Relational Algebra

csc343, Introduction to Databases
Diane Horton, Michelle Craig, and Sina Meraji
Fall 2017
Simplifications

- While learning relational algebra, we will assume:
  - Relations are sets, so now two rows are the same.
  - Every cell has a value.
- In SQL, we will drop these assumptions.
- But for now, they simplify our queries.
- And we will still build a great foundation for SQL.
RA Basics
(covered by your week 2 Prep)
Elementary Algebra

- You did algebra in high school
  - $27y^2 + 8y - 3$

- Operands:

- Operators:
Relational Algebra

- **Operands:** tables
- **Operators:**
  - choose only the rows you want
  - choose only the columns you want
  - combine tables
  - and a few other things
A schema for our examples

Movies(mID, title, director, year, length)
Artists(aID, aName, nationality)
Roles(mID, aID, character)

Foreign key constraints:
- Roles[mID] ⊆ Movies[mID]
- Roles[aID] ⊆ Artists[aID]
Select: choose rows

- **Notation:** $\sigma_c(R)$
  - $R$ is a table.
  - Condition $c$ is a boolean expression.
  - It can use comparison operators and boolean operators.
  - The operands are either constants or attributes of $R$.

- The result is a relation
  - with the same schema as the operand
  - but with only the tuples that satisfy the condition
Exercise

- Write queries to find:
  - All British actors
  - All movies from the 1970s
- What if we only want the names of all British actors?
  We need a way to pare down the columns.
Project: choose columns

- Notation: \( \pi_L(R) \)
  - \( R \) is a table.
  - \( L \) is a subset (not necessarily a proper subset) of the attributes of \( R \).

- The result is a relation
  - with all the tuples from \( R \)
  - but with only the attributes in \( L \), and in that order
About project

- Why is it called “project”?
- What is the value of $\pi_{\text{director}} (\text{Movies})$?
- Exercise: Write an RA expression to find the names of all directors of movies from the 1970s.

- Now, suppose you want the names of all characters in movies from the 1970s.
- We need to be able to combine tables.
Cartesian Product

- Notation: $R_1 \times R_2$
- The result is a relation with
  - every combination of a tuple from $R_1$ concatenated to a tuple from $R_2$
- Its schema is every attribute from $R$ followed by every attribute of $S$, in order
- How many tuples are in $R_1 \times R_2$?
- Example: Movies x Roles
- If an attribute occurs in both relations, it occurs twice in the result (prefixed by relation name)
Continuing on with Relational Algebra
Project and duplicates

- Projecting onto fewer attributes can remove what it was that made two tuples distinct.
- Wherever a project operation might “introduce” duplicates, only one copy of each is kept.
- Example:

<table>
<thead>
<tr>
<th>name</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>Karim</td>
<td>20</td>
</tr>
<tr>
<td>Ruth</td>
<td>18</td>
</tr>
<tr>
<td>Minh</td>
<td>20</td>
</tr>
<tr>
<td>Sofia</td>
<td>19</td>
</tr>
<tr>
<td>Jennifer</td>
<td>19</td>
</tr>
<tr>
<td>Sasha</td>
<td>20</td>
</tr>
</tbody>
</table>

\[ \pi_{\text{age}} \text{ People} \]

<table>
<thead>
<tr>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
</tr>
<tr>
<td>18</td>
</tr>
<tr>
<td>19</td>
</tr>
</tbody>
</table>
### Example of Cartesian product

**profiles:**

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Oprah</td>
<td>Oprah Winfrey</td>
</tr>
<tr>
<td>ginab</td>
<td>Gina Bianchini</td>
</tr>
</tbody>
</table>

**follows:**

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>Oprah</td>
<td>ev</td>
</tr>
<tr>
<td>edyson</td>
<td>ginab</td>
</tr>
<tr>
<td>ginab</td>
<td>ev</td>
</tr>
</tbody>
</table>

**profiles X follows:**

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>Oprah</td>
<td>Oprah Winfrey</td>
<td>Oprah</td>
<td>ev</td>
</tr>
<tr>
<td>Oprah</td>
<td>Oprah Winfrey</td>
<td>edyson</td>
<td>ginab</td>
</tr>
<tr>
<td>Oprah</td>
<td>Oprah Winfrey</td>
<td>ginab</td>
<td>ev</td>
</tr>
<tr>
<td>ginab</td>
<td>Gina Bianchini</td>
<td>Oprah</td>
<td>ev</td>
</tr>
<tr>
<td>ginab</td>
<td>Gina Bianchini</td>
<td>edyson</td>
<td>ginab</td>
</tr>
<tr>
<td>ginab</td>
<td>Gina Bianchini</td>
<td>ginab</td>
<td>ev</td>
</tr>
</tbody>
</table>
Composing larger expressions

- **Math:**
  - The value of any expression is a number.
  - So you can “compose” larger expressions out of smaller ones.
  - There are precedence rules.
  - We can use brackets to override the normal precedence of operators.

