Responsible Data Science

Relational model basics

Prof. Julia Stoyanovich

Center for Data Science &
Computer Science and Engineering
New York University

data RESPONSIBLY

NYU

the relational model

The relational model

- Introduced by Edgar F. Codd in 1970 (Turing award)
- At the heart of relational database management systems (RDBMS)
    - a database consists of a collection of **relations** (tables)
    - **tuples** are stored in table rows
    - **attributes** of tuples are stored in table columns

	A	B	C	D	E	F	G	H
1	UID	sex	race	MarriageSta	DateOfBirth	age	juv_fel_cour	decile_score
2	1	0	1	1	4/18/47	69	0	1
3	2	0	2	1	1/22/82	34	0	3
4	3	0	2	1	5/14/91	24	0	4
5	4	0	2	1	1/21/93	23	0	8
6	5	0	1	2	1/22/73	43	0	1
7	6	0	1	3	8/22/71	44	0	1
8	7	0	3	1	7/23/74	41	0	6
9	8	0	1	2	2/25/73	43	0	4
10	9	0	3	1	6/10/94	21	0	3
11	10	0	3	1	6/1/88	27	0	4
12	11	1	3	2	8/22/78	37	0	1
13	12	0	2	1	12/2/74	41	0	4
14	13	1	3	1	6/14/68	47	0	1
15	14	0	2	1	3/25/85	31	0	3
16	15	0	4	4	1/25/79	37	0	1
17	16	0	2	1	6/22/90	25	0	10
18	17	0	3	1	12/24/84	31	0	5
19	18	0	3	1	1/8/85	31	0	3
20	19	0	2	3	6/28/51	64	0	6
21	20	0	2	1	11/29/94	21	0	9
22	21	0	3	1	8/6/88	27	0	2
23	22	1	3	1	3/22/95	21	0	4
24	23	0	4	1	1/23/92	24	0	4
25	24	0	3	3	1/10/73	43	0	1
26	25	0	1	1	8/24/83	32	0	3
27	26	0	2	1	2/8/89	27	0	3
28	27	1	3	1	9/3/79	36	0	3

The relational model

- Relations are **unordered collections** of tuples
  - conceptually, a relation is a **set** of tuples
  - however, SQL implements a relation as a **multiset** (bag) of tuples
- Why this model?
  - Simple yet powerful. Great for processing very large data sets in bulk

	A	B	C	D	E	F	G	H
1	UID	sex	race	MarriageSta	DateOfBirth	age	juv_fel_cour	decile_score
2	1	0	1	1	4/18/47	69	0	1
3	2	0	2	1	1/22/82	34	0	3
4	3	0	2	1	5/14/91	24	0	4
5	4	0	2	1	1/21/93	23	0	8
6	5	0	1	2	1/22/73	43	0	1
7	6	0	1	3	8/22/71	44	0	1
8	7	0	3	1	7/23/74	41	0	6
9	8	0	1	2	2/25/73	43	0	4
10	9	0	3	1	6/10/94	21	0	3
11	10	0	3	1	6/1/88	27	0	4
12	11	1	3	2	8/22/78	37	0	1
13	12	0	2	1	12/2/74	41	0	4
14	13	1	3	1	6/14/68	47	0	1
15	14	0	2	1	3/25/85	31	0	3
16	15	0	4	4	1/25/79	37	0	1
17	16	0	2	1	6/22/90	25	0	10
18	17	0	3	1	12/24/84	31	0	5
19	18	0	3	1	1/8/85	31	0	3
20	19	0	2	3	6/28/51	64	0	6
21	20	0	2	1	11/29/94	21	0	9
22	21	0	3	1	8/6/88	27	0	2
23	22	1	3	1	3/22/95	21	0	4
24	23	0	4	1	1/23/92	24	0	4
25	24	0	3	3	1/10/73	43	0	1
26	25	0	1	1	8/24/83	32	0	3
27	26	0	2	1	2/8/89	27	0	3
28	27	1	3	1	9/3/79	36	0	3
29	28	0	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]

The relational model

Episodes (<u>season</u>: int, <u>num</u>: int, title: string, viewers: long)

<u>season</u>	<u>num</u>	title	viewers
1	1	Winter is Coming	2.2 M
1	2	The Kingsroad	2.2 M
2	1	The North Remembers	3.9 M
2	2	The Night Lands	3.8 M

- **Relation**: a set or tuples - order doesn’t matter, all tuples are distinct
- **Attribute**: a column in a relation (e.g., season)
- **Domain**: data type of an attribute (e.g., season: int)
- **Tuple**: a row in a relation, e.g., (1, 2, The Kingsroad, 2.2 M)

