CSC 261/461 – Database Systems
Lecture 14

Fall 2017
Announcement

• Midterm on this Wednesday

• Please arrive 5 minutes early; We will start at 12:30 pm sharp

• I will post a sitting arrangement today.

• Bluehive (CIRC) account

• Start studying for MT
Are you ready?

YEAH, IF YOU COULD STOP SCHEDULING THREE EXAMS ON THE SAME DAY

THAT'D BE GREAT
Night before the exam

Night before an exam

What my teacher thinks I do

What my parents think I do

What my mates think I do

What I think I do

What I actually do

facebook

Twitter

YouTube
During the exam
Or Finishing too early

I HAVE NO IDEA
WHAT I AM DOING
MIDTERM REVIEW
Chapters to Read

• Chapter 1
• Chapter 2
• Chapter 3
• Chapter 4 (Just the basics. Only ISA relationships. Even studying the slides is fine)
• Chapter 5, 6, 7 (SQL)
• Chapter 8 (8.1-8.3)
• Chapter 9
• Chapter 14 (14.1-14.5)
• Chapter 15 (15.1-15.3)
Exam Structure

• Problem 1 (20 pts)  
  – Short answers, True/False, or One liner
• Other problems (40 pts)

• Total: 60 pts

• 2 bonus points for writing the class id correctly.
Time distribution

• Time crunch
  – Not really
  – Should take more than 60 minutes to finish.
  – Not as relaxed as the quiz
MATERIALS COVERED
Questions to ponder

• Why not Lists? Why database?

• How related tables avoid problems associated with lists?
Problems with Lists

• Multiple Concepts or Themes:
  – Microsoft Excel vs Microsoft Access

• Redundancy

• Anomalies:
  – Deletion anomalies
  – Update anomalies
  – Insertion anomalies
List vs Database

• Lists do not provide information about relations!

• Break lists into tables

• Facilitates:
  – Insert
  – Delete
  – Update

• Input and Output interface (Forms and Reports)

• Query!
Again, Why database?

- To store data
- To provide structure
- Mechanism for querying, creating, modifying and deleting data.
- CRED (Create, Read, Update, Delete)
- Store information and relationships

- Database Schema vs. Database State
Simplified Database System Environment

Figure 1.1
A simplified database system environment.
General form SQL

```
SELECT S
FROM R_1,...,R_n
WHERE C_1
GROUP BY a_1,...,a_k
HAVING C_2
```

Evaluation steps:
1. Evaluate **FROM-WHERE**: apply condition $C_1$ on the attributes in $R_1,...,R_n$
2. **GROUP BY** the attributes $a_1,...,a_k$
3. Apply condition $C_2$ to each group (may have aggregates)
4. Compute aggregates in $S$ and return the result
Grouping and Aggregation

Semantics of the query:

1. Compute the FROM and WHERE clauses

2. Group by the attributes in the GROUP BY

3. Compute the SELECT clause: grouped attributes and aggregates
1. Compute the **FROM** and **WHERE** clauses

```sql
SELECT product, SUM(price*quantity) AS TotalSales
FROM Purchase
WHERE date > '10/1/2005'
GROUP BY product
```

<table>
<thead>
<tr>
<th>Product</th>
<th>Date</th>
<th>Price</th>
<th>Quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bagel</td>
<td>10/21</td>
<td>1</td>
<td>20</td>
</tr>
<tr>
<td>Bagel</td>
<td>10/25</td>
<td>1.50</td>
<td>20</td>
</tr>
<tr>
<td>Banana</td>
<td>10/3</td>
<td>0.5</td>
<td>10</td>
</tr>
<tr>
<td>Banana</td>
<td>10/10</td>
<td>1</td>
<td>10</td>
</tr>
</tbody>
</table>
2. Group by the attributes in the **GROUP BY**

```sql
SELECT   product, SUM(price*quantity) AS TotalSales
FROM     Purchase
WHERE    date > '10/1/2005'
GROUP BY product
```
3. Compute the **SELECT** clause: grouped attributes and aggregates

```sql
SELECT product, SUM(price*quantity) AS TotalSales
FROM Purchase
WHERE date > '10/1/2005'
GROUP BY product
```

<table>
<thead>
<tr>
<th>Product</th>
<th>Date</th>
<th>Price</th>
<th>Quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bagel</td>
<td>10/21</td>
<td>1</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td>10/25</td>
<td>1.5</td>
<td>20</td>
</tr>
<tr>
<td>Banana</td>
<td>10/3</td>
<td>0.5</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>10/10</td>
<td>1</td>
<td>10</td>
</tr>
</tbody>
</table>

![SELECT](image)

<table>
<thead>
<tr>
<th>Product</th>
<th>TotalSales</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bagel</td>
<td>50</td>
</tr>
<tr>
<td>Banana</td>
<td>15</td>
</tr>
</tbody>
</table>
HAVING Clause

`SELECT product, SUM(price*quantity) FROM Purchase WHERE date > '10/1/2005' GROUP BY product HAVING SUM(quantity) > 100`

Same query as before, except that we consider only products that have more than 100 buyers.

HAVING clauses contain conditions on aggregates.

Whereas WHERE clauses condition on individual tuples...
Null Values

Unexpected behavior:

```
SELECT * 
FROM Person 
WHERE age < 25 OR age >= 25 
```

Some Persons are not included!
Null Values

Can test for NULL explicitly:

- $x$ IS NULL
- $x$ IS NOT NULL

```sql
SELECT *
FROM Person
WHERE age < 25 OR age >= 25
  OR age IS NULL
```

Now it includes all Persons!
By default, joins in SQL are “inner joins”:

Product(name, category)
Purchase(prodName, store)

SELECT Product.name, Purchase.store
FROM Product
JOIN Purchase ON Product.name = Purchase.prodName

SELECT Product.name, Purchase.store
FROM Product, Purchase
WHERE Product.name = Purchase.prodName

Both equivalent: Both INNER JOINS!
By default, joins in SQL are “inner joins”:

```
Product(name, category)
Purchase(prodName, store)
```

```
SELECT Product.name, Purchase.store
FROM   Product
JOIN   Purchase ON Product.name = Purchase.prodName
```

```
SELECT Product.name, Purchase.store
FROM   Product, Purchase
WHERE  Product.name = Purchase.prodName
```

