Data Management and Warehousing

Introduction

Stéphane Bressan

NUS
National University
of Singapore





We want to develop a sales analysis application for our online gaming store. We would like to store several items of information about our customers: their first name, last name, date of birth, e-mail, date and country of registration on our online sales service and the customer identifier that they have chosen . We also want to manage the list of our products, the games, their name, their version and their price. The price is fixed for each version of each game. Finally, our customers buy and download games. So we must remember which version of which game each client has downloaded. It is not important to keep the download date for this application.

1. Go to `www.sqlite.org`
2. Go to `www.sqlite.org/download.html`
3. Download the command-line shell for accessing and modifying SQLite databases ("`A bundle of` …")
4. Extract the executable
5. You will find a short documentation at www.sqlite.org/sqlite.html


```
› .open bt5110.db
› .mode column
› .headers on
› .help
› ...
› ...

› .quit
```

You may skip the "`.open bt5110.db`" for now. That step creates a persistent database but also may slow down some update operations.

Let us create a table to store customer information. Let us call this table "`customers`". Each record in the table represents a customer. Each record contains a field to record the customer's first name, "`first_name`", a field for saving the last name of the customer, "`last_name`", a field for saving the customer's email, "`email`", a field for saving the customer's date of birth, "`dob`", a field to record the customer's registration date, "`since`", a field to record the customer's identifier, "`customerid`" and finally a field to register the country of registration of the customer, "`country`".

The SQL "CREATE TABLE" command creates the table. The name of the table and the name and domain (type) of each field must be specified: string of variable length, "VARCHAR ( )" and date, "DATE".

```
CREATE TABLE customers (
      first_name VARCHAR(64),
      last_name VARCHAR(64) ,
      email VARCHAR(64),
      dob DATE,
      since DATE,
      customerid VARCHAR(16) ,
      country VARCHAR(16));
```

Let us create a table to save the information about the games. Let us call this table "`games`". Each record in this table represents a version of a game. It contains a field to save the name of the game, "`name`", a field to save the version of the game, "`version`" and a field to record the sale price of the game, "`price`".

We see two new domains: fixed character strings "CHAR ()" and "NUMERIC" numbers. There are many others.

```
CREATE TABLE games (
      name VARCHAR(32),
      version CHAR(3),
      price NUMERIC );
```

Let us create a table to save information about downloads. Let us call this table "`downloads`". Each record in this table represents the download of a version of a game by a customer. It contains a field to record the customer ID, "`customerid`", a field to save the name of the game, "`name`" and a field to save the version of the game, "`version`".

Note that these three fields have the same names and domains as the corresponding fields in the "games" and "customers" tables.

```
CREATE TABLE downloads (
      customerid VARCHAR(16),
      name VARCHAR(32),
      version CHAR(3));
```

We can now start inserting data. Let us create a record for the customer Carole Yoga. Carole was born on August 1, 1989. She registered with our online service on September 15, 2016 from France with the identifier "Carole89".

```
INSERT INTO customers VALUES(
      'Carole',
      'Yoga',
      'cyoga@glarge.org',
      '1989-08-01',
      '2016-09-15',
      'Carole89',
      'France');
```

We remove the three tables and their content.

```
DROP TABLE downloads;

DROP TABLE customers;

DROP TABLE games;
```

The files below contain the SQL commands for creating the "customers", "games" and "downloads" tables as well as the data insertion commands, respectively.

customers.sql

games.sql

downloads.sql

```
CREATE TABLE downloads(
        customerid VARCHAR(16) NOT NULL REFERENCES customers(customerid),
        name VARCHAR(32) NOT NULL,
        version CHAR(3) NOT NULL,
        FOREIGN KEY (name, version) REFERENCES games(name, version),
        PRIMARY KEY(customerid, name, version));


INSERT INTO downloads VALUES ('Adam1983', 'Biodex', '1.0');
INSERT INTO downloads VALUES ('Adam1983', 'Domainer', '2.1');
```

