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

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., `'Trader Joe''s Hofbrau Bock'`
- Any value can be NULL
  - (like Objects in Java)

Dates and Times

- DATE and TIME are types in SQL
- The form of a date value is:

  DATE 'yyyy-mm-dd'

  - Example: `DATE '2009-02-04'` for February 4, 2009

Times as Values

- The form of a time value is:

  TIME 'hh:mm:ss'

  with an optional decimal point and fractions of a second following

  - Example: `TIME '15:30:02.5'` = two and a half seconds after 15:30

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

Changing a Relation Schema

- To delete an attribute:

  ALTER TABLE <name> DROP <attribute>;

- To add an attribute:

  ALTER TABLE <name> ADD <element>;

- Examples:

  ALTER TABLE Beers ADD prize CHAR(10);

  ALTER TABLE Drinkers DROP phone;

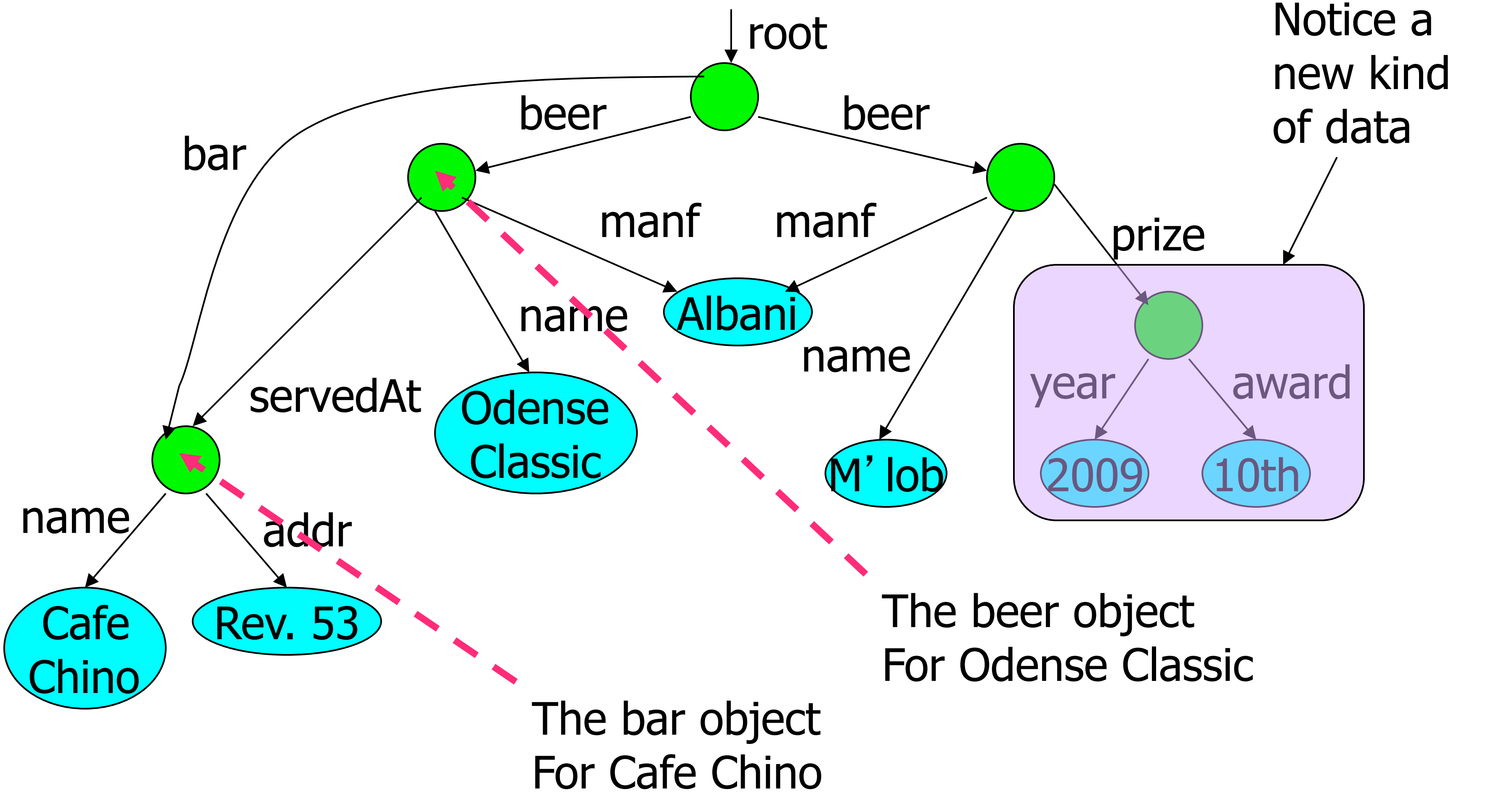
Semistructured Data

- Another data model, based on trees
- Motivation: flexible representation of data
- Motivation: sharing of *documents* among systems and databases

Graphs of Semistructured Data

- Nodes = objects
- Labels on arcs (like attribute names)
- Atomic values at leaf nodes (nodes with no arcs out)
- Flexibility: no restriction on:
  - Labels out of a node
  - Number of successors with a given label

Example: Data Graph



XML

- XML = *Extensible Markup Language*
- While HTML uses tags for formatting (e.g., "italic"), XML uses tags for semantics (e.g., "this is an address")
- Key idea: create tag sets for a domain (e.g., genomics), and translate all data into properly tagged XML documents

XML Documents

- Start the document with a *declaration*, surrounded by <?xml … ?>
