Lecture 02: Relational Query Languages and Database Design

Tuesday, January 14, 2014
Brief Review of 1\textsuperscript{st} Lecture

• Database = collection of related files
• Physical data independence
• SQL:
  – Select-from-where
  – Nested loop semantics
  – Group by (you read the slides, right?)
  – Advanced stuff: nested queries, outerjoins
Outline

• Stonebraker’s blog on *Big Data*

• Relational Query Languages

• Database Design: Book Chapters 2, 3

• Functional Dependencies and BCNF
Big Data

What is it?
Big Data

What is it?

• Gartner report*
  – High Volume
  – High Variety
  – High Velocity

* [http://www.gartner.com/newsroom/id/1731916](http://www.gartner.com/newsroom/id/1731916)
Big Data

What is it?

• Stonebraker:
  – Big volumes, small analytics
  – Big analytics, on big volumes
  – Big velocity
  – Big variety

• What do you think about Big Data?
Outline

• Stonebraker’s blog on *Big Data*

• Relational Query Languages
  – Relational algebra
  – Recursion-free datalog with negation
  – Relational calculus

• Database Design

• Functional Dependencies and BCNF
Running Example

Find all actors who acted both in 1910 and in 1940:

Q: SELECT DISTINCT a.fname, a.lname
    FROM   Actor a, Casts c1, Movie m1, Casts c2, Movie m2
    WHERE  a.id = c1.pid  AND c1.mid = m1.id
            AND a.id = c2.pid  AND c2.mid = m2.id
            AND m1.year = 1910  AND m2.year = 1940;
Two Perspectives

• Named Perspective:
  Actor(id, fname, lname)
  Casts(pid, mid)
  Movie(id, name, year)

• Unnamed Perspective:
  Actor = arity 3
  Casts = arity 2
  Movie = arity 3
1. Relational Algebra

- Used internally by the database engine to execute queries

- Book: chapter 4.2

- We will return to RA when we discuss query execution
1. Relational Algebra

The Basic Five operators:

- Union: $\cup$
- Difference: $-$
- Selection: $\sigma$
- Projection: $\Pi$
- Join: $\Join$

Renaming: $\rho$ (for named perspective)
1. Relational Algebra (Details)

• **Selection**: returns tuples that satisfy condition
  
  – Named perspective: \( \sigma_{\text{year} = '1910'}(\text{Movie}) \)
  
  – Unamed perspective: \( \sigma_3 = '1910' \) (Movie)
1. Relational Algebra (Details)

- **Selection**: returns tuples that satisfy condition
  - Named perspective: \( \sigma_{\text{year} = '1910'}(\text{Movie}) \)
  - Unnamed perspective: \( \sigma_3 = '1910' \) (Movie)

- **Projection**: returns only some attributes
  - Named perspective: \( \Pi_{\text{fname, lname}}(\text{Actor}) \)
  - Unnamed perspective: \( \Pi_{2,3}(\text{Actor}) \)
1. Relational Algebra (Details)

- **Selection**: returns tuples that satisfy condition
  - Named perspective: \( \sigma_{\text{year} = \text{‘1910’}}(\text{Movie}) \)
  - Unnamed perspective: \( \sigma_3 = \text{‘1910’} \) (Movie)

- **Projection**: returns only some attributes
  - Named perspective: \( \Pi_{\text{fname, lname}}(\text{Actor}) \)
  - Unnamed perspective: \( \Pi_{2,3}(\text{Actor}) \)

- **Join**: joins two tables on a condition
  - Named perspective: \( \text{Casts} \Join_{\text{mid} = \text{id}} \text{Movie} \)
  - Unnamed perspective: \( \text{Casts} \Join_{2=1} \text{Movie} \)
1. Relational Algebra Example

Q: SELECT DISTINCT a.fname, a.lname
    FROM   Actor a, Casts c1, Movie m1, Casts c2, Movie m2
    WHERE  a.id = c1.pid AND c1.mid = m1.id
            AND a.id = c2.pid AND c2.mid = m2.id
            AND m1.year = 1910 AND m2.year = 1940;

Note how we renamed year to year1, year2

Actor(id, fname, lname)
Casts(pid, mid)
Movie(id, name, year)
1. Relational Algebra Example

Q: SELECT DISTINCT a.fname, a.lname
    FROM Actor a, Casts c1, Movie m1, Casts c2, Movie m2
    WHERE a.id = c1.pid AND c1.mid = m1.id
        AND a.id = c2.pid AND c2.mid = m2.id
        AND m1.year = 1910 AND m2.year = 1940;

Actor(id, fname, lname)
Casts(pid,mid)
Movie(id,name,year)
Joins and Cartesian Product

• Each tuple in R1 with each tuple in R2

\[ R1 \times R2 \]

• Rare in practice; mainly used to express joins
Cartesian Product (aka Cross Product)

**Employee**

<table>
<thead>
<tr>
<th>Name</th>
<th>SSN</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>9999999999</td>
</tr>
<tr>
<td>Tony</td>
<td>7777777777</td>
</tr>
</tbody>
</table>

**Dependent**

<table>
<thead>
<tr>
<th>EmpSSN</th>
<th>DepName</th>
</tr>
</thead>
<tbody>
<tr>
<td>9999999999</td>
<td>Emily</td>
</tr>
<tr>
<td>7777777777</td>
<td>Joe</td>
</tr>
</tbody>
</table>

**Employee \times Dependent**

<table>
<thead>
<tr>
<th>Name</th>
<th>SSN</th>
<th>EmpSSN</th>
<th>DepName</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>9999999999</td>
<td>9999999999</td>
<td>Emily</td>
</tr>
<tr>
<td>John</td>
<td>9999999999</td>
<td>7777777777</td>
<td>Joe</td>
</tr>
<tr>
<td>Tony</td>
<td>7777777777</td>
<td>9999999999</td>
<td>Emily</td>
</tr>
<tr>
<td>Tony</td>
<td>7777777777</td>
<td>7777777777</td>
<td>Joe</td>
</tr>
</tbody>
</table>
Natural Join

\[ R1 \bowtie R2 \]

• Meaning: \[ R1 \bowtie R2 = \Pi_A(\sigma(R1 \times R2)) \]

• Where:
  – Selection \( \sigma \) checks equality of all common attributes
  – Projection eliminates duplicate common attributes
### Natural Join Example

