Introduction to Database Systems
CSE 444

Lecture 18: Relational Algebra
Outline

• Motivation and sets v.s. bags
• Relational Algebra
• Translation from SQL to the Relational Algebra

• Read Sections 2.4, 5.1, and 5.2
  – [Old edition: 5.1 through 5.4]
  – These book sections go over relational operators
The WHAT and the HOW

• In SQL, we write **WHAT** we want to get from the data

• The database system needs to figure out **HOW** to get the data we want

• The passage from **WHAT** to **HOW** goes through the **Relational Algebra**
SQL = WHAT

Product(pid, name, price)
Purchase(pid, cid, store)
Customer(cid, name, city)

SELECT DISTINCT x.name, z.name
FROM Product x, Purchase y, Customer z
WHERE x.pid = y.pid and y.cid = z.cid and
x.price > 100 and z.city = 'Seattle'

It’s clear WHAT we want, unclear HOW to get it
Relational Algebra = HOW

Product(pid, name, price)
Purchase(pid, cid, store)
Customer(cid, name, city)

Temporary tables T1, T2, . . .

\[ \delta \]
\[ \Pi \]
\[ \sigma \text{ price}>100 \text{ and city='Seattle'} \]

Final answer

T4(name,name)

T3(. . .)

T2( . . .)

T1(pid,name,price,pid,cid,store)
Relational Algebra = HOW

The order is now clearly specified:
• Iterate over PRODUCT…
• …join with PURCHASE…
• …join with CUSTOMER…
• …select tuples with Price>100 and City='Seattle'…
• …eliminate duplicates…
• …and that’s the final answer !
Relations

• A relation is a set of tuples
  – Sets: \{a, b, c\}, \{a, d, e, f\}, \{\}\, . . .

• But, commercial DBMS’s implement relations that are bags rather than sets
  – Bags: \{a, a, b, c\}, \{b, b, b, b, b\}, . . .
Sets v.s. Bags

Relational Algebra has two flavors:
• Over sets: theoretically elegant but limited
• Over bags: needed for SQL queries + more efficient
  – Example: Compute average price of all products

We discuss set semantics
• We mention bag semantics only where needed
Outline

• Motivation and sets v.s. bags
• Relational Algebra
• Translation from SQL to the Relational Algebra

• Read Sections 2.4, 5.1, and 5.2
  – [Old edition: 5.1 through 5.4]
  – These book sections go over relational operators
Relational Algebra

• **Query language** associated with relational model

• **Queries specified in an operational manner**
  – A query gives a step-by-step procedure

• **Relational operators**
  – Take one or two relation instances as argument
  – Return one relation instance as result
  – Easy to **compose** into **relational algebra expressions**
Relational Algebra (1/3)

Five basic operators:

• **Union** ($\cup$) and **Set difference** ($-$)

• **Selection**: $\sigma_{\text{condition}}(S)$
  - Condition is Boolean combination ($\wedge, \vee$) of terms
  - Term is: attribute op constant, attr. op attr.
  - Op is: $<$, $\leq$, $=$, $\neq$, $\geq$, or $>$

• **Projection**: $\pi_{\text{list-of-attributes}}(S)$

• **Cross-product** or **cartesian product** ($\times$)
Relational Algebra (2/3)

Derived or auxiliary operators:

- **Intersection** $(\cap)$, **Division** $(R/S)$

- **Join**: $R \bowtie_\theta S = \sigma_\theta(R \times S)$

- **Variations of joins**
  - Natural, equijoin, theta-join
  - Outer join and semi-join

- **Rename** $\rho_{B_1,\ldots,B_n}(S)$
Extensions for bags

• Duplicate elimination: $\delta$
• Group by: $\gamma$ [Same symbol as aggregation]
  – Partitions tuples of a relation into “groups”
• Sorting: $\tau$

Other extensions

• Aggregation: $\gamma$ (min, max, sum, average, count)
Union and Difference

• R1 ∪ R2
• Example:
  – ActiveEmployees ∪ RetiredEmployees

• R1 – R2
• Example:
  – AllEmployees – RetiredEmployees

Be careful when applying to bags!
What about Intersection?

