Relational Design

Relational Design

- Basic design approaches.
- What makes a good design better than a bad design?
- How do we tell we have a "good" design?
- How to we go about creating a good design?

Georgia Tech

Basic Design Approaches

- Bottom-up, a.k.a. synthethis
  - Start with individual attributes and large set of binary relationships among attributes
  - Unpopular
- Top-down, a.k.a. analysis
  - Start with groupings of attributes, e.g., a schema derived from ER-relational mapping
  - Decompose until design properties are met

Relational Design Desiderata

1. The sematics of relation schemas and their attributes should be clear
2. There should be little or no redundant information in tuples
3. The should be few or no NULL values in tuples
4. It should be impossible to generate spurious (invalid) tuples
5. It should be easy to join tables

Design Goal 1: Convey cohesive meaning in relational schemas

Example:
EMP(Ename, Ssn, Bdate, Address, Dno)
DEPT(Dno, Dname, Dmgr_ssn)

- Each EMPT tuple represents a single employee
- Each DEPT tuple represents a single department

Guidlene 2: Minimize Redundancy

Redundant information in schemas:

- wastes storage space, and
- leads to data manipulation anomalies.

One way to think of schemas with redundancy: they are joined tables from well-designed schemas.
Redundancy leads to data manipulation anomalies . . .

Redundancy leads to Insertion Anomalies

Ssn	Ename	Bdate	Addr	Dmanaged	Dno	Dname
123	Alice	1990	ATL	1	1	Research
124	Bob	1991	BOS	NULL	1	Research
125	Cheng	1992	CHS	NULL	1	Research
126	Drhuv	1993	DET	2	2	Engineering
127	Earl	1994	EWR	NULL	2	Engineering

Every time we insert a new employee, we have to repeat the department information.

Redundancy leads to Deletion Anomalies

Ssn	Ename	Bdate	Addr	Dmanaged	Dno	Dname
123	Alice	1990	ATL	1	1	Research
124	Bob	1991	BOS	NULL	1	Research
125	Cheng	1992	CHS	NULL	1	Research
126	Drhuv	1993	DET	2	2	Engineering
127	Earl	1994	EWR	NULL	2	Engineering

If we delete the last member of a department, we lose the information about the department itself. Does it cease to exist?

Redundancy leads to Update (Modification) Anomalies

Ssn	Ename	Bdate	Addr	Dmanaged	Dno	Dname
123	Alice	1990	ATL	1	1	Research
124	Bob	1991	BOS	NULL	1	Research
125	Cheng	1992	CHS	NULL	1	Research
126	Drhuv	1993	DET	2	2	Engineering
127	Earl	1994	EWR	NULL	2	Engineering

If we change the name of the Research department to the "Playing with lasers" department, we have to change multiple tuples.

Design Goal 3: Minimize Nulls in Tuples

Ssn	Ename	Bdate	Addr	Dmanaged	Dno	Dname
123	Alice	1990	ATL	1	1	Research
124	Bob	1991	BOS	NULL	1	Research
125	Cheng	1992	CHS	NULL	1	Research
126	Drhuv	1993	DET	2	2	Engineering
127	Earl	1994	EWR	NULL	2	Engineering

Bad design: Dmanaged has many nulls because most employees aren't managers.

Design Goal 3: Minimize the need for NULL values in tuples

- Nulls don't have certain meaning - could be absent, N/A, false
- Aren't used in joins
- Aren't counted in aggregate functions
- Waste space

We reduce NULLS by normalization using functional dependency theory.

Design Goal 4: Avoid Spurious Tuples

Say we have a relation state r(R) =

student	course	instructor
Narayan	Database	Mark
Narayan	Operating Systems	Ammar
Smith	Database	Navathe
Smith	Operating Systems	Ammar
Smith	Theory	Schulman
Wallace	Database	Mark
Wallace	Operating Systems	Ahamad
Wong	Database	Omiecinski
Zelaya	Database	Navathe

Bad Decomposition

r(R1) =

student	instructor
Narayan	Ammar
Narayan	Mark
Smith	Ammar
Smith	Navathe
Smith	Schulman
Wallace	Ahamad
Wallace	Mark
Wong	Omiecinski
Zelaya	Navathe

r(R2) =

student	course
Narayan	Database
Narayan	Operating Systems
Smith	Database
Smith	Operating Systems
Smith	Theory
Wallace	Database
Wallace	Operating Systems
Wong	Database
Zelaya	Database

We would join on student and end up with ...

Join with Spurious Tuples

student	course	instructor
Narayan	Database	Ammar
Narayan	Database	Mark
Narayan	Operating Systems	Ammar
Narayan	Operating Systems	Mark
Smith	Database	Ammar
Smith	Database	Navathe

... and 13 more tuples, which is way more tuples than the original relation due to spurious tuples, so the join is not non-additive.
Lost the association between Instructor and Course. E.g., Mark does not teach Operating Systems.

