Data Types in SQL

• Values in relational model have a data type
  – Implicit in the formal relational model
• Types available vary by manufacturer
  – Number, String, Date, …
  – Constants, variables – familiar concepts
  – Often extensible (similar to a new class in Java)
• For now, just assume whatever is convenient
  – You'll need to use Oracle conventions for projects
**Data Type Example: Date and Time**

- **date**: Dates, containing a (4 digit) year, month and date
  - Example: `date '2005-7-27'`
- **time**: Time of day, in hours, minutes and seconds.
  - Example: `time '09:00:30'`  `time '09:00:30.75'`
- **timestamp**: date plus time of day
  - Example: `timestamp '2005-7-27 09:00:30.75'`
- **interval**: period of time
  - Example: interval ‘1’ day
  - Subtracting a date/time/timestamp value from another gives an interval value
  - Interval values can be added to date/time/timestamp values

**Oracle uses a “Date” type that corresponds to timestamp**

- Very specific (and adjustable, but inflexible) constant format

---

**Null Values**

- It is possible for tuples to have a null value, denoted by `null`, for some of their attributes
- `null` signifies an unknown value or that a value does not exist.
- The result of any arithmetic expression involving `null` is `null`
  - Example: `5 + null` returns null
- The predicate `is null` can be used to check for null values.
  - Example: Find all instructors whose salary is null.
    ```sql
    select name
    from instructor
    where salary is null
    ```
Null Values and Three Valued Logic

- Three values – true, false, unknown
- Any comparison with null returns unknown
  - Example: 5 < null or null <> null or null = null
- Three-valued logic using the value unknown:
  - OR: (unknown or true) = true,
    (unknown or false) = unknown
    (unknown or unknown) = unknown
  - AND: (true and unknown) = unknown,
    (false and unknown) = false,
    (unknown and unknown) = unknown
  - NOT: (not unknown) = unknown
  - “P is unknown” evaluates to true if predicate P evaluates to unknown
- Result of where clause predicate is treated as false if it evaluates to unknown

Relational Algebra

Mathematical view of SQL queries

- Formal view of what we’ve seen in SQL
  - Selection (where clause)
  - Projection (select clause)
- Cartesian Product
- Join
- Set operations
- Aggregation
  - And how these are done in SQL
“Core” Relational Algebra

A small set of operators that allow us to manipulate relations in limited but useful ways. The operators are:

1. Union, intersection, and difference: the usual set operators.
   - But the relation schemas must be the same.
2. Selection: Picking certain rows from a relation.
4. Products and joins: Composing relations in useful ways.
5. Renaming of relations and their attributes.

Relational Algebra

<table>
<thead>
<tr>
<th>Operator</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \sigma )</td>
<td>UNARY</td>
</tr>
<tr>
<td>( \pi )</td>
<td>UNARY</td>
</tr>
<tr>
<td>( \times )</td>
<td>FUNDAMENTAL</td>
</tr>
<tr>
<td>( \cup )</td>
<td>BINARY</td>
</tr>
<tr>
<td>( \setminus )</td>
<td></td>
</tr>
<tr>
<td>( \cap )</td>
<td></td>
</tr>
<tr>
<td>( \theta )</td>
<td></td>
</tr>
<tr>
<td>( \bowtie )</td>
<td></td>
</tr>
<tr>
<td>( \div )</td>
<td></td>
</tr>
</tbody>
</table>

- **Properties:**
  - Limited expressive power (subset of possible queries), but rich enough to be useful
  - Closed (input is relation, output is relation)
  - Finite (result always defined)
  - Easy to automatically determine how to efficiently process
Select Operation

- Notation: $\sigma_p(r)$
- $p$ is called the selection predicate
  - Corresponds to SQL where clause
- Defined as:
  $$\sigma_p(r) = \{ t | t \in r \text{ and } p(t) \}$$

Where $p$ is a formula in propositional calculus consisting of terms connected by: $\land$ (and), $\lor$ (or), $\neg$ (not)
Each term is one of:
- $<attribute>$ op $<attribute>$ or $<constant>$
  where op is one of: $=, \neq, >, \geq, <, \leq$

