons2(last_name TEXT, first_name TEXT, born DATETIME
check(born IS datetime(born)), sex TEXT CHECK( sex IN ('Woman','Man')));
INSERT INTO "persons2" VALUES('Joplin','Janis','1943-01-19
00:00:00','Woman');
COMMIT;
```

The books2 table with constraints

```
PRAGMA foreign_keys=ON;
BEGIN TRANSACTION;
CREATE TABLE "books2"(author TEXT, title TEXT,
                      isbn TEXT PRIMARY KEY,
                  publisherid INTEGER,
                  FOREIGN KEY(publisherid)
                   REFERENCES publishers(publisher_id)
);
INSERT INTO "books2" VALUES('John Smith','Life','0-0-0-0-0-1',1);
INSERT INTO "books2" VALUES('James Woody','Love','0-0-0-0-0-2',1);
INSERT INTO "books2" VALUES('Joan Carmen','Guns','0-0-0-0-0-3',1);
INSERT INTO "books2" VALUES('Johnanna Boyd','Code','0-0-0-0-0-4',1);
INSERT INTO "books2" VALUES('Eva Peron','Cars','0-0-0-0-0-5',2);
COMMIT;
```