- Typical:

```
<?xml version = "1.0" encoding
  = "utf-8" ?>
```

- Document consists of one *root tag* surrounding nested tags

Tags

- Tags, as in HTML, are normally matched pairs, as <FOO> … </FOO>
  - Optional single tag <FOO/>
- Tags may be nested arbitrarily
- XML tags are case sensitive

Example: an XML Document

```
<?xml version = "1.0" encoding = "utf-8" ?>
<BARS>
  <BAR><NAME>Cafe Chino</NAME>
    <BEER><NAME>Odense Classic</NAME>
        <PRICE>20</PRICE></BEER>
    <BEER><NAME>Erdinger Weißbier</NAME>
        <PRICE>35</PRICE></BEER>
  </BAR>
  <BAR> …
</BARS>
```

A NAME subobject

A BEER subobject

Attributes

- Like HTML, the opening tag in XML can have attribute = value pairs
- Attributes also allow linking among elements (discussed later)

Bars, Using Attributes

```
<?xml version = “1.0” encoding = “utf-8” ?>
<BARS>
  <BAR name = “Cafe Chino”>
    <BEER name = “Odense Classic” price = 20 />
    <BEER name = “Erdinger Weißbier” price =
  35 />
  </BAR>
  <BAR> …
</BARS>
```

name and price are attributes

Notice Beer elements have only opening tags with attributes.

DTD's (Document Type Definitions)

- A grammatical notation for describing allowed use of tags.
- Definition form:

```
<!DOCTYPE <root tag> [
  <!ELEMENT <name>(<components>)>
  . . . more elements . . .
]>
```

Example: DTD

```
<!DOCTYPE BARS [
    <!ELEMENT BARS (BAR*)>
    <!ELEMENT BAR (NAME, BEER+)>
    <!ELEMENT NAME (#PCDATA)>
    <!ELEMENT BEER (NAME, PRICE)>
    <!ELEMENT PRICE (#PCDATA)>
]>
```

A BARS object has zero or more BAR's nested within.

A BAR has one NAME and one or more BEER subobjects.

A BEER has a NAME and a PRICE.

NAME and PRICE are HTML text.

