Algebraic Data Models

Algebra

An **algebra** is a mathematical structure that consists of:

1. **A Set (A):** The domain or universe of elements.
2. **A Signature (F):** A set of operations defined on the set A.
3. **Axioms (Optional):** Properties that the operations must satisfy.

Formally, an algebra is a tuple:

$$\mathcal{A} = \langle A, \{f_i\}_{i \in I} \rangle$$

Where:

- A is a set (the carrier set).
- $\{f_i : A^n \to A\}_{i \in I}$ is a family of operations defined on A, where f_i is an n-ary operation ($n \geq 0$).

A Simple Example

The algebra of natural numbers under addition and multiplication:

$$\mathcal{A} = \langle \mathbb{N}, +, \times \rangle$$

- $A = \mathbb{N}$ (the set of natural numbers).
- $f_1 = +$ (binary addition: $\mathbb{N} \times \mathbb{N} \to \mathbb{N}$).
- $f_2 = \times$ (binary multiplication: $\mathbb{N} \times \mathbb{N} \to \mathbb{N}$).

Set Algebra

Set algebra is a structure consisting of a carrier set (the power set of a given universe) and operations like union, intersection, and complement.

Formal Representation

$$\mathcal{A} = \langle \mathcal{P}(U), \cup, \cap, \setminus, \emptyset, U \rangle$$

- **Carrier Set ($\mathcal{P}(U)$)**: The set of all subsets of a universe U, i.e., the power set of U.
- **Operations ($\cup, \cap, \setminus$)**:
  - **Union ($A \cup B$)**: The set of all elements in A or B.
  - **Intersection ($A \cap B$)**: The set of all elements in both A and B.
  - **Set Difference ($A \setminus B$)**: The set of elements in A but not in B.
  - **Complement ($\overline{A}$)**: The set of elements in $U \setminus A$.
- **Distinguished Elements ($\emptyset, U$)**:
  - **Empty Set ($\emptyset$)**: The set containing no elements.
  - **Universal Set (U)**: The set containing all elements of the universe.

Is this algebra minimal?

1. **Associativity:**
   - $(A \cup B) \cup C = A \cup (B \cup C)$
   - $(A \cap B) \cap C = A \cap (B \cap C)$
2. **Commutativity:**
   - $A \cup B = B \cup A$
   - $A \cap B = B \cap A$
3. **Distributivity:**
   - $A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$
   - $A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$
4. **Identity:**
   - $A \cup \emptyset = A$
   - $A \cap U = A$
5. **Complement:**
   - $A \cup \overline{A} = U$
   - $A \cap \overline{A} = \emptyset$

Boolean Algebra

Boolean algebra is a mathematical structure that models logical operations. It is defined over a set of truth values ($\{0, 1\}$) with operations like AND, OR, and NOT.

Formal Representation

$$\mathcal{B} = \langle B, \vee, \wedge, \neg, 0, 1 \rangle$$

- **Carrier Set (B):** The set of Boolean values: $B = \{0, 1\}$.
- **Operations ($\vee, \wedge, \neg$):**
  - **OR ($a \vee b$):** Returns 1 if $a = 1$ or $b = 1$.
  - **AND ($a \wedge b$):** Returns 1 if both $a = 1$ and $b = 1$.
  - **NOT ($\neg a$):** Returns 1 if $a = 0$, and 0 if $a = 1$.
- **Distinguished Elements ($0, 1$):**
  - 0: Represents false.
  - 1: Represents true.

