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 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)
1	2
3	4

R_2(

B,	C)
5	6
7	8
9	10

R_3(

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 $R_1 \times R_2$
  - Then apply σ_C to the result
- As for σ, C can be any boolean-valued condition
  - Historic versions of this operator allowed only $A \theta B$, 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: $R_3 := \sigma_C (R_1 \times R_2)$
- Precedence of relational operators:
  1. $[\sigma, \pi, \rho]$ (highest)
  2. $[\times, \bowtie]$
  3. $\cap$
  4. $[\cup, —]$

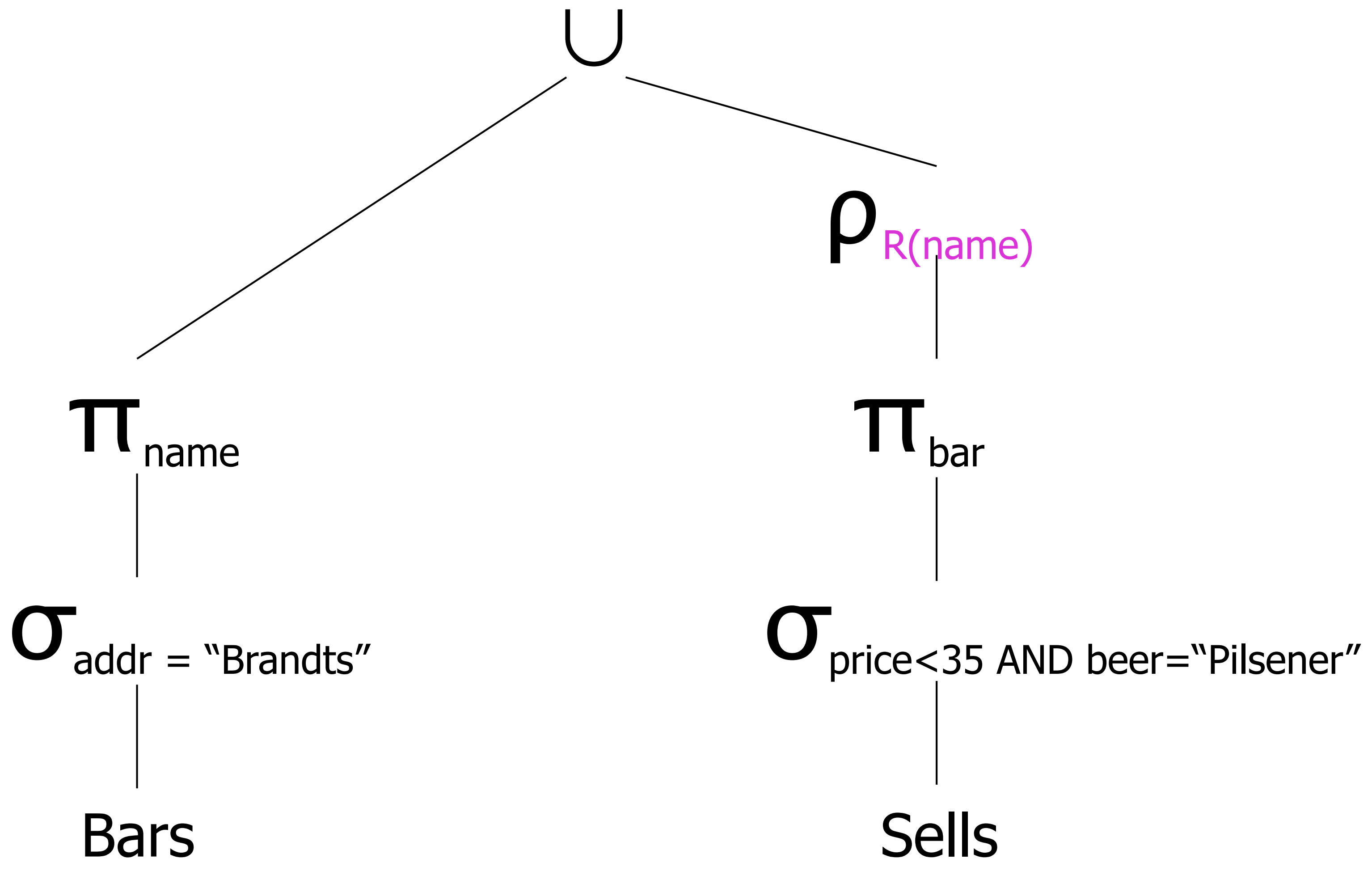
Expression Trees

- 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:

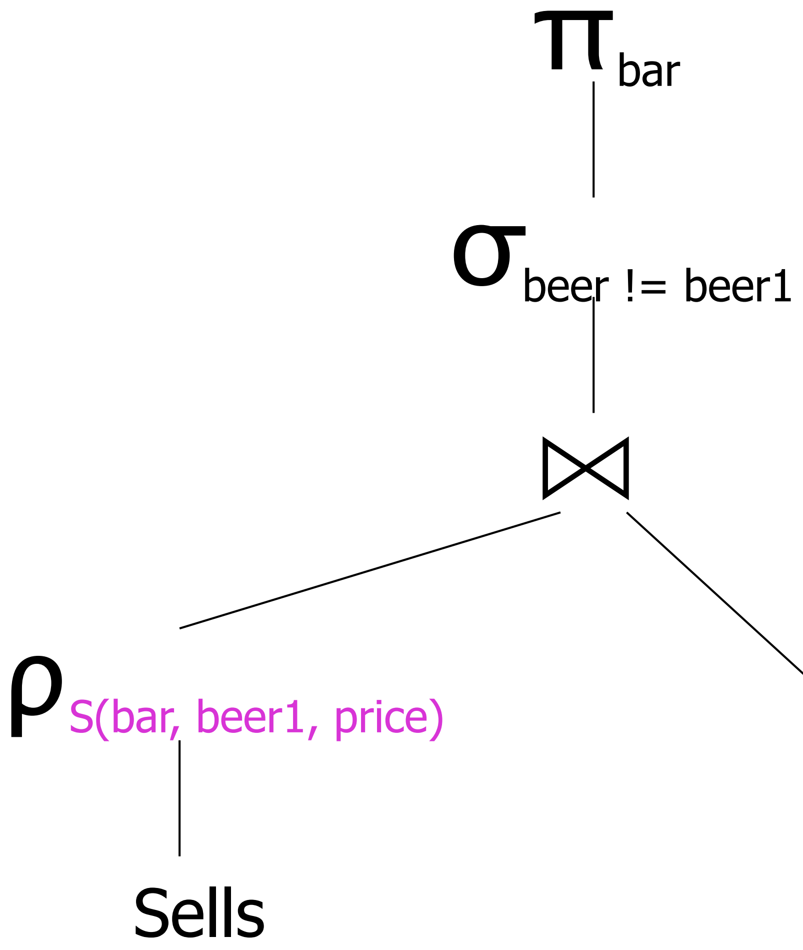
As a Tree:


$\cup$
$\rho_{R(name)}$
π_{name}
π_{bar}
$\sigma_{addr = \text{"Brandts"}}$
$\sigma_{price<35 \text{ AND } beer=\text{"Pilsener"}}$
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(name, manf)

Bars(name, addr, license)

Drinkers(name, addr, phone)

Likes(drinker, beer)

Sells(bar, beer, price)

Frequents(drinker, bar)

