Design Theory and Normalization (part 2)

CSC343
Mark Kazakevich
(with slides from Diane Horton, Jeff Ullman)

Last time

- We learned about a set of constraints that we can put on a relation called **functional dependencies**
- We learned about how to test if an FD follows from a set of FDs using the **closure test**
- Learned how to **project** FDs onto a subset of attributes
- Learned how to find a **minimal basis**, or a set of equivalent non-redundant FDs

Today

- How do we use these tools to make **better schemas?**

Anomalies

- When we try to represent too much information in a relation, we run into issues known as ***anomalies***
- **Three** main types:
  - *Redundancy*
    - Unnecessary repetition of information
  - *Update anomalies*
    - Updating a tuple creates inconsistent data
  - *Deletion anomalies*
    - Removing a tuple results in unwanted loss of information

Bad Tables

- What is bad about this table?

title	*year*	*length*	*genre*	*studioName*	*starName*
Star Wars	1977	124	SciFi	Fox	Carrie Fisher
Star Wars	1977	124	SciFi	Fox	Mark Hamill
Star Wars	1977	124	SciFi	Fox	Harrison Ford
Gone With the Wind	1939	231	drama	MGM	Vivien Leigh
Wayne's World	1992	95	comedy	Paramount	Dana Carvey
Wayne's World	1992	95	comedy	Paramount	Mike Meyers

- It has <u>all</u> the anomalies

- Redundancy
  - Lots of duplicate information for the `Star Wars` tuples
- Update
  - Changing the genre for `Star Wars` in one tuple requires updating all `Star Wars` tuples
- Deletion
  - Removing `Vivien Leigh` as a star can remove “`Gone With the Wind`” entirely

title	*year*	*length*	*genre*	*studioName*	*starName*
Star Wars	1977	124	SciFi	Fox	Carrie Fisher
Star Wars	1977	124	SciFi	Fox	Mark Hamill
Star Wars	1977	124	SciFi	Fox	Harrison Ford
Gone With the Wind	1939	231	drama	MGM	Vivien Leigh
Wayne's World	1992	95	comedy	Paramount	Dana Carvey
Wayne's World	1992	95	comedy	Paramount	Mike Meyers

Recall: Database ‘Design’ Theory

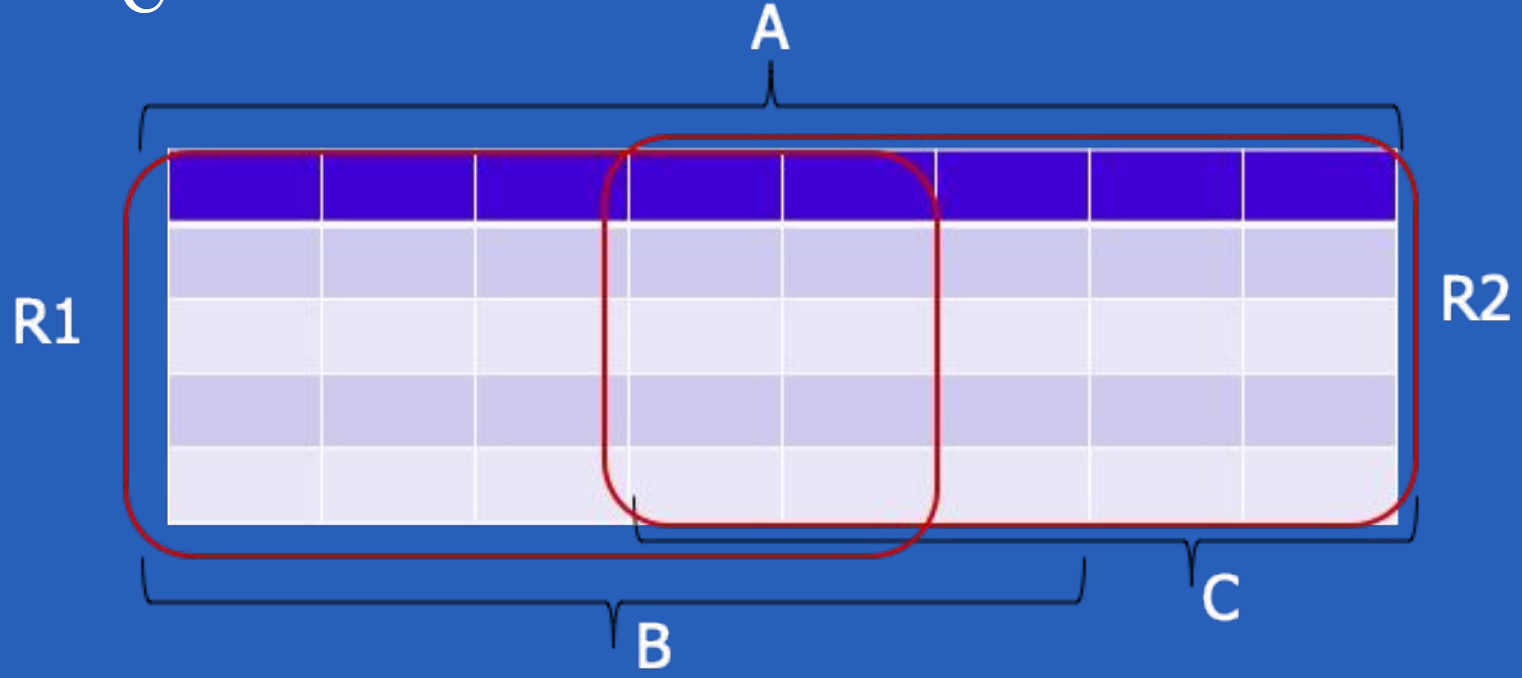
- **General idea:**
  - Express *constraints* on the relationships between attributes
  - <u>Use these constraints to ***decompose*** the relations</u>
    - Create smaller, ‘better’ relations

