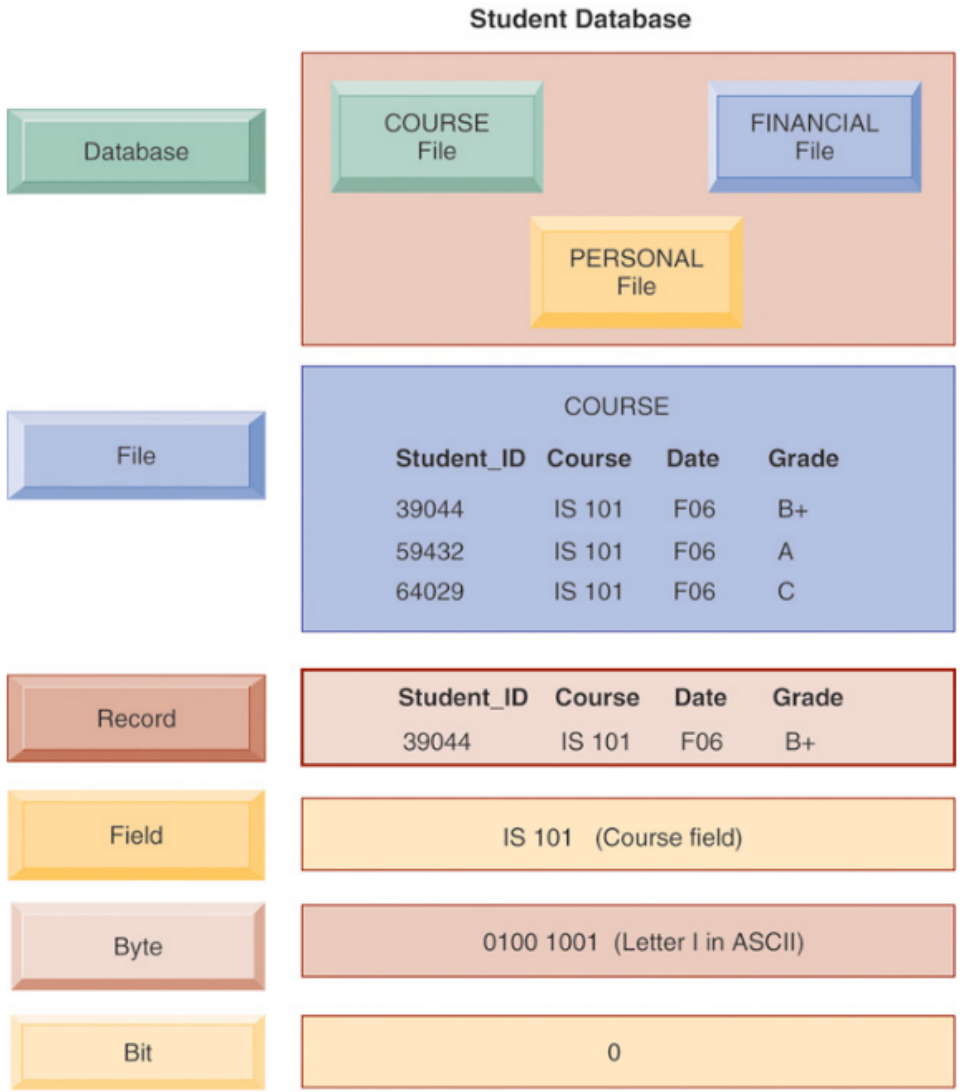
Introduction to Database Systems: CS312 A Small Database System

Oliver Bonham-Carter

18 January 2019

All Types of Data!

