>>

Relational Design

- Relational Design Theory
- Relational Design and Redundancy
- Database Design (revisited)

^ >>

❖ Relational Design Theory

The aim of studying relational design theory:

- improve understanding of relationships among data
- gain enough formalism to assist practical database design

What we study here:

- basic theory and definition of functional dependencies
- methodology for improving schema designs (normalisation)

Functional dependencies

- describe relationships between attributes within a relation
- have implications for "good" relational schema design

<< ^ >>

❖ Relational Design and Redundancy

A good relational database design:

- must capture *all* necessary attributes/associations
- do this with *minimal* amount of stored information

Minimal stored information ⇒ no redundant data.

In database design, redundancy is generally a "bad thing":

- causes problems maintaining consistency after updates

don't want to have to update it in multiple places, if you forget then they become out of sync

But ... redundancy may give performance improvements

- e.g. avoid a join to collect pieces of data together

<< ^ >>

❖ Relational Design and Redundancy (cont)

Consider the following relation defining bank accounts/branches:

accountNo	balance	customer	branch	address	assets
A-101	500	1313131	Downtown	Brooklyn	9000000
A-102	400	1313131	Perryridge	Horseneck	1700000
A-113	600	9876543	Round Hill	Horseneck	8000000
A-201	900	9876543	Brighton	Brooklyn	7100000
A-215	700	1111111	Mianus	Horseneck	400000
A-222	700	1111111	Redwood	Palo Alto	2100000
A-305	350	1234567	Round Hill	Horseneck	8000000
...	...	...	...	...	...

Careless updating of this data may introduce inconsistencies.

i.e. if the Horseneck road gets renamed, then we have to change that address for every instance of the round hill branch, likewise every time the round hill branch's assets change, we need to update the assets entry for every round hill tuple that exists

<< ^ >>

❖ Relational Design and Redundancy (cont)

If we add $300 to account A-113 ...

accountNo	balance	customer	branch	address	assets
A-101	500	1313131	Downtown	Brooklyn	9000000
A-102	400	1313131	Perryridge	Horseneck	1700000
A-113	900	9876543	Round Hill	Horseneck	8000300
A-201	900	9876543	Brighton	Brooklyn	7100000
A-215	700	1111111	Mianus	Horseneck	400000
A-222	700	1111111	Redwood	Palo Alto	2100000
A-305	350	1234567	Round Hill	Horseneck	8000000
...	...	...	...	...	...

i.e. customer a-113 just deposited 300 dollars into their account (600+300 = 900)
but we only updated the assets on this tuple, not the other one, leading to inconsistency

<< ^ >>

❖ Relational Design and Redundancy (cont)

If we add a new account A-306 at the Round Hill branch ...

accountNo	balance	customer	branch	address	assets
A-101	500	1313131	Downtown	Brooklyn	9000000
A-102	400	1313131	Perryridge	Horseneck	1700000
A-113	900	9876543	Round Hill	Horseneck	8000300
A-201	900	9876543	Brighton	Brooklyn	7100000
A-215	700	1111111	Mianus	Horseneck	400000
A-222	700	1111111	Redwood	Palo Alto	2100000
A-305	350	1234567	Round Hill	Horseneck	8000000
A-306	500	7654321	Round Hill	Horseneck	8000500?
...	...	...	...	...	...

do we add 500 to the ...300 or ...000 version? ideally, all entries should read 8000800

<< ^ >>

❖ Relational Design and Redundancy (cont)

If we close account A-101 ...

accountNo	balance	customer	branch	address	assets
~~A-101~~	~~500~~	~~1313131~~	~~Downtown~~	~~Brooklyn~~	~~9000000~~
A-102	400	1313131	Perryridge	Horseneck	1700000
A-113	900	9876543	Round Hill	Horseneck	8000300
A-201	900	9876543	Brighton	Brooklyn	7100000
A-215	700	1111111	Mianus	Horseneck	400000
A-222	700	1111111	Redwood	Palo Alto	2100000
A-305	350	1234567	Round Hill	Horseneck	8000000
A-306	500	7654321	Round Hill	Horseneck	8000500?
...	...	...	...	...	...

What is the address of the Downtown branch?

we've now lost this information for good because it no longer exists in any of the tuples. If the branches were stored seperately, we'd still have this information after deleting a specific customer.

<< ^ >>

❖ Relational Design and Redundancy (cont)

Insertion anomaly:

- when we insert a new record, we need to check that branch data is consistent with existing tuples

Update anomaly:

- if a branch changes address, we need to update all tuples referring to that branch

Deletion anomaly:

- if we remove information about the last account at a branch, all of the branch information disappears

Insertion/update anomalies can be handled, e.g. by triggers

- but this requires extra DBMS work on every change to the database

and we can't even handle the deletion anomaly at all!

<< ^

❖ Database Design (revisited)

To avoid these kinds of update problems:

- need a schema with "minimal overlap" between tables
- each table contains a "coherent" collection of data values

Such schemas have little/no redundancy

ER → SQL mapping tends to give non-redundant schemas

- but does not guarantee no redundancy

The methods we describe in this section

- can reduce redundancy in schemas ⇒ eliminate update anomalies

Produced: 1 Nov 2020