Carnegie Mellon Univ.
Dept. of Computer Science
15-415/615 - DB Applications

C. Faloutsos – A. Pavlo
Lecture#17: Schema Refinement & Normalization

Correction: Implied FDs

Relational Model: Keys

Administrivia

• HW6 is due Tuesday March 24th.

Product(name, color, category, dept, price)

<table>
<thead>
<tr>
<th>name</th>
<th>color</th>
<th>category</th>
<th>dept</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>Green</td>
<td>Gadget</td>
<td>Toys</td>
<td>9.99</td>
</tr>
<tr>
<td>Widget</td>
<td>Black</td>
<td>Gadget</td>
<td>Toys</td>
<td>49.99</td>
</tr>
<tr>
<td>Gizmo</td>
<td>Green</td>
<td>Squirrels</td>
<td>Garden</td>
<td>19.99</td>
</tr>
</tbody>
</table>

Provided FDs
name → color
category → dept
color, category → price

Implied FDs
name, category → price

color, category → dept

• Super Key:
  – A set of attributes that uniquely identifies a tuple.

• Candidate Key:
  – A *minimal* set of attributes that uniquely identifies a tuple.

• Primary Key:
  – Usually just the candidate key.
Correction: Super Key Example

Provided FDs

Provided FDs

Implied FDs

Implied FDs

Example

Loans(bname, bcity, assets, cname, loanId, amt)

Redundancy Problems

Today’s Class

• The dangers of bad database design
• Decomposition Goals
• Normal Forms
• Relational Model vs. NoSQL

Tuple meaning: Christos has a loan (L-17) for $1000 with DJ Snake taken out of the Downtown branch in Pittsburgh, which has assets of $9M.
Decomposition

- Split a single relation $R$ into a set of relations $\{R_1, \ldots, R_n\}$.
- Not all decompositions are “good”

Goals of Decomposition

- **Loseless Joins** (chpt. 19.5.1)
  - Want to be able to reconstruct original relation by joining smaller ones using a natural join.
- **Dependency Preservation** (chpt. 19.5.2)
  - Want to minimize the cost of global integrity constraints based on FD’s.
- **Redundancy Avoidance**
  - Avoid unnecessary data duplication.

Lossless Decomposition

- **Main Idea:** Intersecting attributes must form a super key for one of the resulting smaller relations.

<table>
<thead>
<tr>
<th>$R_1$</th>
<th>$R_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>A</td>
</tr>
<tr>
<td>...</td>
<td>A</td>
</tr>
</tbody>
</table>

$|R_1|=n$

$|R_2|\leq n$

A not a key for $R_1$

A is a key for $R_2$ and Cardinality with $R_1$ is $n:1$

Loans(bname, bcity, assets, cname, loanId, amt)

Functional Dependencies:

- $\text{bname} \rightarrow \text{bcity}, \text{assets}$
- $\text{loanId} \rightarrow \text{amt}, \text{bname}$
Lossless Decomposition

R1(bname, bcity, assets, cname)  R2(cname, loanId, amt)

<table>
<thead>
<tr>
<th>bname</th>
<th>bcity</th>
<th>assets</th>
<th>cname</th>
<th>loanId</th>
<th>amt</th>
</tr>
</thead>
<tbody>
<tr>
<td>Downtown</td>
<td>Pittsburgh</td>
<td>$9M</td>
<td>Christos</td>
<td>L-17</td>
<td>$1000</td>
</tr>
<tr>
<td>Downtown</td>
<td>Pittsburgh</td>
<td>$9M</td>
<td>Obama</td>
<td>L-17</td>
<td>$2000</td>
</tr>
<tr>
<td>Downtown</td>
<td>Pittsburgh</td>
<td>$9M</td>
<td>Obama</td>
<td>L-23</td>
<td>$2000</td>
</tr>
<tr>
<td>Compton</td>
<td>Los Angeles</td>
<td>$2M</td>
<td>Christos</td>
<td>L-17</td>
<td>$1000</td>
</tr>
<tr>
<td>Compton</td>
<td>Los Angeles</td>
<td>$2M</td>
<td>Christos</td>
<td>L-93</td>
<td>$500</td>
</tr>
<tr>
<td>Downtown</td>
<td>Pittsburgh</td>
<td>$9M</td>
<td>DJ Snake</td>
<td>L-17</td>
<td>$1000</td>
</tr>
</tbody>
</table>

Cannot recover original table with a join!