However: Products that never sold (with no Purchase tuple) will be lost!
INNER JOIN:

<table>
<thead>
<tr>
<th>Product</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
</tr>
<tr>
<td>Gizmo</td>
</tr>
<tr>
<td>Camera</td>
</tr>
<tr>
<td>OneClick</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Purchase</th>
</tr>
</thead>
<tbody>
<tr>
<td>prodName</td>
</tr>
<tr>
<td>Gizmo</td>
</tr>
<tr>
<td>Camera</td>
</tr>
<tr>
<td>Camera</td>
</tr>
</tbody>
</table>

SELECT Product.name, Purchase.store
FROM Product
INNER JOIN Purchase
ON Product.name = Purchase.prodName

Note: another equivalent way to write an INNER JOIN!
**LEFT OUTER JOIN:**

**Product**

<table>
<thead>
<tr>
<th>name</th>
<th>category</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>gadget</td>
</tr>
<tr>
<td>Camera</td>
<td>Photo</td>
</tr>
<tr>
<td>OneClick</td>
<td>Photo</td>
</tr>
</tbody>
</table>

**Purchase**

<table>
<thead>
<tr>
<th>prodName</th>
<th>store</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>Wiz</td>
</tr>
<tr>
<td>Camera</td>
<td>Ritz</td>
</tr>
<tr>
<td>Camera</td>
<td>Wiz</td>
</tr>
</tbody>
</table>

**SELECT** `Product.name, Purchase.store`  
**FROM** `Product`  
**LEFT OUTER JOIN** `Purchase`  
**ON** `Product.name = Purchase.prodName`
Other Outer Joins

• Left outer join:
  – Include the left tuple even if there’s no match

• Right outer join:
  – Include the right tuple even if there’s no match

• Full outer join:
  – Include the both left and right tuples even if there’s no match
Also read

• Triggers
• Views
• Operations on Databases
  – Create, Drop, and Alter
• Operations of Tables
  – Insert, Delete and Update
• Constraint:
  – Key constraint
  – Foreign key
  – On Delete cascade, Set Null, Set Default;
Database Design Process

1. **Requirements analysis**
   - What is going to be stored?
   - How is it going to be used?
   - What are we going to do with the data?
   - Who should access the data?

Technical and non-technical people are involved.
2. Conceptual Design

- A high-level description of the database
- Sufficiently precise that technical people can understand it
- But, not so precise that non-technical people can’t participate

This is where E/R fits in.
Database Design Process

3. Implementation:

- Logical Database Design
- Physical Database Design
- Security Design
Database Design Process

1. Requirements Analysis
2. Conceptual Design
3. Logical, Physical, Security, etc.

E/R Model & Diagrams used

E/R is a visual syntax for DB design which is precise enough for technical points, but abstracted enough for non-technical people.

This process is iterated many times.

CSC 261, Fall 2017, UR
• After the requirements have been gathered, they are transformed into an Entity Relationship (E-R) Data Model

• E-R Models consist of
  1. Entities
  2. Attributes
     a) Identifiers (Keys)
     b) Non-key attributes
  3. Relationships
1. E/R BASICS: ENTITIES & RELATIONS
• An entity set has **attributes**  
  – Represented by ovals attached to an entity set

Shapes **are** important. Colors **are not**.
• A **key** is a **minimal** set of attributes that uniquely identifies an entity.

Denote elements of the primary key by underlining.

Here, \{name, category\} is **not** a key (it is not **minimal**).

The E/R model forces us to designate a single **primary** key, though there may be multiple candidate keys.
• A relationship connects two or more entity sets.
• It is represented by a diamond, with lines to each of the entity sets involved.
• The degree of the relationship defines the number of entity classes that participate in the relationship
  – Degree 1 is a unary relationship
  – Degree 2 is a binary relationship
  – Degree 3 is a ternary relationship
Conceptual Unary Relationship

Person \(\rightarrow\) Marries
Conceptual Binary Relationship

Person — Owns — Cars
Conceptual Ternary Relationship

- Patient
- Prescription
- Drug
- Doctor

CSC 261, Fall 2017, UR
The R in E/R: Relationships

• E, R and A together:

![Diagram showing relationships between Product and Company]
What is a Relationship?

A relationship between entity sets P and C is a subset of all possible pairs of entities in P and C, with tuples uniquely identified by P and C’s keys.
A relationship between entity sets P and C is a \textit{subset of all possible pairs of entities in P and C}, with tuples uniquely identified by \textit{P and C's keys}.
What is a Relationship?

A relationship between entity sets $P$ and $C$ is a subset of all possible pairs of entities in $P$ and $C$, with tuples uniquely identified by $P$ and $C$'s keys.

<table>
<thead>
<tr>
<th>Company</th>
<th>Product</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name</td>
</tr>
<tr>
<td>GizmoWorks</td>
<td>Gizmo</td>
</tr>
<tr>
<td>GizmoLite</td>
<td>GizmoLite</td>
</tr>
<tr>
<td>Gadget</td>
<td>Gadget</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Company C</th>
<th>Product P</th>
</tr>
</thead>
<tbody>
<tr>
<td>C.name</td>
<td>P.name</td>
</tr>
<tr>
<td>GizmoWorks</td>
<td>Gizmo</td>
</tr>
<tr>
<td>GizmoWorks</td>
<td>GizmoLite</td>
</tr>
<tr>
<td>GizmoWorks</td>
<td>Gadget</td>
</tr>
<tr>
<td>GadgetCorp</td>
<td>Gizmo</td>
</tr>
<tr>
<td>GadgetCorp</td>
<td>GizmoLite</td>
</tr>
<tr>
<td>GadgetCorp</td>
<td>Gadget</td>
</tr>
</tbody>
</table>
- Relationships may have attributes as well.

