CS 317/387

Towards SQL -
Relational Algebra

A Relation is a Table

Attributes (column headers)

Tuples (rows)

name	manf
Winterbrew	Pete's
Bud Lite	Anheuser-Busch

Beers

Schemas

- *Relation schema* = relation name and attribute list.
  - Optionally: types of attributes.
  - Example: Beers(name, manf) or Beers(name: string, manf: string)
- *Database* = collection of relations.
- *Database schema* = set of all relation schemas in the database.

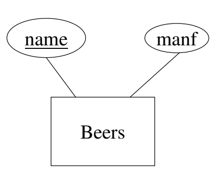
Why Relations?

- Very simple model.
- *Often* matches how we think about data.
- Abstract model that underlies SQL, the most important database language today.

From E/R Diagrams to Relations

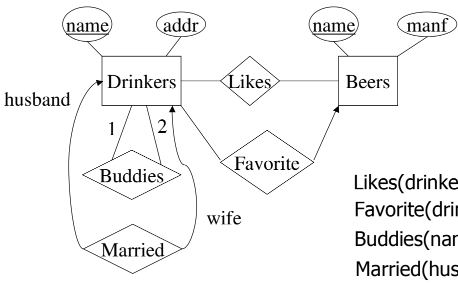
- Entity set -> relation.
  - Attributes -> attributes.
- Relationships -> relations whose attributes are only:
  - The keys of the connected entity sets.
  - Attributes of the relationship itself.

Entity Set -> Relation



Relation: Beers(name, manf)

Relationship -> Relation



Likes(drinker, beer)
Favorite(drinker, beer)
Buddies(name1, name2)
Married(husband, wife)

What is an "Algebra"

Mathematical system consisting of:

- *Operands* --- variables or values from which new values can be constructed.
- *Operators* --- symbols denoting procedures that construct new values from given values.

More on Algebras

- **Arithmetic**: operands are variables and constants, operators are +,−,×,÷, /, etc.
- **Set algebra:** operands are sets and operators are Union, Intersection Set Difference etc.
- An algebra allows us to *construct expressions* by combining operands and expression using operators.
  - a2 + 2 × a × b + b2, (a + b)2.
  - R – (R-S) , R ∩ S etc.

What is Relational Algebra?

- An algebra whose operands are relations or variables that represent relations.
- Operators are designed to do the most common things that we need to do with relations in a database.
  - The result is an algebra that can be used as a basis for a *query language* for relations.
- Relational algebra is a notation for specifying queries about the contents of relations.
- Notation of relational algebra eases the task of reasoning about queries.
  - Operations in relational algebra have counterparts in SQL.

Roadmap

- There is a core relational algebra that has traditionally been thought of as *the* relational algebra.
- But there are several other operators we shall add to the core in order to model better the language SQL --- the principal language used in relational database systems.

Core Relational Algebra

- Union, intersection, and difference.
  - Usual set operations, but require both operands have the same relation schema.
- Selection: picking certain rows.
- Projection: picking certain columns.
  - Both Selection and Projection remove certain parts of a relation!
- Products and joins: compositions of relations.
- Renaming of relations and attributes.

Union

- The union of two relations R and S is the set of tuples that are in R or in S or in both.
- R and S must have identical sets of attributes and the types of the attributes must be the same.
- The attributes of R must occur in the same order as the attributes in S.
- RA $R \cup S$

```
(SELECT * FROM R) UNION (SELECT * FROM S);
```

Intersection

- The intersection of two relations R and S is the set of tuples that are in both R and S.
- Same conditions hold on R and S as for the union operator.
- RA $R \cap S$

```
(SELECT * FROM R) INTERSECT (SELECT * FROM
  S);
```

Difference

- The difference of two relations R and S is the set of tuples that are in R but not in S.
- Same conditions hold on R and S as for the union operator.
- RA: $R - S$
- `(SELECT * FROM R) EXCEPT (SELECT * FROM S);`
- $R - (R-S)$ = ??
- $R \cap S$

Selection

- **R1 := SELECT$_C$ (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*.
- Basis of:

```
SELECT *  FROM R WHERE C;
```

More on Selection

- Syntax of C: similar to conditionals in programming languages.
- Values compared are constants and attributes of the relations mentioned in the FROM clause.
- We may apply usual arithmetic operators to numeric values before comparing them.
  - SQL Compare values using =, <, >, <=, >=.