- Relational algebra is the same.
More about joining relations
Cartesian product can be inconvenient

- It can introduce nonsense tuples.
- You can get rid of them with selects.
- But this is so highly common, an operation was defined to make it easier: natural join.
Natural Join

- Notation: \( R \bowtie S \)
- The result is defined by
  - taking the Cartesian product
  - selecting to ensure equality on attributes that are in both relations (as determined \textit{by name})
  - projecting to remove duplicate attributes.
- Example:
  Artists \( \bowtie \) Roles gets rid of the nonsense tuples.
Examples

- The following examples show what natural join does when the tables have:
  - no attributes in common
  - one attribute in common
  - a different attribute in common

- (Note that we change the attribute names for relation follows to set up these scenarios.)
### profiles:

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Oprah</td>
<td>Oprah Winfrey</td>
</tr>
<tr>
<td>ginab</td>
<td>Gina Bianchini</td>
</tr>
</tbody>
</table>

### follows:

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>Oprah</td>
<td>ev</td>
</tr>
<tr>
<td>edyson</td>
<td>ginab</td>
</tr>
<tr>
<td>ginab</td>
<td>ev</td>
</tr>
</tbody>
</table>

### profiles ⋈ follows:

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>Oprah</td>
<td>Oprah Winfrey</td>
<td>Oprah</td>
<td>ev</td>
</tr>
<tr>
<td>Oprah</td>
<td>Oprah Winfrey</td>
<td>edyson</td>
<td>ginab</td>
</tr>
<tr>
<td>Oprah</td>
<td>Oprah Winfrey</td>
<td>ginab</td>
<td>ev</td>
</tr>
<tr>
<td>ginab</td>
<td>Gina Bianchini</td>
<td>Oprah</td>
<td>ev</td>
</tr>
<tr>
<td>ginab</td>
<td>Gina Bianchini</td>
<td>edyson</td>
<td>ginab</td>
</tr>
<tr>
<td>ginab</td>
<td>Gina Bianchini</td>
<td>ginab</td>
<td>ev</td>
</tr>
</tbody>
</table>
profiles:

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Oprah</td>
<td>Oprah Winfrey</td>
</tr>
<tr>
<td>ginab</td>
<td>Gina Bianchini</td>
</tr>
</tbody>
</table>

follows:

<table>
<thead>
<tr>
<th>ID</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>Oprah</td>
<td>ev</td>
</tr>
<tr>
<td>edyson</td>
<td>ginab</td>
</tr>
<tr>
<td>ginab</td>
<td>ev</td>
</tr>
</tbody>
</table>

profiles ⋈ follows:

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>ID</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>Oprah</td>
<td>Oprah Winfrey</td>
<td>Oprah</td>
<td>ev</td>
</tr>
<tr>
<td>Oprah</td>
<td>Oprah Winfrey</td>
<td>edyson</td>
<td>ginab</td>
</tr>
<tr>
<td>Oprah</td>
<td>Oprah Winfrey</td>
<td>ginab</td>
<td>ev</td>
</tr>
<tr>
<td>ginab</td>
<td>Gina Bianchini</td>
<td>Oprah</td>
<td>ev</td>
</tr>
<tr>
<td>ginab</td>
<td>Gina Bianchini</td>
<td>edyson</td>
<td>ginab</td>
</tr>
<tr>
<td>ginab</td>
<td>Gina Bianchini</td>
<td>ginab</td>
<td>ev</td>
</tr>
</tbody>
</table>

(The redundant ID column is omitted in the result)
### Profiles

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Oprah</td>
<td>Oprah Winfrey</td>
</tr>
<tr>
<td>ginab</td>
<td>Gina Bianchini</td>
</tr>
</tbody>
</table>

### Follows

<table>
<thead>
<tr>
<th>a</th>
<th>ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Oprah</td>
<td>ev</td>
</tr>
<tr>
<td>edyson</td>
<td>ginab</td>
</tr>
<tr>
<td>ginab</td>
<td>ev</td>
</tr>
</tbody>
</table>

