

SQL Data Definition Language & Relational Algebra

Outline

- Database definition with SQL
 - SQL Data Definition Language
 - Static Constraints
 - (Dynamic constraints later -- require query language)
 - Foreign Key Constraints
 - Attribute/Tuple-based Checks
 - Assertions
- Relational Algebra
 - Relational Algebra on sets
 - Relational Algebra on bags

Database Schemas in SQL

- Structured Query Language
 - "sequel" if you worked for IBM in the 80s
 - "es-queue-el" for the rest of us
- SQL is primarily a query language, for getting information from a database.
- But SQL also includes a *data-definition* component for describing database schemas.

Creating (Declaring) a Relation

- Simplest form is:

```
CREATE TABLE <name> (  
    <list of elements>  
);
```

- To delete a relation:

```
DROP TABLE <name>;
```

Elements of Table Declarations

- Most basic element: an attribute and its type.
- The most common types are:
 - INT or INTEGER (synonyms).
 - REAL or FLOAT (synonyms).
 - CHAR(n) = fixed-length string of n characters.
 - VARCHAR(n) = variable-length string of up to n characters.
- **1NF**: Each attribute has a primitive type (not a composite type such as set, or relation)

Example: Create Table

```
CREATE TABLE Sells (  
    bar        CHAR(20),  
    beer       VARCHAR(20),  
    price      REAL  
);
```

SQL Values

- Integers and reals are represented as you would expect.
- Strings are too, except they require single quotes.
 - Two single quotes = real quote, e.g., ' Joe' ' s Bar' .
- Any value can be NULL.

Kinds of Constraints

- **Keys.**
- **Foreign-key**, or referential-integrity.
- **Value-based** constraints.
 - Constrain values of a particular attribute.
- **Tuple-based** constraints.
 - Relationship among components.
- **Assertions**: any SQL boolean expression.
 - We will cover this after we have covered SQL query language

Declaring Keys

- An attribute or list of attributes may be declared PRIMARY KEY or UNIQUE.
- Either says that no two tuples of the relation may agree in all the attribute(s) on the list.
- There are a few distinctions to be mentioned later.

Declaring Single-Attribute Keys

- Place PRIMARY KEY or UNIQUE after the type in the declaration of the attribute.
- Example:

```
CREATE TABLE Beers (  
    name        CHAR(20)  UNIQUE,  
    manf        CHAR(20)  
);
```

Declaring Multiattribute Keys

- A key declaration can also be another element in the list of elements of a CREATE TABLE statement.
- This form is essential if the key consists of more than one attribute.
 - May be used even for one-attribute keys.

Example: Multiattribute Key

- The bar and beer together are the key for Sells:

```
CREATE TABLE Sells (  
    bar        CHAR(20),  
    beer       VARCHAR(20),  
    price      REAL,  
    PRIMARY KEY (bar, beer)  
);
```

PRIMARY KEY vs. UNIQUE

1. There can be only one PRIMARY KEY for a relation, but several UNIQUE attributes.
2. No attribute of a PRIMARY KEY can ever be NULL in any tuple. But attributes declared UNIQUE may have NULL's, and there may be several tuples with NULL.

Foreign Keys

- Values appearing in attributes of one relation must be key of another relation.
- **Example:** in **Sells(bar, beer, price)**, we might expect that a beer value also appears in **Beers.name** .

Expressing Foreign Keys

- ◆ Use keyword REFERENCES, either:
 1. After an attribute (for one-attribute keys).
 2. As an element of the schema:
FOREIGN KEY (<list of attributes>
REFERENCES <relation> (<attributes>)
- ◆ Referenced attributes must be declared PRIMARY KEY or UNIQUE.