- Example of selection:
  $$\sigma_{\text{dept\_name}=\text{Physics}}(\text{instructor})$$

Project Operation

- Notation:
  $$\Pi_{A_1, A_2, \ldots, A_k}(r)$$
  where $A_1, A_2$ are attribute names and $r$ is a relation name.
  - Corresponds to SQL Select distinct clause
- The result is defined as the relation of $k$ columns obtained by erasing the columns that are not listed
- Duplicate rows removed from result, since relations are sets
- Example: To eliminate the $\text{dept\_name}$ attribute of $\text{instructor}$
  $$\Pi_{\text{ID}, \text{name}, \text{salary}}(\text{instructor})$$
Cartesian Product

\[ R = R_1 \times R_2 \]

pairs each tuple \( t_1 \) of \( R_1 \) with each tuple \( t_2 \) of \( R_2 \) and puts in \( R \) a tuple \( t_1 t_2 \).

### Cartesian Product Operation

- Notation \( r \times s \)
- Defined as:
  \[ r \times s = \{ t q \mid t \in r \text{ and } q \in s \} \]
- Assume that attributes of \( r(R) \) and \( s(S) \) are disjoint. (That is, \( R \cap S = \emptyset \)).
  - If attributes of \( r(R) \) and \( s(S) \) are not disjoint, then renaming must be used (later)
- SQL: This is the “from” clause
  - select *
    from \( r, s \)
The from Clause

- The `from` clause lists the relations involved in the query
  - Correlates to the Cartesian product operation of the relational algebra.
- Find the Cartesian product `instructor X teaches`

```sql
select *
from instructor, teaches
```
- Generates every possible `instructor` – `teaches` pair, with all attributes from both relations.
- For common attributes (e.g., `ID`), the attributes in the resulting table are renamed using the relation name (e.g., `instructor.ID`)
- Cartesian product not very useful directly, but useful combined with where-clause condition (selection operation in relational algebra).

---

### Cartesian Product

<table>
<thead>
<tr>
<th>instructor</th>
<th>teaches</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>name</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>10101</td>
<td>Srinivasan</td>
</tr>
<tr>
<td>10101</td>
<td>Srinivasan</td>
</tr>
<tr>
<td>10101</td>
<td>Srinivasan</td>
</tr>
<tr>
<td>12121</td>
<td>Wu</td>
</tr>
<tr>
<td>15151</td>
<td>Mozart</td>
</tr>
<tr>
<td>22222</td>
<td>Einstein</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Instructor ID</th>
<th>name</th>
<th>dept_name</th>
<th>salary</th>
<th>teaches.ID</th>
<th>course_id</th>
<th>sec_id</th>
<th>semester</th>
<th>year</th>
</tr>
</thead>
<tbody>
<tr>
<td>10101</td>
<td>Srinivasan</td>
<td>Comp. Sci.</td>
<td>65000</td>
<td>10101</td>
<td>CS-101</td>
<td>1</td>
<td>Fall</td>
<td>2009</td>
</tr>
<tr>
<td>10101</td>
<td>Srinivasan</td>
<td>Comp. Sci.</td>
<td>65000</td>
<td>10101</td>
<td>CS-315</td>
<td>1</td>
<td>Spring</td>
<td>2010</td>
</tr>
<tr>
<td>10101</td>
<td>Srinivasan</td>
<td>Comp. Sci.</td>
<td>65000</td>
<td>10101</td>
<td>CS-347</td>
<td>1</td>
<td>Fall</td>
<td>2009</td>
</tr>
<tr>
<td>12121</td>
<td>Wu</td>
<td>Finance</td>
<td>90000</td>
<td>12121</td>
<td>FIN-201</td>
<td>1</td>
<td>Spring</td>
<td>2010</td>
</tr>
<tr>
<td>15151</td>
<td>Mozart</td>
<td>Music</td>
<td>40000</td>
<td>15151</td>
<td>MU-199</td>
<td>1</td>
<td>Spring</td>
<td>2010</td>
</tr>
<tr>
<td>22222</td>
<td>Einstein</td>
<td>Physics</td>
<td>95000</td>
<td>22222</td>
<td>PHY-101</td>
<td>1</td>
<td>Fall</td>
<td>2009</td>
</tr>
</tbody>
</table>
Examples