Decomposing a relation

$$R1 = \pi_{B}(R)$$

$$R2 = \pi_{C}(R)$$

Here we decompose into two relations, but often we decompose into more.



But *which* decomposition

- Decomposition can definitely improve a schema.
- But which decomposition?
  - There are many possibilities.
- And how can we be sure a new schema doesn't exhibit other anomalies?

Boyce-Codd Normal Form (BCNF)

We say a relation R is in BCNF if:

- For every nontrivial FD X->Y that holds in R, X is a superkey.

Remember:

- nontrivial means Y is not contained in X.
- a superkey doesn't have to be minimal.
  - i.e., the left side of every nontrivial FD must contain a key.

Intuition

- In other words, BCNF requires that:
    - Only things that functionally determine **everything**
      can functionally determine **anything**.
- Why is the BCNF property valuable?

BCNF Decomposition

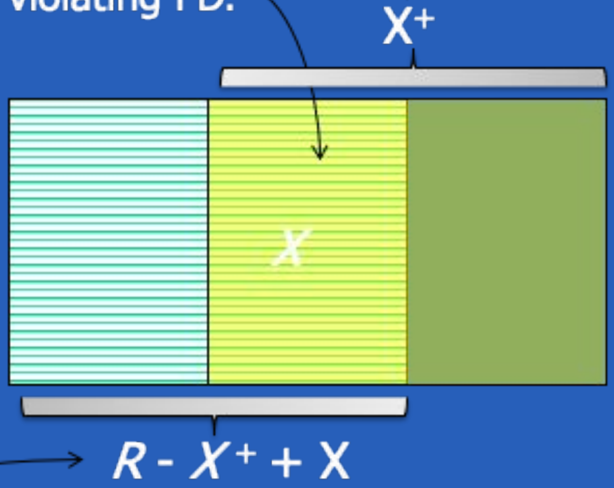
- R is a relation; F is a set of FDs.
- Return the BCNF decomposition of R, given these FDs.

```
BCNF_decomp(R, F):
If an FD X -> Y in F violates BCNF:
    Compute X+.
    Replace R by two relations with schemas:
       R1 = X+
       R2 = R – (X+ – X)
    Project the FD’s F onto R1 and R2.
    Recursively decompose R1 and R2 into BCNF.
```

Decomposition Picture

1) Start with the LHS
of the violating FD.

2) Close the LHS
to get one new relation



3) Everything except the new stuff is the other new relation.
X is in both new relations to make a connection between them.

Notes on decomposition

- If more than one FD violates BCNF, you may decompose based on any one of them.
    - So there may be *multiple* results possible.
- The new relations we create may not be in BCNF. We must recurse.
    - We only keep the relations at the "leaves".

Speed-up

- When projecting FDs onto a new relation, check each new FD:
  - Does the new relation violate BCNF because of this FD?
- If so, abort the projection.
  - You are about to discard this relation anyway (and decompose further).

Decomposition Properties

What we want from a decomposition

1. **No anomalies.**
2. **Lossless Join**: It should be possible to
   - project the original relations onto the decomposed schema
   - then reconstruct the original by joining. We should get back exactly the original tuples.
3. **Dependency Preservation**: All the original FD's should be satisfied.

What BCNF decomposition offers

1. **No anomalies** : ✓
   (Due to no redundancy)
2. **Lossless Join** : ✓
   (textbook section 3.4.1 argues this)
3. **Dependency Preservation** : ✗

What is a 'lossy' join?