#### Table R

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>Y</td>
</tr>
<tr>
<td>X</td>
<td>Z</td>
</tr>
<tr>
<td>Y</td>
<td>Z</td>
</tr>
<tr>
<td>Z</td>
<td>V</td>
</tr>
</tbody>
</table>

#### Table S

<table>
<thead>
<tr>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Z</td>
<td>U</td>
</tr>
<tr>
<td>V</td>
<td>W</td>
</tr>
<tr>
<td>Z</td>
<td>V</td>
</tr>
</tbody>
</table>

#### Expression

\[
R \bowtie S = \Pi_{ABC}(\sigma_{R.B=S.B}(R \times S))
\]

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>Z</td>
<td>U</td>
</tr>
<tr>
<td>X</td>
<td>Z</td>
<td>V</td>
</tr>
<tr>
<td>Y</td>
<td>Z</td>
<td>U</td>
</tr>
<tr>
<td>Y</td>
<td>Z</td>
<td>V</td>
</tr>
<tr>
<td>Z</td>
<td>V</td>
<td>W</td>
</tr>
</tbody>
</table>
**Natural Join Example 2**

**AnonPatient P**

<table>
<thead>
<tr>
<th>age</th>
<th>zip</th>
<th>disease</th>
</tr>
</thead>
<tbody>
<tr>
<td>54</td>
<td>98125</td>
<td>heart</td>
</tr>
<tr>
<td>20</td>
<td>98120</td>
<td>flu</td>
</tr>
</tbody>
</table>

**Voters V**

<table>
<thead>
<tr>
<th>name</th>
<th>age</th>
<th>zip</th>
</tr>
</thead>
<tbody>
<tr>
<td>p1</td>
<td>54</td>
<td>98125</td>
</tr>
<tr>
<td>p2</td>
<td>20</td>
<td>98120</td>
</tr>
</tbody>
</table>

\[ P \bowtie V \]

<table>
<thead>
<tr>
<th>age</th>
<th>zip</th>
<th>disease</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>54</td>
<td>98125</td>
<td>heart</td>
<td>p1</td>
</tr>
<tr>
<td>20</td>
<td>98120</td>
<td>flu</td>
<td>p2</td>
</tr>
</tbody>
</table>
Natural Join

• Given schemas $R(A, B, C, D)$, $S(A, C, E)$, what is the schema of $R \Join S$?

• Given $R(A, B, C)$, $S(D, E)$, what is $R \Join S$?

• Given $R(A, B)$, $S(A, B)$, what is $R \Join S$?
Theta Join

• A join that involves a predicate

$$R1 \bowtie_{\theta} R2 = \sigma_{\theta} (R1 \times R2)$$

• Here $\theta$ can be any condition

• For our voters/disease example:

$$P \bowtie P.zip = V.zip \text{ and } P.age < V.age + 5 \text{ and } P.age > V.age - 5$$
Equijoin

• A theta join where $\theta$ is an equality

$$R1 \bowtie_{A=B} R2 = \sigma_{A=B} (R1 \times R2)$$

• This is by far the most used variant of join in practice
## Equijoin Example

**AnonPatient** $P$

<table>
<thead>
<tr>
<th>age</th>
<th>zip</th>
<th>disease</th>
</tr>
</thead>
<tbody>
<tr>
<td>54</td>
<td>98125</td>
<td>heart</td>
</tr>
<tr>
<td>20</td>
<td>98120</td>
<td>flu</td>
</tr>
</tbody>
</table>

**Voters** $V$

<table>
<thead>
<tr>
<th>name</th>
<th>age</th>
<th>zip</th>
</tr>
</thead>
<tbody>
<tr>
<td>p1</td>
<td>54</td>
<td>98125</td>
</tr>
<tr>
<td>p2</td>
<td>20</td>
<td>98120</td>
</tr>
</tbody>
</table>

**Equijoin $P \bowtie_{P.age = V.age} V$**

<table>
<thead>
<tr>
<th>age</th>
<th>P.zip</th>
<th>disease</th>
<th>name</th>
<th>V.zip</th>
</tr>
</thead>
<tbody>
<tr>
<td>54</td>
<td>98125</td>
<td>heart</td>
<td>p1</td>
<td>98125</td>
</tr>
<tr>
<td>20</td>
<td>98120</td>
<td>flu</td>
<td>p2</td>
<td>98120</td>
</tr>
</tbody>
</table>
Join Summary

• **Theta-join:** $R \bowtie_\theta S = \sigma_\theta(R \times S)$
  - Join of $R$ and $S$ with a join condition $\theta$
  - Cross-product followed by selection $\theta$

• **Equijoin:** $R \bowtie_\theta S = \pi_A (\sigma_\theta(R \times S))$
  - Join condition $\theta$ consists only of equalities
  - Projection $\pi_A$ drops all redundant attributes

• **Natural join:** $R \bowtie S = \pi_A (\sigma_\theta(R \times S))$
  - Equijoin
  - Equality on **all** fields with same name in $R$ and in $S$
So Which Join Is It?

- When we write $R \bowtie S$ we usually mean an equijoin, but we often omit the equality predicate when it is clear from the context.
More Joins

• **Outer join**
  – Include tuples with no matches in the output
  – Use NULL values for missing attributes

• **Variants**
  – Left outer join
  – Right outer join
  – Full outer join
### Outer Join Example

#### AnonPatient $P$

<table>
<thead>
<tr>
<th>age</th>
<th>zip</th>
<th>disease</th>
</tr>
</thead>
<tbody>
<tr>
<td>54</td>
<td>98125</td>
<td>heart</td>
</tr>
<tr>
<td>20</td>
<td>98120</td>
<td>flu</td>
</tr>
<tr>
<td>33</td>
<td>98120</td>
<td>lung</td>
</tr>
</tbody>
</table>

#### AnonJob $J$

<table>
<thead>
<tr>
<th>job</th>
<th>age</th>
<th>zip</th>
</tr>
</thead>
<tbody>
<tr>
<td>lawyer</td>
<td>54</td>
<td>98125</td>
</tr>
<tr>
<td>cashier</td>
<td>20</td>
<td>98120</td>
</tr>
</tbody>
</table>

#### $P \bowtie_{V} J$

<table>
<thead>
<tr>
<th>age</th>
<th>zip</th>
<th>disease</th>
<th>job</th>
</tr>
</thead>
<tbody>
<tr>
<td>54</td>
<td>98125</td>
<td>heart</td>
<td>lawyer</td>
</tr>
<tr>
<td>20</td>
<td>98120</td>
<td>flu</td>
<td>cashier</td>
</tr>
<tr>
<td>33</td>
<td>98120</td>
<td>lung</td>
<td>null</td>
</tr>
</tbody>
</table>
Some Examples