Example: With Attribute

```
CREATE TABLE Beers (  
    name    CHAR(20) PRIMARY KEY,  
    manf    CHAR(20) );  
  
CREATE TABLE Sells (  
    bar     CHAR(20),  
    beer    CHAR(20) REFERENCES Beers(name),  
    price   REAL );
```

Example: As Schema Element

```
CREATE TABLE Beers (  
    name    CHAR(20) PRIMARY KEY,  
    manf    CHAR(20) );  
CREATE TABLE Sells (  
    bar     CHAR(20),  
    beer    CHAR(20),  
    price   REAL,  
    FOREIGN KEY (beer) REFERENCES  
    Beers (name) );
```

Running Example

Beers(name, manf)

Bars(name, addr, license)

Drinkers(name, addr, phone)

Likes(drinker, beer)

Sells(bar, beer, price)

Frequents(drinker, bar)

- Underline = *key* (tuples cannot have the same value in all key attributes)

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.

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 *query language* for relations.

Core Relational Algebra

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

Selection

- $R1 := \sigma_C(R2)$
 - 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 := $\sigma_{\text{bar}=\text{"Joe's"}}(\text{Sells})$:

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

Projection

- $R1 := \pi_L(R2)$
 - 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 := $\Pi_{\text{beer,price}}$ (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:
 - ▶ Arithmetic on attributes, e.g., $A+B \rightarrow C$.
 - ▶ Duplicate occurrences of the same attribute.

Example: Extended Projection

$R =$

A	B
1	2
3	4

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

C	A1	A2
3	1	1
7	3	3

Product

- $R3 := R1 \times R2$
 - 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 $R2.A$.

Example: $R3 := R1 \times R2$

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

- $R3 := R1 \bowtie_C R2$
 - 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 $A \theta B$, where θ is $=, <, \text{etc.}$; hence the name “theta-join.”

Example: Theta Join

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

BarInfo := Sells $\bowtie_{\text{Sells.bar} = \text{Bars.name}}$ 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 $R3 := R1 \bowtie R2$.

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	Miller	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.
- $R1 := \rho_{R1(A1, \dots, An)}(R2)$ makes R1 be a relation with attributes $A1, \dots, An$ and the same tuples as R2.
- Simplified notation: $R1(A1, \dots, An) := R2$.

Example: Renaming

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.
- ◆ 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:** $R3 := R1 \bowtie_C R2$ can be written:

$R4 := R1 \times R2$

$R3 := \sigma_C(R4)$

Expressions in a Single Assignment

- ◆ **Example:** the theta-join $R3 := R1 \bowtie_C R2$ can be written: $R3 := \sigma_C (R1 \times R2)$
- ◆ Precedence of relational operators:
 - ◆ $[\sigma, \pi, \rho]$ (highest).
 - ◆ $[\times, \bowtie]$.
 - ◆ \cap .
 - ◆ $[\cup, \text{---}]$