For example: “since” records when company started making a product

Note: “since” is implicitly unique per pair here! Why?
• Summary

**Figure 3.14**
Summary of the notation for ER diagrams.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Entity</td>
</tr>
<tr>
<td></td>
<td>Weak Entity</td>
</tr>
<tr>
<td></td>
<td>Relationship</td>
</tr>
<tr>
<td></td>
<td>Attribute</td>
</tr>
<tr>
<td></td>
<td>Key Attribute</td>
</tr>
<tr>
<td></td>
<td>Multivalued Attribute</td>
</tr>
<tr>
<td></td>
<td>Composite Attribute</td>
</tr>
<tr>
<td></td>
<td>Derived Attribute</td>
</tr>
<tr>
<td>$E_1$</td>
<td>$R$</td>
</tr>
<tr>
<td>$E_2$</td>
<td>Total Participation of $E_2$ in $R$</td>
</tr>
<tr>
<td>$E_1$</td>
<td>$R$</td>
</tr>
<tr>
<td>$E_2$</td>
<td>Cardinality Ratio 1: $N$ for $E_1$: $E_2$ in $R$</td>
</tr>
<tr>
<td>$R$</td>
<td>Structural Constraint ($\text{min}$, $\text{max}$) on Participation of $E$ in $R$</td>
</tr>
</tbody>
</table>
Design Theory (ER model to Relations)
CREATE TABLE Employees (  
  ssn char(11),  
  name varchar(30),  
  lot Integer,  
  PRIMARY KEY (ssn))
Other Conversions (ER model to Tables)

• Relationships:
  • Many to Many
  • One to Many
  • One to One
• One customer can have at max 2 loans. One loan can be given to multiple customers.

What it really means:
– One customer can have \((0,2)\) loans
– One loan can be given to \((1,n)\) customer
– This is a many to many scenario
Crow’s foot Notation

Chen Notation
FUNCTIONAL DEPENDENCIES
Prime and Non-prime attributes

- A **Prime attribute** must be a member of *some* candidate key.

- A **Nonprime attribute** is not a prime attribute—that is, it is not a member of any candidate key.

![Diagram](image-url)
Now that we know how to find FDs, it’s a straight-forward process:

1. Search for “bad” FDs

2. If there are any, then keep decomposing the table into sub-tables until no more bad FDs

3. When done, the database schema is normalized
Main idea is that we define “good” and “bad” FDs as follows:

- $X \rightarrow A$ is a “good FD” if $X$ is a (super)key
  - In other words, if $A$ is the set of all attributes

- $X \rightarrow A$ is a “bad FD” otherwise

We will try to eliminate the “bad” FDs!
  - Via normalization
Normalizing into 2NF and 3NF

(a) EMP_PROJ

FD1
FD2
FD3

2NF Normalization

(b) EMP_DEPT

3NF Normalization
Figure 14.12 Normalization into 2NF and 3NF. (a) The LOTS relation with its functional dependencies FD1 through FD4. (b) Decomposing into the 2NF relations LOTS1 and LOTS2. (c) Decomposing LOTS1 into the 3NF relations LOTS1A and LOTS1B. (d) Progressive normalization of LOTS into a 3NF design.
Normal Forms Defined Informally

- **1st normal form**
  - All attributes depend on the key
- **2nd normal form**
  - All attributes depend on the whole key
- **3rd normal form**
  - All attributes depend on nothing but the key
• A relation schema R is in **second normal form (2NF)** if every non-prime attribute A in R is fully functionally dependent on *every* key of R

• A relation schema R is in **third normal form (3NF)** if it is in 2NF *and* no non-prime attribute A in R is transitively dependent on *any* key of R
1. BOYCE-CODD NORMAL FORM
Figure 14.14 A relation TEACH that is in 3NF but not in BCNF

- Two FDs exist in the relation TEACH:
  - fd₁: \{ student, course \} → instructor
  - fd₂: instructor → course
- \{student, course\} is a candidate key for this relation
- So this relation is in 3NF but not in BCNF
- A relation NOT in BCNF should be decomposed so as to meet this property,
  - while possibly forgoing the preservation of all functional dependencies in the decomposed relations.
Three possible decompositions for relation TEACH

- **D1**: \{student, instructor\} and \{student, course\}
- **D2**: \{course, instructor\} and \{course, student\}
- **D3**: \{instructor, course\} and \{instructor, student\} ✔
**ALTERNATIVE DEFINITION of 3NF:** We can restate the definition as:

A relation schema R is in *third normal form* (3NF) if, whenever a nontrivial FD $X \rightarrow A$ holds in R, either

a) $X$ is a superkey of R or

b) $A$ is a prime attribute of R

The condition (b) takes care of the dependencies that “slip through” (are allowable to) 3NF but are “caught by” BCNF which we discuss next.
1. BOYCE-CODD NORMAL FORM
What you will learn about in this section

1. Boyce-Codd Normal Form

2. The BCNF Decomposition Algorithm
5. BCNF (Boyce-Codd Normal Form)

- **Definition of 3NF:**
  - A relation schema R is in 3NF if, whenever a nontrivial FD $X \rightarrow A$ holds in R, either
    - a) $X$ is a superkey of R or
    - b) $A$ is a prime attribute of R

- A relation schema R is in **Boyce-Codd Normal Form (BCNF)** if whenever an FD $X \rightarrow A$ holds in R, then
  - a) $X$ is a superkey of R
  - b) There is no $b$

- Each normal form is strictly stronger than the previous one
  - Every 2NF relation is in 1NF
  - Every 3NF relation is in 2NF
  - Every BCNF relation is in 3NF
Boyce-Codd normal form

(a) LOTS1A

<table>
<thead>
<tr>
<th>Property_id#</th>
<th>County_name</th>
<th>Lot#</th>
<th>Area</th>
</tr>
</thead>
</table>

FD1

FD2

FD5

BCNF Normalization

LOTS1AX

<table>
<thead>
<tr>
<th>Property_id#</th>
<th>Area</th>
<th>Lot#</th>
</tr>
</thead>
</table>