Q2: Name of supplier of parts with size greater than 10
\[ \pi_{\text{sname}}(\text{Supplier} \Join \text{Supply} \Join (\sigma_{\text{psize}>10} (\text{Part})) \]

Q3: Name of supplier of red parts or parts with size greater than 10
\[ \pi_{\text{sname}}(\text{Supplier} \Join \text{Supply} \Join (\sigma_{\text{psize}>10} (\text{Part}) \cup \sigma_{\text{pcolor}='red'} (\text{Part}))) \]
Outline

• Stonebraker’s blog on *Big Data*
• Relational Query Languages
  – Relational algebra
  – Recursion-free datalog with negation
  – Relational calculus
• Database Design
• Functional Dependencies and BCNF
2. Datalog

• Very friendly notation for queries
• Initially designed for recursive queries
• Some companies offer datalog implementation for data analytics, e.g. LogicBlox
• Today: only recursion-free or non-recursive datalog, and add negation
• Later: full datalog
2. Datalog

How to try out datalog quickly:

- Download DLV from [http://www.dbai.tuwien.ac.at/proj/dlv/](http://www.dbai.tuwien.ac.at/proj/dlv/)
- Run DLV on this file:

```
parent(william, john).
parent(john, james).
parent(james, bill).
parent(sue, bill).
parent(james, carol).
parent(sue, carol).

male(john).
male(james).
female(sue).
male(bill).
female(carol).

grandparent(X, Y) :- parent(X, Z), parent(Z, Y).
father(X, Y) :- parent(X, Y), male(X).
mother(X, Y) :- parent(X, Y), female(X).
brother(X, Y) :- parent(P, X), parent(P, Y), male(X), X != Y.
sister(X, Y) :- parent(P, X), parent(P, Y), female(X), X != Y.
```
2. Datalog: Facts and Rules

**Facts**

<table>
<thead>
<tr>
<th>Actor</th>
<th>Name</th>
<th>Movie</th>
</tr>
</thead>
<tbody>
<tr>
<td>344759</td>
<td>Douglas</td>
<td>A Night in Armour, 1910</td>
</tr>
<tr>
<td>344759</td>
<td></td>
<td>Arizona, 1940</td>
</tr>
<tr>
<td>344759</td>
<td></td>
<td>Ave Maria, 1940</td>
</tr>
<tr>
<td>355713</td>
<td></td>
<td></td>
</tr>
<tr>
<td>29000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>29445</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Rules**

- Q1(y) :- Movie(x,y,z), z='1940'.
- Q2(f, l) :- Actor(z,f,l), Casts(z,x), Movie(x,y,'1940').
- Q3(f,l) :- Actor(z,f,l), Casts(z,x1), Movie(x1,y1,1910), Casts(z,x2), Movie(x2,y2,1940).

**Notes**

- Facts = tuples in the database
- Rules = queries
- Extensional Database Predicates = EDB
- Intensional Database Predicates = IDB
2. Datalog: Terminology

Q2(f, l) :- Actor(z,f,l), Casts(z,x), Movie(x,y,’1940’).

f, l = head variables
x,y,z = existential variables
2. Datalog program

Find all actors with Bacon number ≤ 2

B0(x) :- Actor(x,'Kevin', 'Bacon')
B1(x) :- Actor(x,f,l), Casts(x,z), Casts(y,z), B0(y)
B2(x) :- Actor(x,f,l), Casts(x,z), Casts(y,z), B1(y)
Q4(x) :- B1(x)
Q4(x) :- B2(x)

Note: Q4 is the union of B1 and B2
2. Datalog with negation

Find all actors with Bacon number $\geq 2$

\begin{verbatim}
B0(x) :- Actor(x,'Kevin', 'Bacon')
B1(x) :- Actor(x,f,l), Casts(x,z), Casts(y,z), B0(y)
Q6(x) :- Actor(x,f,l), not B1(x), not B0(x)
\end{verbatim}
2. Safe Datalog Rules

Here are *unsafe* datalog rules. What’s “unsafe” about them?

\[
U1(x,y) :- \text{Movie}(x,z,1994), y>1910
\]

\[
U2(x) :- \text{Movie}(x,z,1994), \neg \text{Casts}(u,x)
\]

A datalog rule is *safe* if every variable appears in some positive relational atom.
2. Datalog v.s. SQL

- Non-recursive datalog with negation is very close to SQL; with some practice, you should be able to translate between them back and forth without difficulty; see example in the paper
Outline

• Stonebraker’s blog on *Big Data*

• Relational Query Languages
  – Relational algebra
  – Recursion-free datalog with negation
  – Relational calculus

• Database Design

• Functional Dependencies and BCNF
3. Relational Calculus

• Also known as *predicate calculus*, or *first order logic*

• The most expressive formalism for queries: easy to write complex queries

• TRC = Tuple RC = named perspective

• DRC = Domain RC = unnamed perspective
3. Relational Calculus

Predicate P:

\[ P ::= \text{atom} \mid P \land P \mid P \lor P \mid P \Rightarrow P \mid \text{not}(P) \mid \forall x. P \mid \exists x. P \]

Query Q:

\[ Q(x_1, \ldots, x_k) = P \]

Example: find the first/last names of actors who acted in 1940

\[ Q(f, l) = \exists x. \exists y. \exists z. (\text{Actor}(z, f, l) \land \text{Casts}(z, x) \land \text{Movie}(x, y, 1940)) \]

What does this query return?

\[ Q(f, l) = \exists z. (\text{Actor}(z, f, l) \land \forall x. (\text{Casts}(z, x) \Rightarrow \exists y. \text{Movie}(x, y, 1940))) \]
3. Relational Calculus:

Example

Likes(drinker, beer)
Frequents(drinker, bar)
Serves(bar, beer)

Find drinkers that frequent some bar that serves some beer they like.

Q(x) = ∃y. ∃z. Frequents(x, y) ∧ Serves(y,z) ∧ Likes(x,z)
3. Relational Calculus: Example

Likes(drinker, beer)  
Frequents(drinker, bar)  
Serves(bar, beer)

Find drinkers that frequent some bar that serves some beer they like.

\[ Q(x) = \exists y. \exists z. \text{Frequents}(x, y) \land \text{Serves}(y, z) \land \text{Likes}(x, z) \]

Find drinkers that frequent only bars that serves some beer they like.

\[ Q(x) = \forall y. \text{Frequents}(x, y) \Rightarrow (\exists z. \text{Serves}(y, z) \land \text{Likes}(x, z)) \]
3. Relational Calculus: Example

Likes(drinker, beer)
Frequents(drinker, bar)
Serves(bar, beer)

Find drinkers that frequent some bar that serves some beer they like.

