Overview
- Operations on whole relations
  - Inputs are sets, output is a set
- Can nest arbitrarily complex expressions
- SELECT, PROJECT
- Set union $\cup$, intersection $\cap$, difference $-$, on compatible relations
- Cartesian product $\otimes$ and various flavors of JOIN
  - Division $\div$ sort of inverse of product
- Aggregate functions (unofficial)

Select
- Unary operation
- Select a subset of tuples from a relation, based upon a condition
  - Use AND, OR, NOT for compound conditions
- Result: a table with same attributes as original: a proper subset
  - May be given a name (temporary)
- Notation:
  $\sigma_{\text{condition}}(\text{relationname})$

Project
- Unary operation
- Select a subset of columns
- Result: a table with same number of rows as original
  - Not actually a subset of the original (unlike $\sigma$)
  - May be given a name (temporary)
- Notation:
  $\pi_{\text{col-list}}(\text{relationname})$

Join
- A binary operation on relations
- Result is a whole relation
- General description: a $\otimes$ followed by a $\sigma$.
  - The $\sigma$ condition equates or otherwise relates common attributes between the two relations
  - Often a superfluous common attribute is removed
- Notation (these slides):
  $R_1 \_\_\_ \text{join-condition} \_\_\_ R_2$

"Equi" and "Natural" join
- Common attributes are compared for equality
  - No need to specify a join condition
  - Could join on more than one attribute
  - Need to list attributes if names are not the same
- "Natural join": Superfluous columns are removed automatically
- Notation (our text): $R_1 *_{\text{attr-list}} R_2$
Division: R1 ÷ R2
• Sort of the reverse of Cartesian product
• Like integer division in that any "remainder" is discarded
• Main idea: find all the tuples in R1 which are joined to all the values in R2
  – the R2 attributes are discarded
• Same thing can be accomplished with combination of \( \Pi \), \( \otimes \), –

Division Details of R = R1 ÷ R2
• R1 (dividend): attribute set X \( \cup \) Y, |R1| rows
• R2 (divisor): attribute set Y, |R2| rows
• R (quotient):
  – attribute set X, i.e., the attributes of R1 not in R2
  – at most |R1|/|R2| rows
  – A row is in the answer (R) if that row (X attributes) occurs in R1 with each combination of the rows (Y attributes) of R2.

Division Examples
• Who would have lots to talk about with Bessie? "Find (all) customers who have rented (all) the same movies as Bessie has."
• What airlines compete with Horizon Air? "Find the airlines which serve a city also served by Horizon" (not a division query).
• Which airline is best positioned to put Horizon Air out of business? "Find the airlines which fly to (all) the cities served by Horizon" (a division query).

Aggregate Functions
• Technically, not part of R.A.
• Actual query languages will implement many of these
  – (Usually) unary operators, take a whole relation and compute a value
  – COUNT, AVERAGE, MAX, MIN
• Result is returned as a relation with one row and one column
  – i.e., not as a scalar number

Grouping and Aggregates
• Rows may be grouped based on attribute values
  – Think of it as a sort on those attributes
• Aggregate functions can be applied to the grouped relation
  – Computes a value for each group
• Result returned as a relation with one row for each group, one column for each aggregate function

Grouping Notation and Example
• \(<\text{grouping attributes}> \otimes <\text{agg. function list}> (\text{relation})\)
• "List number of employees and average salary for each department"

<table>
<thead>
<tr>
<th>DNAME</th>
<th>COUNT (SSN)</th>
<th>AVERAGE (SALARY)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SW Support</td>
<td>54</td>
<td>$30,301</td>
</tr>
<tr>
<td>HW Support</td>
<td>18</td>
<td>$72,600</td>
</tr>
<tr>
<td>Grounds</td>
<td>5</td>
<td>$89,000</td>
</tr>
</tbody>
</table>
Looking Ahead

• Order of operations affects efficiency
• Example: $\sigma(R1) \ast \sigma(R2)$ probably much faster than $\sigma(R1 \ast R2)$
• Large joins can be particularly taxing
• Ideally, we do not let this affect how we write queries!
• Smart DBMSs do "query optimization"
  – automatically reorder operations for efficiency