Boolean algebra satisfies:

1. **Commutativity:**
   - $a \vee b = b \vee a$
   - $a \wedge b = b \wedge a$
2. **Associativity:**
   - $(a \vee b) \vee c = a \vee (b \vee c)$
   - $(a \wedge b) \wedge c = a \wedge (b \wedge c)$
3. **Distributivity:**
   - $a \wedge (b \vee c) = (a \wedge b) \vee (a \wedge c)$
   - $a \vee (b \wedge c) = (a \vee b) \wedge (a \vee c)$
4. **Identity:**
   - $a \vee 0 = a$
   - $a \wedge 1 = a$
5. **Complement:**
   - $a \vee \neg a = 1$
   - $a \wedge \neg a = 0$

An Algebra of Time Points

Carrier Set (P)

$$P = \{\text{all possible time points (e.g., timestamps)}\}$$

Example: $P = \{\text{January 1, 2025, 12:00 PM, etc.}\}$

Operations on Time Points (f_p)

1. **Addition ($+$):**
   - $f_{\text{add}}(p, d) = p'$: Add a duration d to a time point p.
   - Example: $\text{Jan 1, 2025} + 3 \text{ days} = \text{Jan 4, 2025}$.
2. **Subtraction ($-$):**
   - $f_{\text{sub}}(p, d) = p'$: Subtract a duration d from a time point p.
   - Example: $\text{Jan 4, 2025} - 3 \text{ days} = \text{Jan 1, 2025}$.

Relations on Time Points (R_p)

1. **Before ($<$):**
   - $R_{\text{before}}(p_1, p_2)$: True if p_1 occurs before p_2.
   - Example: $\text{Jan 1, 2025} < \text{Feb 1, 2025}$.
2. **After ($>$):**
   - $R_{\text{after}}(p_1, p_2)$: True if p_1 occurs after p_2.
   - Example: $\text{Feb 1, 2025} > \text{Jan 1, 2025}$.
3. **Equals ($=$):**
   - $R_{\text{equals}}(p_1, p_2)$: True if $p_1 = p_2$.

An Algebra of Time Intervals

Carrier Set (I)

$$I = \{[p_1, p_2] \mid p_1, p_2 \in P, p_1 \leq p_2\}$$

Example: $I = \{[\text{Jan 1, 2025}, \text{Jan 10, 2025}], [\text{Feb 1, 2025}, \text{Feb 20, 2025}]\}$

Operations on Time Intervals (f_i)

1. **Interval Addition ($+$):**
   - $f_{\text{add}}([p_1, p_2], d) = [p_1 + d, p_2 + d]$: Add a duration d to both endpoints of the interval.
   - Example: $[\text{Jan 1, 2025}, \text{Jan 10, 2025}] + 5 \text{ days} = [\text{Jan 6, 2025}, \text{Jan 15, 2025}]$.
2. **Interval Intersection ($\cap$):**
   - $f_{\text{intersect}}([p_1, p_2], [q_1, q_2]) = [\max(p_1, q_1), \min(p_2, q_2)]$: The intersection of two intervals.
   - Example: $[\text{Jan 1, 2025}, \text{Jan 10, 2025}] \cap [\text{Jan 5, 2025}, \text{Jan 15, 2025}] = [\text{Jan 5, 2025}, \text{Jan 10, 2025}]$.
3. **Interval Union ($\cup$):**
   - $f_{\text{union}}([p_1, p_2], [q_1, q_2])$: Combines two intervals if they overlap or are adjacent.

Properties

1. **Closure:**
   - Time points are closed under addition and subtraction.
   - Intervals are closed under intersection and addition of durations.
2. **Associativity:**
   - $(A + B) + C = A + (B + C)$ for addition of durations.
3. **Order Relations:**
   - Time points and intervals respect a total order.

Relations on Time Intervals (R_i)

1. **Overlaps:**
   - $R_{\text{overlaps}}([p_1, p_2], [q_1, q_2])$: True if the intervals overlap.
   - Example: $[\text{Jan 1, 2025}, \text{Jan 10, 2025}]$ overlaps $[\text{Jan 5, 2025}, \text{Jan 15, 2025}]$.
2. **Before:**
   - $R_{\text{before}}([p_1, p_2], [q_1, q_2])$: True if $p_2 < q_1$.
3. **Contains:**
   - $R_{\text{contains}}([p_1, p_2], [q_1, q_2])$: True if $p_1 \leq q_1$ and $p_2 \geq q_2$.

Tuples

A **tuple** is a finite ordered collection of elements, where each element can be of a different type.

