Introduction to Database Systems

The Relational Model: Creating Tables with Constraints in SQL

Stéphane Bressan

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 version and 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 customer 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 cs2102.db`	open or create a database
› `.mode column`	display results in columns
› `.headers on`	display names of fields
› `.help`	
› `...`	
› `...`	
› `.quit`	save and quit

You may skip the "`.open cs2102.db`“ for now. That step creates a persistent database but also may slow down some update operations.

Data Models

Hierarchical Model 1965 (IMS)

Network Model 1965 (DBTG)

Relational Model (1NF) 1970s

(E.F. Codd “A Relational Model for Large Shared Data Banks” Communication of the ACM, Vol 13, #6)

Nested Relational Model 1970s

Complex Object 1980s

Object Model 1980 (OQL)

Object Relational Model 1990s (SQL)

XML (DTD), XML Schema 1990s (Xpath, Xquery)

NoSQL Databases (MongoDB)

The Relational Model

Use mathematics to describe and represent records and collections of records: the relation can be understood formally, leads to formal query languages and properties can be explained and proven

Use a simple data structure: the table is simple to understand, is a useful data structure (capture many situations) and leads to useful yet not too complex query languages

(SQL was invented by D. Chamberlain and R. Boyce in 1974 at IBM for the first relational database management system "System R". SQL is an ANSI standard since 1986. SQL is an ISO standard since 1987. We refer to the SQL-92 (or SQL2) standard)

Idea

The relation is a set of t-uples. It has no duplicate (not a multi-set). It has no order (not a sequence).

The table is a sequence of records. It can have duplicates. It has order.

SQL compromises between the relation and the table. It has duplicates. It uses "`PRIMARY KEY`", "`UNIQUE`" and "`DISTINCT`" and "`GROUP BY`" to eliminate duplicates. It has no order. It uses "`ORDER BY`" to order data

SQL DDL Statement

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

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

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

Inserting Data

```
INSERT INTO customers(first_name, last_name, email,
dob, since, customerid, country) VALUES ('Deborah',
'Ruiz', 'druiz0@drupal.org', '1984-08-01', '2016-10-
17', 'Deborah84', 'Singapore');

INSERT INTO customers(last_name, first_name, email,
dob, since, customerid, country) VALUES ('Gonzalez',
'Denise', 'dgonzalez9@slate.com', '1982-12-06', '2016-
07-26', 'Denise82', 'Malaysia');

INSERT INTO customers (email, last_name, country)
VALUES ('wleong3@shop-pro.jp', 'Leong', 'Singapore');

INSERT INTO customers(first_name VALUES ('Deborah',
'Ruiz', 'druiz0@drupal.org', '1984-08-01', '2016-10-
17', 'Deborah84', 'Singapore');

SELECT * FROM customers;
```

Relation Instance

column (fields)

attribute name:

domain

(or type)

table relation

table name relation name

customers

relation schema

row

t-uple

first_name: VARCHAR(64)	last_name: VARCHAR(64)	email: VARCHAR(64)	dob: DATE	since: DATE	customers: VARCHAR(16)	country: VARCHAR(16)
Deborah	Ruiz	druiz0@drupal.org	1984-08-01	2016-10-17	Deborah84	Singapore
Denise	Gonzalez	dgonzalez9@slate.com	1982-12-06	2016-07-26	Denise82	Malaysia
null	Leong	wleong3@shop-pro.jp	null	null	null	Singapore
Deborah	Ruiz	~~druiz0@drupal.org~~	~~1984-08-01~~	~~2016-10-17~~	~~Deborah84~~	~~Singapore~~

number of columns: degree or arity

number of rows: cardinality

Inserting Data From Existing Tables

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

INSERT INTO singapore_customers
SELECT first_name, last_name, email, dob, since, customerid
FROM customers WHERE country='Singapore';

SELECT * FROM singapore_customers;
```

Although SQL allows to insert the results of a query into a table, it is generally an unnecessary and generally bad idea for either permanent or temporary results.

Creating Views

Instead one can create a view. A view is a table defined by a query. it is always up-to-date. Crating a view is s a very good idea.

```
DROP TABLE singapore_customers;
CREATE VIEW singapore_customers AS
SELECT first_name, last_name, email, dob, since, customerid
       FROM customers
       WHERE country='Singapore';
SELECT * FROM singapore_customers;

INSERT INTO customers VALUES ('Hellen', 'Vasquez',
'hvasqueza@nymag.com', '1988-10-31', '2016-06-15',
'Hellen88', 'Singapore');
SELECT * FROM singapore_customers;

DELETE FROM customers WHERE email='hvasqueza@nymag.com';
SELECT * FROM singapore_customers;
```

Deleting and Updating Rows

```
DELETE FROM customers WHERE country='Malaysia';

UPDATE customers
SET country='SG'
WHERE country='Singapore';

INSERT INTO games VALUES ('Skype', '5.1', 15);