Lossless Decomposition

Loans(bname, bcity, assets, cname, loanId, amt)

R1(bname, bcity, assets, cname)  R2(cname, loanId, amt)

- This is a **bad** decomposition because it causes a lossy join:
  - The ⨝ adds meaningless tuples.
  - By adding noise, have lost meaningful information as a result of the decomposition.

Functional Dependencies:

- bname → bcity, assets
- loanId → amt, bname

More garbage data!
This one is **lossy** too because $R_1 \bowtie R_2$ has 7 tuples where as Loans originally had 4.

Functional Dependencies:
- $bname \rightarrow bcity, assets$
- $loanId \rightarrow amt, bname$

### Lossless Decomposition

- **R1**($bname, assets, cname, loanId$)
- **R2**($loanId, bcity, amt$)

This one is **lossless**! $R_1 \bowtie R_2$ has 4 tuples

A decomposition of $R$ into $R_1 \cup R_2$ is loseless iff:
- $R_1 \cap R_2 \rightarrow R_1$ or $R_1 \cap R_2 \rightarrow R_2$
- Intersecting attributes must form a super key for one of the resulting smaller relations
Dependency Preservation

- **Main Idea:** Original FDs cannot span multiple tables.
- Why does this matter?
  - It would be expensive to check (assuming that our DBMS supports ASSERTIONS).

### Functional Dependencies:

- \( bname \rightarrow bcity, \text{assets} \)
- \( \text{loanId} \rightarrow \text{amt}, \text{bname} \)

### CREATE ASSERTION

```
CREATE ASSERTION bname-bcity
CHECK (NOT EXISTS
  (SELECT *
  FROM R1 AS x1, R2 AS y1,
  R1 AS x2, R2 AS y2
  WHERE x1.loanId = y1.loanId
  AND x2.loanId = y2.loanId
  AND x1.loanId = x2.loanId
  AND x1.bname = x2.bname
  AND y1.bcity <> y2.bcity))
```

### Loans Table

<table>
<thead>
<tr>
<th>bname</th>
<th>bcity</th>
<th>assets</th>
<th>cname</th>
<th>loanId</th>
<th>amt</th>
</tr>
</thead>
<tbody>
<tr>
<td>Downtown</td>
<td>Pittsburgh</td>
<td>$9M</td>
<td>Christos</td>
<td>L-17</td>
<td>$1000</td>
</tr>
<tr>
<td>Downtown</td>
<td>Pittsburgh</td>
<td>$9M</td>
<td>Obama</td>
<td>L-23</td>
<td>$2000</td>
</tr>
<tr>
<td>Compton</td>
<td>Los Angeles</td>
<td>$2M</td>
<td>Christos</td>
<td>L-93</td>
<td>$500</td>
</tr>
<tr>
<td>Downtown</td>
<td>Pittsburgh</td>
<td>$9M</td>
<td>DJ Snake</td>
<td>L-17</td>
<td>$1000</td>
</tr>
</tbody>
</table>

### R1 and R2 Relations

<table>
<thead>
<tr>
<th>bname</th>
<th>assets</th>
<th>cname</th>
<th>loanId</th>
</tr>
</thead>
<tbody>
<tr>
<td>Downtown</td>
<td>$9M</td>
<td>Christos</td>
<td>L-17</td>
</tr>
<tr>
<td>Downtown</td>
<td>$9M</td>
<td>Obama</td>
<td>L-23</td>
</tr>
<tr>
<td>Compton</td>
<td>$2M</td>
<td>Christos</td>
<td>L-93</td>
</tr>
<tr>
<td>Downtown</td>
<td>$9M</td>
<td>DJ Snake</td>
<td>L-17</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>loanId</th>
<th>bcity</th>
<th>amt</th>
</tr>
</thead>
<tbody>
<tr>
<td>L-17</td>
<td>Pittsburgh</td>
<td>$1000</td>
</tr>
<tr>
<td>L-23</td>
<td>Pittsburgh</td>
<td>$2000</td>
</tr>
<tr>
<td>L-93</td>
<td>Los Angeles</td>
<td>$500</td>
</tr>
</tbody>
</table>
Dependency Preservation

To ensure that FD checking is efficient, only a single relation should be examined for each FD.