A tuple is defined as:

$$t = (t_1, t_2, \ldots, t_n)$$

Where:

- n: The **arity** or length of the tuple (number of elements).
- t_i: The i-th element of the tuple, which belongs to a specified domain D_i ($t_i \in D_i$).

The tuple t is ordered, meaning:

$$(t_1, t_2, \ldots, t_n) \neq (t_2, t_1, \ldots, t_n) \text{ unless } t_1 = t_2.$$

Key Properties

1. **Ordered**: The order of elements in a tuple matters.
   - Example: $(1, 2) \neq (2, 1)$.
2. **Finite Length**: A tuple has a fixed, finite number of elements.
   - Example: $(1, 2, 3)$ is a 3-tuple (arity = 3).
3. **Heterogeneous Types**: Elements in a tuple can belong to different domains.
   - Example: $(\text{"Alice"}, 30, 50000.0)$, where:
     - $\text{"Alice"} \in \text{Strings}$,
     - $30 \in \mathbb{N}$,
     - $50000.0 \in \mathbb{R}$.

Operations for Atomic Tuples

Operation	Definition	Example
Accessing Elements	Retrieves the i-th element of a tuple t.	$t = (1, 2, 3), t[2] = 2$
Concatenation	Combines two tuples into one longer tuple.	$(1, 2) + (3, 4) = (1, 2, 3, 4)$
Projection	Extracts specific attributes (indices) from a tuple.	$\pi_{1,3}((1, 2, 3)) = (1, 3)$
Slicing	Extracts a contiguous subset of elements.	$\text{slice}((1, 2, 3, 4, 5), 2, 4) = (2, 3, 4)$
Filtering	Retains elements that satisfy a predicate.	$\text{filter}((1, 2, 3, 4, 5), p(x) = x\%2 = 0) = (2, 4)$
Element-Wise Transformation	Applies a function f to each element in the tuple.	$\text{map}((1, 2, 3), f(x) = x^2) = (1, 4, 9)$
Tuple Equality	Checks if two tuples are equal (element-wise comparison).	$(1, 2, 3) = (1, 2, 3) \rightarrow \text{true}$, $(1, 2, 3) = (3, 2, 1) \rightarrow \text{false}$
Attribute-Wise Comparison	Compares tuples element by element (e.g., $<$, $>$).	$(1, 2, 3) < (1, 3, 4) \rightarrow \text{true}$
Lexicographical Comparison	Compares tuples based on lexicographical order.	$(1, 2, 3) < (1, 2, 4) \rightarrow \text{true}$

Many-sorted Algebra

A **many-sorted algebra** generalizes the concept of a single-sorted algebra by allowing **multiple carrier sets** (sorts), with operations that can involve elements from different sorts.

Formally, a many-sorted algebra is a tuple:

$$\mathcal{A} = \langle S, \{A_s\}_{s \in S}, \{f_i\}_{i \in I} \rangle$$

Where:

1. S: A finite set of **sorts** (types). Each sort represents a distinct type of element.
2. $\{A_s\}_{s \in S}$: A family of **carrier sets**, one for each sort $s \in S$. These sets represent the domain of elements for each sort.
3. $\{f_i\}_{i \in I}$: A family of **operations**, where each operation f_i has an arity $(s_1, s_2, \ldots, s_n; t)$, meaning:
   - The operation takes n inputs, with types $s_1, s_2, \ldots, s_n \in S$.
   - The output is of type $t \in S$.

Example: A 2-sorted Algebra

- The Sorts and Carrier Sets
  - Sorts:
    - String (A_{String}): Elements are strings.
    - Number (A_{Number}): Elements are numbers.
  - Carrier Sets:
    - A_{String} ={"apple","banana","cherry"}
    - A_{Number} ={1,2,3,4,5,6}

A many-sorted algebra must be specified in terms of:

- **Sorts:** Different types of elements (e.g., Strings, Numbers).
- **Carrier Sets:** Sets associated with each sort.
- **Operations:** Functions defined within or across sorts.
- **Relations:** Logical comparisons between elements.