LOTS1AY

<table>
<thead>
<tr>
<th>Area</th>
<th>County_name</th>
</tr>
</thead>
</table>

(b) $R$

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
</table>

FD1

FD2

Figure 14.13

Boyce-Codd normal form. (a) BCNF normalization of LOTS1A with the functional dependency FD2 being lost in the decomposition. (b) A schematic relation with FDs; it is in 3NF, but not in BCNF due to the f.d. $C \rightarrow B$. 
A relation `TEACH` that is in 3NF but not in BCNF

- Two FDs exist in the relation `TEACH`:
  - `{student, course}` → instructor
  - instructor → course

- `{student, course}` is a candidate key for this relation
- So this relation is in 3NF but not in BCNF
- A relation NOT in BCNF should be decomposed

<table>
<thead>
<tr>
<th>Student</th>
<th>Course</th>
<th>Instructor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Narayan</td>
<td>Database</td>
<td>Mark</td>
</tr>
<tr>
<td>Smith</td>
<td>Database</td>
<td>Navathe</td>
</tr>
<tr>
<td>Smith</td>
<td>Operating Systems</td>
<td>Ammar</td>
</tr>
<tr>
<td>Smith</td>
<td>Theory</td>
<td>Schulman</td>
</tr>
<tr>
<td>Wallace</td>
<td>Database</td>
<td>Mark</td>
</tr>
<tr>
<td>Wallace</td>
<td>Operating Systems</td>
<td>Ahamad</td>
</tr>
<tr>
<td>Wong</td>
<td>Database</td>
<td>Omiecinski</td>
</tr>
<tr>
<td>Zelaya</td>
<td>Database</td>
<td>Navathe</td>
</tr>
<tr>
<td>Narayan</td>
<td>Operating Systems</td>
<td>Ammar</td>
</tr>
</tbody>
</table>
Achieving the BCNF by Decomposition

• Three possible decompositions for relation TEACH
  – D1: \{\text{student, instructor}\} and \{\text{student, course}\}

  – D2: \{\text{course, instructor}\} and \{\text{course, student}\}

  ✓ D3: \{\text{instructor, course}\} and \{\text{instructor, student}\}
BCNF is a simple condition for removing anomalies from relations:

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

if \( \{X_1, \ldots, X_n\} \rightarrow A \) is a *non-trivial* FD in R

then \( \{X_1, \ldots, X_n\} \) is a superkey for R

In other words: there are no “bad” FDs
Example

<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>

\[\{\text{SSN}\} \rightarrow \{\text{Name}, \text{City}\}\]

This FD is \textit{bad} because it is \textbf{not} a superkey

What is the key? \{\text{SSN, PhoneNumber}\}

\[\Rightarrow \textbf{Not in BCNF}\]
### Example

<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>Madison</td>
</tr>
</tbody>
</table>

Let’s check anomalies:
- Redundancy?
- Update?
- Delete?

This FD is now **good** because it is the key.

{SSN} → {Name, City}

Now in BCNF!
BCNF Decomposition

BCNFDcomp(R):

If \(X \rightarrow A\) causes BCNF violation:
Decompose R into
\(R1 = XA\)
\(R2 = R - A\)
(Note: X is present in both R1 and R2)

Return BCNFDcomp(R1), BCNFDcomp(R2)
Example

BCNFDcomp(R):

If $X \rightarrow A$ causes BCNF violation:
Decompose $R$ into
$R_1 = XA$
$R_2 = R - A$
(Note: $X$ is present in both $R_1$ and $R_2$)

Return BCNFDcomp($R_1$),
BCNFDcomp($R_2$)
Example

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

\[ R_1(A,B,C,D) \]
\[ \{C\}^+ = \{C,D\} \neq \{A,B,C,D\} \]

\[ R_{11}(C,D) \]
\[ R_{12}(A,B,C) \]

\[ R_2(A,E) \]

\{A\} \rightarrow \{B,C\}
\{C\} \rightarrow \{D\}
2. DECOMPOSITIONS
1. We saw that redundancies in the data (“bad FDs”) can lead to data anomalies.

2. We developed mechanisms to detect and remove redundancies by decomposing tables into BCNF.
   1. BCNF decomposition is standard practice—very powerful & widely used!

3. However, sometimes decompositions can lead to more subtle unwanted effects…

When does this happen?
Decompositions in General

\[ R(A_1, \ldots, A_n, B_1, \ldots, B_m, C_1, \ldots, C_p) \]

\[ R_1(A_1, \ldots, A_n, B_1, \ldots, B_m) \]

\[ R_2(A_1, \ldots, A_n, C_1, \ldots, C_p) \]

\[ R_1 = \text{the projection of } R \text{ on } A_1, \ldots, A_n, B_1, \ldots, B_m \]

\[ R_2 = \text{the projection of } R \text{ on } A_1, \ldots, A_n, C_1, \ldots, C_p \]
### Theory of Decomposition

- **Name** | **Price** | **Category**
- Gizmo | 19.99 | Gadget
- OneClick | 24.99 | Camera
- Gizmo | 19.99 | Camera

Sometimes a decomposition is “correct”

I.e. it is a **Lossless decomposition**

---

**CSC 261, Fall 2017, UR**
**Lossy Decomposition**

<table>
<thead>
<tr>
<th>Name</th>
<th>Price</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>19.99</td>
<td>Gadget</td>
</tr>
<tr>
<td>OneClick</td>
<td>24.99</td>
<td>Camera</td>
</tr>
<tr>
<td>Gizmo</td>
<td>19.99</td>
<td>Camera</td>
</tr>
</tbody>
</table>

However, sometimes it isn’t.

What’s wrong here?

<table>
<thead>
<tr>
<th>Name</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>Gadget</td>
</tr>
<tr>
<td>OneClick</td>
<td>Camera</td>
</tr>
<tr>
<td>Gizmo</td>
<td>Camera</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Price</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>19.99</td>
<td>Gadget</td>
</tr>
<tr>
<td>24.99</td>
<td>Camera</td>
</tr>
<tr>
<td>19.99</td>
<td>Camera</td>
</tr>
</tbody>
</table>
A decomposition $R$ to $(R_1, R_2)$ is **lossless** if $R = R_1 \text{ Join } R_2$.
Lossless Decompositions

BCNF decomposition is always lossless. Why?