UPDATE games
SET price = price * 1.17;

SELECT * FROM games;

DELETE FROM customers;
```

Modifying Tables

```
ALTER TABLE games ADD expiry VARCHAR(10);

SELECT * FROM games;

ALTER TABLE games DROP COLUMN expiry; (not in SQLite)

ALTER TABLE games MODIFY expiry DATE; (not in SQLite)

ALTER TABLE games DROP COLUMN expiry; (not in SQLite)
```

The exact syntax may vary from one system to another. One can also alter domains and constraints.

Dropping Tables

```
DROP TABLE customers;

DROP TABLE games;

DROP TABLE downloads;
```

While "`DELETE`" removes the content of a table and leaves the table empty, "`DROP`" removes the table and its content.

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

SQL Integrity Constraints

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

Distributed and Concurrent Access

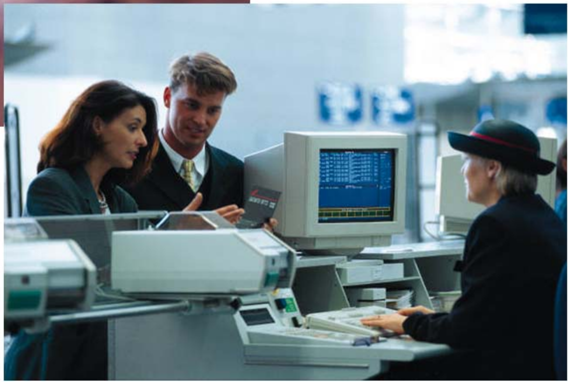
How can data be shared by users and processes that are possibly distributed over a network?

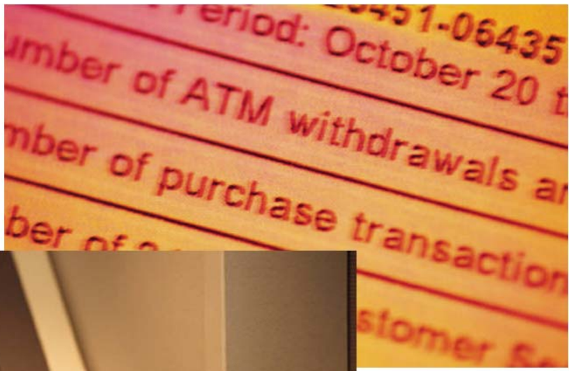
Integrity of Data should be Maintained

How to maintain the integrity of data in spite of possible application, system, or media failures?

Transactions

A transaction is a logical unit of work carried out by a user or an application





Consistent States

A consistent state of the database is a state which complies with the business rules as usually defined by integrity constraints

"students who have not passed cs2102 cannot take cs3223"

If the integrity constraints are violated by a transaction, the transaction is aborted and rolled back, otherwise, it is committed.

ACID Properties

Recovery

- Atomicity: all actions in a transaction happen or none happen (transactions are committed or aborted and rolled back)
- Durability: effects of committed transactions last

Concurrency Control

- Isolation: Transactions can be understood independently from each other (to be safe be serializable!)
- Consistency: If individual transactions leave the application in a consistent state, their concurrent execution should do the same

Integrity Constraint: What Do They Do?

Integrity constraints are checked by the DBMS before a transaction (`BEGIN...END`) modifying the data is committed;

If an integrity constraint is violated, the transaction is aborted and rolled back, the changes are not reflected;

Otherwise the transaction is committed and the changes are effective.

Note: In SQL integrity constraints can be immediate or deferred. You should always use deferred constraints.

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.

```
apps (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.

NULL Values

Every domain (type) has an additional value: the `null` value (read Ramakrishnan). In general the semantics of null could be ambiguous. It could be "unknown", "does not exists", "unknown or does not exists". In SQL it is generally "unknown".

`"something = null"` is unknown (even is "`something`" is "`null`").

`"null IS NULL"` is true.

`"something < null"` is unknown.

`"something > null"` is unknown.

`"10 + null"` is null.

`"0 * null"` is null.

`"COUNT(*)"` counts `NULL` values.
`"COUNT(att)", "AVG(att)", "MAX(att)", "MIN(att)"` eliminate null values.

NULL Values Logic

"`SELECT FROM WHERE condition`" returns results when the condition is true.

P	Q	P AND Q	P OR Q	NOT P
True	True	True	True	False
False	True	False	True	True
Unknown	True	Unknown	True	Unknown
True	False	False	True	False
False	False	False	False	True
Unknown	False	False	Unknown	Unknown
True	Unknown	Unknown	True	False
False	Unknown	False	Unknown	True
Unknown	Unknown	Unknown	Unknown	Unknown

NOT NULL

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

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

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,
```

Column Constraint – NOT NULL

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));
```

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

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.