Operations on This Algebra – 1

Operations Within Strings

1. **Concatenation (f_{concat}):**
   - Combines two strings.
   - Type: $f_{\text{concat}} : A_{\text{String}} \times A_{\text{String}} \to A_{\text{String}}$
   - Example: $f_{\text{concat}}(\text{"apple"}, \text{"banana"}) = \text{"applebanana"}$
2. **String Length (f_{length}):**
   - Returns the length of a string as a number.
   - Type: $f_{\text{length}} : A_{\text{String}} \to A_{\text{Number}}$
   - Example: $f_{\text{length}}(\text{"apple"}) = 5$

Operations Within Numbers

1. **Addition (f_{add}):**
   - Adds two numbers.
   - Type: $f_{\text{add}} : A_{\text{Number}} \times A_{\text{Number}} \to A_{\text{Number}}$
   - Example: $f_{\text{add}}(2, 3) = 5$
2. **Multiplication (f_{multiply}):**
   - Multiplies two numbers.
   - Type: $f_{\text{multiply}} : A_{\text{Number}} \times A_{\text{Number}} \to A_{\text{Number}}$
   - Example: $f_{\text{multiply}}(2, 4) = 8$

Operations on this Algebra – 2

- Cross-sort operations

1. Repeat String by Number (f_{repeat})

- **Type:**

$$f_{\text{repeat}} : A_{\text{String}} \times A_{\text{Number}} \to A_{\text{String}}$$

- **Description:** Repeats a string n times, where n is a number.
- **Example:**

$$f_{\text{repeat}}(\text{"apple"}, 3) = \text{"appleappleapple"}$$

---

2. String Length (f_{length})

- **Type:**

$$f_{\text{length}} : A_{\text{String}} \to A_{\text{Number}}$$

- **Description:** The operation f_{length} maps a string $s \in A_{\text{String}}$ to its length as a number $n \in A_{\text{Number}}$. This is a **cross-sort operation** because it maps elements from the sort of strings (A_{String}) to the sort of numbers (A_{Number}).
- **Example:** If $A_{\text{String}} = \{\text{"apple"}, \text{"banana"}, \text{"cherry"}\}$ and $A_{\text{Number}} = \{1, 2, 3, 4, 5, 6\}$, then:

$$f_{\text{length}}(\text{"apple"}) = 5, \quad f_{\text{length}}(\text{"banana"}) = 6, \quad f_{\text{length}}(\text{"cherry"}) = 6$$

Relational Algebra

- Relational Algebra is a many-sorted algebra
- Sorts (S):We maintain the following sorts:
  - Domains (D): Sets of atomic values for attributes.
  - Attributes (A): Names of attributes, associated with specific domains.
  - Tuples (T): Ordered collections of values, one for each attribute, constrained by attribute-domain mappings.
  - Relations (R): Sets of tuples constrained by a schema (attribute names and domains).

Relational Algebra

1. **Carrier Set for Domains (A_{Domains}):**
   - Domains are sets of atomic values.
   - Example: $\mathrm{D_A} = \{1, 2, 3\}, \mathrm{D_B} = \{\text{"apple"}, \text{"banana"}\}$.
2. **Carrier Set for Attributes ($A_{\text{Attributes}}$):**
   - Attributes are pairs of attribute names and their associated domains:
     $$A_{\text{Attributes}} = \{(\mathrm{A}, \mathrm{D_A}), (\mathrm{B}, \mathrm{D_B})\}$$
   - Dependency: Each attribute is defined by a name and a domain.