Example

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

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

Result of Query

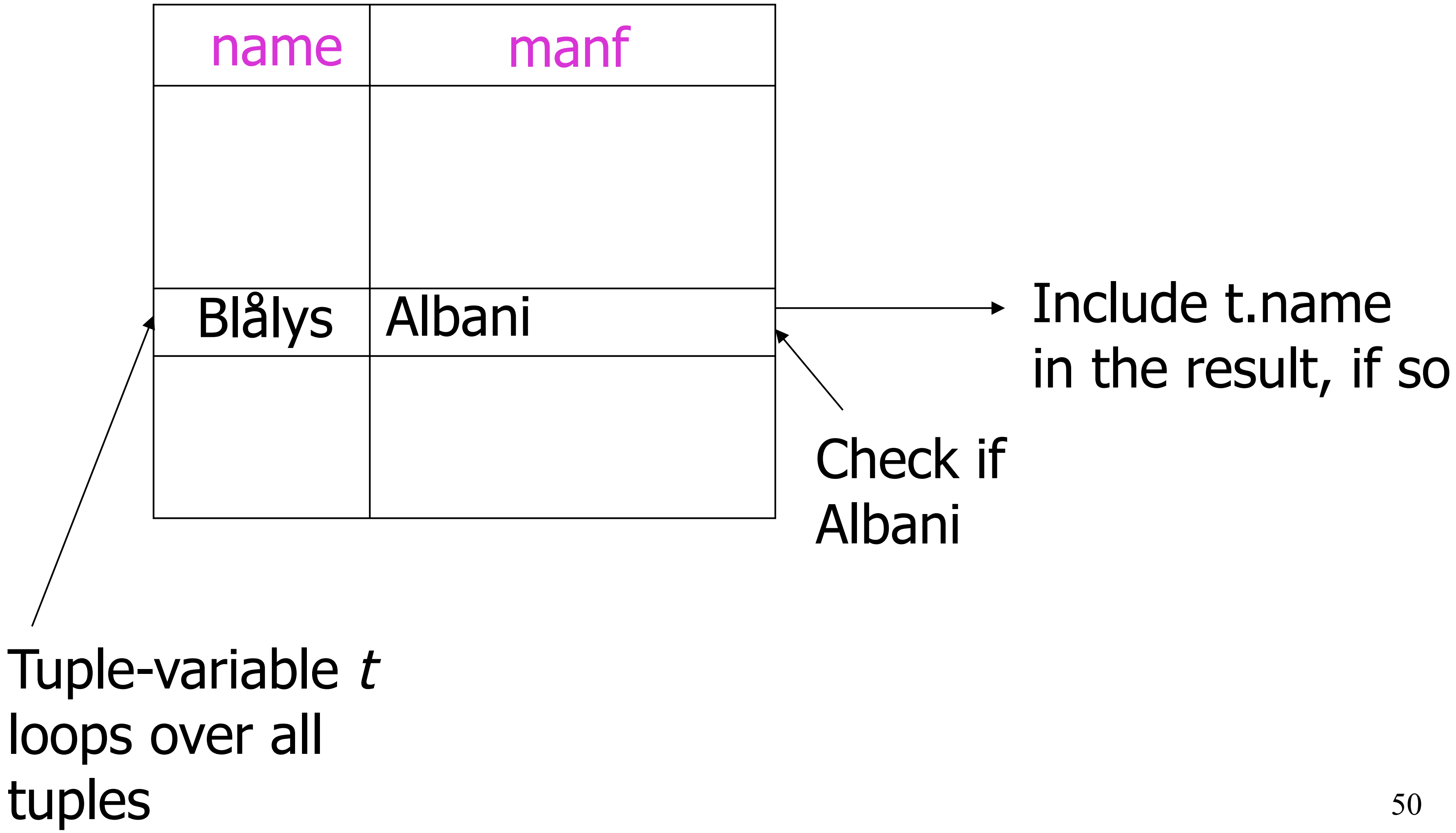
name
Od. Cl.
Eventyr
Blålys
. . .

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

Renaming Attributes

- If you want the result to have different attribute names, use "AS <new name>" to rename an attribute
- Example: Using Beers(name, manf):

```
SELECT name AS beer, manf
   FROM Beers
   WHERE manf = 'Albani'
```

Result of Query:

beer	manf
Od.Cl.	Albani
Eventyr	Albani
Blålys	Albani
. . .	. . .