Design Goal 5: Design relation schemas for natural joins

Design relation schemas to be naturally joined on attributes that are related by foreign key-primary key relationships.
Acheived by normalization based on functional dependency theory - foreign keys reference primary keys.

Functional Dependencies

A generalization of superkeys.

Given a relation schema R, and subsets of attributes X and Y, the functional dependency

$$X \rightarrow Y$$

Means that for any pair of tuples t_1 and t_2 in $r(R)$

$$\begin{array}{c}\text{if } t_1[X] = t_2[X] \\ \text{then } t_1[Y] = t_2[Y]\end{array}$$

In other words, whenever the attributes on the left side of a functional dependency are the same for two tuples in the relation, the attributes on the right side of the functional dependency will also be equal.

Relations Satisfy FDs

A	B	C	D
a_1	b_1	c_1	d_1
a_1	b_2	c_1	d_2
a_2	b_2	c_2	d_2
a_2	b_2	c_2	d_3
a_3	b_3	c_2	d_4

$A \rightarrow C$ is satisfied because no two tuples with the same A value have different C values.

$C \rightarrow A$ is not satisfied because

$t_4 = (a_2, b_3, c_2, d_3)$ and

$t_5 = (a_3, b_3, c_2, d_4)$

Satisfying vs. Holding

We say that a functional dependency f holds on a relation if it is not legal to create a tuple that does not satisfy f. Alternately, we say that a relation schema (not just a particular state) satisfies a functional dependency.

name	street	city
Alice	Elm	Charlotte
Bob	Peachtree	Atlanta
Charlie	Elm	Charlotte

Here $street \rightarrow city$ is satisifed by this relation state. However, we would not say that the functional dependency holds, or that the relation schema satisfies the functional dependency because we know there can be different cities with the same street names.

Trivial Functional Dependencies

A functional dependency is trivial if it is satisfied by all relations.
Formally, a functional dependency $X \to Y$ is trivial if $Y \subseteq X$
For example:

- $A \to A$
- $AB \to A$
- $AB \to B$

are trivial.
We don't write trivial functional dependencies when we enumerate a set of functional dependencies that hold on a schema for the purposes of normalization or normal form testing.

Normal Forms

A normal form is a set of conditions based on functional dependencies that acts as tests for the "goodness" of the design of a relation schema. Normalization is the process of decomposing existing relation schemas into new relation schemas that satisfy normal forms for the purpose of:

- minimizing redundancy, and
- minimizing insertion, deletion, and update anomalies (we'll learn later)

We cover first, second, third, and Boyce-Codd normal forms in this class (only 3NF for today). Each higher normal form subsumes the normal forms below it, e.g., a 3NF schema is also in 2NF and 1NF. The normal form of a relation schema is the highest normal form it satisfies.

First Normal Form (1NF)

Every attribute value is atomic, which is effectively guaranteed by most RDBMS systems today.
The following relation is not in 1NF:

Dname	Dnumber	Dmgr_ssn	Dlocations
Research	5	333445555	{Bellaire, Sugarland, Houston}
Admin	4	987654321	{Stafford}
HQ	1	888665555	{Houston}

Because Dlocations values are not atomic.

Fixing Non 1NF Schemas

Many ways to fix (see book). Best way is to decompose into two schemas:

Dname	<u>Dnumber</u>	`Dmgr_ssn`
Research	5	333445555
Admin	4	987654321
HQ	1	888665555

<u>Dnumber</u>	<u>Dlocation</u>
5	Bellaire
5	Sugarland
5	Houston
4	Stafford
1	Houston

General Definition of 2NF and 3NF

Definitions in previous lecture based on primary key. General definitions based on all candidate keys.
Remember:

- An attribute is prime if it is part of a candidate key,
- otherwise it is nonprime.

General definition of 2NF: A relation schema R is in 2NF if every nonprime attribute A in R is not partially dependent on any key of R.

A Non-2NF Schema

$LOTS(\underline{Property\_id}, County\_name, Lot\#, Area, Price, Tax\_rate)$

- FD1: Property_id $\rightarrow$ County_name, Lot#, Area, Price, Tax_rate
- FD2: County_name, Lot# $\rightarrow$ Property_id, Area, Price, Tax_rate
- FD3: County_name $\rightarrow$ Tax_rate
- FD4: Area $\rightarrow$ Price

Both Property_id and {County_name, Lot#} are candidate keys. So, by the general definition of 2NF LOTS is not in 2NF due to FD3, i.e., Tax_rate is partially dependent on a candidate key.