Q(x) = \exists y. \exists z. \text{Frequents}(x, y) \land \text{Serves}(y,z) \land \text{Likes}(x,z)

Find drinkers that frequent only bars that serves some beer they like.

Q(x) = \forall y. \text{Frequents}(x, y) \Rightarrow (\exists z. \text{Serves}(y,z) \land \text{Likes}(x,z))

Find drinkers that frequent some bar that serves only beers they like.

Q(x) = \exists y. \text{Frequents}(x, y) \land \forall z. (\text{Serves}(y,z) \Rightarrow \text{Likes}(x,z))
3. Relational Calculus: Example

Find drinkers that frequent some bar that serves some beer they like.

\[ Q(x) = \exists y. \exists z. \text{Frequents}(x, y) \land \text{Serves}(y,z) \land \text{Likes}(x,z) \]

Find drinkers that frequent only bars that serves some beer they like.

\[ Q(x) = \forall y. \text{Frequents}(x, y) \rightarrow (\exists z. \text{Serves}(y,z) \land \text{Likes}(x,z)) \]

Find drinkers that frequent some bar that serves only beers they like.

\[ Q(x) = \exists y. \text{Frequents}(x, y) \land \forall z. (\text{Serves}(y,z) \rightarrow \text{Likes}(x,z)) \]

Find drinkers that frequent only bars that serves only beer they like.

\[ Q(x) = \forall y. \text{Frequents}(x, y) \rightarrow \forall z. (\text{Serves}(y,z) \rightarrow \text{Likes}(x,z)) \]
3. Domain Independent
Relational Calculus

• As in datalog, one can write “unsafe”
RC queries; they are also called domain
dependent

• See examples in the paper

• Moral: make sure your RC queries are
always domain independent
3. Relational Calculus

Take home message:
• Need to write a complex SQL query:
  • First, write it in RC
  • Next, translate it to datalog (see next)
  • Finally, write it in SQL

As you gain experience, take shortcuts
3. From RC to Non-recursive Datalog w/ negation

Query: Find drinkers that like some beer so much that they frequent all bars that serve it

\[ Q(x) = \exists y. \text{Likes}(x, y) \land \forall z. (\text{Serves}(z, y) \Rightarrow \text{Frequents}(x, z)) \]
3. From RC to Non-recursive Datalog w/ negation

**Query:** Find drinkers that like some beer so much that they frequent all bars that serve it

\[ Q(x) = \exists y. \text{Likes}(x, y) \land \forall z. (\text{Serves}(z, y) \Rightarrow \text{Frequents}(x, z)) \]

**Step 1:** Replace \( \forall \) with \( \exists \) using de Morgan’s Laws

\[ Q(x) = \exists y. \text{Likes}(x, y) \land \neg \exists z. (\text{Serves}(z, y) \land \neg \text{Frequents}(x, z)) \]
3. From RC to Non-recursive Datalog w/ negation

**Query:** Find drinkers that like some beer so much that they frequent all bars that serve it

\[ Q(x) = \exists y. \text{Likes}(x, y) \land \forall z. (\text{Serves}(z,y) \Rightarrow \text{Frequents}(x,z)) \]

**Step 1:** Replace \( \forall \) with \( \exists \) using de Morgan’s Laws

\[ Q(x) = \exists y. \text{Likes}(x, y) \land \neg \exists z. (\text{Serves}(z,y) \land \neg \text{Frequents}(x,z)) \]

**Step 2:** Make all subqueries domain independent

\[ Q(x) = \exists y. \text{Likes}(x, y) \land \neg \exists z. (\text{Likes}(x,y) \land \text{Serves}(z,y) \land \neg \text{Frequents}(x,z)) \]
3. From RC to Non-recursive Datalog w/ negation

Step 3: Create a datalog rule for each subexpression;
  (shortcut: only for “important” subexpressions)

\[ Q(x) = \exists y. \text{Likes}(x, y) \land \neg \exists z. (\text{Likes}(x,y) \land \text{Serves}(z,y) \land \neg \text{Frequents}(x,z)) \]

\[ H(x,y) \]

\[ H(x,y) \quad :\quad \text{Likes}(x,y), \text{Serves}(y,z), \text{not Frequents}(x,z) \]
\[ Q(x) \quad :\quad \text{Likes}(x,y), \text{not } H(x,y) \]
3. From RC to Non-recursive Datalog w/ negation

\[
\begin{align*}
H(x,y) & \ :- \ Likes(x,y), Serves(y,z), \text{not Frequents}(x,z) \\
Q(x) & \ :- \ Likes(x,y), \text{not } H(x,y)
\end{align*}
\]

Step 4: Write it in SQL

```sql
SELECT DISTINCT L.drinker FROM Likes L
WHERE not exists
  (SELECT * FROM Likes L2, Serves S
   WHERE L2.drinker=L.drinker and L2.beer=L.beer
   and L2.beer=S.beer
   and not exists (SELECT * FROM Frequents F
                      WHERE F.drinker=L2.drinker
                      and F.bar=S.bar))
```
3. From RC to Non-recursive Datalog w/ negation

H(x,y) :- Likes(x,y), Serves(y,z), not Frequents(x,z)
Q(x)   :- Likes(x,y), not H(x,y)

Unsafe rule

Improve the SQL query by using an unsafe datalog rule

```
SELECT DISTINCT L.drinker FROM Likes L
WHERE not exists
  (SELECT * FROM Serves S
   WHERE L.beer=S.beer
   and not exists (SELECT * FROM Frequents F
                     WHERE F.drinker=L.drinker
                     and F.bar=S.bar))
```
Summary of Translation

• RC $\rightarrow$ recursion-free datalog w/ negation
  – Subtle: as we saw; more details in the paper

• Recursion-free datalog w/ negation $\rightarrow$ RA
  – Easy: see paper

• RA $\rightarrow$ RC
  – Easy: see paper
Summary

• All three have same expressive power:
  – RA
  – Non-recursive datalog w/ neg. (= “core” SQL)
  – RC

• Write complex queries in RC first, then translate to SQL
Outline

• Stonebraker’s blog on *Big Data*
• Relational Query Languages
  – Relational algebra
  – Recursion-free datalog with negation
  – Relational calculus
• Database Design
• Functional Dependencies and BCNF
Database Design
Database Design Process

Conceptual Model:

Relational Model:
Tables + constraints
And also functional dep.