Attributes

- Opening tags in XML can have *attributes*
- In a DTD,

`<!ATTLIST` *E* `...>`

declares an attribute for element *E*, along with its datatype

Example: Attributes

No closing tag or subelements

```
<!ELEMENT BEER EMPTY>
  <!ATTLIST name CDATA #REQUIRED,
     manf CDATA #IMPLIED>
```

Character string

Required = "must occur";
Implied = "optional

Example use:
<BEER name="Odense Classic" />

Summary 1

Things you should know now:

- Basic ideas about databases and DBMSs
- What is a data model?
- Idea and Details of the relational model
- SQL as a data definition language

Things given as background:

- History of database systems
- Semistructured data model

Relational Algebra

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
- Example:
  - Integers ..., -1, 0, 1, ... as operands
  - Arithmetic operations +/- as operators

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

- $R_1 := \sigma_C(R_2)$
  - C is a condition (as in "if" statements) that refers to attributes of R_2
  - R_1 is all those tuples of R_2 that satisfy C

Example: Selection

Relation Sells:

bar	beer	price
Cafe Chino	Od. Cla.	20
Cafe Chino	Erd. Wei.	35
Cafe Bio	Od. Cla.	20
Bryggeriet	Pilsener	31

ChinoMenu := $\sigma_{bar=\text{“Cafe Chino”}}$(Sells):

bar	beer	price
Cafe Chino	Od. Cla.	20
Cafe Chino	Erd. Wei.	35

Projection

- $R_1 := \pi_L(R_2)$
  - L is a list of attributes from the schema of R_2
  - R_1 is constructed by looking at each tuple of R_2, extracting the attributes on list L, in the order specified, and creating from those components a tuple for R_1
  - Eliminate duplicate tuples, if any

Example: Projection

Relation Sells:

bar	beer	price
Cafe Chino	Od. Cla.	20
Cafe Chino	Erd. Wei.	35
Cafe Bio	Od. Cla.	20
Bryggeriet	Pilsener	31

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

beer	price
Od. Cla.	20
Erd. Wei.	35
Pilsener	31

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., $A+B\text{->}C$
  2. Duplicate occurrences of the same attribute

Example: Extended Projection

R =

A	B
1	2
3	4

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

C	A_1	A_2
3	1	1
7	3	3

Product

- $R_3 := R_1 \text{ X } R_2$
  - Pair each tuple t_1 of R_1 with each tuple t_2 of R_2
  - Concatenation t_1t_2 is a tuple of R_3
  - Schema of R_3 is the attributes of R_1 and then R_2, in order
  - But beware attribute *A* of the same name in R_1 and R_2: use $R_1.A$ and $R_2.A$

Example: $R_3 := R_1 \times R_2$

$R_1($ A, B $)$

A,	B
1	2
3	4

$R_2($ B, C $)$

B,	C
5	6
7	8
9	10

$R_3($ A, R_1.B, R_2.B, C $)$

A,	R_1.B,	R_2.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

- $R_3 := R_1 \bowtie_C R_2$
  - Take the product R_1 X R_2
  - 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
C.Ch.	Od.C.	20
C.Ch.	Er.W.	35
C.Bi.	Od.C.	20
Bryg.	Pils.	31

)

Bars(

name,	addr
C.Ch.	Reventlo.
C.Bi.	Brandts
Bryg.	Flakhaven

)

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

BarInfo(

bar,	beer,	price,	name,	addr
C.Ch.	Od.C.	20	C.Ch.	Reventlo.
C.Ch.	Er.W.	35	C.Ch.	Reventlo.
C.Bi.	Od.C.	20	C.Bi.	Brandts
Bryg.	Pils.	31	Bryg.	Flakhaven

)

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 $R_3 := R_1 \bowtie R_2$

Example: Natural Join

Sells(

bar,	beer,	price
C.Ch.	Od.Cl.	20
C.Ch.	Er.We.	35
C.Bi.	Od.Cl.	20
Bryg.	Pils.	31

)

Bars(

bar,	addr
C.Ch.	Reventlo.
C.Bi.	Brandts
Bryg.	Flakhaven

)