3. **Carrier Set for Tuples (A_{Tuples}):**
   - Tuples are mappings of attributes to values, constrained by their associated domains:
     $$A_{\text{Tuples}} = \{t : A_{\text{Attributes}} \to A_{\text{Values}} \mid t(A) \in \mathrm{D}_A, \forall A \in A_{\text{Attributes}}\}$$
   - Example:
     $$t_1 = \{(\mathrm{A}, 1), (\mathrm{B}, \text{"apple"})\}, \quad t_2 = \{(\mathrm{A}, 2), (\mathrm{B}, \text{"banana"})\}$$
4. **Carrier Set for Relations ($A_{\text{Relations}}$):**
   - Relations are sets of tuples constrained by a schema (attributes and their domains):
     $$A_{\text{Relations}} = \{R \subseteq A_{\text{Tuples}} \mid \text{Schema}(R) = \text{Attributes of } R\}$$
   - Example: If the schema is $\{\mathrm{A}, \mathrm{B}\}$, then:
     $$R = \{\{(\mathrm{A}, 1), (\mathrm{B}, \text{"apple"})\}, \{(\mathrm{A}, 2), (\mathrm{B}, \text{"banana"})\}\}$$

Core Relational Algebra

- Union, intersection, and difference:
  - Usual set operations, but both operands must have the same relation schema.
- Selection: picking certain rows based on predicates.
- Projection: picking certain columns
  - Extended projection: reporting results of computation on columns
- Products and joins: compositions of relations
- Renaming of relations and attributes.

Selection

- R1 := σ_C(R2)
  - C is a condition (as in “if” statements) that refers to attributes of R2.
- R1 is all those tuples of R2 that satisfy C.

Example: Selection

Relation Sells:

bar	beer	price
Joe's	Bud	2.50
Joe's	Miller	2.75
Sue's	Bud	2.50
Sue's	Miller	3.00

JoeMenu := $\sigma_{bar="Joe's"}(Sells)$:

bar	beer	price
Joe's	Bud	2.50
Joe's	Miller	2.75

Projection

- R1 := π_L (R2)
  - L is a *list of attributes* from the schema of R2
- R1 is constructed by looking at each tuple of R2, extracting the attributes on list L, in the order specified, and creating from those components a tuple for R1
- Eliminate duplicate tuples, if any

Example: Projection

Relation Sells:

bar	beer	price
Joe's	Bud	2.50
Joe's	Miller	2.75
Sue's	Bud	2.50
Sue's	Miller	3.00

Prices := $\pi_{beer,price}$(Sells):

beer	price
Bud	2.50
Miller	2.75
Miller	3.00

Extended Projection

- Using the same π_L operator, we allow the list L to contain arbitrary expressions involving attributes:
    1. Arithmetic on attributes, e.g., A+B → C
    2. Concatenation of attributes
    3. Duplicate occurrences of the same attribute

Example: Extended Projection

R =

A	B
1	2
3	4

$\pi_{A+B->C,A,A}(R)$ =

C	A1	A2
3	1	1
7	3	3

(Cartesian) Product

- R3 := R1 $\times$ R2
  - Pair each tuple t1 of R1 with each tuple t2 of R2
  - Concatenation t1 and t2 is a tuple of R3
  - Schema of R3 is the attributes of R1 and then R2, in order
  - But beware attribute A of the same name in R1 and R2: use R1.A and R2.A.

Example: Product

R1(

A,	B
1	2
3	4

)

R2(

B,	C
5	6
7	8
9	10

)

R3(

A,	R1.B,	R2.B,	C
1	2	5	6
1	2	7	8
1	2	9	10
3	4	5	6
3	4	7	8
3	4	9	10

)

Theta Join

- R3 := R1 $\bowtie_C$ R2
  - Take the product R1 $\times$ R2
  - Then apply σ_C to the result
- As for σ, C can be any Boolean-valued condition
- Historic versions of this operator allowed only A θ B, where θ is =, <, etc.; hence the name “theta-join.”

Example: Theta Join

Sells(

bar,	beer,	price
Joe's	Bud	2.50
Joe's	Miller	2.75
Sue's	Bud	2.50
Sue's	Coors	3.00

)

Bars(

name,	addr
Joe's	Maple St.
Sue's	River Rd.

)

BarInfo := Sells $\bowtie_{\text{Sells.bar = Bars.name}}$ Bars

BarInfo(

bar,	beer,	price,	name,	addr
Joe's	Bud	2.50	Joe's	Maple St.
Joe's	Miller	2.75	Joe's	Maple St.
Sue's	Bud	2.50	Sue's	River Rd.
Sue's	Coors	3.00	Sue's	River Rd.