Schema vs. instances

Relation schema is a description of a relation in terms of relation name, attribute names, attribute datatypes, constraints (e.g., keys). A schema describes **all valid instances** of a relation.

schema Episodes (<u>season</u>: integer, <u>num</u>: integer, title: string, viewers: integer)

instance 1

<u>season</u>	<u>num</u>	title	viewers
1	1	Winter is Coming	2.2 M
1	2	The Kingsroad	2.2 M
2	1	The North Remembers	3.9 M

instance 2

<u>season</u>	<u>num</u>	title	viewers
1	20	Blah, Blah and Blah	0
4	7	Yet Another Title	10 B

instance 3

<u>season</u>	<u>num</u>	title	viewers

Integrity constraints

- Ensure that data adheres to the rules of the application
- Specified **when schema is defined**
- Checked and enforced by the DBMS when relations are modified (tuples added / removed / updated)
- Must **hold on every valid instance** of the database

1. **Domain constraints** - specify valid data types for each attribute, e.g., Students (sid: integer, name: string, gpa: decimal)
2. **Key constraints** - define a unique identifier for each tuple
3. **Referential integrity constraints** - specify links between tuples
4. **Functional dependencies** - show relationships within a table

Key constraints

A set of attributes is a **candidate key** for a relation if:
(1) no two distinct tuples can have the same values for all key attributes (candidate key **uniquely identifies** a tuple), *and*
(2) this is not true for any subset of the key attributes (candidate key **is minimal**)

- If condition (2) is not met, we have a **superkey**
- There may be more than one candidate key for a relation, if so, one is designated as the **primary key**
- All candidate key should be known to property enforce data integrity

Example: name possible candidate keys

Students (sid: integer, login: string, name: string, dob: date)

Key constraints

Example: Students (sid: integer, login: string, name: string, dob: date)

three possible SQL implementations

```
create table Students (
   sid   integer      primary key,
   login varchar(128) unique,
   name  varchar(128),
   dob   date,
   gpa   decimal,
   unique (name, dob) );
```

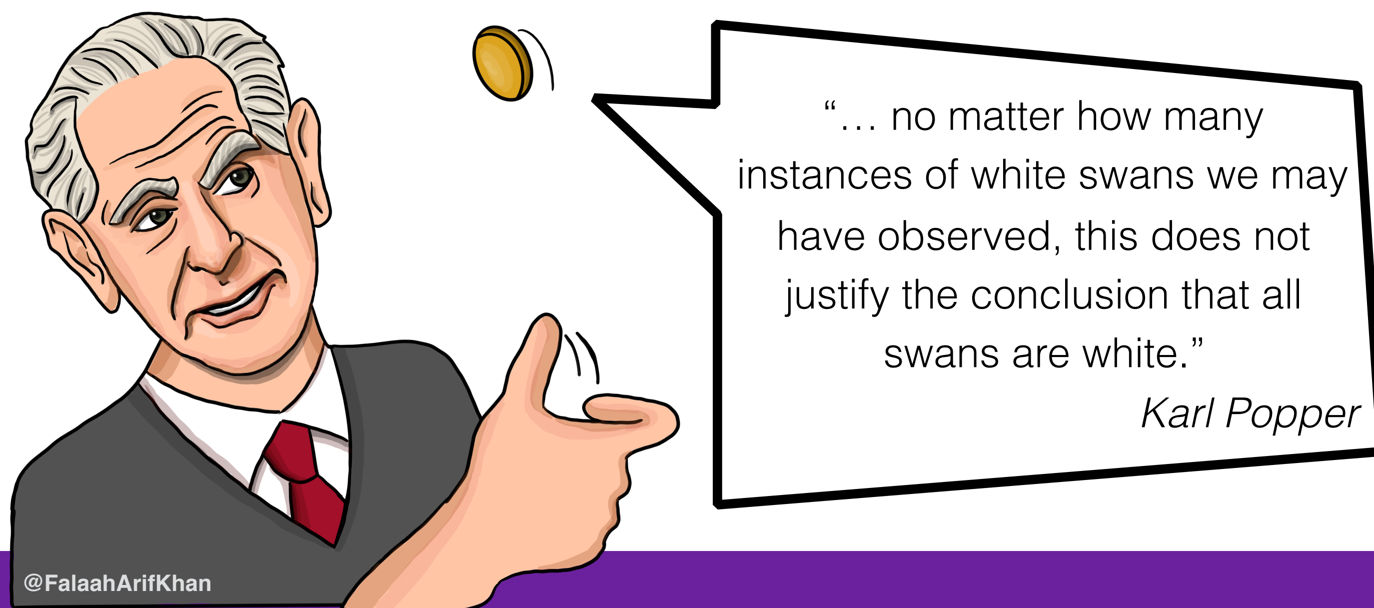
```
create table Students (
   sid   integer      unique,
   login varchar(128) primary key,
   name  varchar(128),
   dob   date,
   gpa   decimal,
   unique (name, dob) );
```

```
create table Students (
   sid   integer       unique,
   login varchar(128)  unique,
   name  varchar(128),
   dob   date,
   gpa   decimal,
   primary key (name, dob) );
```

NB: every relation must have exactly one primary key

DB 101: Where do business rules come from?