- Find the names of all instructors who have taught some course and the course_id
  
  ```sql
  select name, course_id
  from instructor, teaches
  where instructor.ID = teaches.ID
  ```

- Find the names of all instructors in the Art department who have taught some course and the course_id
  
  ```sql
  select name, course_id
  from instructor, teaches
  where instructor.ID = teaches.ID and instructor.dept_name = 'Art'
  ```
Theta-Join

- \( R \bowtie \Theta \ C \bowtie \ S \)

- What would be an interesting \( \Theta \)?
  - \( \text{instructor}.\text{ID} = \text{teaches}.\text{ID} \)
- We said this isn't fundamental, why not?
  - Equivalent to \( R = \sigma_\Theta(R_1 \times R_2) \).

<table>
<thead>
<tr>
<th>ID</th>
<th>name</th>
<th>dept_name</th>
<th>salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>10101</td>
<td>Srinivasan</td>
<td>Comp. Sci.</td>
<td>65000</td>
</tr>
<tr>
<td>12121</td>
<td>Wu</td>
<td>Finance</td>
<td>90000</td>
</tr>
<tr>
<td>15151</td>
<td>Mozart</td>
<td>Music</td>
<td>40000</td>
</tr>
<tr>
<td>22222</td>
<td>Einstein</td>
<td>Physics</td>
<td>95000</td>
</tr>
<tr>
<td>23243</td>
<td>El Said</td>
<td>History</td>
<td>60000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID</th>
<th>course_id</th>
<th>sec_id</th>
<th>semester</th>
<th>year</th>
</tr>
</thead>
<tbody>
<tr>
<td>10101</td>
<td>CS-101</td>
<td>1</td>
<td>Fall</td>
<td>2009</td>
</tr>
<tr>
<td>10101</td>
<td>CS-315</td>
<td>1</td>
<td>Spring</td>
<td>2010</td>
</tr>
<tr>
<td>10101</td>
<td>CS-347</td>
<td>1</td>
<td>Fall</td>
<td>2009</td>
</tr>
<tr>
<td>12121</td>
<td>FIN-201</td>
<td>1</td>
<td>Spring</td>
<td>2010</td>
</tr>
<tr>
<td>15151</td>
<td>MU-199</td>
<td>1</td>
<td>Spring</td>
<td>2010</td>
</tr>
<tr>
<td>22222</td>
<td>PHY-101</td>
<td>1</td>
<td>Fall</td>
<td>2009</td>
</tr>
</tbody>
</table>

Theta-Join: \( R \bowtie \ S \) where \( \sigma_{\Theta}(R \times S) \)

- \( \sigma_{\Theta}(R \times S) \)
  - \( \Theta \) can be \(< \), \(=\), \(\neq\), \(\leq\), \(\geq\)
  - If equal \(=\), then it is an EQUIJOIN

\[
\begin{align*}
R \bowtie \Theta \ S &= \sigma_{\Theta}(R \times S) \\
R(A B C) \bowtie \Theta S(C D E) &= R(A B C) \bowtie \Theta S(C D E)
\end{align*}
\]

result has schema \( T(A B C C' D E) \)
Natural Join

\[ R = R_1 \bowtie R_2 \]

calls for the theta-join of \( R_1 \) and \( R_2 \) with the condition that all attributes of the same name be equated. Then, one column for each pair of equated attributes is projected out.