Normalization:
Eliminates anomalies

Conceptual Schema

Physical storage details
Physical Schema
Entity / Relationship Diagrams

- Entity set = a class
  - An entity = an object

- Attribute

- Relationship
Keys in E/R Diagrams

- Every entity set must have a key

```
Product
  name
  price
```
What is a Relation?

• A mathematical definition:
  – if A, B are sets, then a relation R is a subset of $A \times B$

• $A=\{1,2,3\}$, $B=\{a,b,c,d\}$,
  $A \times B = \{(1,a),(1,b), \ldots, (3,d)\}$
  $R = \{(1,a), (1,c), (3,b)\}$

• **makes** is a subset of **Product $\times$ Company**:
Multiplicity of E/R Relations

• one-one:

• many-one

• many-many
What does this say?
Notation in Class v.s. the Book

In class:

Product \(\rightarrow\) makes \(\rightarrow\) Company

In the book:

Product \(\rightarrow\) makes \(\rightarrow\) Company
Multi-way Relationships

How do we model a purchase relationship between buyers, products and stores?

Can still model as a mathematical set (Q. how?)

A. As a set of triples \( \subseteq \text{Person} \times \text{Product} \times \text{Store} \)
Q: What does the arrow mean?

A: A given person buys a given product from at most one store.

[Arrow pointing to E means that if we select one entity from each of the other entity sets in the relationship, those entities are related to at most one entity in E]
Q: What does the arrow mean?

A: A given person buys a given product from at most one store AND every store sells to every person at most one product.
Q: How do we say that every person shops at at most one store?

A: Cannot. This is the best approximation. (Why only approximation?)
Converting Multi-way Relationships to Binary

Arrows go in which direction?

date

Purchase

ProductOf

StoreOf

BuyerOf

Product

Store

Person
Converting Multi-way Relationships to Binary

Make sure you understand why!
Design Principles

What’s wrong?

Moral: be faithful to the specifications of the app!
Design Principles: What’s Wrong?

Moral: pick the right kind of entities.
Design Principles: What’s Wrong?

Product

Purchase

Dates

Store

Person

date

Moral: don’t complicate life more than it already is.
From E/R Diagrams to Relational Schema

- Entity set $\rightarrow$ relation
- Relationship $\rightarrow$ relation
Entity Set to Relation

\[
\text{Product}(\text{prod-ID}, \text{category}, \text{price})
\]

<table>
<thead>
<tr>
<th>prod-ID</th>
<th>category</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo55</td>
<td>Camera</td>
<td>99.99</td>
</tr>
<tr>
<td>Pokenm19</td>
<td>Toy</td>
<td>29.99</td>
</tr>
</tbody>
</table>
CREATE TABLE Product ( prod-ID CHAR(30) PRIMARY KEY, category VARCHAR(20), price double)
N-N Relationships to Relations

Represent that in relations!
**Orders** (prod-ID, cust-ID, date)

**Shipment** (prod-ID, cust-ID, name, date)

**Shipping-Co** (name, address)

<table>
<thead>
<tr>
<th>prod-ID</th>
<th>cust-ID</th>
<th>name</th>
<th>date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo55</td>
<td>Joe12</td>
<td>UPS</td>
<td>4/10/2011</td>
</tr>
<tr>
<td>Gizmo55</td>
<td>Joe12</td>
<td>FEDEX</td>
<td>4/9/2011</td>
</tr>
</tbody>
</table>
CREATE TABLE Shipment(
    name CHAR(30)
    REFERENCES Shipping-Co,
    prod-ID CHAR(30),
    cust-ID VARCHAR(20),
    date DATETIME,
    PRIMARY KEY (name, prod-ID, cust-ID),
    FOREIGN KEY (prod-ID, cust-ID)
    REFERENCES Orders
)
N-1 Relationships to Relations

Represent this in relations!
Orders\( (\text{prod-ID}, \text{cust-ID}, \text{date1}, \text{name}, \text{date2}) \)

Shipping-Co\( (\text{name}, \text{address}) \)

Remember: no separate relations for many-one relationship
Multi-way Relationships to Relations

Product

- prod-ID
- price

Purchase

- cust-ssn
- store-name

Person

- ssn
- name

Store

- name
- address

Purchase(prod-ID, cust-ssn, store-name)
Modeling Subclasses

Some objects in a class may be special
define a new class
better: define a *subclass*

Products

Software products

Educational products

So --- we define subclasses in E/R
Understanding Subclasses

Think in terms of records:

Product

- field1
- field2

SoftwareProduct

- field1
- field2
- field3

EducationalProduct

- field1
- field2
- field4
- field5
Subclasses to Relations

Other ways to convert are possible

<table>
<thead>
<tr>
<th>Name</th>
<th>Price</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>99</td>
<td>gadget</td>
</tr>
<tr>
<td>Camera</td>
<td>49</td>
<td>photo</td>
</tr>
<tr>
<td>Toy</td>
<td>39</td>
<td>gadget</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>platforms</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>unix</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Age Group</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>toddler</td>
</tr>
<tr>
<td>Toy</td>
<td>retired</td>
</tr>
</tbody>
</table>
Modeling Union Types With Subclasses

Say: each piece of furniture is owned either by a person or by a company
Modeling Union Types With Subclasses

Say: each piece of furniture is owned either by a person or by a company

Solution 1. Acceptable but imperfect (What’s wrong ?)
Modeling Union Types With Subclasses

Solution 2: better, more laborious

- Person
- Company
- FurniturePiece

isa

ownedBy

isa
Weak Entity Sets

Entity sets are weak when their key comes from other classes to which they are related.

Team(sport, number, universityName)
University(name)
What Are the Keys of R?
Constraints in E/R Diagrams

• Finding constraints is part of the modeling process.
• Commonly used constraints:

  • **Keys**: social security number uniquely identifies a person.
  
  • **Single-value constraints**: a person can have only one father.
  
  • **Referential integrity constraints**: if you work for a company, it must exist in the database.

  • **Other constraints**: peoples’ ages are between 0 and 150.
No formal way to specify multiple keys in E/R diagrams.
Single Value Constraints

makes

V. S.

makes
Referential Integrity Constraints

Each product made by at most one company.
Some products made by no company

Each product made by exactly one company.