If \{A_1, \ldots, A_n\} \rightarrow \{B_1, \ldots, B_m\}
Then the decomposition is lossless

Note: don’t need \{A_1, \ldots, A_n\} \rightarrow \{C_1, \ldots, C_p\}
A relation TEACH that is in 3NF but not in BCNF

<table>
<thead>
<tr>
<th>Student</th>
<th>Course</th>
<th>Instructor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Narayan</td>
<td>Database</td>
<td>Mark</td>
</tr>
<tr>
<td>Smith</td>
<td>Database</td>
<td>Navathe</td>
</tr>
<tr>
<td>Smith</td>
<td>Operating Systems</td>
<td>Ammar</td>
</tr>
<tr>
<td>Smith</td>
<td>Theory</td>
<td>Schulman</td>
</tr>
<tr>
<td>Wallace</td>
<td>Database</td>
<td>Mark</td>
</tr>
<tr>
<td>Wallace</td>
<td>Operating Systems</td>
<td>Ahamad</td>
</tr>
<tr>
<td>Wong</td>
<td>Database</td>
<td>Omiecinski</td>
</tr>
<tr>
<td>Zelaya</td>
<td>Database</td>
<td>Navathe</td>
</tr>
<tr>
<td>Narayan</td>
<td>Operating Systems</td>
<td>Ammar</td>
</tr>
</tbody>
</table>

• Two FDs exist in the relation TEACH:
  - \{\text{student, course}\} \rightarrow \text{instructor}
  - \text{instructor} \rightarrow \text{course}

• \{\text{student, course}\} is a candidate key for this relation
• So this relation is in 3NF but not in BCNF
• A relation NOT in BCNF should be decomposed
Achieving the BCNF by Decomposition (2)

- Three possible decompositions for relation TEACH
  - D1: \{student, instructor\} and \{student, course\}
  - D2: \{course, instructor\} and \{course, student\}
  - D3: \{instructor, course\} and \{instructor, student\}
A problem with BCNF

**Problem**: To enforce a FD, must reconstruct original relation—*on each insert!*
A Problem with BCNF

We do a BCNF decomposition on a “bad” FD:
\{Unit\}+ = \{Unit, Company\}

We lose the FD \{Company, Product\} → \{Unit\}!!
So Why is that a Problem?

No problem so far. All local FD’s are satisfied.

Let’s put all the data back into a single table again:

Violates the FD \{Company, Product\} → \{Unit\}!!
The Problem

• We started with a table R and FDs F

• We decomposed R into BCNF tables R₁, R₂, …
  with their own FDs F₁, F₂, …

• We insert some tuples into each of the relations—which
  satisfy their local FDs but when reconstruct it violates some
  FD across tables!

Practical Problem: To enforce FD, must
reconstruct R—on each insert!
Possible Solutions

• Various ways to handle so that decompositions are all lossless / no FDs lost
  – For example 3NF- stop short of full BCNF decompositions.

• Usually a tradeoff between redundancy / data anomalies and FD preservation...

BCNF still most common- with additional steps to keep track of lost FDs...
• **Problem Set 5** *(Really important)*
  – Cover
  – Minimal Cover
  – BCNF violations and Decomposition
RELATIONAL ALGEBRA & CALCULUS
1. Selection ($\sigma$)

- Returns all tuples which satisfy a condition
- Notation: $\sigma_c(R)$
- Examples
  - $\sigma_{\text{Salary} > 40000}$ (Employee)
  - $\sigma_{\text{name} = \text{"Smith"}}$ (Employee)
- The condition $c$ can be $=$, $<$, $\leq$, $>$, $\geq$, $\neq$

SQL:
```
SELECT *
FROM Students
WHERE gpa > 3.5;
```

RA:
```
$\sigma_{\text{gpa} > 3.5}(\text{Students})$
```
Another example:

\[ \sigma_{\text{Salary} > 40000} (\text{Employee}) \]

<table>
<thead>
<tr>
<th>SSN</th>
<th>Name</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234545</td>
<td>John</td>
<td>200000</td>
</tr>
<tr>
<td>5423341</td>
<td>Smith</td>
<td>600000</td>
</tr>
<tr>
<td>4352342</td>
<td>Fred</td>
<td>500000</td>
</tr>
<tr>
<td>5423341</td>
<td>Smith</td>
<td>600000</td>
</tr>
<tr>
<td>4352342</td>
<td>Fred</td>
<td>500000</td>
</tr>
</tbody>
</table>
2. Projection (\(\Pi\))

- Eliminates columns, then removes duplicates
- Notation: \(\Pi_{A_1, \ldots, A_n}(R)\)
- Example: project social-security number and names:
  - \(\Pi_{SSN, Name}(Employee)\)
  - Output schema: \(Answer(SSN, Name)\)

SQL:
```sql
SELECT DISTINCT sname, gpa
FROM Students;
```

RA:
\(\Pi_{sname, gpa}(Students)\)
Another example:

<table>
<thead>
<tr>
<th>SSN</th>
<th>Name</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234545</td>
<td>John</td>
<td>200000</td>
</tr>
<tr>
<td>5423341</td>
<td>John</td>
<td>600000</td>
</tr>
<tr>
<td>4352342</td>
<td>John</td>
<td>200000</td>
</tr>
</tbody>
</table>

\[ \Pi_{\text{Name,Salary}} \left( \text{Employee} \right) \]

<table>
<thead>
<tr>
<th>Name</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>200000</td>
</tr>
<tr>
<td>John</td>
<td>600000</td>
</tr>
</tbody>
</table>
Note that RA Operators are Compositional!