1. \( R_1(bname, assets, cname, loanId) \)
2. \( R_2(loanId, bcity, amt) \)

Loans(bname, bcity, assets, cname, loanId, amt)

Dependency Preservation

• To test whether the decomposition \( R = \{R_1, \ldots, R_n\} \) preserves \( R \)’s FD set \( F \):
  1. Compute \( F^+ \)
  2. Compute \( G \) as the union of the set of FDs in \( F^+ \) that are covered by \( \{R_1, \ldots, R_n\} \)
  3. Compute \( G^+ \)
  4. If \( F^+ = G^+ \), then \( \{R_1, \ldots, R_n\} \) is Dependency Preserving

Is \( R = \{R_1, R_2\} \) dependency preserving?

1. \( F^+ = \{A \rightarrow B, AB \rightarrow D, C \rightarrow D\} \)
2. \( G = \{A \rightarrow B\} \cup \{C \rightarrow D\} \)
3. \( G^+ = \{A \rightarrow B, C \rightarrow D\} \)
4. \( F^+ \neq G^+ \) because \( (A \rightarrow D) \in (F^+ \setminus G^+) \)

\( F^+ \) cannot have FD’s not in \( F^+ \)

Decomposition is not DP

Is \( R = \{R_1, R_2\} \) dependency preserving?

1. \( F^+ = \{A \rightarrow B, AB \rightarrow D, C \rightarrow D\} \)
2. \( G = \{A \rightarrow B, A \rightarrow D\} \cup \{C \rightarrow D\} \)
3. \( G^+ = \{A \rightarrow B, AB \rightarrow D, A \rightarrow D, C \rightarrow D\} \)
4. \( F^+ = G^+ \)

Decomposition is DP
Dependency Preservation

**R1**(A, B, D) **R2**(C, D)

\[ F = \{ A \rightarrow B, AB \rightarrow D, C \rightarrow D \} \]

- Bonus Question: Is the decomposition **R**={**R1**, **R2**} lossless?

Redundancy Avoidance

- **Main Idea:** Want to avoid duplicate entries in a relation for a FD.
- When there exists some FD X→Y covered by relation and X is not a super key.

```markdown
<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1</td>
<td>B1</td>
<td>C1</td>
<td>D1</td>
</tr>
<tr>
<td>A2</td>
<td>B2</td>
<td>C2</td>
<td>D2</td>
</tr>
<tr>
<td>A3</td>
<td>B3</td>
<td>C3</td>
<td>D3</td>
</tr>
</tbody>
</table>
```

Decomposition is not Lossless

```markdown
<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1</td>
<td>B1</td>
<td>C1</td>
<td>D1</td>
</tr>
<tr>
<td>A2</td>
<td>B2</td>
<td>C2</td>
<td>D2</td>
</tr>
<tr>
<td>A3</td>
<td>B3</td>
<td>C3</td>
<td>D3</td>
</tr>
</tbody>
</table>
```

Redundancy Avoidance

**R**(A, B, C)

\[ F = \{ B \rightarrow C \} \]

- The super keys for **R** are all sets of attributes that include A.

```markdown
<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1</td>
<td>B1</td>
<td>C1</td>
</tr>
<tr>
<td>A2</td>
<td>B1</td>
<td>C1</td>
</tr>
<tr>
<td>A3</td>
<td>B2</td>
<td>C2</td>
</tr>
<tr>
<td>A4</td>
<td>B2</td>
<td>C2</td>
</tr>
<tr>
<td>A5</td>
<td>B2</td>
<td>C2</td>
</tr>
<tr>
<td>A6</td>
<td>B3</td>
<td>C3</td>
</tr>
<tr>
<td>A7</td>
<td>B3</td>
<td>C3</td>
</tr>
</tbody>
</table>
```
Decomposition Summary

• **Lossless Joins**
  – Motivation: Avoid information loss.
  – Goal: No noise introduced when reconstituting universal relation via joins.
  – Test: At each decomposition \( R=(R_1 \cup R_2) \), check whether \( (R_1 \cap R_2) \rightarrow R_1 \) or \( (R_1 \cap R_2) \rightarrow R_2 \)

• **Dependency Preservation**
  – Motivation: Efficient FD assertions.
  – Goal: No global integrity constraints that require joins of more than one table with itself.
  – Test: \( R=(R_1 \cup \ldots \cup R_n) \) is dependency preserving if closure of FD’s covered by each \( R_i = \text{closure of FD’s covered by } R=F \)

• **Redundancy Avoidance**
  – Motivation: Avoid update, delete anomalies.
  – Goal: Avoid update anomalies, wasted space.
  – Test: For an \( X \rightarrow Y \) covered by \( R_n \), \( X \) should be a super key of \( R_n \).