- **Business rules are given**: by the client, by the application designer, by your boss, by nature
- Once you know the rules, you create a **relational schema** that encodes them
- We can **never-ever-ever deduce business rules by looking at an instance** of a relation!
- We can sometimes know which rules do not hold, but we cannot be sure which rules do hold



DB 101: Where do business rules come from?

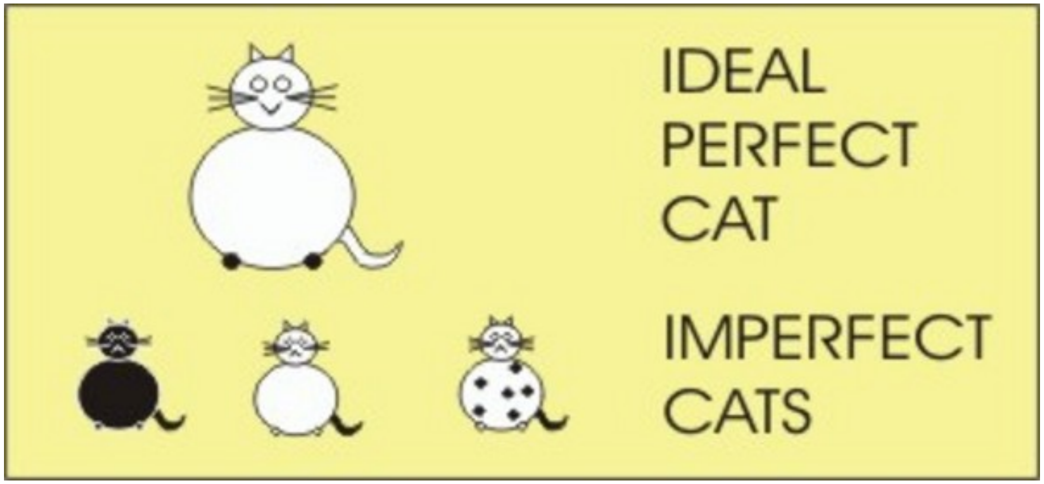
- **Business rules are given**: by the client, by the application designer, by your boss, by nature
- Once you know the rules, you create a **relational schema** that encodes them
- We can **never-ever-ever deduce business rules by looking at an instance** of a relation!
- We can sometimes know which rules do not hold, but we cannot be sure which rules do hold

Employee

id	login	name
1	jim	Jim Morrison
2	amy	Amy Winehouse
3	amy	Amy Pohler
4	raj	Raj Kapoor

1. Which column **is not** a candidate key?
2. Which column(s) **may be** a candidate key?
3. Give 2 create table statements for which this instance is valid.

DB (databases) vs. DS (data science)

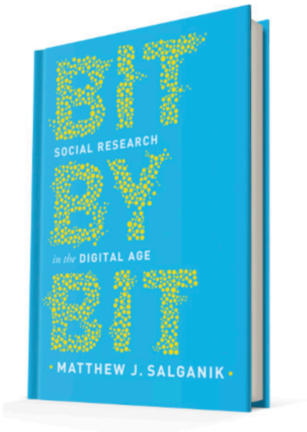


https://midnightmediamusings.wordpress.com/2014/07/01/plato-and-the-theory-of-forms/

- **DB**: start with the schema, admit only data that fits; iterative refinement is possible, and common, but we are still schema-first
- **DS**: start with the data, figure out what schema it fits, or almost fits - reasons of usability, repurposing, low start-up cost

the "right" approach is somewhere between these two, **data profiling aims to bridge** between the two world views / methodologies

Readymade vs. Custom-made



Readymade

Custommade

"Duchamp is best known for his readymades, such as ***Fountain***, where he took ordinary objects and repurposed them as art. Michelangelo, on the other hand, didn't repurpose. When he wanted to create a statue of ***David***, he didn't look for a piece of marble that kind of looked like ***David***: he spent three years laboring to create his masterpiece. ***David*** is not a readymade; it is a custommade."

https://www.bitbybitbook.com/en/1st-ed/introduction/themes/

relational algebra

What is an algebra?