Georgia Tech

2NF Decomposition

LOTS(<u>Property_id</u> , County_name, Lot#, Area, Price, Tax_rate)
becomes
LOTS1(<u>Property_id</u> , County_name, Lot#, Area, Price)

- FD1: Property_id $\rightarrow$ County_name, Lot#, Area, Price, Tax_rate
- FD2: County_name, Lot# $\rightarrow$ Property_id, Area, Price, Tax_rate
- FD4: Area $\rightarrow$ Price

LOTS2(<u>County_name</u> , Tax_rate)

- FD3: County_name $\rightarrow$ Tax_rate

General Definition of 3NF

A relation schema R is in 3NF if, whenever a nontrivial functional dependency $X \rightarrow A$ holds in R, either
(a) X is a superkey of R, or (b) A is a prime attribute of R.
LOTS1(Property_id , County_name, Lot#, Area, Price)

- FD1: Property_id $\rightarrow$ County_name, Lot#, Area, Price, Tax_rate
- FD2: County_name, Lot# $\rightarrow$ Property_id, Area, Price, Tax_rate
- FD4: Area $\rightarrow$ Price

not in 3NF due to FD4. Area is not a superkey and Price is not a prime attribute. Note that Price is transitively dependent on each candidate key.

3NF Decomposition

LOTS1(Property_id , County_name, Lot#, Area, Price)
becomes
LOTS1A(Property_id , County_name, Lot#, Area)

- FD1: Property_id $\rightarrow$ County_name, Lot#, Area, Price, Tax_rate
- FD2: County_name, Lot# $\rightarrow$ Property_id, Area, Price, Tax_rate

and
LOTS1B(Area , Price)

- FD4: Area $\rightarrow$ Price

Straight to 3NF

Though we present a progression through 2NF to 3NF for historical reasons, it's not necessary. Given our origial LOTS
LOTS(Property_id , County_name, Lot#, Area, Price, Tax_rate)

- FD1: Property_id → County_name, Lot#, Area, Price, Tax_rate
- FD2: County_name, Lot# → Property_id, Area, Price, Tax_rate
- FD3: County_name → Tax_rate
- FD4: Area → Price

We see that FD3 and FD4 are problem FDs because neither County_name nor Area is a superkey.

Decomposition Straight to 3NF

So we can decompose
LOTS(<u>Property_id</u> , County_name, Lot#, Area, Price, Tax_rate)
directly into:
LOTS1A(<u>Property_id</u> , County_name, Lot#, Area)

- FD1: Property_id $\rightarrow$ County_name, Lot#, Area
- FD2: County_name, Lot# $\rightarrow$ Property_id, Area

LOTS1B(<u>Area</u> , Price)

- FD4: Area $\rightarrow$ Price

LOTS2(<u>County_name</u> , Tax_rate)

- FD3: County_name $\rightarrow$ Tax_rate

Observations of General 3NF Tests

Two types of problematic FDs:

- A nonprime attribute determines another nonprime attribute, giving rise to a transitive dependency on a key.
- Some subset of a key determines a nonprime attribute, giving rise to a partial dependencey on a key which violates 2NF.

Boyce-Codd Normal Form (BCNF)

A relation schema R is in BCNF if whenever a nontrivial functional dependency $X \to A$ holds in R, then X is a superkey of R
Note that this is the same as 3NF except that it doesn't allow any attributes (even prime attributes) to be determined by non-keys.
General non-BCNF pattern: given $R(A, B, C)$
and FDs

- $AB \to C$
- $C \to B$

R is in 3NF but not BCNF due to the FD $C \to B$.

BCNF Example 1

Say we add FD5 to LOTS1A(Property_id , County_name, Lot#, Area)

- FD1: Property_id → County_name, Lot#, Area
- FD2: County_name, Lot# → Property_id, Area
- FD5: Area → County_name

And say that Fulton county lots are restriced to 1.1, 1.2, . . . , 2.0 acres and DeKalb county lots are restricted to 0.5, 0.6, . . . , 1.0 acres. LOTS1A will have a great deal of redundancy. BCNF doesn't allow this schema because of FD5: Area is not a superkey.

BCNF Example 1 Decomposition

LOTS1A(Property_id , County_name, Lot#, Area)
becomes
LOTS1AX(Property_id , Area, Lot#)

- FD1: Property_id $\rightarrow$ County_name, Lot#, Area

and
LOTS1AY(Area , County_name)

- FD5: Area $\rightarrow$ County_name

Note that FD2 is lost because its attributes are no longer in the same relation schema. In general, FDs may not be preservable in BCNF decompositions.

BCNF Example 2

Given TEACH(Student, Course, Instructor) and

- FD1: {Student, Course} $\rightarrow$ Instructor
- FD2: Instructor $\rightarrow$ Course.

FD2 violates BCNF. There are three possible BCNF decompositions:

1. R1(_Student_, Instructor) and R2(_Student_, Course)
2. R1(_Instructor_, Course) and R2(_Student_, Course)
3. R1(_Instructor_, Course) and R2(_Instructor_, Student)

All three decompositions lose FD1. Which decompositions are good?

Desirable Properties of Decompositions

A decomposition of R into R_1 and R_2 must preserve attributes, that is, $R = R_1 \cup R_2$. We'd also like:

1. Dependency preservation, and
2. Non-additive (lossless) joins.

Dependencies can be preserved in all 3NF decompositions, but not in all BCNF decompositions. In all decompositions we must have non-additive join property.

In the next lecture we'll learn more theory which enables us to test these conditions.