- It is a derived operator
- \( R_1 \cap R_2 = R_1 - (R_1 - R_2) \)
- Also expressed as a join (will see later)
- Example
  - UnionizedEmployees \( \cap \) RetiredEmployees
Relational Algebra (1/3)

Five basic operators:

- **Union** ($\cup$) and **Set difference** ($-$)
- **Selection**: $\sigma_{\text{condition}}(S)$
  - Condition is Boolean combination ($\land, \lor$) of terms
  - Term is: attribute op constant, attr. op attr.
  - Op is: $<$, $\leq$, $=$, $\neq$, $\geq$, or $>$
- **Projection**: $\pi_{\text{list-of-attributes}}(S)$
- **Cross-product** or **cartesian product** ($\times$)
Selection

• Returns all tuples that satisfy a condition
• Notation: $\sigma_c(R)$
• Examples
  – $\sigma_{\text{Salary} > 40000}$ (Employee)
  – $\sigma_{\text{name} = \text{"Smith"}}$ (Employee)
• The condition $c$ can be
  – Boolean combination ($\land, \lor$) of terms
  – Term is: attribute op constant, attr. op attr.
  – Op is: $<$, $<=$, $=$, $\neq$, $>=$, or $>$
\[ \sigma_{\text{Salary} > 40000} \text{(Employee)} \]

<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>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SSN</th>
<th>Name</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>5423341</td>
<td>Smith</td>
<td>600000</td>
</tr>
<tr>
<td>4352342</td>
<td>Fred</td>
<td>500000</td>
</tr>
</tbody>
</table>
Projection

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

Semantics differs over set or over bags
<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>

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>20000</td>
</tr>
<tr>
<td>John</td>
<td>60000</td>
</tr>
</tbody>
</table>

Set semantics: duplicate elimination automatic
Bag semantics: no duplicate elimination; need explicit \( \delta \)
## Selection & Projection Examples

### Patient

<table>
<thead>
<tr>
<th>no</th>
<th>name</th>
<th>zip</th>
<th>disease</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>p1</td>
<td>98125</td>
<td>flu</td>
</tr>
<tr>
<td>2</td>
<td>p2</td>
<td>98125</td>
<td>heart</td>
</tr>
<tr>
<td>3</td>
<td>p3</td>
<td>98120</td>
<td>lung</td>
</tr>
<tr>
<td>4</td>
<td>p4</td>
<td>98120</td>
<td>heart</td>
</tr>
</tbody>
</table>

\[
\sigma_{\text{disease}=\text{\textquoteleft heart\textquoteright}}(\text{Patient})
\]

<table>
<thead>
<tr>
<th>no</th>
<th>name</th>
<th>zip</th>
<th>disease</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>p2</td>
<td>98125</td>
<td>heart</td>
</tr>
<tr>
<td>4</td>
<td>p4</td>
<td>98120</td>
<td>heart</td>
</tr>
</tbody>
</table>

\[
\pi_{\text{zip, disease}}(\text{Patient})
\]

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

\[
\pi_{\text{zip}}(\sigma_{\text{disease}=\text{\textquoteleft heart\textquoteright}}(\text{Patient}))
\]

<table>
<thead>
<tr>
<th>zip</th>
</tr>
</thead>
<tbody>
<tr>
<td>98120</td>
</tr>
<tr>
<td>98125</td>
</tr>
</tbody>
</table>
Relational Algebra (1/3)

Five basic operators:

- **Union** (\(\cup\)) and **Set difference** (\(-\))
- **Selection**: \(\sigma_{condition}(S)\)
  - Condition is Boolean combination (\(\wedge, \vee\)) of terms
  - Term is: attribute op constant, attr. op attr.
  - Op is: <, <=, =, \(\neq\), >=, or >
- **Projection**: \(\pi_{list-of-attributes}(S)\)
- **Cross-product** or **cartesian product** (\(\times\))
Cartesian Product

- Each tuple in R1 with each tuple in R2
- Notation: $R_1 \times R_2$
- Example:
  - Employee $\times$ Dependents
- Rare in practice; mainly used to express joins
## Cartesian Product Example

**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>

**Dependents**

<table>
<thead>
<tr>
<th>Employee</th>
<th>SSN</th>
<th>Dname</th>
</tr>
</thead>
<tbody>
<tr>
<td>9999999999</td>
<td>Emily</td>
<td></td>
</tr>
<tr>
<td>7777777777</td>
<td>Joe</td>
<td></td>
</tr>
</tbody>
</table>

**Employee x Dependents**

<table>
<thead>
<tr>
<th>Name</th>
<th>SSN</th>
<th>Employee</th>
<th>Dname</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>
Relational Algebra (2/3)

Derived or auxiliary operators:

- **Intersection** ($\cap$), **Division** ($R/S$)
- **Join**: $R \bowtie_{\theta} S = \sigma_{\theta}(R \times S)$
- **Variations of joins**
  - Natural, equijoin, theta-join
  - Outer join and semi-join
- **Rename** $\rho_{B_1,\ldots,B_n}(S)$
Renaming

- Changes the schema, not the instance
- Notation: \( \rho_{B_1,\ldots,B_n}(R) \)
- Example:
  - \( \rho_{\text{LastName}, \text{SocSocNo}}(\text{Employee}) \)
  - Output schema: \( \text{Answer(LastName, SocSocNo)} \)
### Renaming Example

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

\[ \rho_{\text{LastName}, \text{SocSocNo}} (\text{Employee}) \]

<table>
<thead>
<tr>
<th>LastName</th>
<th>SocSocNo</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>9999999999</td>
</tr>
<tr>
<td>Tony</td>
<td>7777777777</td>
</tr>
</tbody>
</table>

Magda Balazinska - CSE 444, Fall 2010
Relational Algebra (2/3)

Derived or auxiliary operators:

• **Intersection** (\(\cap\)), **Division** (\(R/S\))

• **Join**: \(R \bowtie_\theta S = \sigma_\theta(R \times S)\)

• **Variations of joins**
  – Natural, equijoin, theta-join
  – Outer join and semi-join

• **Rename** \(\rho_{B_1, \ldots, B_n}(S)\)
Different Types of Join

- **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 $S$
Theta-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>
</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>

$P \Join_{P.age=J.age \land P.zip=J.zip \land P.age < 50} J$

<table>
<thead>
<tr>
<th>P.age</th>
<th>P.zip</th>
<th>disease</th>
<th>job</th>
<th>J.age</th>
<th>J.zip</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>98120</td>
<td>flu</td>
<td>cashier</td>
<td>20</td>
<td>98120</td>
</tr>
</tbody>
</table>
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>

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>

$P \bowtie_{P.age=J.age} J$

<table>
<thead>
<tr>
<th>age</th>
<th>P.zip</th>
<th>disease</th>
<th>job</th>
<th>J.zip</th>
</tr>
</thead>
<tbody>
<tr>
<td>54</td>
<td>98125</td>
<td>heart</td>
<td>lawyer</td>
<td>98125</td>
</tr>
<tr>
<td>20</td>
<td>98120</td>
<td>flu</td>
<td>cashier</td>
<td>98120</td>
</tr>
</tbody>
</table>
Natural 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>
</tbody>
</table>

AnnonJob 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>

\[ P \bowtie J \]

<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>
</tbody>
</table>
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**

<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>
Semijoin

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

- Semijoins are used in distributed databases

\[
\text{Employee} \times \left|_{snn=ssn} \right. (\sigma_{\text{age}>71} \left(\text{Dependents}\right))
\]

\[
T = \Pi_{snn} (\sigma_{\text{age}>71} \left(\text{Dependents}\right))
\]

\[
R = \text{Employee} \bowtie T
\]

\[
\text{Answer} = R \bowtie \text{Dependents}
\]
Complex RA Expressions

\[ \Pi_{\text{name}} \]

\[ \sigma_{\text{name}=\text{fred}} \]

\[ \Pi_{\text{ssn}} \]

\[ \sigma_{\text{name}=\text{gizmo}} \]

\[ \Pi_{\text{pid}} \]

Person         Purchase            Person          Product
Example of Algebra Queries

Q1: Jobs of patients who have heart disease

\[ \pi_{\text{job}}(\text{AnnonJob} \bowtie (\sigma_{\text{disease} = 'heart'} (\text{AnonPatient}))) \]
More Examples

Supplier(sno,sname,scity,sstate)
Part(pno,pname,psize,pcolor)
Supply(sno,pno,qty,price)

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

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

• An Algebra Expression is like a program
  – Several operations
  – Strictly specified order

• But Algebra expressions have limitations
RA and Transitive Closure

- Cannot compute “transitive closure”

<table>
<thead>
<tr>
<th>Name1</th>
<th>Name2</th>
<th>Relationship</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fred</td>
<td>Mary</td>
<td>Father</td>
</tr>
<tr>
<td>Mary</td>
<td>Joe</td>
<td>Cousin</td>
</tr>
<tr>
<td>Mary</td>
<td>Bill</td>
<td>Spouse</td>
</tr>
<tr>
<td>Nancy</td>
<td>Lou</td>
<td>Sister</td>
</tr>
</tbody>
</table>

- Find all direct and indirect relatives of Fred
- Cannot express in RA !!! Need to write Java program
Outline