Today’s Class

• The dangers of bad database design
• Decomposition Goals
• Normal Forms
• Relational Model vs. NoSQL
Normal Forms

• Now we know how to derive more FDs, we can then:
  – Search for “bad” FDs
  – If there are such, then decompose the table into two tables, repeat for the sub-tables.
  – When done, the database schema is normalized

The Universe of Relations

All Relations
• 1st Normal Form = All Tables are Flat
• 2nd Normal Form = Obsolete
• 3rd Normal Form = Today
• Boyce-Codd Normal Form = Today
• 4th and 5th = See textbook
• 6th = Most (normal) people never need this.
First Normal Form

• All types must be atomic.

**Loans**(bname, bcity, assets, cname, loanId, amt)

<table>
<thead>
<tr>
<th>bname</th>
<th>bcity</th>
<th>assets</th>
<th>cname</th>
<th>loanId</th>
<th>amt</th>
</tr>
</thead>
<tbody>
<tr>
<td>Downtown</td>
<td>Pittsburgh</td>
<td>$9M</td>
<td>Christos, DJ Snake</td>
<td>L-17</td>
<td>$1000</td>
</tr>
<tr>
<td>Downtown</td>
<td>Pittsburgh</td>
<td>$9M</td>
<td>Obama</td>
<td>L-23</td>
<td>$2000</td>
</tr>
<tr>
<td>Compton</td>
<td>Los Angeles</td>
<td>$2M</td>
<td>Christos</td>
<td>L-93</td>
<td>$500</td>
</tr>
</tbody>
</table>

**Violates 1NF**

Loans(bname, bcity, assets, cname, loanId, amt)

<table>
<thead>
<tr>
<th>bname</th>
<th>bcity</th>
<th>assets</th>
<th>cname</th>
<th>loanId</th>
<th>amt</th>
</tr>
</thead>
<tbody>
<tr>
<td>Downtown</td>
<td>Pittsburgh</td>
<td>$9M</td>
<td>Christos</td>
<td>L-17</td>
<td>$1000</td>
</tr>
<tr>
<td>Downtown</td>
<td>Pittsburgh</td>
<td>$9M</td>
<td>Obama</td>
<td>L-23</td>
<td>$2000</td>
</tr>
<tr>
<td>Compton</td>
<td>Los Angeles</td>
<td>$2M</td>
<td>Christos</td>
<td>L-93</td>
<td>$500</td>
</tr>
<tr>
<td>Downtown</td>
<td>Pittsburgh</td>
<td>$9M</td>
<td>DJ Snake</td>
<td>L-17</td>
<td>$1000</td>
</tr>
</tbody>
</table>

**Valid 1NF!**

Second Normal Form

• 1NF and non-key attribute fully depend on the candidate key.

**R1**(bname, assets, cname, loanId)

<table>
<thead>
<tr>
<th>bname</th>
<th>assets</th>
<th>cname</th>
<th>loanId</th>
</tr>
</thead>
<tbody>
<tr>
<td>Downtown</td>
<td>$9M</td>
<td>Christos</td>
<td>L-17</td>
</tr>
<tr>
<td>Downtown</td>
<td>$9M</td>
<td>Obama</td>
<td>L-23</td>
</tr>
<tr>
<td>Compton</td>
<td>$2M</td>
<td>Christos</td>
<td>L-93</td>
</tr>
<tr>
<td>Downtown</td>
<td>$9M</td>
<td>DJ Snake</td>
<td>L-17</td>
</tr>
</tbody>
</table>

**R2**(loanId, bcity, amt)

<table>
<thead>
<tr>
<th>loanId</th>
<th>bcity</th>
<th>amt</th>
</tr>
</thead>
<tbody>
<tr>
<td>L-17</td>
<td>Pittsburgh</td>
<td>$1000</td>
</tr>
<tr>
<td>L-23</td>
<td>Pittsburgh</td>
<td>$2000</td>
</tr>
<tr>
<td>L-93</td>
<td>Los Angeles</td>
<td>$500</td>
</tr>
</tbody>
</table>

Boyce-Codd Normal Form

• BCNF guarantees no redundancies and no lossless joins (but not DP).