Note: For weak entity sets should be replaced by (sec 4.4.2)
Q: What does this mean?
A: A Company entity cannot be connected by relationship to more than 99 Product entities.

Note: For “at least one”, you can use “≥ 1” in a many-many relationship.
Database Design Summary

• Conceptual modeling = design the database schema
  – Usually done with Entity-Relationship diagrams
  – It is a form of documentation the database schema; it is not executable code
  – Straightforward conversion to SQL tables
  – Big problem in the real world: the SQL tables are updated, the E/R documentation is not maintained

• Schema refinement using normal forms
  – Functional dependencies, normalization
Outline

• Stonebraker’s blog on Big Data

• Relational Query Languages

• Database Design

• Functional Dependencies and BCNF
Relational Schema Design

<table>
<thead>
<tr>
<th>Name</th>
<th>SSN</th>
<th>PhoneNumber</th>
<th>City</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fred</td>
<td>123-45-6789</td>
<td>206-555-1234</td>
<td>Seattle</td>
</tr>
<tr>
<td>Fred</td>
<td>123-45-6789</td>
<td>206-555-6543</td>
<td>Seattle</td>
</tr>
<tr>
<td>Joe</td>
<td>987-65-4321</td>
<td>908-555-2121</td>
<td>Westfield</td>
</tr>
</tbody>
</table>

One person may have multiple phones, but lives in only one city.

Primary key is thus (SSN,PhoneNumber)

What is the problem with this schema?
Relational Schema Design

Anomalies:

Redundancy = repeat data
Update anomalies = what if Fred moves to “Bellevue”?
Deletion anomalies = what if Joe deletes his phone number?

<table>
<thead>
<tr>
<th>Name</th>
<th>SSN</th>
<th>PhoneNumber</th>
<th>City</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fred</td>
<td>123-45-6789</td>
<td>206-555-1234</td>
<td>Seattle</td>
</tr>
<tr>
<td>Fred</td>
<td>123-45-6789</td>
<td>206-555-6543</td>
<td>Seattle</td>
</tr>
<tr>
<td>Joe</td>
<td>987-65-4321</td>
<td>908-555-2121</td>
<td>Westfield</td>
</tr>
</tbody>
</table>
Relation Decomposition

Break the relation into two:

<table>
<thead>
<tr>
<th>Name</th>
<th>SSN</th>
<th>PhoneNumber</th>
<th>City</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fred</td>
<td>123-45-6789</td>
<td>206-555-1234</td>
<td>Seattle</td>
</tr>
<tr>
<td>Fred</td>
<td>123-45-6789</td>
<td>206-555-6543</td>
<td>Seattle</td>
</tr>
<tr>
<td>Joe</td>
<td>987-65-4321</td>
<td>908-555-2121</td>
<td>Westfield</td>
</tr>
</tbody>
</table>

Anomalies have gone:

No more repeated data
Easy to move Fred to “Bellevue” (how ?)
Easy to delete all Joe’s phone numbers (how ?)
Relational Schema Design (or Logical Design)

How do we do this systematically?

Start with some relational schema

Find out its \textit{functional dependencies} (FDs)

Use FDs to \textit{normalize} the relational schema
Functional Dependencies (FDs)

Definition

If two tuples agree on the attributes

\[ A_1, A_2, \ldots, A_n \]

then they must also agree on the attributes

\[ B_1, B_2, \ldots, B_m \]

Formally:

\[ A_1, A_2, \ldots, A_n \rightarrow B_1, B_2, \ldots, B_m \]
**Functional Dependencies (FDs)**

**Definition**  \( A_1, ..., A_m \rightarrow B_1, ..., B_n \) holds in \( R \) if:

\[
\forall t, t' \in R,
(t.A_1 = t'.A_1 \land ... \land t.A_m = t'.A_m \Rightarrow t.B_1 = t'.B_1 \land ... \land t.B_n = t'.B_n)
\]

If \( t, t' \) agree here then \( t, t' \) agree here
Example

An FD holds, or does not hold on an instance:

<table>
<thead>
<tr>
<th>EmpID</th>
<th>Name</th>
<th>Phone</th>
<th>Position</th>
</tr>
</thead>
<tbody>
<tr>
<td>E0045</td>
<td>Smith</td>
<td>1234</td>
<td>Clerk</td>
</tr>
<tr>
<td>E3542</td>
<td>Mike</td>
<td>9876</td>
<td>Salesrep</td>
</tr>
<tr>
<td>E1111</td>
<td>Smith</td>
<td>9876</td>
<td>Salesrep</td>
</tr>
<tr>
<td>E9999</td>
<td>Mary</td>
<td>1234</td>
<td>Lawyer</td>
</tr>
</tbody>
</table>

EmpID → Name, Phone, Position
Position → Phone
but not Phone → Position
<table>
<thead>
<tr>
<th>EmpID</th>
<th>Name</th>
<th>Phone</th>
<th>Position</th>
</tr>
</thead>
<tbody>
<tr>
<td>E0045</td>
<td>Smith</td>
<td>1234</td>
<td>Clerk</td>
</tr>
<tr>
<td>E3542</td>
<td>Mike</td>
<td>9876</td>
<td>Salesrep</td>
</tr>
<tr>
<td>E1111</td>
<td>Smith</td>
<td>9876</td>
<td>Salesrep</td>
</tr>
<tr>
<td>E9999</td>
<td>Mary</td>
<td>1234</td>
<td>Lawyer</td>
</tr>
</tbody>
</table>

**Position → Phone**
<table>
<thead>
<tr>
<th>EmpID</th>
<th>Name</th>
<th>Phone</th>
<th>Position</th>
</tr>
</thead>
<tbody>
<tr>
<td>E0045</td>
<td>Smith</td>
<td>1234</td>
<td>Clerk</td>
</tr>
<tr>
<td>E3542</td>
<td>Mike</td>
<td>9876</td>
<td>Salesrep</td>
</tr>
<tr>
<td>E1111</td>
<td>Smith</td>
<td>9876</td>
<td>Salesrep</td>
</tr>
<tr>
<td>E9999</td>
<td>Mary</td>
<td>1234</td>
<td>Lawyer</td>
</tr>
</tbody>
</table>

Example

But not Phone ➔ Position
Example

Do all the FDs hold on this instance?