BarInfo := Sells $\bowtie$ Bars

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

BarInfo(

bar,	beer,	price,	addr
C.Ch.	Od.Cl.	20	Reventlo.
C.Ch.	Er.We.	35	Reventlo.
C.Bi.	Od.Cl.	20	Brandts
Bryg.	Pils.	31	Flakhaven

)

Renaming

- The ρ operator gives a new schema to a relation
- $R_1 := \rho_{R_1(A_1,\ldots,A_n)}(R_2)$ makes R_1 be a relation with attributes $A_1,\ldots,A_n$ and the same tuples as R_2
- Simplified notation: $R_1(A_1,\ldots,A_n) := R_2$

Example: Renaming

Bars(

name,	addr
C.Ch.	Reventlo.
C.Bi.	Brandts
Bryg.	Flakhaven

)

R(bar, addr) := Bars

R(

bar,	addr
C.Ch.	Reventlo.
C.Bi.	Brandts
Bryg.	Flakhaven

)

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: $R_3 := R_1 \bowtie_C R_2$ can be written:

  $R_4 := R_1 \text{ X } R_2$

  $R_3 := \sigma_C(R_4)$

Expressions in a Single Assignment

- Example: the theta-join $R_3 := R_1 \bowtie_C R_2$ can be written: $R_3 := \sigma_C (R_1 \text{ X } R_2)$
- Precedence of relational operators:
  1. [σ, π, ρ] (highest)
  2. [X, $\bowtie$]
  3. $\cap$
  4. [$\cup$, —]