• A relation R with FD set F is in BCNF if for all non-trivial X→Y in F+
  – X→R (i.e., X is a super key)

**Functional Dependencies:**

bname → bcity, assets
loanId → amt, bname
Boyce-Codd Normal Form

- Is \( R \) in BCNF?
- Consider the non-trivial dependencies in \( F^+ \):
  1. \( A \to B, \ A \to R \) (A is a key)
  2. \( A \to C, \ A \to R \) (A is a key)
  3. \( B \to C, \ B \not\to A \) (B is not a key)

\( R \) is not in BCNF

Boyce-Codd Normal Form

- Are \( R_1, R_2 \) in BCNF?
- Test \( R_1 \)
  1. \( A \to B, \ A \to R_1 \) (A is a key)
- Test \( R_2 \)
  1. \( B \to C, \ B \to R_2 \) (B is a key)

\( R_1, R_2 \) are in BCNF

Boyce-Codd Normal Form

- Given a schema \( R \) and a set of FDs \( F \), we can always decompose \( R \) into \( \{R_1, \ldots, R_n\} \) such that
  - \( \{R_1, \ldots, R_n\} \) are in BCNF
  - The decompositions are lossless.
- But some BCNF decompositions might lose dependencies.

BCNF Decomposition Algorithm

- Given a relation \( R \) and a FD set \( F \):
  1. Compute \( F^+ \)
  2. \( \text{Result} \leftarrow \{R\} \)
  3. While some \( R_i \in \text{Result} \) not in BCNF, do:
     a) Choose \((X \to Y) \in F^+ \) such that \((X \to Y) \) is covered
        - \( R_i \) and \( X \not\to R_i \)
        - Decompose \( R_i \) on \((X \to Y)\):
          \( R_{i,1} \leftarrow X \cup Y \)
          \( R_{i,2} \leftarrow R_i - Y \)
     c) \( \text{Result} \leftarrow (\text{Result} - \{R_i\}) \cup \{R_{i,1}, R_{i,2}\} \)
BCNF Example

\[ R(\text{name, ssn, phone#}, \text{city}) \]
\[ F = \{\text{ssn} \rightarrow \text{name, city} \} \]

<table>
<thead>
<tr>
<th>name</th>
<th>ssn</th>
<th>phone#</th>
<th>city</th>
</tr>
</thead>
<tbody>
<tr>
<td>Christos</td>
<td>123-45-6789</td>
<td>555-555-5555</td>
<td>Pittsburgh</td>
</tr>
<tr>
<td>Christos</td>
<td>123-45-6789</td>
<td>666-666-6666</td>
<td>Pittsburgh</td>
</tr>
<tr>
<td>Lil' Fame</td>
<td>987-65-4321</td>
<td>777-777-7777</td>
<td>Brooklyn</td>
</tr>
<tr>
<td>Lil' Fame</td>
<td>987-65-4321</td>
<td>888-888-8888</td>
<td>Brooklyn</td>
</tr>
</tbody>
</table>

- Step #1:
  - \( F^+ = \{\text{ssn} \rightarrow \text{name, city, (ssn\rightarrow\text{name,city)}\} \}

- Step #3: \( R \) is not in BCNF
  - 3(a): We choose \( (\text{ssn} \rightarrow \text{name, city}) \) as the FD to split on because \( \text{ssn} \) does not get us the \( \text{phone#} \) (i.e., it is not the super key).

  - 3(b): Split \( R \) based on \( (\text{ssn} \rightarrow \text{name, city}) \) such that \( R_1 = (\text{name, ssn, city}) \) and \( R_2 = (\text{ssn, phone#}) \)

  - 3(c): The resulting schema is now \( R = \{R_1, R_2\} \)
BCNF Example

\[ R_1(\text{name}, \text{ssn}, \text{city}) \quad R_2(\text{ssn}, \text{phone#}) \]
\[ F = \{\text{ssn} \rightarrow \text{name, city}\} \]

- Step #3: Check if \{R_1, R_2\} are not in BCNF
  - Lossless?
  - Anomalies?