Excerpt from the file downloads.sql


```
› PRAGMA foreign_keys = ON;
› .read customers.sql
› .read games.sql
› .read downloads.sql
```

A simple SQL query must include the "SELECT" clause, which specifies the fields to be printed, the "FROM" clause, which indicates the tables to be queried, and possibly the "WHERE" clause, which contains a possible condition on the records to be considered.
For example, the following query displays the first and last names of registered customers from Singapore.

```
SELECT first_name, last_name
FROM customers
WHERE country = 'Singapore';
```

```
UPDATE games
SET price = price *1.1
WHERE version = '3.0';

DELETE FROM customers
WHERE last_name = 'Yoga';
```

NUS National University of Singapore

SQL Integrity Constraints

```
PRIMARY KEY
NOT NULL
UNIQUE
FOREIGN KEY
CHECK
```

NUS National University of Singapore

Structural Constraints

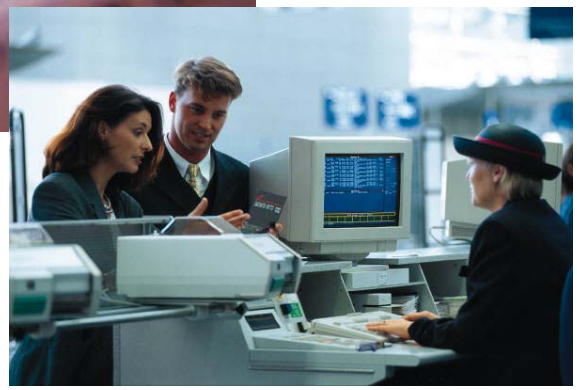
```
CREATE TABLE downloads(
first_name VARCHAR(64),
last_name VARCHAR(64),
email VARCHAR(64),
dob DATE,
since DATE,
customerid VARCHAR(16),
country VARCHAR(16)
name VARCHAR(32),
version CHAR(3),
price NUMERIC);
```

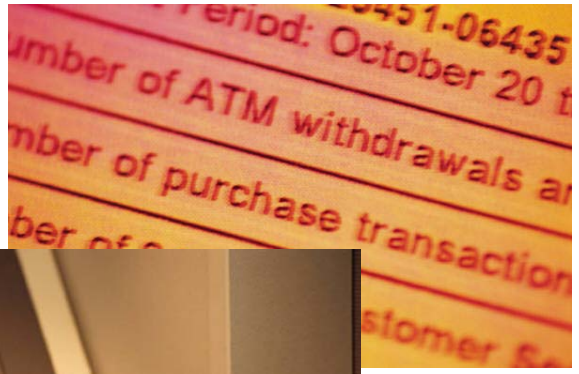
The choice of the number of columns and their domains imposes structural constraints.

If we use only one table, there can be no customer without a game and no game without a customer, unless we use NULL values.

Transactions

A transaction is a logical unit of work carried out by a user or an application. Make sure that constrained are `'DEFERRED'` that is checked just before the transaction is committed. Whether it is possible and how it is done may depend on the DBMS.





Column (Value) Constraint – PRIMARY KEY

A primary key is a set of attributes that identifies uniquely a record. You cannot have two records with the same value of their primary key in the same table.

```
CREATE TABLE customers (
first_name VARCHAR(64),
last_name VARCHAR(64),
email VARCHAR(64),
dob DATE,
since DATE,
customerid VARCHAR(16) PRIMARY KEY,
country VARCHAR(16));
```

The primary key attribute is underlined.

```
customers (first_name, last_name, email, dob, since,
customerid, country)
```

Table Constraint – COMPOSITE PRIMARY KEY

For convenience, the primary key can be composite: it is the combination of several attributes

```
CREATE TABLE games(
name VARCHAR(32),
version CHAR(3),
price NUMERIC,
PRIMARY KEY (name, version));
```

The primary key attributes (prime attributes) are underlined.

games(name, version, price);

SQLite: PRIMARY KEY

```
SELECT * FROM games
WHERE name= 'Aerified'
AND version = '1.0';

INSERT INTO games
VALUES ('Aerified', '1.0', 5);
```

The operation or the transaction violating the constraints is aborted and rolled back. An error is raised that the programmer can catch.