- For **any** decomposition, it is the case that:
  - $r \subseteq r_1 \bowtie \ldots \bowtie r_n$
  - i.e., we will get back every tuple.
- But it may not be the case that:
  - $r \supseteq r_1 \bowtie \ldots \bowtie r_n$
  - i.e., we can get spurious tuples.

The BCNF ***property*** does not guarantee lossless join

- If you use the BCNF decomposition algorithm, a lossless join is *guaranteed*.
- If you generate a decomposition some other way
  - you have to check to make sure you have a lossless join
  - **even if your schema satisfies BCNF!**
- We'll learn an algorithm for this check later.

Preservation of dependencies

- BCNF decomposition *does not* guarantee preservation of dependencies.
- i.e., in the schema that results, it may be possible to create an instance that:
  - satisfies all the FDs in the final schema,
  - but violates one of the original FDs.
- Why? Because the algorithm goes too far — breaks relations down too much.

3NF is less strict than BCNF

- **3rd Normal Form (3NF)** modifies the BCNF condition to be less strict.
- An attribute is ***prime*** if it is a member of any key
- `X -> A` satisfies 3NF iff
  - X is a superkey **or** A is prime.
- i.e., it's ok if X is not a superkey as long as A is prime.

3NF ‘synthesis’

- BCNF *decomposition*
  - starts from a large relation and gets schema by decomposing into smaller ones
- 3NF ***synthesis***
  - Builds up schema by constructing relations from FDs

3NF Synthesis

- F is a set of FDs; L is a set of attributes.
- Synthesize and return a schema in 3rd Normal Form

```
3NF_synthesis(F, L):
    Construct a minimal basis M for F.
    For each FD X->Y in M:
        Define a new relation with
        schema X U Y.
    If no relation is a superkey for L:
        Add a relation whose schema is
        some key.
```

3NF synthesis doesn't "go too far"

- BCNF decomposition doesn't stop decomposing until in all relations:
  - if X -> A then X is a superkey.
- 3NF generates relations where:
  - X -> A and yet X is not a superkey, but A is at least prime.

How can we get anomalies with 3NF?

- 3NF synthesis guarantees that the resulting schema will be in 3rd normal form.
- This allows FDs with a **non-superkey** on the LHS.
- This allows **redundancy**, and thus anomalies.

How do we know...?

… that the 3NF synthesis algorithm guarantees:

- **3NF**: A property of minimal bases [see the textbook for more]
- **Preservation of dependencies**: Each FD from a minimal basis is contained in a relation, thus preserved.
- **Lossless join**: We'll return to this once we know how to test for lossless join.

3NF synthesis doesn't "go too far"

1. *No anomalies* : ✗
2. *Lossless Join* : ✓
3. *Dependency Preservation* : ✓

- Neither BCNF nor 3NF can guarantee all three! We must be satisfied with 2 of 3.
- Decompose too far ⇒ can't always enforce original FDs (BCNF)
- Not far enough ⇒ can have redundancy (3NF)
- We consider a schema "good" if it is in either BCNF or 3NF

Back to **Lossless** Join

- If we project R onto R1, R2, ..., Rk , can we recover R by rejoining?
- We will get all of R.
  - Any tuple in R can be recovered from its projected fragments. This is guaranteed.
- But will we get only R?
  - Can we get a tuple we *didn't have* in R? This part we must check.

Aside: when we don't need to test for lossless Join

- Both BCNF decomposition and 3NF synthesis guarantee lossless join.
- So we don't need to test for lossless join if the schema was generated via BCNF decomposition or 3NF synthesis.
- But merely satisfying BCNF or 3NF does not guarantee a lossless join!

The **Chase Test**

- Suppose tuple t appears in the join.
- Then t is the join of projections of some tuples of R, one for each R_i of the decomposition.
- Can we use the given FD's to show that one of these tuples must be t ?

Setup for the Chase Test

- Start by assuming $t = abc\ldots$.
- For each i, there is a tuple s_i of R that has $a, b, c, \ldots$ in the attributes of R_i.
- s_i can have any values in other attributes.
- We'll use the same letter as in t, but with a subscript, for these components.

The algorithm

1. If two rows agree in the left side of a FD, make their right sides agree too.
2. Always replace a subscripted symbol by the corresponding unsubscripted one, if possible.
3. If we ever get a completely unsubscripted row, we know any tuple in the project-join is in the original (*i.e.*, the join is lossless).
4. Otherwise, the final tableau is a counterexample (*i.e.*, the join is lossy).

Practise!

- Additional exercises will be posted on Quercus to help you get better at these techniques