Faloutsos/Pavlo
CMU SCS 15-415/615 53

A Problem with BCNF

\[ R(\text{unit}, \text{comp}, \text{product}) \]
\[ F = \{\text{unit} \rightarrow \text{comp}, (\text{comp, product} \rightarrow \text{unit})\} \]

- At this point we don’t have any problems:
  - We’re in BCNF and all local FDs are satisfied.

Faloutsos/Pavlo
CMU SCS 15-415/615 54

A Problem with BCNF

\[ R_1(\text{unit, comp}) \quad R_2(\text{unit, product}) \]
\[ F = \{\text{unit} \rightarrow \text{comp}, (\text{comp,product} \rightarrow \text{unit})\} \]

- We started with a relation \( R \) and its dependency set \( FD \).
- We decomposed \( R \) into BCNF relations \{R_1, \ldots, R_n\} with their own \{FD_1, \ldots, FD_n\}.
- We can reconstruct R from \{R_1, \ldots, R_n\}.
- But we cannot reconstruct FD from \{FD_1, \ldots, FD_n\}.

Faloutsos/Pavlo
CMU SCS 15-415/615 55
Third Normal Form

• 3NF preserves dependencies but may have some anomalies.
• A relation $R$ with FD set $F$ is in 3NF if for every $X \rightarrow Y$ in $F+$:
  – $X \rightarrow Y$ is trivial, or
  – $X$ is a super key, or
  – $Y$ is part of a candidate key

3NF Decomposition Algorithm

• Given a relation $R$ and a FD set $F$:
  1. Compute $F_c$
  2. $\text{Result} \leftarrow \emptyset$
  3. For each FD $(X \rightarrow Y) \in F_c$, add a relation $R(X, Y)$ to $\text{Result}$
  4. If $\text{Result}$ is not lossless, add a table with an appropriate key.

3NF Example

Step #1: Compute canonical cover
– $F_c \leftarrow \{A \rightarrow B, B \rightarrow C\}$

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1</td>
<td>B_A1</td>
<td></td>
<td>C_B_A1</td>
</tr>
<tr>
<td>A2</td>
<td>B_A2</td>
<td></td>
<td>C_B_A2</td>
</tr>
<tr>
<td>A3</td>
<td>B_A3</td>
<td></td>
<td>C_B_A3</td>
</tr>
<tr>
<td>A2</td>
<td>B_A2</td>
<td></td>
<td>C_B_A2</td>
</tr>
</tbody>
</table>

Step #3: Split $R$ based on its FDs
– $R_1(A, B)$ because $A \rightarrow B$
– $R_2(B, C)$ because $B \rightarrow C$
• Step #3: Split $R$ based on its FDs
  – $R_1(A, B)$ because $A \rightarrow B$
  – $R_2(B, C)$ because $B \rightarrow C$

• Step #4: Check whether $\{R_1, R_2\}$ is lossless.
  – Nope!
  – Add $R_3$ based on the join attribute $A \rightarrow C$
BCNF vs. 3NF

- **BCNF:**
  - No Anomalies, but may lose some FDS.
  - In practice, this is what you want.
- **3NF:**
  - Keeps all FDs, but may have some anomalies.

Today’s Class

- The dangers of bad database design
- Decomposition Goals
- Normal forms
- Relational Model vs. NoSQL

The Rise of NoSQL

- Prior to the early 2000s, few people needed a high-performance DBMS.
- Key tenants of the NoSQL movement:
  - Joins are slow, so we will denormalize tables.
  - Transactions are slow and we need to be on-line 24/7, so let’s drop ACID.

MongoDB

- Document Model = JSON / XML
- No server-side joins:
  - “Pre-join” collections by embedding related documents inside of each other.

Next Week
BCNF Example

- A customer has orders and each order has order items.

Denormalization Example

- A customer has orders and each order has order items.

Denormalization Example

- A customer has orders and each order has order items.

Denormalization Example

• No joins is not a by-product of using the document model, but it makes logical denormalization more “natural”:
  - MongoDB isn’t the only document DBMS:
    - CouchDB
    - RavenDB
    - RethinkDB
    - MarkLogic (XML)
Next Week

- Physical Database Design + Tuning
- The (Awesome) World of Transactions