Example

<table>
<thead>
<tr>
<th>( ID )</th>
<th>name</th>
<th>dept_name</th>
<th>salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>10101</td>
<td>Srinivasan</td>
<td>Comp. Sci.</td>
<td>65000</td>
</tr>
<tr>
<td>12121</td>
<td>Wu</td>
<td>Finance</td>
<td>90000</td>
</tr>
<tr>
<td>15151</td>
<td>Mozart</td>
<td>Music</td>
<td>40000</td>
</tr>
<tr>
<td>22222</td>
<td>Einstein</td>
<td>Physics</td>
<td>95000</td>
</tr>
<tr>
<td>32343</td>
<td>El Said</td>
<td>History</td>
<td>60000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( ID )</th>
<th>course_id</th>
<th>sec_id</th>
<th>semester</th>
<th>year</th>
</tr>
</thead>
<tbody>
<tr>
<td>10101</td>
<td>CS-101</td>
<td>1</td>
<td>Fall</td>
<td>2009</td>
</tr>
<tr>
<td>10101</td>
<td>CS-315</td>
<td>1</td>
<td>Spring</td>
<td>2010</td>
</tr>
<tr>
<td>10101</td>
<td>CS-347</td>
<td>1</td>
<td>Fall</td>
<td>2009</td>
</tr>
<tr>
<td>12121</td>
<td>FIN-201</td>
<td>1</td>
<td>Spring</td>
<td>2010</td>
</tr>
<tr>
<td>15151</td>
<td>MU-199</td>
<td>1</td>
<td>Spring</td>
<td>2010</td>
</tr>
<tr>
<td>22222</td>
<td>PHY-101</td>
<td>1</td>
<td>Fall</td>
<td>2009</td>
</tr>
</tbody>
</table>

Related Pages:

- Page 627

Joined Relations

- **Join operations** take two relations and return as a result another relation.
- A join operation is a Cartesian product which requires that tuples in the two relations match (under some condition). It also specifies the attributes that are present in the result of the join.
- The join operations are typically used as subquery expressions in the \textit{from} clause.
Join operations – Example

- Relation course

<table>
<thead>
<tr>
<th>course_id</th>
<th>title</th>
<th>dept_name</th>
<th>credits</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIO-301</td>
<td>Genetics</td>
<td>Biology</td>
<td>4</td>
</tr>
<tr>
<td>CS-190</td>
<td>Game Design</td>
<td>Comp. Sci.</td>
<td>4</td>
</tr>
<tr>
<td>CS-315</td>
<td>Robotics</td>
<td>Comp. Sci.</td>
<td>3</td>
</tr>
</tbody>
</table>

- Relation prereq

<table>
<thead>
<tr>
<th>course_id</th>
<th>prereq_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIO-301</td>
<td>BIO-101</td>
</tr>
<tr>
<td>CS-190</td>
<td>CS-101</td>
</tr>
<tr>
<td>CS-347</td>
<td>CS-101</td>
</tr>
</tbody>
</table>

- Observe that
  prereq information is missing for CS-315 and
course information is missing for CS-437
Announcements

• Assignment 1 due today at 11:59pm
• Project 1 (SQL queries) will be released later today – watch web site and email
  – Due September 16
• No class on Monday – Labor day
  – PSOs will be held as usual next week

Rename Operation

- Allows us to name, and therefore to refer to, the results of relational-algebra expressions.
- Allows us to refer to a relation by more than one name.
- Example:

  \( \rho_X(E) \)

  returns the expression \( E \) under the name \( X \)

- If a relational-algebra expression \( E \) has arity \( n \), then

  \( \rho_{X(A_1, A_2, \ldots, A_n)}(E) \)

  returns the result of expression \( E \) under the name \( X \), and with the attributes renamed to \( A_1, A_2, \ldots, A_n \).

- SQL: `select E.first=A1, E.second=A2 as x`
Union Operation

- Notation: \( r \cup s \)
- Defined as:
  \[
  r \cup s = \{ t | t \in r \text{ or } t \in s \}
  \]
- For \( r \cup s \) to be valid:
  1. \( r, s \) must have the same arity (same number of attributes)
  2. The attribute domains must be compatible (example: 2\textsuperscript{nd} column of \( r \) deals with the same type of values as does the 2\textsuperscript{nd} column of \( s \))
- Example: to find all courses taught in the Fall 2009 semester, or in the Spring 2010 semester, or in both
  \[
  \Pi_{\text{course}_id}(\sigma\text{ semester}='\text{Fall}' \land \text{year}=2009(\text{section})) \cup \\
  \Pi_{\text{course}_id}(\sigma\text{ semester}='\text{Spring}' \land \text{year}=2010(\text{section}))
  \]
- SQL: `select * from r union select * from s`