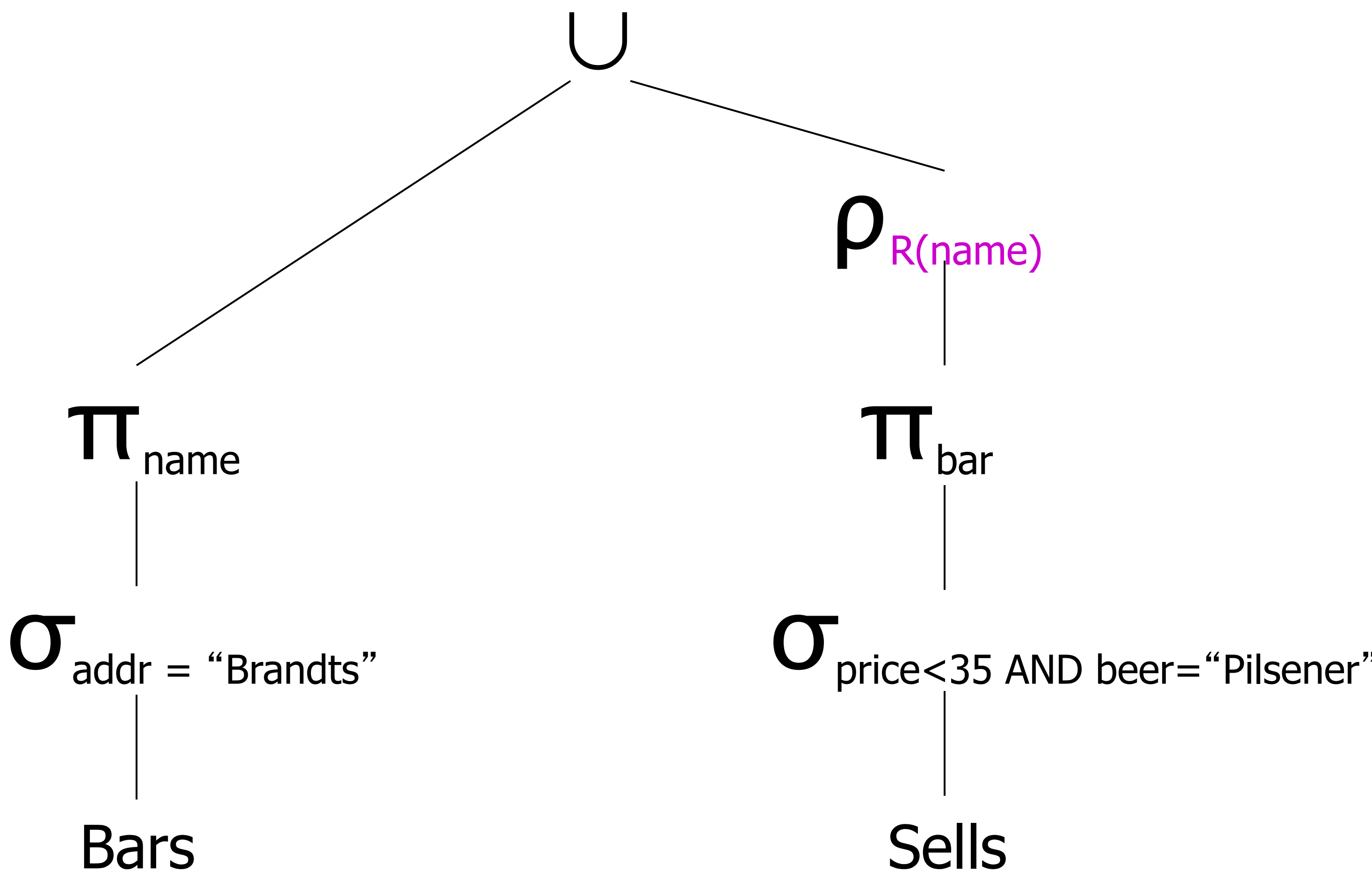
Expression Trees

- 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 at Brandts or sell Pilsener for less than 35:

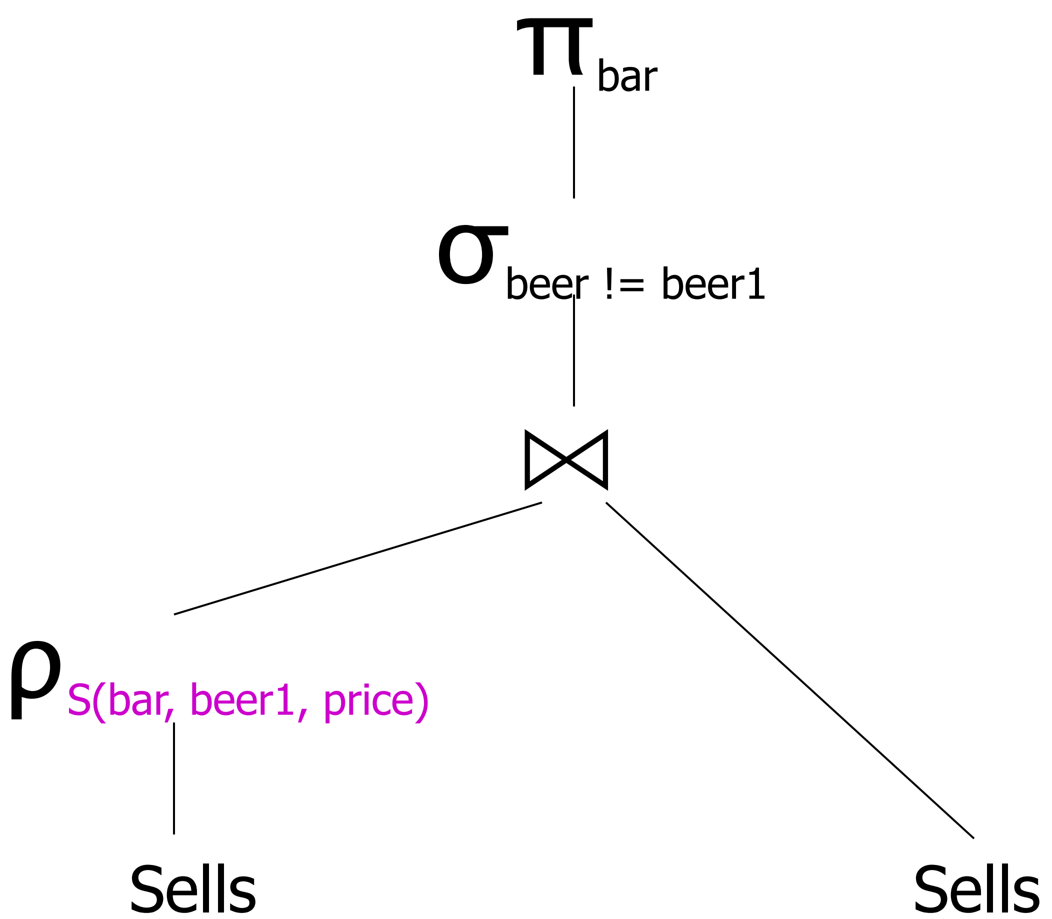
As a Tree:


∪
ρ R(name)
π name
π bar
σ addr = “Brandts”
σ price<35 AND beer=“Pilsener”
Bars
Sells

Example: 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

The Tree



π_{bar}
$\sigma_{beer\ != \ beer1}$
$\bowtie$
$\rho_{S(bar,\ beer1,\ price)}$
Sells
Sells

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

- Product: schema is the attributes of both relations
  - Use $R_1.A$ and $R_2.A$, 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 != 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 does not 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 $S \cup S$!= S in general
  - e.g., {1} ∪ {1} = {1,1} != {1}

Summary 2

More things you should know:

- Relational Algebra
- Selection, (Extended) Projection, Product, Join, Natural Join, Renaming
- Complex Operations as Sequences, Expressions, or Trees
- Difference between Sets and Bags

Basic SQL Queries

Why SQL?

- SQL is a very-high-level language
  - Say "what to do" rather than "how to do it"
  - Avoid a lot of data-manipulation details needed in procedural languages like C++ or Java
- Database management system figures out "best" way to execute query
  - Called "query optimization"

Select-From-Where Statements

SELECT desired attributes
FROM one or more tables
WHERE condition about tuples of
the tables

Our Running Example

- All our SQL queries will be based on the following database schema.
  - Underline indicates key attributes.

Beers(<u>name</u>, manf)

Bars(<u>name</u>, addr, license)

Drinkers(<u>name</u>, addr, phone)

Likes(<u>drinker</u>, <u>beer</u>)

Sells(<u>bar</u>, <u>beer</u>, price)

Frequents(<u>drinker</u>, <u>bar</u>)

Example

- Using Beers(name, manf), what beers are made by Albani Bryggerierne?

```
SELECT name
FROM Beers
WHERE manf = 'Albani';
```

Result of Query

name
Od. Cl.
Eventyr
Blålys
. . .

The answer is a relation with a single attribute, name, and tuples with the name of each beer by Albani Bryggerierne, such as Odense Classic.

Meaning of Single-Relation Query

- Begin with the relation in the FROM clause
- Apply the selection indicated by the WHERE clause
- Apply the extended projection indicated by the SELECT clause

Operational Semantics

name	manf
Blålys	Albani

Include t.name in the result, if so

Check if Albani

Tuple-variable *t* loops over all tuples

Operational Semantics – General

- Think of a *tuple variable* visiting each tuple of the relation mentioned in FROM
- Check if the "current" tuple satisfies the WHERE clause
- If so, compute the attributes or expressions of the SELECT clause using the components of this tuple

* In SELECT clauses

- When there is one relation in the FROM clause, * in the SELECT clause stands for "all attributes of this relation"
- Example: Using Beers(name, manf):

```
SELECT *
FROM Beers
WHERE manf = 'Albani';
```

Result of Query:

name	manf
Od.Cl.	Albani
Eventyr	Albani
Blålys	Albani
. . .	. . .

Now, the result has each of the attributes of Beers