### Profiles \(\bowtie\) Follows

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>a</th>
<th>ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Oprah</td>
<td>Oprah Winfrey</td>
<td>Oprah</td>
<td>ev</td>
</tr>
<tr>
<td>Oprah</td>
<td>Oprah Winfrey</td>
<td>edyson</td>
<td>ginab</td>
</tr>
<tr>
<td>Oprah</td>
<td>Oprah Winfrey</td>
<td>ginab</td>
<td>ev</td>
</tr>
<tr>
<td>ginab</td>
<td>Gina Bianchini</td>
<td>Oprah</td>
<td>ev</td>
</tr>
<tr>
<td>ginab</td>
<td>Gina Bianchini</td>
<td>edyson</td>
<td>ginab</td>
</tr>
<tr>
<td>ginab</td>
<td>Gina Bianchini</td>
<td>ginab</td>
<td>ev</td>
</tr>
</tbody>
</table>

(The redundant ID column is omitted in the result)
Properties of Natural Join

- **Commutative:**
  \[ R \Join S = S \Join R \]
  (although attribute order may vary; this will matter later when we use set operations)

- **Associative:**
  \[ R \Join (S \Join T) = (R \Join S) \Join T \]

- So when writing n-ary joins, brackets are irrelevant. We can just write:
  \[ R_1 \Join R_2 \Join \ldots \Join R_n \]
Questions

For the instance on our Movies worksheet:

1. How many tuples are in Artists $\times$ Roles?
2. How many tuples are in Artists $\bowtie$ Roles?
3. What is the result of:
   $$\pi_{\text{aName}} \sigma_{\text{director} = "Kubrick"}(\text{Artists} \bowtie \text{Roles} \bowtie \text{Movies})$$
4. What is the result of:
   $$\pi_{\text{aName}}((\sigma_{\text{director} = "Kubrick"}\text{Artists}) \bowtie \text{Roles} \bowtie \text{Movies})$$
1. How many tuples are in Artists $\times$ Roles?

**Artists:**

<table>
<thead>
<tr>
<th>aID</th>
<th>aName</th>
<th>nationality</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Nicholson</td>
<td>American</td>
</tr>
<tr>
<td>2</td>
<td>Ford</td>
<td>American</td>
</tr>
<tr>
<td>3</td>
<td>Stone</td>
<td>British</td>
</tr>
<tr>
<td>4</td>
<td>Fisher</td>
<td>American</td>
</tr>
</tbody>
</table>

**Roles:**

<table>
<thead>
<tr>
<th>mID</th>
<th>aID</th>
<th>character</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>Jack Torrance</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>Jake ‘J.J.’ Gittes</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>Delbert Grady</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>Han Solo</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>Bob Falfa</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>Princess Leia Organa</td>
</tr>
</tbody>
</table>

2. How many tuples are in Artists \( \bowtie \) Roles?

### Artists:

<table>
<thead>
<tr>
<th>aID</th>
<th>aName</th>
<th>nationality</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Nicholson</td>
<td>American</td>
</tr>
<tr>
<td>2</td>
<td>Ford</td>
<td>American</td>
</tr>
<tr>
<td>3</td>
<td>Stone</td>
<td>British</td>
</tr>
<tr>
<td>4</td>
<td>Fisher</td>
<td>American</td>
</tr>
</tbody>
</table>

### Roles:

<table>
<thead>
<tr>
<th>mID</th>
<th>aID</th>
<th>character</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>Jack Torrance</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>Jake ‘J.J.’ Gittes</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>Delbert Grady</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>Han Solo</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>Bob Falfa</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>Princess Leia Organa</td>
</tr>
</tbody>
</table>
### 3. What is the result of:

\[
\Pi_{aName} \sigma_{\text{director}="Kubrick"} (\text{Artists} \bowtie \text{Roles} \bowtie \text{Movies})
\]

#### Movies:

<table>
<thead>
<tr>
<th>mID</th>
<th>title</th>
<th>director</th>
<th>year</th>
<th>length</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Shining</td>
<td>Kubrick</td>
<td>1980</td>
<td>146</td>
</tr>
<tr>
<td>2</td>
<td>Player</td>
<td>Altman</td>
<td>1992</td>
<td>146</td>
</tr>
<tr>
<td>3</td>
<td>Chinatown</td>
<td>Polaski</td>
<td>1974</td>
<td>131</td>
</tr>
<tr>
<td>4</td>
<td>Repulsion</td>
<td>Polaski</td>
<td>1965</td>
<td>143</td>
</tr>
<tr>
<td>5</td>
<td>Star Wars IV</td>
<td>Lucas</td>
<td>1977</td>
<td>126</td>
</tr>
<tr>
<td>6</td>
<td>American Graffiti</td>
<td>Lucas</td>
<td>1973</td>
<td>110</td>
</tr>
<tr>
<td>7</td>
<td>Full Metal Jacket</td>
<td>Kubrick</td>
<td>1987</td>
<td>156</td>
</tr>
</tbody>
</table>