Example

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 := SELECT$_{bar=\text{"Joe's"}}$(Sells):

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

Projection

- **R1 := PROJ$_L$ (R2)**
- *L* is a list of attributes from the schema of R2.
  - Same as selecting select columns from a table.
- 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.

```
SELECT A1, A2, . . . , An  FROM R;
```

Example

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 := PROJ$_{beer,price}$(Sells):

beer	price
Bud	2.50
Miller	2.75
Miller	3.00

Product

- **R3 := R1 * R2**
- Pair each tuple t1 of R1 with each tuple t2 of R2.
- Concatenation ***t1t2*** 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*.

```
SELECT * FROM R1, R2
```

Example: R3 := R1 * R2

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 JOIN$_C$ R2**
- Set of tuples in the Cartesian product that satisfies some condition C
- Computing it
  - Take the product R1 * R2.
  - Then apply SELECT$_C$ to the result.
- *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."

```
SELECT * from R1, R2 WHERE C
```

Example

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 JOIN $_{\text{Sells.bar = Bars.name}}$ 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

- The *natural join* of two relations R and S is a set of pairs of tuples, one from R and one from S, that agree on whatever attributes are common to the schemas of R and S
- Connect two relations by:
  - Equating attributes of the same name, and
  - Projecting out one copy of each pair of equated attributes.
- The schema for the result contains the union of the attributes of R and S.
- Denoted **R3 := R1 JOIN R2**

Example

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.
Ben's	Central Av

)

BarInfo := Sells JOIN 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.

)

More on Natural Join

- A dangling tuple is one that fails to pair with any tuple in the other relation.
  - Ben's bar on Central Ave. in the previous example

```
R(A,B, C) and S(B, C,D).
SELECT R.A, R.B, R.C, S.D FROM R,S
    WHERE R.B = S.B AND R.C = S.C;
```

Renaming

- The RENAME operator gives a new schema to a relation.
- R1 := $RENAME_{R1(A1,\dots,An)}$(R2) makes R1 be a relation with attributes A1,…,A*n* and the same tuples as R2.
- Simplified notation: R1(A1,…,A*n*) := R2.

Example

Bars(

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

)

Pubs(bar, addr) := Bars

Pubs(

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

)

Building Complex Expressions

- Combine operators with parentheses and precedence rules.
- Three notations, just as in arithmetic:
  1. *Sequences of assignment* statements.
  2. Expressions with several operators.
  3. Expression trees.

Sequences of Assignments

- Create temporary relation names.
- Renaming can be implied by giving relations a list of attributes.
- Example: R3 := R1 $JOIN_C$ R2 can be written:

R4 := R1 * R2

R3 := $SELECT_C$ (R4)

Expressions in a Single Assignment

- Example: the theta-join R3 := R1 $JOIN_C$ R2 can be written: R3 := $SELECT_C$ (R1 * R2)
- Precedence of relational operators:
    1. [SELECT, PROJECT, RENAME] (highest).
    2. [PRODUCT, JOIN].
    3. INTERSECTION.
    4. [UNION, --]

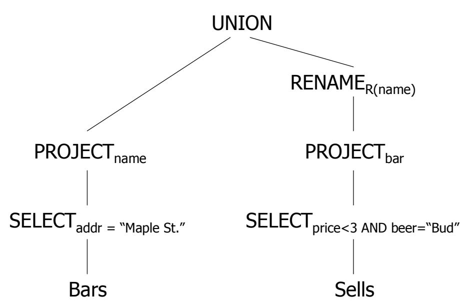
Expression Trees

- Leaves are operands --- either variables standing for relations or particular, constant relations.
- Interior nodes are operators, applied to their child or children.

Example

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

As a Tree:

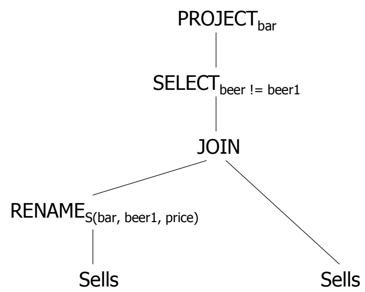


Example

- Using **`Sells(bar, beer, price)`**, find the bars that sell two different beers at the same price.

- **<u>Strategy</u>**: 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.

The Tree



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.

Why Bags?

- SQL, the most important query language for relational databases, is actually a bag language.
- Some operations, like projection, are much more efficient on bags than sets.

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.