)

Natural Join

- A useful join variant (natural join) connects two relations by:
  - Equating attributes of the same name, and
  - Projecting out one copy of each pair of equated attributes
- Denoted R3 := R1 ⋈ R2

Example: Natural Join

Sells(

bar,	beer,	price
Joe's	Bud	2.50
Joe's	Miller	2.75
Sue's	Bud	2.50
Sue's	Coors	3.00

)

Bars(

bar,	addr
Joe's	Maple St.
Sue's	River Rd.

)

BarInfo := Sells ⋈ Bars

Note: Bars.name has become Bars.bar to make the natural join "work."

BarInfo(

bar,	beer,	price,	addr
Joe's	Bud	2.50	Maple St.
Joe's	Milller	2.75	Maple St.
Sue's	Bud	2.50	River Rd.
Sue's	Coors	3.00	River Rd.

)

Renaming

- The ρ operator gives a new schema to a relation.
- R1 := $\rho_{R1(A1,\ldots,An)}$(R2) makes R1 a relation with attributes A1,…,An and the same tuples as R2.
- Simplified notation: R1(A1,…,An) := R2

Bars(

name,	addr
Joe's	Maple St.
Sue's	River Rd.

)

R(bar, addr) := Bars

R(

bar,	addr
Joe's	Maple St.
Sue's	River Rd.

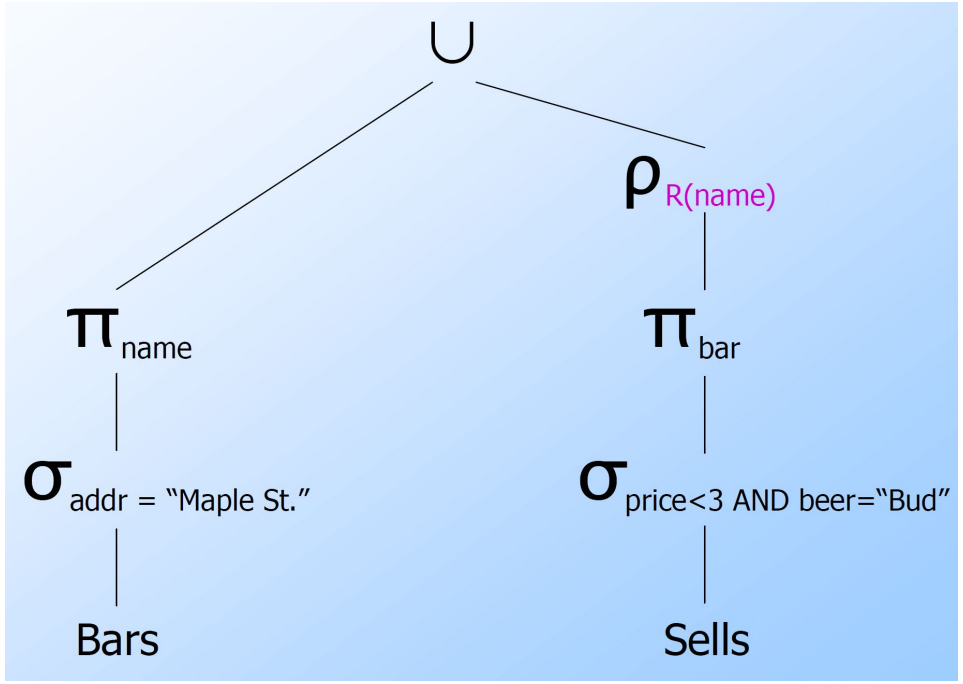
)

Building Complex Expressions

- Combine operators with parentheses and precedence rules
- the theta-join R3 := R1 $\bowtie_C$ R2 can be written: R3 := σ_C (R1 $\times$ R2)
- Precedence of relational operators:
    1. [σ, π, ρ] (highest)
    2. [$\times$, $\bowtie$]
    3. $\cap$
    4. [$\cup$, —]