**Students(sid,sname,gpa)**

**SELECT DISTINCT**
**sname,**
**gpa**
**FROM Students**
**WHERE** **gpa > 3.5;**

**How do we represent this query in RA?**

**π_{sname,gpa}(σ_{gpa>3.5}(Students))**

**σ_{gpa>3.5}(π_{sname,gpa}(Students))**

**Are these logically equivalent?**
3. Cross-Product ($\times$)

- Each tuple in $R_1$ with each tuple in $R_2$
- Notation: $R_1 \times R_2$
- Example:
  - Employee $\times$ Dependents
- Rare in practice; mainly used to express joins

```sql
SELECT * FROM Students, People;
```

```sql
RA:
Students(sid, sname, gpa)
People(ssn, pname, address)

Students $\times$ People
```
Another example:

<table>
<thead>
<tr>
<th>ssn</th>
<th>pname</th>
<th>address</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234545</td>
<td>John</td>
<td>216 Rosse</td>
</tr>
<tr>
<td>5423341</td>
<td>Bob</td>
<td>217 Rosse</td>
</tr>
</tbody>
</table>

Students $\times$ People

<table>
<thead>
<tr>
<th>sid</th>
<th>sname</th>
<th>gpa</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>John</td>
<td>3.4</td>
</tr>
<tr>
<td>002</td>
<td>Bob</td>
<td>1.3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ssn</th>
<th>pname</th>
<th>address</th>
<th>sid</th>
<th>sname</th>
<th>gpa</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234545</td>
<td>John</td>
<td>216 Rosse</td>
<td>001</td>
<td>John</td>
<td>3.4</td>
</tr>
<tr>
<td>5423341</td>
<td>Bob</td>
<td>217 Rosse</td>
<td>001</td>
<td>John</td>
<td>3.4</td>
</tr>
<tr>
<td>1234545</td>
<td>John</td>
<td>216 Rosse</td>
<td>002</td>
<td>Bob</td>
<td>1.3</td>
</tr>
<tr>
<td>5423341</td>
<td>Bob</td>
<td>216 Rosse</td>
<td>002</td>
<td>Bob</td>
<td>1.3</td>
</tr>
</tbody>
</table>
Renaming ($\rho$)

- Changes the schema, not the instance
- A ‘special’ operator- neither basic nor derived
- Notation: $\rho_{B_1, \ldots, B_n}(R)$

Note: this is shorthand for the proper form (since names, not order matters!):

- $\rho_{A_1 \rightarrow B_1, \ldots, A_n \rightarrow B_n}(R)$

SQL:

```sql
SELECT sid AS studId, sname AS name, gpa AS gradePtAvg
FROM Students;
```

RA:

$$\rho_{\text{studId, name, gradePtAvg}}(Students)$$

We care about this operator because we are working in a named perspective
Another example:

\[ \rho_{\text{studId}, \text{name}, \text{gradePtAvg}}(\text{Students}) \]

<table>
<thead>
<tr>
<th>sid</th>
<th>sname</th>
<th>gpa</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>John</td>
<td>3.4</td>
</tr>
<tr>
<td>002</td>
<td>Bob</td>
<td>1.3</td>
</tr>
</tbody>
</table>

\[
\begin{array}{ccc}
\text{studId} & \text{name} & \text{gradePtAvg} \\
001 & John & 3.4 \\
002 & Bob & 1.3 \\
\end{array}
\]
• Notation: \( R_1 \bowtie R_2 \)

• Joins \( R_1 \) and \( R_2 \) on equality of all shared attributes
  – If \( R_1 \) has attribute set \( A \), and \( R_2 \) has attribute set \( B \), and they share attributes \( A \cap B = C \), can also be written: \( R_1 \bowtie_C R_2 \)

• Our first example of a derived RA operator:
  – Meaning: \( R_1 \bowtie R_2 = \Pi_{A \cup B}(\sigma_{C=D}(\rho_{C \rightarrow D}(R_1 \times R_2))) \)
  – Where:
    • The rename \( \rho_{C \rightarrow D} \) renames the shared attributes in one of the relations
    • The selection \( \sigma_{C=D} \) checks equality of the shared attributes
    • The projection \( \Pi_{A \cup B} \) eliminates the duplicate common attributes

SQL:
```
SELECT DISTINCT sid, S.name, gpa, ssn, address
FROM Students S, People P
WHERE S.name = P.name;
```

RA:
```
Students \bowtie People
```
Another example:

**Students S**

```
<table>
<thead>
<tr>
<th>sid</th>
<th>S.name</th>
<th>gpa</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>John</td>
<td>3.4</td>
</tr>
<tr>
<td>002</td>
<td>Bob</td>
<td>1.3</td>
</tr>
</tbody>
</table>
```

**People P**

```
<table>
<thead>
<tr>
<th>ssn</th>
<th>P.name</th>
<th>address</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234545</td>
<td>John</td>
<td>216 Rosse</td>
</tr>
<tr>
<td>5423341</td>
<td>Bob</td>
<td>217 Rosse</td>
</tr>
</tbody>
</table>
```

**Students ⋈ People**

```
<table>
<thead>
<tr>
<th>sid</th>
<th>S.name</th>
<th>gpa</th>
<th>ssn</th>
<th>address</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>John</td>
<td>3.4</td>
<td>1234545</td>
<td>216 Rosse</td>
</tr>
<tr>
<td>002</td>
<td>Bob</td>
<td>1.3</td>
<td>5423341</td>
<td>216 Rosse</td>
</tr>
</tbody>
</table>
```
Natural Join

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

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

• Given $R(A, B), S(A, B)$, what is $R \bowtie S$ ?
Example: Converting SFW Query -> RA

```
SELECT DISTINCT
    gpa,
    address
FROM Students S,
     People P
WHERE gpa > 3.5 AND
    S.name = P.name;
```

How do we represent this query in RA?

\[
\Pi_{gpa, address}(\sigma_{gpa > 3.5}(S \bowtie P))
\]
Logical Equivalence of RA Plans

• Given relations \( R(A,B) \) and \( S(B,C) \):
  
  – Here, projection & selection commute:
    * \( \sigma_{A=5}(\Pi_A(R)) = \Pi_A(\sigma_{A=5}(R)) \)

  – What about here?
    * \( \sigma_{A=5}(\Pi_B(R)) \) ? = \( \Pi_B(\sigma_{A=5}(R)) \)
Relational Algebra (RA)

• **Five basic operators:**
  1. Selection: $\sigma$
  2. Projection: $\Pi$
  3. Cartesian Product: $\times$
  4. Union: $\cup$
  5. Difference: $-$

• **Derived or auxiliary operators:**
  – Intersection
  – Joins (natural, equi-join, theta join, semi-join)
  – Renaming: $\rho$
  – Division

We’ll look at these

And also at some of these derived operators
1. Union (\(\cup\)) and 2. Difference (\(\mathbin{-}\))

- **R_1 \cup R_2**
  - Example: \(\text{ActiveEmployees} \cup \text{RetiredEmployees}\)

- **R_1 \mathbin{-} R_2**
  - Example: \(\text{AllEmployees} \mathbin{-} \text{RetiredEmployees}\)
What about Intersection (\(\cap\))?