Example: Bag Selection

R(

A,	B
1	2
5	6
1	2

)

SELECT$_{A+B<5}$ (R)

=

A	B
1	2
1	2

Example: Bag Projection

R(

A,	B
1	2
5	6
1	2

)

PROJECT$_{A}$ (R)

=

A
1
5
1

Example: Bag Product

R(

A,	B
1	2
5	6
1	2

)

S(

B,	C
3	4
7	8

)

R * 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

Example: Bag Theta-Join

R(

A,	B
1	2
5	6
1	2

)

S(

B,	C
3	4
7	8

)

R JOIN $_{R.B<S.B}$ 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 Union

- An element appears in the union of two bags the sum of the number of times it appears in each bag.

{1,2,1} ∪{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.

{1,2,1,1} ∩ {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.

{1,2,1,1} – {1,2,3} = {1,1}.

Beware: Bag Laws != Set Laws

- Some, but *not all* algebraic laws that hold for sets also hold for bags.
- Question: Does the commutative law for Union hold for bags?
- Answer: 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*.

More on sets Vs bags

- Set union is *idempotent*, meaning that $S \cup S = S$.
- Question: Is Bag union idempotent?
- Answer: If *x* appears *n* times in *S*, then it appears 2*n* times in $S \cup S$. Thus $S \cup S$!= *S* in general.

The Extended Algebra

- DELTA = eliminate duplicates from bags.
- TAU = sort tuples.
- *Extended projection* : arithmetic, duplication of columns.
- GAMMA = grouping and aggregation.
- *Outerjoin* : avoids "dangling tuples" = tuples that do not join with anything.

Duplicate Elimination

- R1 := DELTA(R2).
- R1 consists of one copy of each tuple that appears in R2 one or more times.

Example: Duplicate Elimination

R = (

A	B
1	2
3	4
1	2

)

DELTA(R) =

A	B
1	2
3	4

Sorting

- R1 := TAU_L (R2).
  - *L* is a list of some of the attributes of R2.
- R1 is the list of tuples of R2 sorted first on the value of the first attribute on *L*, then on the second attribute of *L*, and so on.
  - Break ties arbitrarily.

Example: Sorting

R = (

A	B
1	2
3	4
5	2

)

TAU_B (R) = [(5,2), (1,2), (3,4)]

Extended Projection

- Using the same $PROJ_L$ operator, we allow the list *L* to contain ***arbitrary expressions involving attributes***, for example:
  1. Arithmetic on attributes, e.g., *A+B*.
  2. Duplicate occurrences of the same attribute.

Example: Extended Projection

R = (

A	B
1	2
3	4

)

$PROJ_{A+B,A,A}$ (R) =

A+B	A1	A2
3	1	1
7	3	3

Aggregation Operators

- Aggregation operators are not operators of relational algebra.
- Rather, they apply to entire columns of a table and produce a single result.
- The most important examples: SUM, AVG, COUNT, MIN, and MAX.

Example: Aggregation

R = (

A	B
1	3
3	4
3	2

)

SUM(A) = 7
COUNT(A) = 3
MAX(B) = 4
AVG(B) = 3

Grouping Operator

- **R1 := GAMMA$_L$ (R2)**
- *L* is a list of elements that are either:
  1. Individual (*grouping*) attributes.
  2. AGG(*A*), where AGG is one of the aggregation operators and *A* is an attribute.

Applying GAMMA$_L$(R)

- Group *R* according to all the grouping attributes on list *L*.
  - That is: form one group for each distinct list of values for those attributes in *R*.
- Within each group, compute AGG(*A*) for each aggregation on list *L*.
- Result has one tuple for each group:
  1. The grouping attributes and
  2. Their group's aggregations.

Example: Grouping/Aggregation

R = (

A	B	C
1	2	3
4	5	6
1	2	5

)

GAMMA$_{A,B,AVG(C)}$ (R) = ??

First, group *R* by *A* and *B* :

A	B	C
1	2	3
1	2	5
4	5	6

Then, average *C* within groups:

A	B	AVG(C)
1	2	4
4	5	6

Outerjoin

- Suppose we join *R* $JOIN_C$ *S*.
- A tuple of *R* that has no tuple of *S* with which it joins is said to be *dangling*.
  - Similarly for a tuple of *S*.
- Outerjoin preserves dangling tuples by padding them with a special NULL symbol in the result.

Example: Outerjoin

R = (

A	B
1	2
4	5

)

S = (

B	C
2	3
6	7

)

(1,2) joins with (2,3), but the other two tuples are dangling.

R OUTERJOIN S =

A	B	C
1	2	3
4	5	NULL
NULL	6	7