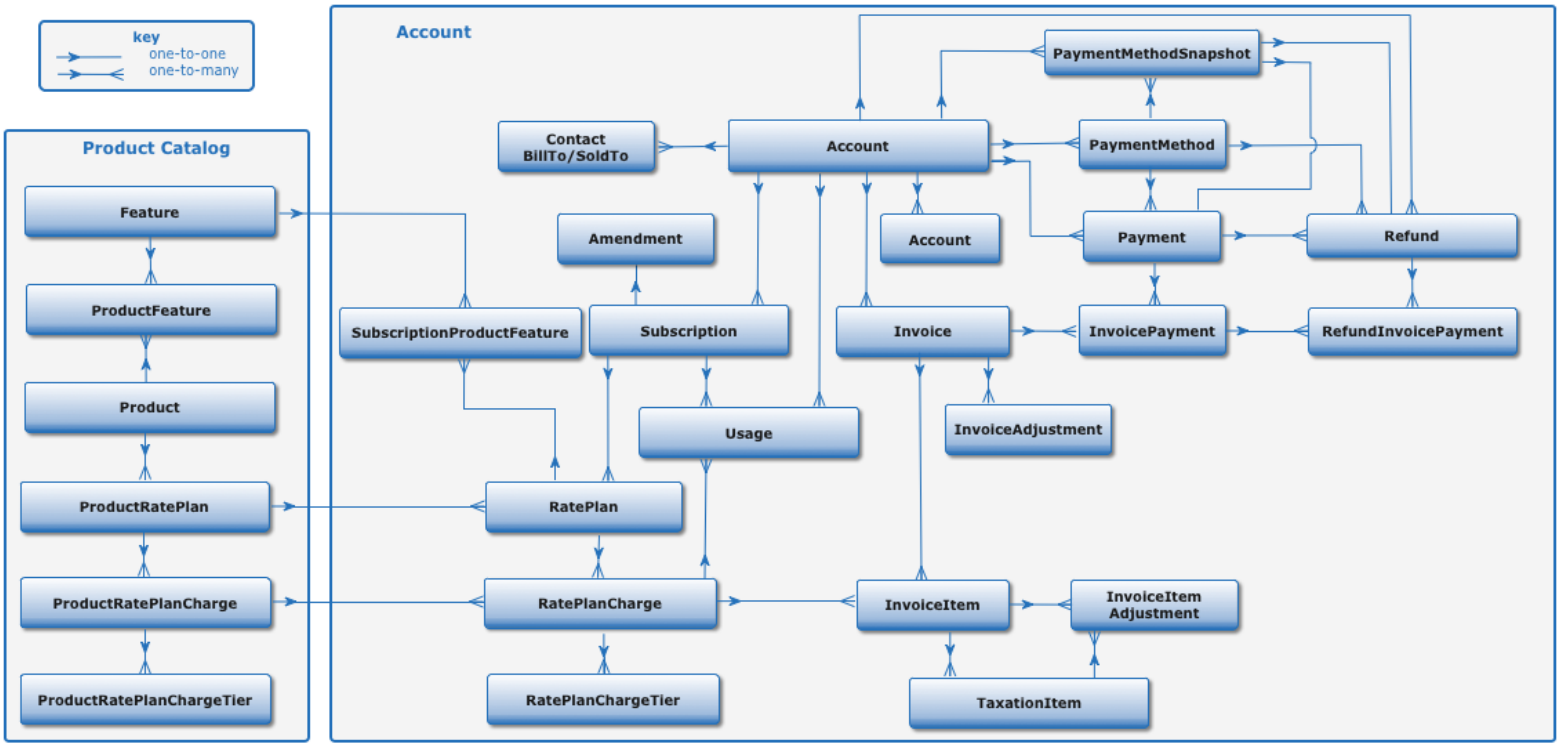


A Database, Simply

ID	Dept	RoomNum
JJ	CS	105
OBC	CS	104
AM	CS	106
GK	CS	108
PL	CS	110
DW	CS	112

- The entire database fits into one table.
- Is the *Department* column unnecessary?