NOT NULL

We can require that the values in a column are not null.

```
CREATE TABLE games(
name VARCHAR(32),
version CHAR(3),
price NUMERIC NOT NULL);
```

NUS National University of Singapore

SQLite: NOT NULL

```
INSERT INTO games (name, version)
VALUES ('Aerified2', '1.0');
```

An error is raised. The operation or the transaction violating the constraints is aborted and rolled back.

Alternatively we could set a default value at table creation time:

```
price NUMERIC DEFAULT 1.00,
```

NUS National University of Singapore

Column Constraint – NOT NULL

According to the SQL standard prime attributes are automatically not null.

```
CREATE TABLE games (
name VARCHAR(32),
version CHAR(3),
price NUMERIC NOT NULL,
PRIMARY KEY (name, version));
```

It is not the case in SQLite and some other systems or versions of other systems.

```
CREATE TABLE games (
name VARCHAR(32) NOT NULL,
version CHAR(3) NOT NULL,
price NUMERIC NOT NULL,
PRIMARY KEY (name, version));
```

NUS National University of Singapore

SQLite's Apology

"According to the SQL standard, PRIMARY KEY should always imply NOT NULL. Unfortunately, due to a bug in some early versions, this is not the case in SQLite. Unless the column is an INTEGER PRIMARY KEY or the table is a WITHOUT ROWID table or the column is declared NOT NULL, SQLite allows NULL values in a PRIMARY KEY column. SQLite could be fixed to conform to the standard, but doing so might break legacy applications. Hence, it has been decided to merely document the fact that SQLite allowing NULLs in most PRIMARY KEY columns."

http://www.sqlite.org/lang_createtable.html

NUS National University of Singapore

Column Constraint - UNIQUE

You cannot have two records with the same value for a unique attribute key in the same table.

```
CREATE TABLE customers (
first_name VARCHAR(64),
last_name VARCHAR(64),
email VARCHAR(64) UNIQUE,
dob DATE,
since DATE,
customerid VARCHAR(16),
country VARCHAR(16));
```

Table Constraint - UNIQUE

```
CREATE TABLE customers (
first_name VARCHAR(64),
last_name VARCHAR(64),
email VARCHAR(64),
dob DATE,
since DATE,
customerid VARCHAR(16),
country VARCHAR(16),
UNIQUE (first_name, last_name));
```

The combination of values of the two attributes must be unique.

UNIQUE NOT NULL vs PRIMARY KEY

```
CREATE TABLE customers (
first_name VARCHAR(64),
last_name VARCHAR(64),
email VARCHAR(64) UNIQUE NOT NULL,
dob DATE,
since DATE,
customerid VARCHAR(16) PRIMARY KEY,
country VARCHAR(16));
```

There can be several candidate keys. They are all unique and not null but SQL requires that only one be declared to be the primary key.

NUS National University of Singapore

SQLite: UNIQUE

```
SELECT * FROM customers
WHERE email = 'druiz0@drupal.org';

insert into customers values ('Deborah', 'Ruiz',
'druiz0@drupal.org', '1984-08-01', '2016-10-17',
'Deborah32', 'Japan');

SELECT * FROM customers
WHERE email = 'druiz0@drupal.org';
```

An error is raised. The operation or the transaction violating the constraints is aborted and rolled back.

Column Constraint – FOREIGN KEY

(referential integrity)

```
CREATE TABLE downloads(
customerid VARCHAR(16) REFERENCES customers
(customerid),
name VARCHAR(32),
version CHAR(3));
```

The column "customerid" in the table downloads references the primary key "customerid" of the table "customers".

The column "customerid" in the table downloads can only take values that appears in the column "customerid" of the table "customers".