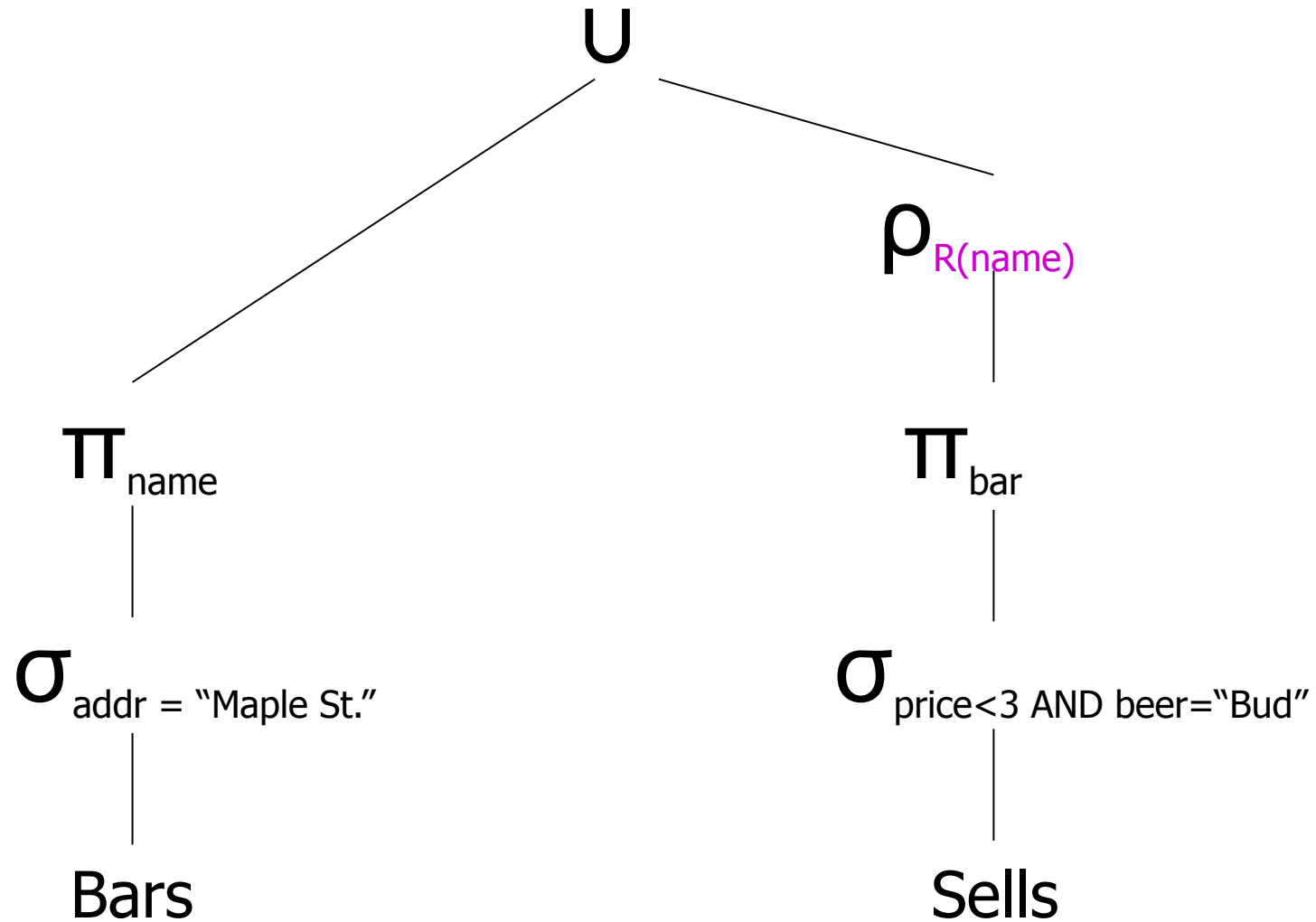
Expression Trees

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

Example: Tree for a Query

- 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.