- A system consisting of operators and operands
- We are all familiar with the algebra of arithmetic: operators are + – ×, operands are constants, like 42, or variables, like x
- Expressions are made up of operators, operands, optionally grouped by parentheses, e.g., (x + 3) / (y – 1)

- In relational algebra:
  - operands are variables that stand for relations
  - constants stand for finite relations (think: a particular set of tuples)
  - let's look at operators

Relational algebra operators

1. The usual **set operations**: union $\cup$, intersection $\cap$, set difference $\setminus$, but applied to relations (sets of tuples)

2. Operations that **remove parts of a relation**
   - **selection** removes rows (tuples)
   - **projection** removes columns (attributes)

3. Operations that combine tuples of two relations
   - **Cartesian product** (a. k. a. **cross product**) - pairs up tuples in two relations in all possible ways
   - **join** - selectively pairs up tuples from two relations

4. A **renaming** operation changes relation schema, re-assigning either relation name or names of attributes

Set operations on relations

Definition: Relations R and S are **union-compatible** if their schemas define attributes with the same (or compatible) domains.

Set operations can only be applied to union-compatible relations.

R

id	name	age
1	Ann	18
2	Jane	22

S

id	name	age
1	Ann	18
3	Mike	21
4	Dave	27

$R \cup S$

id	name	age
1	Ann	18
2	Jane	22
3	Mike	21
4	Dave	27

$R \cap S$

id	name	age
1	Ann	18

R / S

id	name	age
2	Jane	22

S / R

id	name	age
3	Mike	21
4	Dave	27

Note: (1, Ann, 18) appears only once in the result of $R \cup S$

Selection

The **selection operator**, applied to relation *R*, produces a new relation with a **subsets of *R*'s tuples**. Tuples in the new relation are those that satisfy some condition *c*.

$$\sigma_C(R)$$

$Episodes$

season	num	title	viewers
1	1	Winter is Coming	2.2 M
1	2	The Kingsroad	2.2 M
2	1	The North Remembers	3.9 M
2	2	The Night Lands	3.8 M

$\sigma_{viewers>3M} Episodes$

season	num	title	viewers
2	1	The North Remembers	3.9 M
2	2	The Night Lands	3.8 M

Note: $\sigma_C(R)$ has at most as many rows as *R*

Projection

The **projection operator**, applied to relation *R*, produces a new relation with a **subsets of *R*'s attributes.**

$$\pi_{A_1, A_2, \ldots, A_n}(R)$$

Episodes

season	num	title	viewers
1	1	Winter is Coming	2.2 M
1	2	The Kingsroad	2.2 M
2	1	The North Remembers	3.9 M
2	2	The Night Lands	3.8 M

$\pi_{season, title} Episodes$

season	title
1	Winter is Coming
1	The Kingsroad
2	The North Remembers
2	The Night Lands

$\pi_{season} Episodes$

season
1
2

Note: $\pi_{A_1, A_2, \ldots, A_n}(R)$ has at most as many rows as *R*

Why not exactly as many?

Cartsian product

The **Cartesian product** (or **cross product**) of two relations *R* and *S* is the set of pairs, formed by choosing the first element from *R* and the second element from *S*.

$$R \times S$$

Characters

name	house
Tyrion	Lannister
Daenerys	Targaryen

Episodes

season	num	title
1	1	Winter is Coming
1	2	The Kingsroad

$Characters \times Episodes$

name	house	season	num	title
Tyrion	Lannister	1	1	Winter is Coming
Tyrion	Lannister	1	2	The Kingsroad
Daenery	Targaryen	1	1	Winter is Coming
Daenery	Targaryen	1	2	The Kingsroad

Note: there are exactly $|R| * |S|$ tuples in $R \times S$

Join

The **join** of two relations *R* and *S* is the set of pairs, formed by choosing the first element from *R* and the second element from *S*, such that the corresponding tuples in *R* and *S* meet some condition *c* .

$R \bowtie_c S$

Characters

name	house
Tyrion	Lannister
Daenerys	Targaryen

Appearances

name	season	num
Jon Snow	2	1
Tyrion	1	1
Tyrion	2	2
Daenerys	1	2

$$Characters \bowtie_{name} Appearances$$

name	house	name	season	num
Tyrion	Lannister	Tyrion	1	1
Tyrion	Lannister	Tyrion	2	2
Daenerys	Targaryen	Daenery	1	2

Note: there are at most $|R| * |S|$ tuples in $R \bowtie_c S$