SQLite: UNIQUE

```
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`".

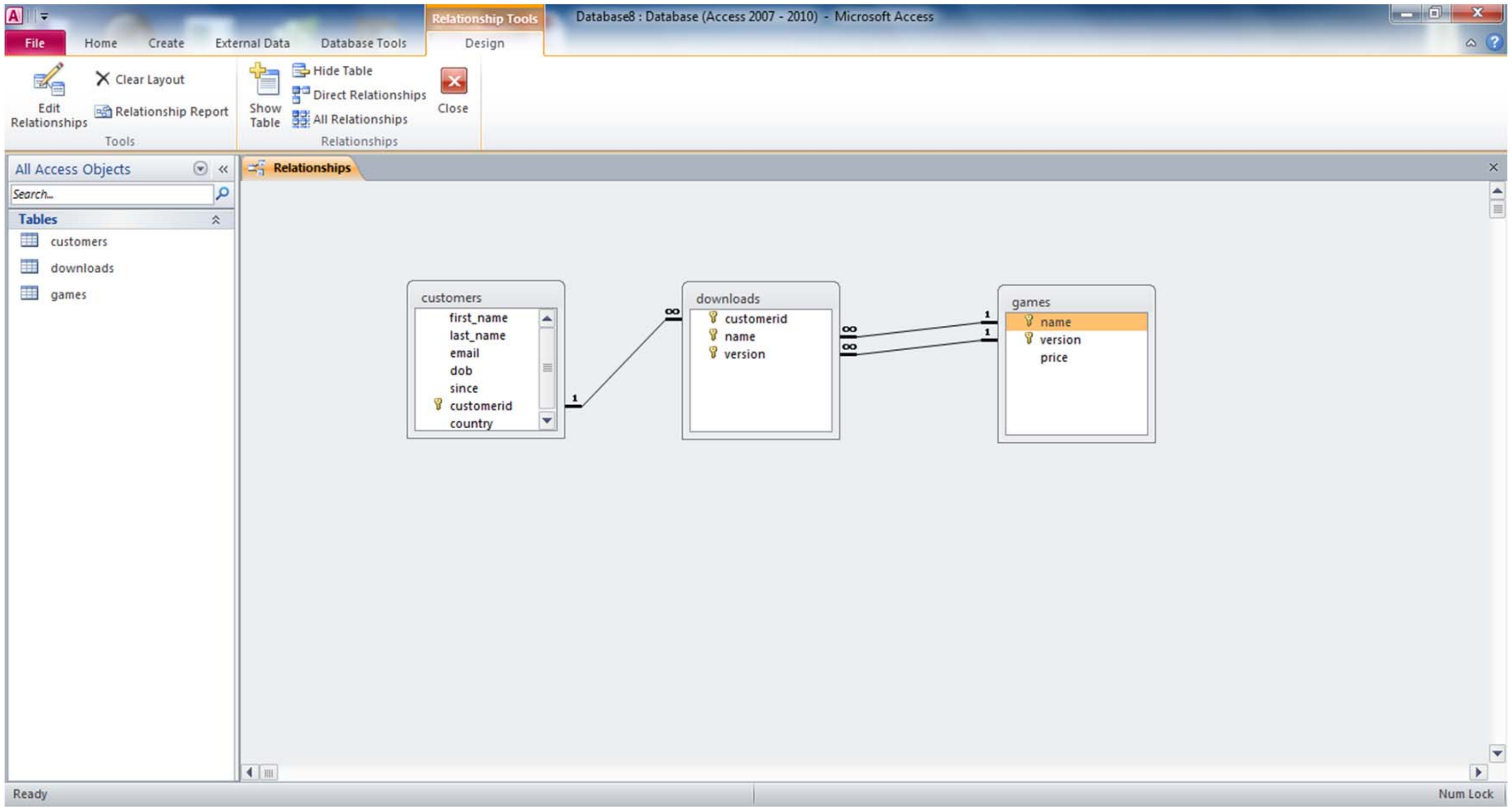
Table Constraint – FOREIGN KEY (referential integrity)

```
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 hal2001 downloads Comfort 5.1

User tom1999 changes his name 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	…
~~tom1999~~ THOM1999	~~tlee@gmail.com~~	~~…~~
john88	al@hotmail.com	…
Walnuts	dcs@nus.edu.sg	…

SQLite: FOREIGN KEY

Let us try:

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

PRAGMA foreign_keys = ON;

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…

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.

Assertions (Not available!)

The SQL standard has provision for constraints over several tables.

```
CREATE ASSERTION email
CHECK( NOT EXISTS (
             SELECT *
             FROM customers u1, customers u2
             WHERE u1.customerid = u2.customerid
             AND u1.email <> u2.email));
```

No database management system implements it yet.

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	…

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 "`customersC.sql`", "`gamesC.sql`" and "`downloadsC.sql`" and identify all the integrity constraints.

Credits

The content of this lecture is based on chapter 2 of the book "Introduction to database Systems"
By
S. Bressan and B. Catania, McGraw Hill publisher

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

steph@nus.edu.sg