Set-Intersection Operation

- Notation: \( r \cap s \)
- Defined as:
  \[
  r \cap s = \{ t | t \in r \text{ and } t \in s \}
  \]
- Assume:
  - \( r, s \) have the same arity
  - attributes of \( r \) and \( s \) are compatible
- Note: \( r \cap s = r - (r - s) \)
- SQL: `select * from r intersect select * from s`
Set Difference Operation

- Notation $r - s$
- Defined as:
  $$r - s = \{ t \mid t \in r \text{ and } t \notin s \}$$

- Set differences must be taken between compatible relations.
  - $r$ and $s$ must have the same arity
  - attribute domains of $r$ and $s$ must be compatible

- Example: to find all courses taught in the Fall 2009 semester, but not in the Spring 2010 semester
  $$\Pi_{course_id} (\sigma_{semester="Fall" \land year=2009} (section)) - \Pi_{course_id} (\sigma_{semester="Spring" \land year=2010} (section))$$

- SQL: `select * from r minus select * from s`

Example

<table>
<thead>
<tr>
<th>Indoor</th>
<th>Date</th>
<th>Time</th>
<th>Room</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bloomberg Day</td>
<td>Aug 30</td>
<td>10:00am</td>
<td>LWSN Commons</td>
</tr>
<tr>
<td>Black Lives Matter Panel Discussion</td>
<td>Sep 13</td>
<td>6:30pm</td>
<td>STEW Fowler Hall</td>
</tr>
<tr>
<td>CERIAS New Student Welcome</td>
<td>Aug 30</td>
<td>6:00pm</td>
<td>LWSN 1142</td>
</tr>
<tr>
<td>Global Fest</td>
<td>Sep 13</td>
<td>11:00am</td>
<td>Morton Center</td>
</tr>
<tr>
<td>Microsoft Day</td>
<td>Aug 31</td>
<td>10:00am</td>
<td>LWSN Commons</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Outdoor</th>
<th>Date</th>
<th>Time</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mosey Down Main Street</td>
<td>Sep 3</td>
<td>6:00pm</td>
<td>Main Street, Lafayette</td>
</tr>
<tr>
<td>Purdue Student Board (PSUB) Night</td>
<td>Aug 26</td>
<td>8:00pm</td>
<td>PMU Front Lawn</td>
</tr>
<tr>
<td>B-Involved Fair</td>
<td>Aug 27</td>
<td>3:00pm</td>
<td>Memorial Mall Square</td>
</tr>
</tbody>
</table>
**select * from indoor union
select * from outdoor**

- A: Table at right
- B: Schema doesn’t match, can’t take Union
- C: Fails, need to rename
- D: Unsure, let’s cover more

### Example

#### Indoor

<table>
<thead>
<tr>
<th>Title</th>
<th>Date</th>
<th>Time</th>
<th>Room</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bloomberg Day</td>
<td>Aug 30</td>
<td>10:00am</td>
<td>LWSN Commons</td>
</tr>
<tr>
<td>Black Lives Matter Panel Discussion</td>
<td>Sep 13</td>
<td>6:30pm</td>
<td>STEW Fowler Hall</td>
</tr>
<tr>
<td>CERIAS New Student Welcome</td>
<td>Aug 30</td>
<td>6:00pm</td>
<td>LWSN 1142</td>
</tr>
<tr>
<td>Global Fest</td>
<td>Sep 13</td>
<td>11:00am</td>
<td>Morton Center</td>
</tr>
<tr>
<td>Microsoft Day</td>
<td>Aug 31</td>
<td>10:00am</td>
<td>LWSN Commons</td>
</tr>
</tbody>
</table>