A Database, Not-So-Simply

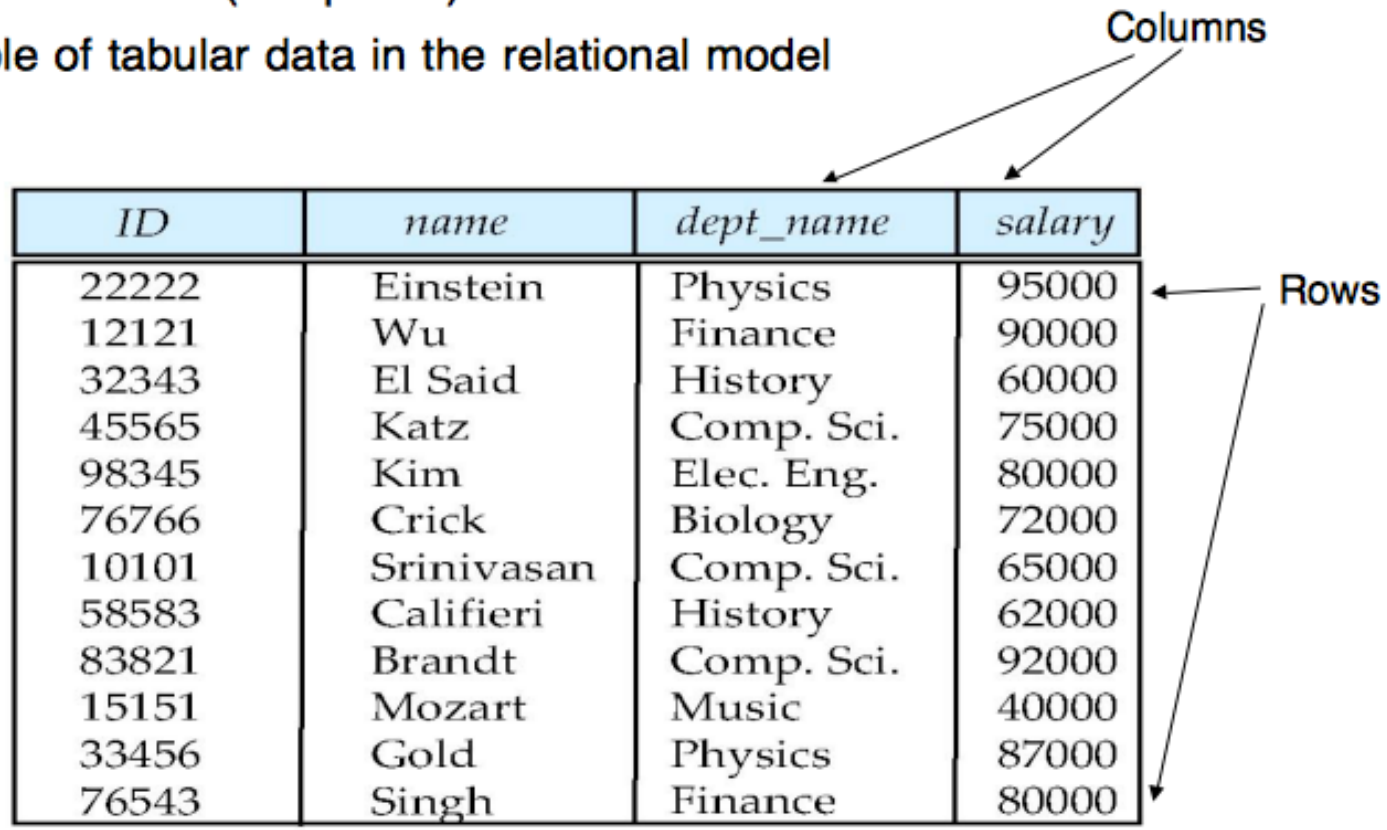


- The entire database is made up of many tables.
- A table must be connected to the others *in some way*.

Relational Models: A single table

Relational model (Chapter 2)

Example of tabular data in the relational model



ID	*name*	*dept_name*	*salary*
22222	Einstein	Physics	95000
12121	Wu	Finance	90000
32343	El Said	History	60000
45565	Katz	Comp. Sci.	75000
98345	Kim	Elec. Eng.	80000
76766	Crick	Biology	72000
10101	Srinivasan	Comp. Sci.	65000
58583	Califieri	History	62000
83821	Brandt	Comp. Sci.	92000
15151	Mozart	Music	40000
33456	Gold	Physics	87000
76543	Singh	Finance	80000

(a) The *instructor* table

- Each field of a row is an "observation"
- Rows are a series of "obserations"
- Each column contains the same type of information for "observation"