• Motivation and sets v.s. bags
• Relational Algebra
• Translation from SQL to the Relational Algebra

• Read Sections 2.4, 5.1, and 5.2
  – [Old edition: 5.1 through 5.4]
  – These book sections go over relational operators
From SQL to RA

Product(pid, name, price)
Purchase(pid, cid, store)
Customer(cid, name, city)

SELECT DISTINCT x.name, z.name
FROM Product x, Purchase y, Customer z
WHERE x.pid = y.pid and y.cid = y.cid and
  x.price > 100 and z.city = 'Seattle'
From SQL to RA

\[ \delta \]

\[ \Pi \ x.\text{name}, z.\text{name} \]

\[ \sigma \ \text{price}>100 \text{ and city}='\text{Seattle}' \]

\[ \Pi \ x.\text{name}, z.\text{name} \]

\[ \sigma \ \text{price}>100 \text{ and city}='\text{Seattle}' \]

\[ \delta \]

\[ \Pi \ x.\text{name}, z.\text{name} \]

\[ \delta \]

\[ \Pi \ x.\text{name}, z.\text{name} \]

\[ \sigma \ \text{price}>100 \text{ and city}='\text{Seattle}' \]

\[ \Pi \ x.\text{name}, z.\text{name} \]

\[ \delta \]

\[ \Pi \ x.\text{name}, z.\text{name} \]

\[ \sigma \ \text{price}>100 \text{ and city}='\text{Seattle}' \]

\[ \Pi \ x.\text{name}, z.\text{name} \]

\[ \delta \]

\[ \Pi \ x.\text{name}, z.\text{name} \]

\[ \sigma \ \text{price}>100 \text{ and city}='\text{Seattle}' \]

\[ \Pi \ x.\text{name}, z.\text{name} \]

\[ \delta \]

\[ \Pi \ x.\text{name}, z.\text{name} \]

\[ \sigma \ \text{price}>100 \text{ and city}='\text{Seattle}' \]

\[ \Pi \ x.\text{name}, z.\text{name} \]

\[ \delta \]

\[ \Pi \ x.\text{name}, z.\text{name} \]

\[ \sigma \ \text{price}>100 \text{ and city}='\text{Seattle}' \]

\[ \Pi \ x.\text{name}, z.\text{name} \]

\[ \delta \]

\[ \Pi \ x.\text{name}, z.\text{name} \]

\[ \sigma \ \text{price}>100 \text{ and city}='\text{Seattle}' \]

\[ \Pi \ x.\text{name}, z.\text{name} \]

\[ \delta \]

\[ \Pi \ x.\text{name}, z.\text{name} \]

\[ \sigma \ \text{price}>100 \text{ and city}='\text{Seattle}' \]

\[ \Pi \ x.\text{name}, z.\text{name} \]

\[ \delta \]

\[ \Pi \ x.\text{name}, z.\text{name} \]

\[ \sigma \ \text{price}>100 \text{ and city}='\text{Seattle}' \]

\[ \Pi \ x.\text{name}, z.\text{name} \]

\[ \delta \]

\[ \Pi \ x.\text{name}, z.\text{name} \]

\[ \sigma \ \text{price}>100 \text{ and city}='\text{Seattle}' \]

\[ \Pi \ x.\text{name}, z.\text{name} \]
Query optimization = finding cheaper equivalent expressions
Operators on Bags

- Duplicate elimination $\delta$
- Grouping $\gamma$
- Sorting $\tau$
Logical Query Plan

\[ \text{SELECT} \text{ city}, \text{count}(\ast) \\text{FROM} \text{ sales} \\text{GROUP BY} \text{city} \\text{HAVING} \text{sum(price)} > 100 \]

\[ \sigma_{p > 100} \]

\[ \Pi_{\text{city, c}} \]

\[ \gamma_{\text{city, sum(price)} \rightarrow p, \text{count}(\ast) \rightarrow c} \]

\[ \text{T1(city, p, c)} \]

\[ \text{T2(city, p, c)} \]

\[ \text{T3(city, c)} \]

T1, T2, T3 = temporary tables

sales(product, city, price)
Non-monontone Queries (at home !)

Product(pid, name, price)
Purchase(pid, cid, store)
Customer(cid, name, city)

```
SELECT DISTINCT z.store
FROM Customer z
WHERE z.city='Seattle' AND
  not exists (select *
               from Product x, Purchase y
               where x.pid= y.pid
               and y.cid = z.cid
               and x.price < 100)
```