#### Artists:

<table>
<thead>
<tr>
<th>aID</th>
<th>aName</th>
<th>nationality</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Nicholson</td>
<td>American</td>
</tr>
<tr>
<td>2</td>
<td>Ford</td>
<td>American</td>
</tr>
<tr>
<td>3</td>
<td>Stone</td>
<td>British</td>
</tr>
<tr>
<td>4</td>
<td>Fisher</td>
<td>American</td>
</tr>
</tbody>
</table>

#### Roles:

<table>
<thead>
<tr>
<th>mID</th>
<th>aID</th>
<th>character</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>Jack Torrance</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>Jake ‘J.J.’ Gittes</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>Delbert Grady</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>Han Solo</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>Bob Falfa</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>Princess Leia Organa</td>
</tr>
</tbody>
</table>
4. What is the result of:

$$\pi_{aName}(\sigma_{\text{director}="Kubrick"}(\text{Artists}) \bowtie \text{Roles} \bowtie \text{Movies})$$

**Movies:**

<table>
<thead>
<tr>
<th>mID</th>
<th>title</th>
<th>director</th>
<th>year</th>
<th>length</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Shining</td>
<td>Kubrick</td>
<td>1980</td>
<td>146</td>
</tr>
<tr>
<td>2</td>
<td>Player</td>
<td>Altman</td>
<td>1992</td>
<td>146</td>
</tr>
<tr>
<td>3</td>
<td>Chinatown</td>
<td>Polaski</td>
<td>1974</td>
<td>131</td>
</tr>
<tr>
<td>4</td>
<td>Repulsion</td>
<td>Polaski</td>
<td>1965</td>
<td>143</td>
</tr>
<tr>
<td>5</td>
<td>Star Wars IV</td>
<td>Lucas</td>
<td>1977</td>
<td>126</td>
</tr>
<tr>
<td>6</td>
<td>American Graffiti</td>
<td>Lucas</td>
<td>1973</td>
<td>110</td>
</tr>
<tr>
<td>7</td>
<td>Full Metal Jacket</td>
<td>Kubrick</td>
<td>1987</td>
<td>156</td>
</tr>
</tbody>
</table>

**Artists:**

<table>
<thead>
<tr>
<th>aID</th>
<th>aName</th>
<th>nationality</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Nicholson</td>
<td>American</td>
</tr>
<tr>
<td>2</td>
<td>Ford</td>
<td>American</td>
</tr>
<tr>
<td>3</td>
<td>Stone</td>
<td>British</td>
</tr>
<tr>
<td>4</td>
<td>Fisher</td>
<td>American</td>
</tr>
</tbody>
</table>

**Roles:**

<table>
<thead>
<tr>
<th>mID</th>
<th>aID</th>
<th>character</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>Jack Torrance</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>Jake ‘J.J.’ Gittes</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>Delbert Grady</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>Han Solo</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>Bob Falfa</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>Princess Leia Organa</td>
</tr>
</tbody>
</table>
Special cases for natural join
No tuples match

<table>
<thead>
<tr>
<th>Employee</th>
<th>Dept</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vista</td>
<td>Sales</td>
</tr>
<tr>
<td>Kagani</td>
<td>Production</td>
</tr>
<tr>
<td>Tzerpos</td>
<td>Production</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Dept</th>
<th>Head</th>
</tr>
</thead>
<tbody>
<tr>
<td>HR</td>
<td>Boutilier</td>
</tr>
</tbody>
</table>
Exactly the same attributes

<table>
<thead>
<tr>
<th>Artist</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>9132</td>
<td>William Shatner</td>
</tr>
<tr>
<td>8762</td>
<td>Harrison Ford</td>
</tr>
<tr>
<td>5555</td>
<td>Patrick Stewart</td>
</tr>
<tr>
<td>1868</td>
<td>Angelina Jolie</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Artist</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234</td>
<td>Brad Pitt</td>
</tr>
<tr>
<td>1868</td>
<td>Angelina Jolie</td>
</tr>
<tr>
<td>5555</td>
<td>Patrick Stewart</td>
</tr>
</tbody>
</table>
No attributes in common