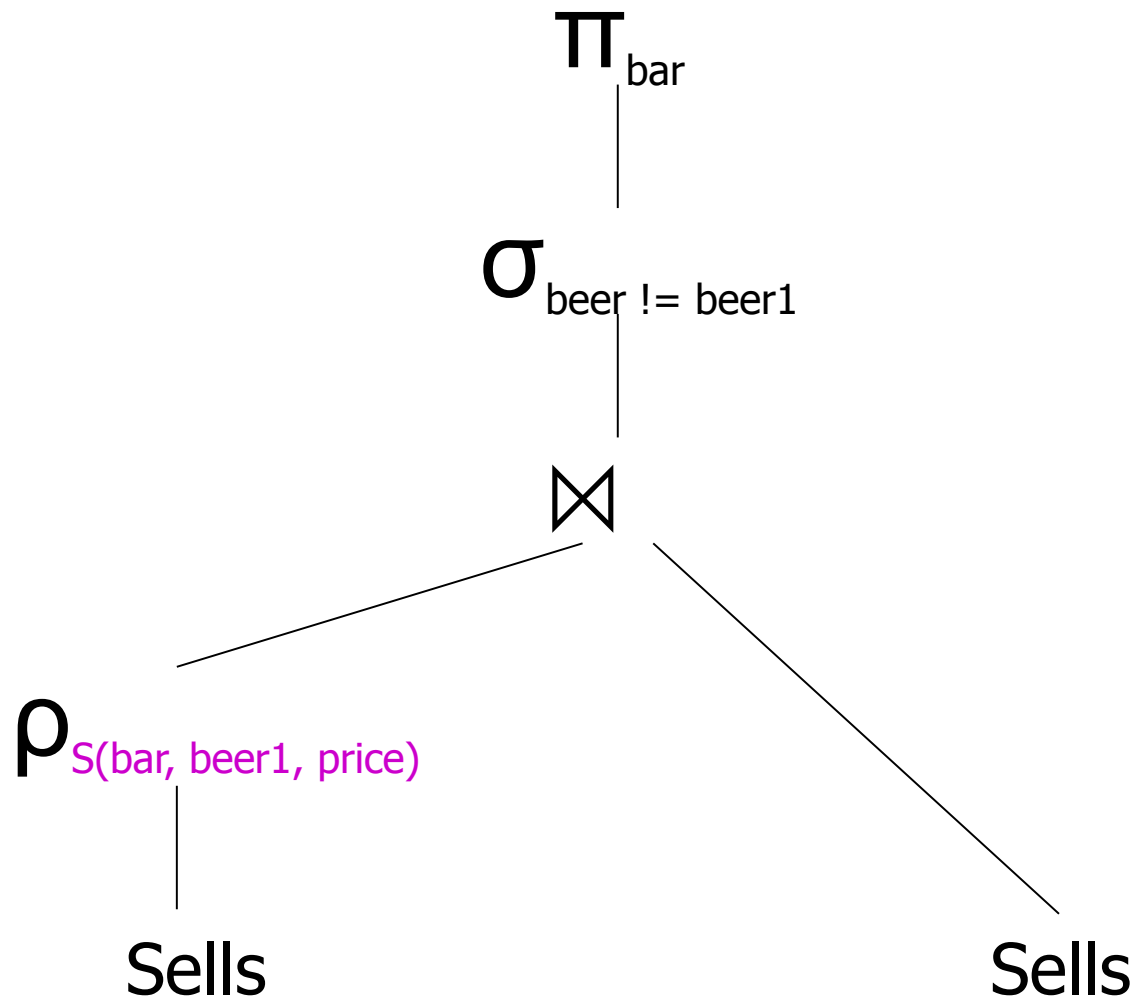
As a Tree:



Example: Self-Join

- Using $\text{Sells}(\text{bar}, \text{beer}, \text{price})$, find the bars that sell two different beers at the same price.
- **Strategy:** by renaming, define a copy of Sells, called $\text{S}(\text{bar}, \text{beer1}, \text{price})$. The natural join of Sells and S consists of quadruples $(\text{bar}, \text{beer}, \text{beer1}, \text{price})$ 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 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

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

A	B
1	2
1	2

Example: Bag Projection

R(

A,	B)
1	2
5	6
1	2

$\Pi_A(R) =$

A
1
5
1

Example: Bag Product

R(

A,	B)
1	2
5	6
1	2

S(

B,	C)
3	4
7	8

R 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

Example: Bag Theta-Join

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 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:** $\{1,2,1,1\} - \{1,2,3\} = \{1,1\}$.

Beware: Bag Laws \neq 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 .

Example: A Law That Fails

- 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 $S \cup S$.
- Thus $S \cup S \neq S$ in general.
 - e.g., $\{1\} \cup \{1\} = \{1,1\} \neq \{1\}$.

Take-away Points

- Relational algebra:
 - Operands are relations (sets of tuples with associated schema)
 - Operations are:
 - Select, project, cartesian product, set difference, union
 - Joins (thetajoin, equijoin, natural join, outer join)
 - Intersection, division