#### Outdoor

<table>
<thead>
<tr>
<th>Title</th>
<th>Date</th>
<th>Time</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mosey Down Main Street</td>
<td>Sep 3</td>
<td>6:00pm</td>
<td>Main Street, Lafayette</td>
</tr>
<tr>
<td>Purdue Student Board (PSUB) Night</td>
<td>Aug 26</td>
<td>8:00pm</td>
<td>PMU Front Lawn</td>
</tr>
<tr>
<td>Bloomberg Day</td>
<td>Aug 30</td>
<td>10:00am</td>
<td>LWSN Commons</td>
</tr>
<tr>
<td>Black Lives Matter Panel Discussion</td>
<td>Sep 13</td>
<td>6:30pm</td>
<td>STEW Fowler Hall</td>
</tr>
<tr>
<td>CERIAS New Student Welcome</td>
<td>Aug 30</td>
<td>6:00pm</td>
<td>LWSN 1142</td>
</tr>
<tr>
<td>Global Fest</td>
<td>Sep 13</td>
<td>11:00am</td>
<td>Morton Center</td>
</tr>
<tr>
<td>B-Involved Fair</td>
<td>Aug 27</td>
<td>3:00pm</td>
<td>Memorial Mall Square</td>
</tr>
</tbody>
</table>
select * from inside intersect
select * from outside

- **A**: Table at left
- **B**: null
- **C**: Fails, need to rename
- **D**: Table below

<table>
<thead>
<tr>
<th>Title</th>
<th>Date</th>
<th>Time</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mosey Down</td>
<td>Sep 3</td>
<td>6:00pm</td>
<td>Main Street, Lafayette</td>
</tr>
<tr>
<td>Purdue Student Board (PSUB)</td>
<td>Aug 26</td>
<td>8:00pm</td>
<td>PMU Front Lawn</td>
</tr>
<tr>
<td>Bloomberg Day</td>
<td>Aug 30</td>
<td>10:00am</td>
<td>LWSN Commons</td>
</tr>
<tr>
<td>Black Lives Matter Panel</td>
<td>Sep 13</td>
<td>6:30pm</td>
<td>STEW Fowler Hall</td>
</tr>
<tr>
<td>Discussion</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CERIAS New Student Welcome</td>
<td>Aug 30</td>
<td>6:00pm</td>
<td>LWSN 1142</td>
</tr>
<tr>
<td>Global Fest</td>
<td>Sep 13</td>
<td>11:00am</td>
<td>Morton Center</td>
</tr>
<tr>
<td>Microsoft Day</td>
<td>Aug 31</td>
<td>10:00am</td>
<td>LWSN Commons</td>
</tr>
<tr>
<td>B-Involved Fair</td>
<td>Aug 27</td>
<td>3:00pm</td>
<td>Memorial Mall Square</td>
</tr>
</tbody>
</table>

Combining Operations

**Algebra =**
1. Basis arguments +
2. Ways of constructing expressions.

For relational algebra:
1. Arguments = variables standing for relations + finite, constant relations.
2. Expressions constructed by applying one of the operators + parentheses.
   - Query = expression of relational algebra.
Operator Precedence

The normal way to group operators is:

1. Unary operators $\sigma$, $\pi$, and $\rho$ have highest precedence.
2. Next highest are the “multiplicative” operators, $\Join$, $\Join^c$, and $\times$.
3. Lowest are the “additive” operators, $\cup$, $\cap$, and $-$.
   - But there is no universal agreement, so we always put parentheses \textit{around} the argument of a unary operator, and it is a good idea to group all binary operators with parentheses \textit{enclosing} their arguments.

Example
Group $R \cup \sigma S \Join T$ as $R \cup (\sigma(S) \Join T)$.

Each Expression Needs a Schema

- If $\cup$, $\cap$, $-$ applied, schemas are the same, so use this schema.
- Projection: use the attributes listed in the projection.
- Selection: no change in schema.
- Product $R \times S$: use attributes of $R$ and $S$.
  - But if they share an attribute $A$, prefix it with the relation name, as $R.A$, $S.A$.
- Theta-join: same as product.
- Natural join: use attributes from each relation; common attributes are merged anyway.
- Renaming: whatever it says.
Formal Definition

A basic expression in the relational algebra consists of either one of the following:
- A relation in the database
- A constant relation