Table Constraint – FOREIGN KEY

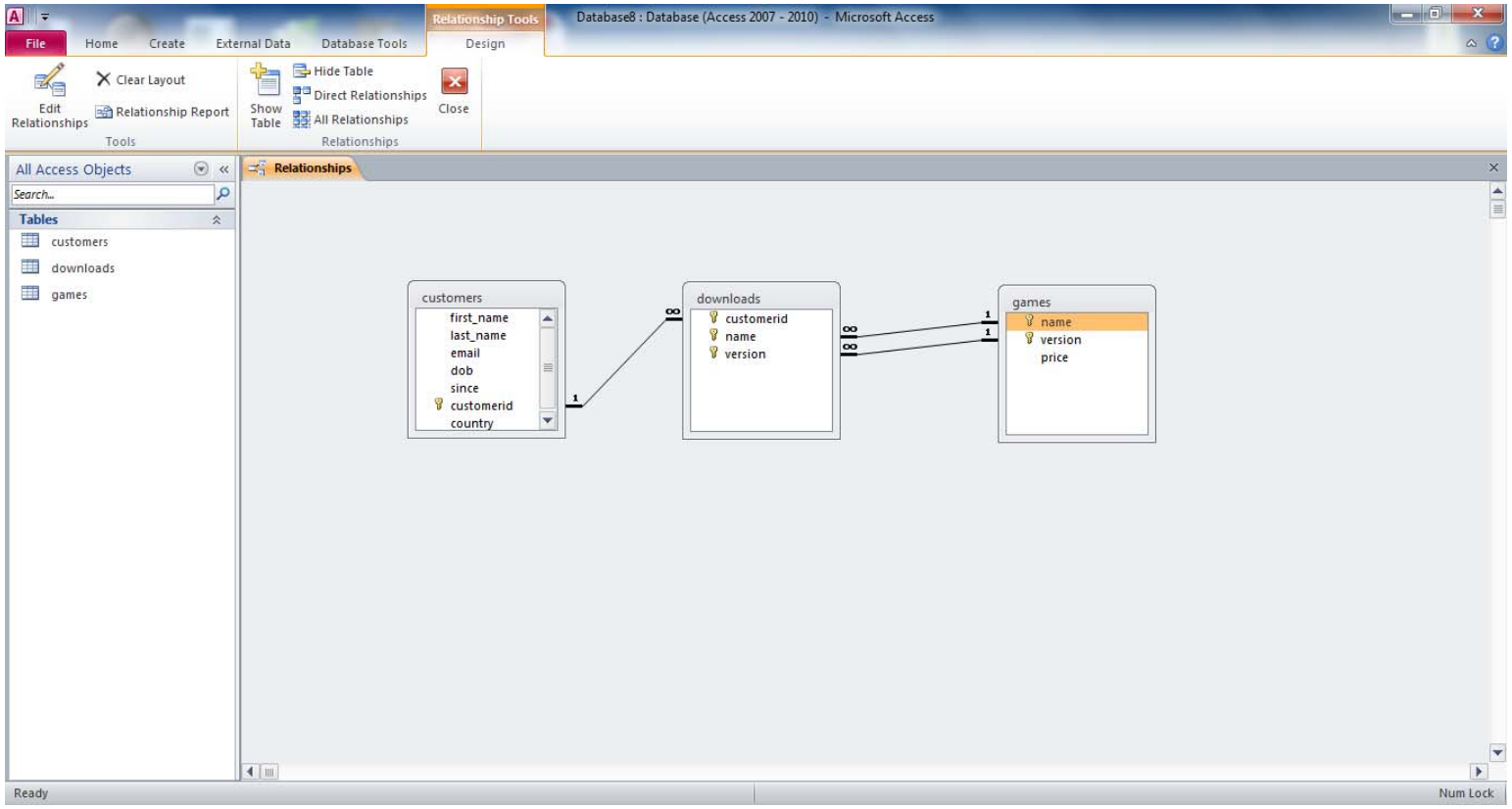
(referential integrity)

```
CREATE TABLE downloads(
customerid VARCHAR(16),
name VARCHAR(32),
version CHAR(3),
FOREIGN KEY (name, version) REFERENCES games (name,
version)
```

The columns "name", "version" in the table "downloads" references the (composite) primary key "name", "version" of the table "games".