<table>
<thead>
<tr>
<th>name</th>
<th>category</th>
<th>color</th>
<th>department</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>Gadget</td>
<td>Green</td>
<td>Toys</td>
<td>49</td>
</tr>
<tr>
<td>Tweaker</td>
<td>Gadget</td>
<td>Green</td>
<td>Toys</td>
<td>99</td>
</tr>
</tbody>
</table>

name → color
category → department
color, category → price
**Example**

<table>
<thead>
<tr>
<th>name</th>
<th>category</th>
<th>color</th>
<th>department</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>Gadget</td>
<td>Green</td>
<td>Toys</td>
<td>49</td>
</tr>
<tr>
<td>Tweaker</td>
<td>Gadget</td>
<td>Black</td>
<td>Toys</td>
<td>99</td>
</tr>
<tr>
<td>Gizmo</td>
<td>Stationary</td>
<td>Green</td>
<td>Office-suppl.</td>
<td>59</td>
</tr>
</tbody>
</table>

What about this one?
Terminology

FD holds or does not hold on an instance

If we can be sure that every instance of $R$ will be one in which a given FD is true, then we say that $R$ satisfies the FD

If we say that $R$ satisfies an FD $F$, we are stating a constraint on $R$
An Interesting Observation

If all these FDs are true:

- name $\rightarrow$ color
- category $\rightarrow$ department
- color, category $\rightarrow$ price

Then this FD also holds:

- name, category $\rightarrow$ price

If we find out from application domain that a relation satisfies some FDs, it doesn’t mean that we found all the FDs that it satisfies! There could be more FDs implied by the ones we have.
Closure of a set of Attributes

Given a set of attributes $A_1, \ldots, A_n$

The closure, $\{A_1, \ldots, A_n\}^+ = \text{the set of attributes } B$

s.t. $A_1, \ldots, A_n \rightarrow B$

Example:

1. name $\rightarrow$ color
2. category $\rightarrow$ department
3. color, category $\rightarrow$ price

Closures:

$name^+ = \{\text{name, color}\}$

$\{\text{name, category}\}^+ = \{\text{name, category, color, department, price}\}$

$\text{color}^+ = \{\text{color}\}$
Closure Algorithm

X={A1, …, An}.

Repeat until X doesn’t change do:
    if B₁, …, Bₙ → C is a FD and B₁, …, Bₙ are all in X
    then add C to X.

Example:

1. name → color
2. category → department
3. color, category → price

{name, category}⁺ =

{   }

{   }
Closure Algorithm

\[ X = \{A_1, \ldots, A_n\}. \]

Repeat until \( X \) doesn’t change do:

1. \( B_1, \ldots, B_n \rightarrow C \) is a FD and \( B_1, \ldots, B_n \) are all in \( X \)
2. then add \( C \) to \( X \).

Example:

1. name \( \rightarrow \) color
2. category \( \rightarrow \) department
3. color, category \( \rightarrow \) price

\{name, category\}^+ =
\{ name, category, color, department, price \}
Closure Algorithm

X = {A₁, ..., An}.

Repeat until X doesn't change do:
  if B₁, ..., Bₙ \rightarrow C is a FD and B₁, ..., Bₙ are all in X
  then add C to X.

Example:

1. name \rightarrow color
2. category \rightarrow department
3. color, category \rightarrow price

\{name, category\}⁺ =
  \{ name, category, color, department, price \}

Hence: name, category \rightarrow color, department, price
Example

In class:

\[ R(A,B,C,D,E,F) \]

\[
\begin{align*}
A, B & \rightarrow C \\
A, D & \rightarrow E \\
B & \rightarrow D \\
A, F & \rightarrow B
\end{align*}
\]

Compute \( \{A,B\}^+ \) \( X = \{A, B, \} \)

Compute \( \{A, F\}^+ \) \( X = \{A, F, \} \)
Example

In class:

R(A,B,C,D,E,F)

Compute \( \{A, B\}^+ \) \( X = \{A, B, C, D, E\} \)

Compute \( \{A, F\}^+ \) \( X = \{A, F, \} \)
Example

In class:

\[ R(A,B,C,D,E,F) \]

\[
\begin{align*}
A, B & \rightarrow C \\
A, D & \rightarrow E \\
B & \rightarrow D \\
A, F & \rightarrow B
\end{align*}
\]

Compute \{A,B\}^+ \quad X = \{A, B, C, D, E\}

Compute \{A, F\}^+ \quad X = \{A, F, B, C, D, E\}
Example

In class:

\[ R(A, B, C, D, E, F) \]

Compute \( \{A, B\}^+ \)
\[ X = \{A, B, C, D, E\} \]

Compute \( \{A, F\}^+ \)
\[ X = \{A, F, B, C, D, E\} \]

What is the key of \( R \)?
Find all FD’s implied by:

- A, B → C
- A, D → B
- B → D
Practice at Home

Find all FD’s implied by:

\[
\begin{array}{ccc}
A, B & \rightarrow & C \\
A, D & \rightarrow & B \\
B & \rightarrow & D \\
\end{array}
\]

Step 1: Compute \( X^+ \), for every \( X \):

\[
\begin{align*}
AB^+ &= ABCD, & AC^+ &= AC, & AD^+ &= ABCD, & \\
& & BC^+ &= BCD, & BD^+ &= BD, & CD^+ &= CD \\
ABC^+ &= ABD^+ = ACD^+ = ABCD \quad & \text{(no need to compute— why ?)} & \\
BCD^+ &= BCD, & ABCD^+ &= ABCD
\end{align*}
\]
Practice at Home

Find all FD’s implied by:

A, B → C
A, D → B
B → D

Step 1: Compute $X^+$, for every $X$:

$A^+ = A$, $B^+ = BD$, $C^+ = C$, $D^+ = D$

$AB^+ = ABCD$, $AC^+ = AC$, $AD^+ = ABCD$

$BC^+ = BCD$, $BD^+ = BD$, $CD^+ = CD$

$ABC^+ = ABD^+ = ACD^+ = ABCD$ (no need to compute— why ?)