- It is a derived operator
- \(R_1 \cap R_2 = R_1 - (R_1 - R_2)\)
- Also expressed as a join!
- Example
  - UnionizedEmployees \(\cap\) RetiredEmployees
**Theta Join** ($\bowtie_\theta$)

- A join that involves a predicate
- $R_1 \bowtie_\theta R_2 = \sigma_\theta (R_1 \times R_2)$
- Here $\theta$ can be any condition

**SQL:**
```
SELECT *
FROM Students, People
WHERE \theta;
```

**RA:**
```
Students $\bowtie_\theta$ People
```
Equi-join ($\bowtie_{A=B}$)

- A theta join where $\theta$ is an equality
- $R_1 \bowtie_{A=B} R_2 = \sigma_{A=B} (R_1 \times R_2)$
- Example:
  - Employee $\bowtie_{SSN(SSN)}$ Dependents

$SQL$:
```
SELECT * 
FROM 
  Students S, 
  People P 
WHERE sname = pname;
```

$RA$:
```
S $\bowtie_{sname=pname}$ P
```
Semijoin (\(\bowtie\))

- \(R \bowtie S = \Pi_{A_1, \ldots, A_n} (R \bowtie S)\)
- Where \(A_1, \ldots, A_n\) are the attributes in \(R\)
- Example:
  - Employee \(\bowtie\) Dependents

SQL:
```
SELECT DISTINCT sid, sname, gpa
FROM Students, People
WHERE sname = pname;
```

RA:
```
Students \(\bowtie\) People
```
Divison ($\div$)

- $T(Y) = R(Y, X) \div S(X)$

- $Y$ is the set of attributes of $R$ that are not attributes of $S$.

- For a tuple $t$ to appear in the result $T$ of the Division, the values in $t$ must appear in $R$ in combination with *every* tuple in $S$. 
R(Y,X) \div S(X) = T(Y)

<table>
<thead>
<tr>
<th>PilotSkills</th>
<th>plane_name</th>
</tr>
</thead>
<tbody>
<tr>
<td>'Celko'</td>
<td>'Piper Cub'</td>
</tr>
<tr>
<td>'Higgins'</td>
<td>'B-52 Bomber'</td>
</tr>
<tr>
<td>'Higgins'</td>
<td>'F-14 Fighter'</td>
</tr>
<tr>
<td>'Higgins'</td>
<td>'Piper Cub'</td>
</tr>
<tr>
<td>'Jones'</td>
<td>'B-52 Bomber'</td>
</tr>
<tr>
<td>'Jones'</td>
<td>'F-14 Fighter'</td>
</tr>
<tr>
<td>'Smith'</td>
<td>'B-1 Bomber'</td>
</tr>
<tr>
<td>'Smith'</td>
<td>'B-52 Bomber'</td>
</tr>
<tr>
<td>'Smith'</td>
<td>'F-14 Fighter'</td>
</tr>
<tr>
<td>'Wilson'</td>
<td>'B-1 Bomber'</td>
</tr>
<tr>
<td>'Wilson'</td>
<td>'B-52 Bomber'</td>
</tr>
<tr>
<td>'Wilson'</td>
<td>'F-14 Fighter'</td>
</tr>
<tr>
<td>'Wilson'</td>
<td>'F-17 Fighter'</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Hangar</th>
<th>plane_name</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>'B-1 Bomber'</td>
</tr>
<tr>
<td></td>
<td>'B-52 Bomber'</td>
</tr>
<tr>
<td></td>
<td>'F-14 Fighter'</td>
</tr>
</tbody>
</table>

```sql
SELECT PS1.pilot_name
    FROM PilotSkills AS PS1, Hangar AS H1
    WHERE PS1.plane_name = H1.plane_name
    GROUP BY PS1.pilot_name
    HAVING COUNT(PS1.plane_name) =
    (SELECT COUNT(plane_name) FROM Hangar);
```

Complete Set of Relational Operations

• The set of operations including
  • Select σ,
  • Project π
  • Union ∪
  • Difference –
  • Rename ρ, and
  • Cartesian Product X

  – is called a *complete set*

  – because any other relational algebra expression can be expressed by a combination of these five operations.

• For example:
  - R ∩ S = (R ∪ S) – ((R – S) ∪ (S – R))
  - R⋈_{<join condition>} S = σ_{<join condition>} (R X S)
**Table 8.1** Operations of Relational Algebra