The columns "name", "version" in the table "downloads" can only take a combination of values that appears in the columns "name", "version" of a row of the table "games".

Database Design: Logical Diagram (MS Access)



This is not an Entity-relationship diagram!

Column Constraint - FOREIGN KEY

(referential integrity)

User tom1999 changed his customerid from tom1999 to THOM1999

downloads

name	version	customerid
Skype	1.0	tom1999
Comfort	1.1	john88
Skype	2.0	tom1999
Comfort	5.1	hal2001

customers

customerid	email	…
THOM1999	tlee@gmail.com	…
john88	al@hotmail.com	…
Walnuts	dcs@nus.edu.sg	…

NUS National University of Singapore

SQLite: FOREIGN KEY

Let us try:

```
PRAGMA foreign_keys = ON;

INSERT INTO downloads VALUES ('Hal2001', 'Comfort',
'5.1');

DELETE FROM customers WHERE customerid = 'Willie90';
```

An error is raised (SQLite forgets to tell us the name of the constraint). The operation or the transaction violating the constraints is aborted and rolled back.

Column Constraint - CHECK

The most general constraint is "`CHECK()`". It checks any condition written in SQL.

```
CREATE TABLE customers(
first_name VARCHAR(64),
last_name VARCHAR(64),
email VARCHAR(64),
dob DATE,
since DATE  CHECK (since >= '2015-02-29'),
customerid VARCHAR(16),
country VARCHAR(16));
```

See also "`CREATE DOMAIN`" and "`CREATE TYPE`". They are not constraints and will raise program errors.

Table Constraint – CHECK (Mostly not available!)

```
create table customers (
        first_name VARCHAR(64),
        last_name VARCHAR(64),
        email VARCHAR(64),
        dob DATE,
        since DATE CONSTRAINT since CHECK (since >=
'2015-02-28'),
        customerid VARCHAR(16),
        country VARCHAR(16),
        CONSTRAINT dob CHECK (since > dob));
```

We should be able to put any complex SQL statement inside CHECK… but it does not work in most database management systems…

NUS National University of Singapore

SQLite: CHECK

```
INSERT INTO customers VALUES ('John', 'Vanne',
'jvanne@usa.com', '2000-02-29', '2014-02-29', 'jv99',
'Singapore');
```

An error is raised and the data is not inserted.

```
INSERT INTO customers VALUES ('John', 'Vanne',
'jvanne@usa.com', '2016-03-29', '2016-02-29', 'jv99',
'Singapore');
```

An error is raised and the data is not inserted.

The operation or the transaction violating the constraints is aborted and rolled back. An error is raised that the programmer can catch.

Propagating Updates and Deletions

downloads

name	version	customerid
Skype	1.0	tom1999
Comfort	1.1	john88
Skype	2.0	tom1999

customers

customerid	email	…
tom1999	tlee@gmail.com	…
john88	al@hotmail.com	…
Walnuts	dcs@nus.edu.sg	…

NUS National University of Singapore

Propagating Updates and Deletions

```
CREATE TABLE downloads(
customerid VARCHAR(16)
        REFERENCES customers (customerid)
        ON UPDATE CASCADE ON DELETE CASCADE,
name VARCHAR(32),
version CHAR(3),
FOREIGN KEY (name, version)
        REFERENCES games (name, version)
        ON UPDATE CASCADE ON DELETE CASCADE);
```

The annotations `"ON UPDATE/DELETE"` with the option `"CASCADE"` propagate the update or deletion. They can also be used with the self-describing options: `"NO ACTION"`, `"SET DEFAULT"` and `"SET NULL"` (note that a null value does not violate referential integrity in SQL).





Read the files "`customers.sql`", "`games.sql`" and "`downloads.sql`" and identify all the integrity constraints.

NUS National University of Singapore

Credits

Images and clips used in this presentation are licensed from Microsoft Office Online Clipart and Media

For questions about the content of this course and about copyrights, please contact Stéphane Bressan

steph@nus.edu.sg