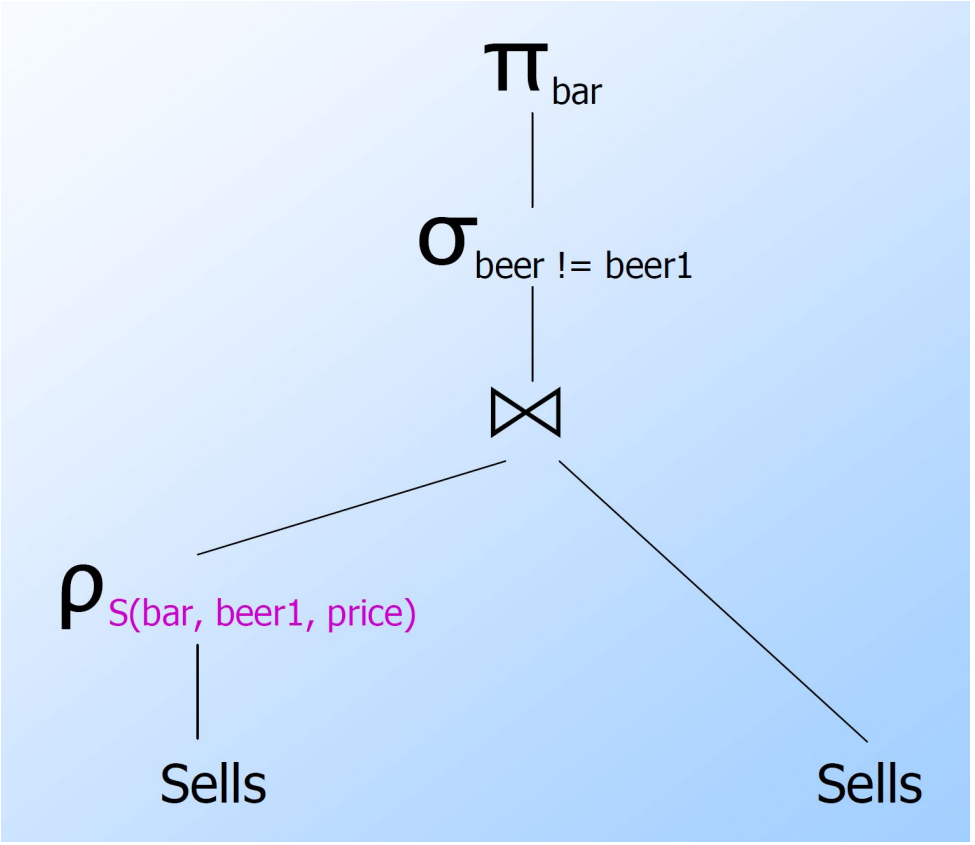
Query Tree

- Using the relations Bars(name, addr) and Sells(bar, beer, price), find the names of all the bars that are either on Maple St. or sell Bud for less than $3



Self-Join

- Using Sells(bar, beer, price), find the bars that sell two different beers at the same price
- Strategy: by renaming, define a copy of Sells, called S(bar, beer1, price)
- The natural join of Sells and S consists of quadruples (bar, beer, beer1, price) such that the bar sells both beers at this price



Schemas for Results

- Union, intersection, and difference: the schemas of the two operands must be the same, so use that schema for the result
- Selection: schema of the result is the same as the schema of the operand
- Projection: list of attributes tells us the schema

Schemas for Results – (2)

- Product: schema is the attributes of both relations.
  - Use R.A, etc., to distinguish two attributes named A
- Theta-join: same as product
- Natural join: union of the attributes of the two relations.
- Renaming: the operator tells the schema

Relational Algebra on Bags

- A bag (or multiset) is like a set, but an element may appear more than once
- Example: {1,2,1,3} is a bag
- Example: {1,2,3} is also a bag that happens to be a set