<table>
<thead>
<tr>
<th>OPERATION</th>
<th>PURPOSE</th>
<th>NOTATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>SELECT</td>
<td>Selects all tuples that satisfy the selection condition from a relation $R$.</td>
<td>$\sigma_{&lt;\text{selection condition}&gt;}(R)$</td>
</tr>
<tr>
<td>PROJECT</td>
<td>Produces a new relation with only some of the attributes of $R$, and removes duplicate tuples.</td>
<td>$\pi_{&lt;\text{attribute list}&gt;}(R)$</td>
</tr>
<tr>
<td>THETA JOIN</td>
<td>Produces all combinations of tuples from $R_1$ and $R_2$ that satisfy the join condition.</td>
<td>$R_1 \bowtie_{&lt;\text{join condition}&gt;} R_2$</td>
</tr>
<tr>
<td>EQUIJOIN</td>
<td>Produces all the combinations of tuples from $R_1$ and $R_2$ that satisfy a join condition with only equality comparisons.</td>
<td>$R_1 \bowtie_{&lt;\text{join condition}&gt;} R_2$, OR $R_1 \bowtie_{(&lt;\text{join attributes 1}&gt;)} R_2$</td>
</tr>
<tr>
<td>NATURAL JOIN</td>
<td>Same as EQUIJOIN except that the join attributes of $R_2$ are not included in the resulting relation; if the join attributes have the same names, they do not have to be specified at all.</td>
<td>$R_1 \bowtie_{&lt;\text{join condition}&gt;} R_2$, OR $R_1 \bowtie_{(&lt;\text{join attributes 1}&gt;)} R_2$ OR $R_1 \bowtie_{(&lt;\text{join attributes 2}&gt;)} R_2$ OR $R_1 \bowtie R_2$</td>
</tr>
<tr>
<td>OPERATION</td>
<td>PURPOSE</td>
<td>NOTATION</td>
</tr>
<tr>
<td>----------------------</td>
<td>-------------------------------------------------------------------------</td>
<td>--------------</td>
</tr>
<tr>
<td>UNION</td>
<td>Produces a relation that includes all the tuples in $R_1$ or $R_2$ or both $R_1$ and $R_2$; $R_1$ and $R_2$ must be union compatible.</td>
<td>$R_1 \cup R_2$</td>
</tr>
<tr>
<td>INTERSECTION</td>
<td>Produces a relation that includes all the tuples in both $R_1$ and $R_2$; $R_1$ and $R_2$ must be union compatible.</td>
<td>$R_1 \cap R_2$</td>
</tr>
<tr>
<td>DIFFERENCE</td>
<td>Produces a relation that includes all the tuples in $R_1$ that are not in $R_2$; $R_1$ and $R_2$ must be union compatible.</td>
<td>$R_1 - R_2$</td>
</tr>
<tr>
<td>CARTESIAN PRODUCT</td>
<td>Produces a relation that has the attributes of $R_1$ and $R_2$ and includes as tuples all possible combinations of tuples from $R_1$ and $R_2$.</td>
<td>$R_1 \times R_2$</td>
</tr>
<tr>
<td>DIVISION</td>
<td>Produces a relation $R(X)$ that includes all tuples $t[X]$ in $R_1(Z)$ that appear in $R_1$ in combination with every tuple from $R_2(Y)$, where $Z = X \cup Y$.</td>
<td>$R_1(Z) \div R_2(Y)$</td>
</tr>
</tbody>
</table>
Examples of Queries in Relational Algebra: Procedural Form

- Q1: Retrieve the name and Address of all Employees who work for the ‘Research’ department.

\[
\begin{align*}
\text{Research} \_\text{dept} & \leftarrow \sigma_{\text{Dname}=\text{Research}}(\text{Department}) \\
\text{Research} \_\text{emps} & \leftarrow (\text{RESEARCH} \_\text{DEP} \bowtie_{\text{DNumber}=\text{Dno}} \text{Employee}) \\
\text{Result} & \leftarrow \pi_{\text{Fname}, \text{Lname}, \text{Address}}(\text{Research} \_\text{emps})
\end{align*}
\]

As a single expression, this query becomes:

\[
\pi_{\text{Fname}, \text{Lname}, \text{Address}}(\sigma_{\text{Dname}=\text{Research}}(\text{Department}) \bowtie_{\text{Dnumber}=\text{Dno}} \text{Employee})
\]
Examples of Queries in Relational Algebra: Procedural Form

Q6: Retrieve the names of Employees who have no dependents.

\[
\text{ALL_EMPS} \leftarrow \pi_{\text{SSN}}(\text{Employee})
\]

\[
\text{EMPS\_WITH\_DEPS}(\text{SSN}) \leftarrow \pi_{\text{Essn}}(\text{DEPENDENT})
\]

\[
\text{EMPS\_WITHOUT\_DEPS} \leftarrow (\text{ALL\_EMPS} - \text{EMPS\_WITH\_DEPS})
\]

\[
\text{RESULT} \leftarrow \pi_{\text{Lname, Fname}}(\text{EMPS\_WITHOUT\_DEPS} \ast \text{Employee})
\]

As a single expression, this query becomes:

\[
\pi_{\text{Lname, Fname}}((\pi_{\text{SSn}}(\text{Employee}) - \rho_{\text{SSn}}(\pi_{\text{Essn}}(\text{Dependent})))) \ast \text{Employee}
\]
Division

- \( T(Y) = R(Y, X) \div S(X) \)

- The complete division expression:
  \[
  R \div S = \pi_Y R - \pi_Y (\pi_Y(R) \times S) - R
  \]

Ignoring the projections, there are just three steps:
- Compute all possible attribute pairings
- Remove the existing pairings
- Remove the non-answers from the possible answers

https://www2.cs.arizona.edu/~mccann/research/divpresentation.pdf
### Division Example

<table>
<thead>
<tr>
<th>R(Y,X)</th>
<th>S(X)</th>
<th>$\pi_Y(R) \times S$</th>
<th>$(\pi_Y(R) \times S) - R$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y</td>
<td>X</td>
<td>Y</td>
<td>X</td>
</tr>
<tr>
<td>y1</td>
<td>x1</td>
<td>y1</td>
<td>x1</td>
</tr>
<tr>
<td>y1</td>
<td>x2</td>
<td>y1</td>
<td>x2</td>
</tr>
<tr>
<td>y2</td>
<td>x1</td>
<td>y2</td>
<td>x1</td>
</tr>
<tr>
<td>y3</td>
<td>x1</td>
<td>y3</td>
<td>x1</td>
</tr>
<tr>
<td>y3</td>
<td>x2</td>
<td>y3</td>
<td>x2</td>
</tr>
</tbody>
</table>

\[
\pi_Y \left( (\pi_Y(R) \times S) - R \right) \quad \pi_Y R - \pi_Y \left( (\pi_Y(R) \times S) - R \right)
\]

\[
R \div S = \frac{\pi_Y \left( (\pi_Y(R) \times S) - R \right)}{\pi_Y R - \pi_Y \left( (\pi_Y(R) \times S) - R \right)}
\]
Acknowledgement

• Some of the slides in this presentation are taken from the slides provided by the authors.
• Many of these slides are taken from cs145 course offered by Stanford University.
• Thanks to YouTube, especially to Dr. Daniel Soper for his useful videos.