$BCD^+ = BCD$, $ABCD^+ = ABCD$

Step 2: Enumerate all FD’s $X \rightarrow Y$, s.t. $Y \subseteq X^+$ and $X \cap Y = \emptyset$:

$AB \rightarrow CD$, $AD \rightarrow BC$, $ABC \rightarrow D$, $ABD \rightarrow C$, $ACD \rightarrow B$
Keys

• A **superkey** is a set of attributes $A_1, \ldots, A_n$ s.t. for any other attribute $B$, we have $A_1, \ldots, A_n \rightarrow B$

• A **key** is a minimal superkey
  – A superkey and for which no subset is a superkey
Computing (Super)Keys

- For all sets X, compute $X^+$

- If $X^+ = [\text{all attributes}]$, then X is a superkey

- Try only the minimal X’s to get the keys
Example

Product(name, price, category, color)

name, category \rightarrow price
category \rightarrow color

What is the key?
Example

Product(name, price, category, color)

(name, category) \rightarrow price

category \rightarrow color

What is the key?

(name, category) + = \{ name, category, price, color \}

Hence (name, category) is a key
Key or Keys?

Can we have more than one key?

Given R(A,B,C) define FD’s s.t. there are two or more keys
Key or Keys?

Can we have more than one key?

Given R(A,B,C) define FD’s s.t. there are two or more keys

A → B
B → C
C → A

or

AB → C
BC → A

or

A → BC
B → AC

What are the keys here?
Eliminating Anomalies

<table>
<thead>
<tr>
<th>Name</th>
<th>SSN</th>
<th>PhoneNumber</th>
<th>City</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fred</td>
<td>123-45-6789</td>
<td>206-555-1234</td>
<td>Seattle</td>
</tr>
<tr>
<td>Fred</td>
<td>123-45-6789</td>
<td>206-555-6543</td>
<td>Seattle</td>
</tr>
<tr>
<td>Joe</td>
<td>987-65-4321</td>
<td>908-555-2121</td>
<td>Westfield</td>
</tr>
<tr>
<td>Joe</td>
<td>987-65-4321</td>
<td>908-555-1234</td>
<td>Westfield</td>
</tr>
</tbody>
</table>

SSN → Name, City

What is the key?

What is the key?

Suggest a rule for decomposing the table to eliminate anomalies
Eliminating Anomalies

Main idea:

• $X \rightarrow A$ is OK if $X$ is a (super)key

• $X \rightarrow A$ is not OK otherwise
  – Need to decompose the table, but how?
There are no “bad” FDs:

**Definition.** A relation R is in BCNF if:

Whenever $X \rightarrow B$ is a non-trivial dependency, then $X$ is a superkey.

Equivalently:

**Definition.** A relation R is in BCNF if:

$\forall\ X$, either $X^+ = X$ or $X^+ = [\text{all attributes}]$
BCNF Decomposition Algorithm

Normalize(R)

find X s.t.: X ≠ X⁺ ≠ [all attributes]

if (not found) then “R is in BCNF”

let Y = X⁺ - X; Z = [all attributes] - X⁺

decompose R into R₁(X ∪ Y) and R₂(X ∪ Z)

Normalize(R₁); Normalize(R₂);
Example

The only key is: \{SSN, PhoneNumber\}
Hence \textbf{SSN} \rightarrow \textbf{Name, City} is a “bad” dependency
In other words:
\textbf{SSN+} = \textbf{Name, City} and is neither \textbf{SSN} nor \textbf{All Attributes}
Example BCNF Decomposition

<table>
<thead>
<tr>
<th>Name</th>
<th>SSN</th>
<th>City</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fred</td>
<td>123-45-6789</td>
<td>Seattle</td>
</tr>
<tr>
<td>Joe</td>
<td>987-65-4321</td>
<td>Westfield</td>
</tr>
</tbody>
</table>

SSN \( \rightarrow \) Name, City

<table>
<thead>
<tr>
<th>SSN</th>
<th>PhoneNumber</th>
</tr>
</thead>
<tbody>
<tr>
<td>123-45-6789</td>
<td>206-555-1234</td>
</tr>
<tr>
<td>123-45-6789</td>
<td>206-555-6543</td>
</tr>
<tr>
<td>987-65-4321</td>
<td>908-555-2121</td>
</tr>
<tr>
<td>987-65-4321</td>
<td>908-555-1234</td>
</tr>
</tbody>
</table>

Let’s check anomalies:
Redundancy?  
Update?  
Delete?
Find X s.t.: $X \neq X^+ \neq [\text{all attributes}]$

Example BCNF Decomposition

Person(name, SSN, age, hairColor, phoneNumber)

- SSN $\rightarrow$ name, age
- age $\rightarrow$ hairColor
Find X s.t.: $X \neq X^+ \neq \{\text{all attributes}\}$

**Example BCNF Decomposition**

Person(name, SSN, age, hairColor, phoneNumber)

SSN $\rightarrow$ name, age

age $\rightarrow$ hairColor

**Iteration 1:** Person: SSN$^+$ = SSN, name, age, hairColor

Decompose into: $P(\text{SSN, name, age, hairColor})$

Phone(SSN, phoneNumber)
Find X s.t.: X $\neq X^+ \neq$ [all attributes]

## Example BCNF Decomposition

### Person(name, SSN, age, hairColor, phoneNumber)

- SSN $\rightarrow$ name, age
- age $\rightarrow$ hairColor

#### Iteration 1:

**Person:** SSN+ = SSN, name, age, hairColor

Decompose into:

- $P($SSN, name, age, hairColor$)$
- Phone(SSN, phoneNumber)

#### Iteration 2:

**P:** age+ = age, hairColor

Decompose:

- People(SSN, name, age)
- Hair(age, hairColor)
- Phone(SSN, phoneNumber)

What are the keys?
Example BCNF Decomposition

Person(name, SSN, age, hairColor, phoneNumber)

SSN → name, age
age → hairColor

Find X s.t.: X ≠ X⁺ ≠ [all attributes]

Iteration 1: Person: SSN⁺ = SSN, name, age, hairColor
Decompose into: P(SSN, name, age, hairColor)
Phone(SSN, phoneNumber)

Iteration 2: P: age⁺ = age, hairColor
Decompose: People(SSN, name, age)
Hair(age, hairColor)
Phone(SSN, phoneNumber)
R(A,B,C,D)

Practice at Home

R(A,B,C,D)
A⁺ = ABC ≠ ABCD
Practice at Home

What are the keys?

R(A,B,C,D)

A⁺ = ABC ≠ ABCD

R₁(A,B,C)
B⁺ = BC ≠ ABC

R₁₁(B,C)

R₁₂(A,B)

R₂(A,D)

What happens if in R we first pick B⁺ ? Or AB⁺ ?
Schema Refinements = Normal Forms

• 1st Normal Form = all tables are flat
• 2nd Normal Form = obsolete
• Boyce Codd Normal Form = today
• 3rd Normal Form = see book