Design Theory for Relational DBs: Normal Forms

Manos Papagelis

Database Design Theory

- Guides systematic improvements to database schemas
- General idea:
  - Express constraints on the data
  - Use these to decompose the relations
- Ultimately, get a schema that is in a "*normal form*"
  - guarantees certain desirable properties
  - "normal" in the sense of conforming to a standard
- The process of converting a schema to a normal form is called *normalization*

Goal #1: remove redundancy

- Consider this schema

Student Name	Student Email	Course	Instructor
Xiao	xiao@gmail	CSC333	Smith
Xiao	xiao@gmail	CSC444	Brown
Jaspreet	jaspreet@gmail	CSC333	Smith

- What if…
  - Xiao changes email addresses? (*update anomaly*)
  - Xiao drops CSC444? (*deletion anomaly*)
  - Need to create a new course, CSC222 (*insertion anomaly*)

Multiple relations => exponentially worse

Goal #2: expressing constraints

- Consider the following sets of schemas:

Students(utorid, name, email)

vs.

Students(utorid, name)
Emails(utorid, address)

- Consider also:

House(street, city, value, owner, propertyTax)

vs.

House(street, city, value, owner)
TaxRates(city, value, propertyTax)

Dependencies, constraints are domain-dependent

NORMAL FORMS

Motivation for normal forms

- Identify a "good" schema
  - For some definition of "good"
  - Avoid anomalies, redundancy, etc.
- Many normal forms
  - 1st
  - 2nd
  - 3rd
  - Boyce-Codd
  - ... and several more we won't discuss...

BCNF $\subseteq$ 3NF $\subseteq$ 2NF $\subseteq$ 1NF (focus on 3NF/BCNF)

1^{st} normal form (1NF)

- No multi-valued attributes allowed
  - Imagine storing a list/set of things in an attribute

  => Not really even expressible in RA
- Counterexample
  - Course(name, instructor, [student,email]*)
  - Redundancy in non-list attributes

Name	Instructor	Student Name	Student Email
CSCC43	Johnson	Xiao	xiao@gmail
		Jaspreet	jaspreet@utsc
		Mary	mary@utsc
CSCD08	Rosenburg	Jaspreet	jaspreet@utsc

2nd normal form (2NF)

- Non-prime attributes depend on candidate keys
  - Consider non-prime (ie. not part of a key) attribute ‘a’
  - Then ∃FD X s.t. X -> a and X is a candidate key
- Counterexample
  - Movies(<u>title</u>, <u>year</u>, <u>star</u>, studio, studioAddress, salary)
  - FD: title, year -> studio; studio -> studioAddress; star->salary

Title	Year	Star	Studio	StudioAddr	Salary
Star Wars	1977	Hamill	Lucasfilm	1 Lucas Way	$100,000
Star Wars	1977	Ford	Lucasfilm	1 Lucas Way	$100,000
Star Wars	1977	Fisher	Lucasfilm	1 Lucas Way	$100,000
Patriot Games	1992	Ford	Paramount	Cloud 9	$2,000,000
Last Crusade	1989	Ford	Lucasfilm	1 Lucas Way	$1,000,000

3rd normal form (3NF)

- Non-prime attr. depend *only* on candidate keys
  - Consider FD X -> a
  - Either a ∈ X OR X is a superkey OR a is prime (part of a key)

  => No transitive dependencies allowed
- Counterexample:
  - studio -> studioAddr

    (*studioAddr* depends on *studio* which is not a candidate key)

Title	Year	Studio	StudioAddr
Star Wars	1977	Lucasfilm	1 Lucas Way
Patriot Games	1992	Paramount	Cloud 9
Last Crusade	1989	Lucasfilm	1 Lucas Way

Boyce-Codd normal form (BCNF)

- One additional restriction over 3NF
  - All non-trivial FD have superkey LHS
- Counterexample
  - CanadianAddress(<u>street</u>, <u>city</u>, <u>province</u>, postalCode)
  - Candidate keys: {street, postalCode}, {street, city, province}
  - FD: postalCode -> city, province

  - Satisfies 3NF: city, province both non-prime
  - Violates BCNF: postalCode is not a superkey

  => Possible anomalies involving postalCode

Do we care? How often do postal codes change?

Limits of decomposition

- Pick two…
  - Lossless-join
  - Dependency-preservation
  - Anomaly-free
- 3NF
  - Always allows join lossless and dependency preserving
  - May allow some anomalies
- BCNF
  - Always excludes anomalies
  - May give up one of lossless-join or dependency-preserving

Use domain knowledge to choose 3NF vs. BCNF