Specific information for each table

ID	*name*	*dept_name*	*salary*
22222	Einstein	Physics	95000
12121	Wu	Finance	90000
32343	El Said	History	60000
45565	Katz	Comp. Sci.	75000
98345	Kim	Elec. Eng.	80000
76766	Crick	Biology	72000
10101	Srinivasan	Comp. Sci.	65000
58583	Califieri	History	62000
83821	Brandt	Comp. Sci.	92000
15151	Mozart	Music	40000
33456	Gold	Physics	87000
76543	Singh	Finance	80000

(a) The *instructor* table

dept_name	*building*	*budget*
Comp. Sci.	Taylor	100000
Biology	Watson	90000
Elec. Eng.	Taylor	85000
Music	Packard	80000
Finance	Painter	120000
History	Painter	50000
Physics	Watson	70000

(b) The *department* table

- Two tables containing *specific* types of data

Specific information for each table

ID	Dept	RoomNum
JJ	CS	105
OBC	CS	104
AM	CS	106
GK	CS	108
PL	CS	110
DW	CS	112
MC	GEO	209
RO	GEO	203
SR	GEO	001
SS	GEO	201
KT	GEO	204

ID	Tea	Sandwich
JJ	1	Ruban
OBC	1	PBJ
AM	1	Chicken
GK	1	Chicken
PL	0	Ruban
DW	0	PBJ
MC	1	Ruban
RO	0	PBJ
SR	1	Ruban
SS	1	Ruban
KT	1	Ruban

- Two tables containing *specific* types of data, using the same ID on a row
- Each table organizes non-redundant information, but needs a way to connect a row to the rest of the base (i.e., the common *ID* column serves as a primary key).

We've got it!

- Let's build a small-sized database using SQLite3!!

SQL is...

- Pronounced "ess-que-el" stands for *Structured Query Language*.
- Used to communicate with a database.
- According to ANSI (American National Standards Institute), it is the standard language for relational database management systems.
- The standard computer language for relational database management and data manipulation.
  - Used to query, insert, update and modify data

Command

$sqlite3

You should see this, or similar:

```
SQLite version 3.19.3 2017-06-27 16:48:08
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite>
```

We are going to build this database

ID	Dept	RoomNum
JJ	CS	105
OBC	CS	104
AM	CS	106
GK	CS	108
PL	CS	110
DW	CS	112
MC	GEO	209
RO	GEO	203
SR	GEO	001
SS	GEO	201
KT	GEO	204

- Our database will contain this same ordering of data

Data and its Schema

We need to tell SQLite3 where to contain the data

Data

- Only three columns in our base:
  1. ID: up to four chars in size
  2. Dept: up to four chars
  3. RoomNum: up to 3 chars

- Plenty of space for as many rows as we want:
  1. Limited by memory

Make a General Table

Pseudo code

```
CREATE TABLE table_name (
    column1 datatype,
    column2 datatype,
    column3 datatype
    ...
);
```

For example, working code would look like the following

```
CREATE TABLE People_I_Know (
    PersonID int,
    LastName varchar(255),
    FirstName varchar(255),
    Address varchar(255),
    City varchar(255)
);
```

Schema

Create a table for the DB

Create database called *dept.sqlite3*

```
$sqlite3 dept.sqlite3
```

```
CREATE TABLE department(
    ID varchar(4),
    Dept varchar(4),
    RoomNum varchar(3)
);
```

- We create a table called *department* to contain our data
- In fact, we have created a container in memory for this task
- Note: $attribute_1$ varchar(n)
  - declaration of the size of string (n) contained by entry called, $attribute_1$
- Why are there differing sizes of *varchars* in the code?

After table is created

Add the data

Check that the table has been created

```
sqlite> .tables
department
```

Insert some data as a *tuple*

```
INSERT INTO department VALUES (
    "OBC",
    "CP",
    "104"
);
```

Query everything in the table, department

```
sqlite> select * from department;
OBC|CS|104
```

Exit and save your database

```
.exit
```

Consider this...

Can you add and populate a new database?

Can you populate your base by adding *more* data?

Can you also check that the data was correctly added?