Operations on Bags

- Selection applies to each tuple, so its effect on bags is like its effect on sets
- Projection also applies to each tuple, but as a bag operator, we do not eliminate duplicates.
- Products and joins are done on each pair of tuples, so duplicates in bags have no effect on how we operate

Examples

R(

A,	B
1	2
5	6
1	2

)

$\sigma_{A+B<5}(R)$ =

A	B
1	2
1	2

R(

A,	B
1	2
5	6
1	2

)

$\pi_{A}(R)$ =

A
1
5
1

R(

A,	B
1	2
5	6
1	2

)

S(

B,	C
3	4
7	8

)

R X S =

A	R.B	S.B	C
1	2	3	4
1	2	7	8
5	6	3	4
5	6	7	8
1	2	3	4
1	2	7	8

R(

A,	B
1	2
5	6
1	2

)

S(

B,	C
3	4
7	8

)

R $\bowtie_{R.B<S.B}$ S =

A	R.B	S.B	C
1	2	3	4
1	2	7	8
5	6	7	8
1	2	3	4
1	2	7	8

Bag Operations

- Bag Union
  - An element appears in the union of two bags the sum of the number of times it appears in each bag.
  - Example: $\{1,2,1\} \cup \{1,1,2,3,1\} = \{1,1,1,1,1,2,2,3\}$
- Bag Intersection
  - An element appears in the intersection of two bags the minimum of the number of times it appears in either
  - Example: $\{1,2,1,1\} \cap \{1,2,1,3\} = \{1,1,2\}$
- Bag Difference
  - An element appears in the difference A – B of bags as many times as it appears in A, minus the number of times it appears in B. But never less than 0 times.
  - Example: $\{1,2,1,1\} - \{1,2,3\} = \{1,1\}$

Bag Laws != Set Laws

- Some, but not all algebraic laws that hold for sets also hold for bags
  - Example: the commutative law for union ($R \cup S = S \cup R$) does hold for bags.
  - Since addition is commutative, adding the number of times x appears in R and S doesn't depend on the order of R and S
- Set union is idempotent, meaning that $S \cup S = S$.
- However, for bags, if x appears n times in S, then it appears 2n times in $S \cup S$.

A tweet as a data item – JSON


```
▼ object {26}
    created_at : Mon Mar 13 21:33:03 +0000 2017
    id    : 841401747822141400
    id_str : 841401747822141440
    text : RT @ashishtikoo31: Have you see any single Media house discussing how Badals
           defeated Captain Amrinder Singh & Bhagwant Mann? Akali Vote sh…
    source : Twitter for Android
    truncated : ☐ false
    in_reply_to_status_id : null
    in_reply_to_status_id_str : null
    in_reply_to_user_id_str : null
    in_reply_to_screen_name : null
  ▼ user {37}
        id    : 782356811273740300
        id_str : 782356811273740288
        name : Kamakshi sharma
        screen_name : Kamaksh17538281
        location : New Delhi, India
        url   : null
        description : Just believe in almighty he will take you forward
        protected : ☐ false
        verified : ☐ false
        followers_count : 6
        friends_count : 120
        listed_count : 0
        favourites__count : 328
        statuses_count : 617
        created_at : Sat Oct 01 23:09:34 +0000 2016
  ▼ entities {4}
    ▶ urls [0]
    ▶ symbols [0]
    ▶ hashtags [0]
    ▼ user_mentions [1]
      ▼ 0 {5}
        ▶ indices [2]
          screen_name : ashishtikoo31
          id_str : 159540538
          name : Ashish Tikoo Rajput
          id    : 159540538
    favorited : ☐ false
    retweeted : ☐ false
    filter_level : low
    lang : en
  ▼ retweeted_status {24}
        in_reply_to_status_id_str : null
        in_reply_to_status_id : null
        coordinates : null
        created_at : Mon Mar 13 17:08:04 +0000 2017
        truncated : ☐ false
        in_reply_to_user_id_str : null
        source : Twitter for iPhone
        retweet_count : 11
        retweeted : ☐ false
        geo   : null
```

Serial and Tree Forms of Semi-structured Data