<table>
<thead>
<tr>
<th>Artist</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234</td>
<td>Brad Pitt</td>
</tr>
<tr>
<td>1868</td>
<td>Angelina Jolie</td>
</tr>
<tr>
<td>5555</td>
<td>Patrick Stewart</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>mID</th>
<th>Title</th>
<th>Director</th>
<th>Year</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>1111</td>
<td>Alien</td>
<td>Scott</td>
<td>1979</td>
<td>152</td>
</tr>
<tr>
<td>1234</td>
<td>Sting</td>
<td>Hill</td>
<td>1973</td>
<td>130</td>
</tr>
</tbody>
</table>
Natural join can “over-match”

- Natural join bases the matching on attribute names.
- What if two attributes have the same name, but we don’t want them to have to match?
- Example: if Artists used “name” for actors’ names and Movies used “name” for movies’ names.
  – Can rename one of them (we’ll see how).
  – Or?
Natural join can “under-match”

- What if two attributes don’t have the same name and we do want them to match?
- Example: Suppose we want aName and director to match.
- Solution?
Theta Join

- It’s common to use $\sigma$ to check conditions after a Cartesian product.
- Theta Join makes this easier.
- Notation: $R \bowtie_{condition} S$
- The result is
  - the same as Cartesian product (not natural join!) followed by select. In other words, $R \bowtie_{condition} S = \sigma_{condition} (R \times S)$.
- The word “theta” has no special connotation. It is an artifact of a definition in an early paper.
- You save just one symbol.
- You still have to write out the conditions, since they are not inferred.
Composing larger expressions (plus a few new operators)
Precedence

- Expressions can be composed recursively.
- Make sure attributes match as you wish.
  - It helps to annotate each subexpression, showing the attributes of its resulting relation.
- Parentheses and precedence rules define the order of evaluation.
- Precedence, from highest to lowest, is:
  - $\sigma$, $\pi$, $\rho$
  - $\times$, $\bowtie$
  - $\cap$
  - $\cup$, $-$

- Unless very sure, use brackets!

The highlighted operators are new. We’ll learn them shortly.
Breaking down expressions

- Complex nested expressions can be hard to read.
- Two alternative notations allow us to break them down:
  - Expression trees.
  - Sequences of assignment statements.
Expression Trees

- Leaves are relations.
- Interior notes are operators.
- Exactly like representing arithmetic expressions as trees.

\[ 3 + 7 \times x \]

- If interested, see Ullman and Widom, section 2.4.10.
Assignment operator

- Notation:
  \[ R := \text{Expression} \]

- Alternate notation:
  \[ R(A_1, ..., A_n) := \text{Expression} \]
  - Lets you name all the attributes of the new relation
  - Sometimes you don’t want the name they would get from Expression.

- \( R \) must be a temporary variable, not one of the relations in the schema.
  I.e., you are not updating the content of a relation!
- Example:
  \[
  \text{CSCoffering} := \sigma_{\text{dept}='\text{csc}'} \text{Offering}
  \]
  \[
  \text{TookCSC}(\text{sid}, \text{grade}) := \pi_{\text{sid}, \text{grade}}(\text{CSCoffering} \bowtie \text{Took})
  \]
  \[
  \text{PassedCSC}(\text{sid}) := \pi_{\text{sid}} \sigma_{\text{grade}\geq 50}(\text{TookCSC})
  \]

- Whether / how small to break things down is up to you. It’s all for readability.
- Assignment helps us break a problem down
- It also allows us to change the names of relations [and attributes].
- There is another way to rename things ...
Rename operation

- Notation: $\rho_{R_1}(R_2)$
- Alternate notation: $\rho_{R_1(A_1, \ldots, A_n)}(R_2)$
  - Lets you rename all the attributes as well as the relation.
- Note that these are equivalent:
  $R_1(A_1, \ldots, A_n) := R_2$
  $R_1 := \rho_{R_1(A_1, \ldots, A_n)}(R_2)$
- $\rho$ is useful if you want to rename *within* an expression.
## Summary of operators