Let $E_1$ and $E_2$ be relational-algebra expressions; the following are all relational-algebra expressions:
- $E_1 \cup E_2$
- $E_1 - E_2$
- $E_1 \times E_2$
- $\sigma_P(E_1)$, $P$ is a predicate on attributes in $E_1$
- $\Pi_S(E_1)$, $S$ is a list consisting of some of the attributes in $E_1$
- $\rho_x(E_1)$, $x$ is the new name for the result of $E_1$

Bag Semantics

A relation (in SQL, at least) is really a bag or multiset.
- It may contain the same tuple more than once, although there is no specified order (unlike a list).
- Example: \{1,2,1,3\} is a bag and not a set.
- Select, project, and join work for bags as well as sets.
  - Just work on a tuple-by-tuple basis, and don’t eliminate duplicates.
- SQL by default uses bag semantics!
  - Except Union, Intersection, Difference
Bag Union
Sum the times an element appears in the two bags.
• Example: \( \{1,2,1\} \cup \{1,2,3,3\} = \{1,1,1,2,2,3,3\} \).

Bag Intersection
Take the minimum of the number of occurrences in each bag.
• Example: \( \{1,2,1\} \cap \{1,2,3,3\} = \{1,2\} \).

Bag Difference
Proper-subtract the number of occurrences in the two bags.
• Example: \( \{1,2,1\} - \{1,2,3,3\} = \{1\} \).

Laws for Bags Differ From Laws for Sets
• Some familiar laws continue to hold for bags.
  – Examples: union and intersection are still commutative and associative.
• But other laws that hold for sets do not hold for bags.

Example
\( R \cap (S \cup T) \equiv (R \cap S) \cup (R \cap T) \) holds for sets.
• Let \( R, S, \) and \( T \) each be the bag \( \{1\} \).
• Left side: \( S \cup T = \{1,1\} \); \( R \cap (S \cup T) = \{1\} \).
• Right side: \( R \cap S = R \cap T = \{1\} \);
  \( (R \cap S) \cup (R \cap T) = \{1\} \cup \{1\} = \{1,1\} \neq \{1\} \).
Extended (“Nonclassical”) Relational Algebra

Adds features needed for SQL, bags.

1. Duplicate-elimination operator $\delta$.
2. Extended projection.
3. Sorting operator $\tau$.
4. Grouping-and-aggregation operator $\gamma$.
5. Outerjoin operator $\bowtie$.

Duplicates

- In relations with duplicates, SQL can define how many copies of tuples appear in the result.
- Multiset versions of some of the relational algebra operators – given multiset relations $r_1$:
  1. $\sigma_\theta(r_1)$: If there are $c_1$ copies of tuple $t_1$ in $r_1$, and $t_1$ satisfies selections $\sigma_\theta$, then there are $c_1$ copies of $t_1$ in $\sigma_\theta(r_1)$.
  2. $\Pi_A(r)$: For each copy of tuple $t_i$ in $r_1$, there is a copy of tuple $\Pi_A(t_i)$ in $\Pi_A(r_1)$ where $\Pi_A(t_i)$ denotes the projection of the single tuple $t_i$. 
Duplicate Elimination

\( \delta(R) = \) relation with one copy of each tuple that appears one or more times in \( R \).

**Example**

\( R = \)

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

\( \delta(R) = \)

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

---

**Duplicates (Cont.)**

- Example: Suppose multiset relations \( r_1 (A, B) \) and \( r_2 (C) \) are as follows:
  \[ r_1 = \{(1, a), (2, a)\} \quad r_2 = \{(2), (3), (3)\} \]
- Then \( \Pi_B(r_1) \) would be \( \{(a), (a)\} \), while \( \Pi_B(r_1) \times r_2 \) would be
  \[ \{(a,2), (a,2), (a,3), (a,3), (a,3), (a,3)\} \]
- SQL duplicate semantics:
  
  ```sql
  select A_1, A_2, ..., A_n
  from r_1, r_2, ..., r_m
  where P
  ```

  is equivalent to the multiset version of the expression:

  \[
  \Pi_{A_1,A_2,\ldots,A_n} (\sigma_P (r_1 \times r_2 \times \ldots \times r_m ))
  \]