<table>
<thead>
<tr>
<th>Operation</th>
<th>Name</th>
<th>Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>choose rows</td>
<td>select</td>
<td>$\sigma$</td>
</tr>
<tr>
<td>choose columns</td>
<td>project</td>
<td>$\pi$</td>
</tr>
<tr>
<td>combine tables</td>
<td>Cartesian product</td>
<td>$\times$</td>
</tr>
<tr>
<td></td>
<td>natural join</td>
<td>$\bowtie$</td>
</tr>
<tr>
<td></td>
<td>theta join</td>
<td>$\bowtie_{condition}$</td>
</tr>
<tr>
<td>rename relation</td>
<td>rename</td>
<td>$\rho$</td>
</tr>
<tr>
<td>[and attributes]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>assignment</td>
<td>assignment</td>
<td>$:=$</td>
</tr>
</tbody>
</table>
“Syntactic sugar”

- Some operations are not necessary.
  - You can get the same effect using a combination of other operations.
- Examples: natural join, theta join.
- We call this “syntactic sugar”.
- This concept also comes up in logic and programming languages.
More practise writing queries
Set operations

- Because relations are sets, we can use set intersection, union and difference.
- But only if the operands are relations over the same attributes (in number, name, and order).
- If the names or order mismatch?
Quick recap about sets in math

Union: \( \{55, 22, 48, 74\} \cup \{22, 23, 48, 9, 50\} = \{55, 22, 48, 74, 23, 9, 50\} \)

Intersection: \( \{55, 22, 48, 74\} \cap \{22, 23, 48, 9, 50\} = \{22, 48\} \)

Difference: \( \{55, 22, 48, 74\} - \{22, 23, 48, 9, 50\} = \{55, 74\} \)

Set operators work the same way in relational algebra.
Expressing Integrity Constraints

- We’ve used this notation to express inclusion dependencies between relations $R_1$ and $R_2$: $R_1[X] \subseteq R_2[Y]$
- We can use RA to express other kinds of integrity constraints.
- Suppose $R$ and $S$ are expressions in RA. We can write an integrity constraint in either of these ways:
  
  \[
  R = \emptyset \\
  R \subseteq S \quad \text{(equivalent to saying } R - S = \emptyset)\]
- We don’t need the second form, but it’s convenient.
Integrity Constraints: Example

- Express the following constraints using the notation $R = \emptyset$ or $R \subseteq S$:

1. 400-level courses cannot count for breadth.
2. In terms when csc490 is offered, csc454 must also be offered.
Summary of techniques for writing queries in relational algebra
Approaching the problem

- Ask yourself which relations need to be involved. Ignore the rest.
- Every time you combine relations, confirm that
  - attributes that should match will be made to match and
  - attributes that will be made to match should match.
- Annotate each subexpression, to show the attributes of its resulting relation.
Breaking down the problem

- Remember that you must look one tuple at a time.
  - If you need info from two different tuples, you must make a new relation where it’s in one tuple.

- Is there an intermediate relation that would help you get the final answer?
  - Draw it out with actual data in it.

- Use assignment to define those intermediate relations.
  - Use good names for the new relations.
  - Name the attributes on the LHS each time, so you don’t forget what you have in hand.
  - Add a comment explaining exactly what’s in the relation.
Specific types of query

- **Max** (min is analogous):
  - Pair tuples and find those that are *not* the max.
  - Then subtract from all to find the maxes.

- **“k or more”**:
  - Make all combos of k different tuples that satisfy the condition.

- **“exactly k”**:
  - “k or more” - “(k+1) or more”.

- **“every”**:
  - Make all combos that should have occurred.
  - Subtract those that *did* occur to find those that didn’t always. These are the failures.
  - Subtract the failures from all to get the answer.
Relational algebra wrap-up
RA is procedural

- An RA query itself suggests a procedure for constructing the result (i.e., how one could implement the query).
- We say that it is “procedural.”
Evaluating queries

- Any problem has multiple RA solutions.
  - Each solution suggests a “query execution plan”.
  - Some may seem a more efficient.
- But in RA, we won’t care about efficiency; it’s an algebra.
- In a DBMS, queries actually are executed, and efficiency matters.
  - Which query execution plan is most efficient depends on the data in the database and what indices you have.
  - Fortunately, the DBMS optimizes our queries.
  - We can focus on what we want, not how to get it.
Relational Calculus

- Another abstract query language for the relational model.
- Based on first-order logic.
- RC is “declarative”: the query describes what you want, but not how to get it.
- Queries look like this:
  \[ \{ t \mid t \in \text{Movies} \land t[\text{director}] = \text{“Scott”} \} \]
- Expressive power (when limited to queries that generate finite results) is the same as RA. It is “relationally complete.”