Expressions in SELECT Clauses

- Any expression that makes sense can appear as an element of a SELECT clause
- Example: Using Sells(bar, beer, price):

```
SELECT bar, beer,
   price*0.134 AS priceInEuro
FROM Sells;
```

Result of Query

bar	beer	priceInEuro
C.Ch.	Od.Cl.	2.68
C.Ch.	Er.Wei.	4.69
…	…	…

Example: Constants as Expressions

- Using Likes(drinker, beer):

```
SELECT drinker, ’ likes Albani ’
   AS whoLikesAlbani
FROM Likes
WHERE beer = ’Od.Cl.’;
```

Result of Query

drinker	whoLikesAlbani
Peter	likes Albani
Kim	likes Albani
…	…

Example: Information Integration

- We often build "data warehouses" from the data at many "sources"
- Suppose each bar has its own relation Menu(beer, price)
- To contribute to Sells(bar, beer, price) we need to query each bar and insert the name of the bar

Information Integration

- For instance, at the Cafe Biografen we can issue the query:

```
SELECT ’Cafe Bio’, beer, price
FROM Menu;
```

Complex Conditions in WHERE Clause

- Boolean operators AND, OR, NOT
- Comparisons =, <>, <, >, <=, >=
  - And many other operators that produce boolean-valued results

Example: Complex Condition

- Using Sells(bar, beer, price), find the price Cafe Biografen charges for Odense Classic:

```
SELECT price
  FROM Sells
  WHERE bar = 'Cafe Bio' AND
     beer = 'Od.Cl.';
```

Patterns

- A condition can compare a string to a pattern by:
  - <Attribute> LIKE <pattern> or <Attribute> NOT LIKE <pattern>
- *Pattern* is a quoted string with
  % = "any string"
  _ = "any character"

Example: LIKE

- Using Drinkers(name, addr, phone) find the drinkers with address in Fynen:

```
SELECT name
FROM Drinkers
WHERE phone LIKE ’%, 5___ %’;
```

NULL Values

- Tuples in SQL relations can have NULL as a value for one or more components
- Meaning depends on context
- Two common cases:
  - *Missing value:* e.g., we know Cafe Chino has some address, but we don’t know what it is
  - *Inapplicable:* e.g., the value of attribute spouse for an unmarried person

Comparing NULL’s to Values

- The logic of conditions in SQL is really 3-valued logic: TRUE, FALSE, UNKNOWN
- Comparing any value (including NULL itself) with NULL yields UNKNOWN
- A tuple is in a query answer iff the WHERE clause is TRUE (not FALSE or UNKNOWN)

Three-Valued Logic

- To understand how AND, OR, and NOT work in 3-valued logic, think of TRUE = 1, FALSE = 0, and UNKNOWN = ½
- AND = MIN; OR = MAX; NOT(*x*) = 1-*x*
- Example:

TRUE AND (FALSE OR NOT(UNKNOWN)) =
MIN(1, MAX(0, (1 - ½))) =
MIN(1, MAX(0, ½)) = MIN(1, ½) = ½

Surprising Example

- From the following Sells relation:

bar	beer	price
C.Ch.	Od.Cl.	NULL

```
SELECT bar
FROM Sells
```

$$\text{WHERE } \underbrace{\underbrace{\text{price < 20}}_{\text{UNKNOWN}} \text{ OR } \underbrace{\text{price >= 20;}}_{\text{UNKNOWN}}}_{\text{UNKNOWN}}$$

2-Valued Laws != 3-Valued Laws

- Some common laws, like commutativity of AND, hold in 3-valued logic
- But not others, e.g., the *law of the excluded middle:* *p* OR NOT *p* = TRUE
  - When *p* = UNKNOWN, the left side is MAX(½, (1 – ½)) = ½ != 1

Multirelation Queries

- Interesting queries often combine data from more than one relation
- We can address several relations in one query by listing them all in the FROM clause
- Distinguish attributes of the same name by "<relation>.<attribute>"

Example: Joining Two Relations

- Using relations Likes(drinker, beer) and Frequents(drinker, bar), find the beers liked by at least one person who frequents C. Ch.

```
SELECT beer
FROM Likes, Frequents
WHERE bar = 'C.Ch.' AND
   Frequents.drinker =
                  Likes.drinker;
```

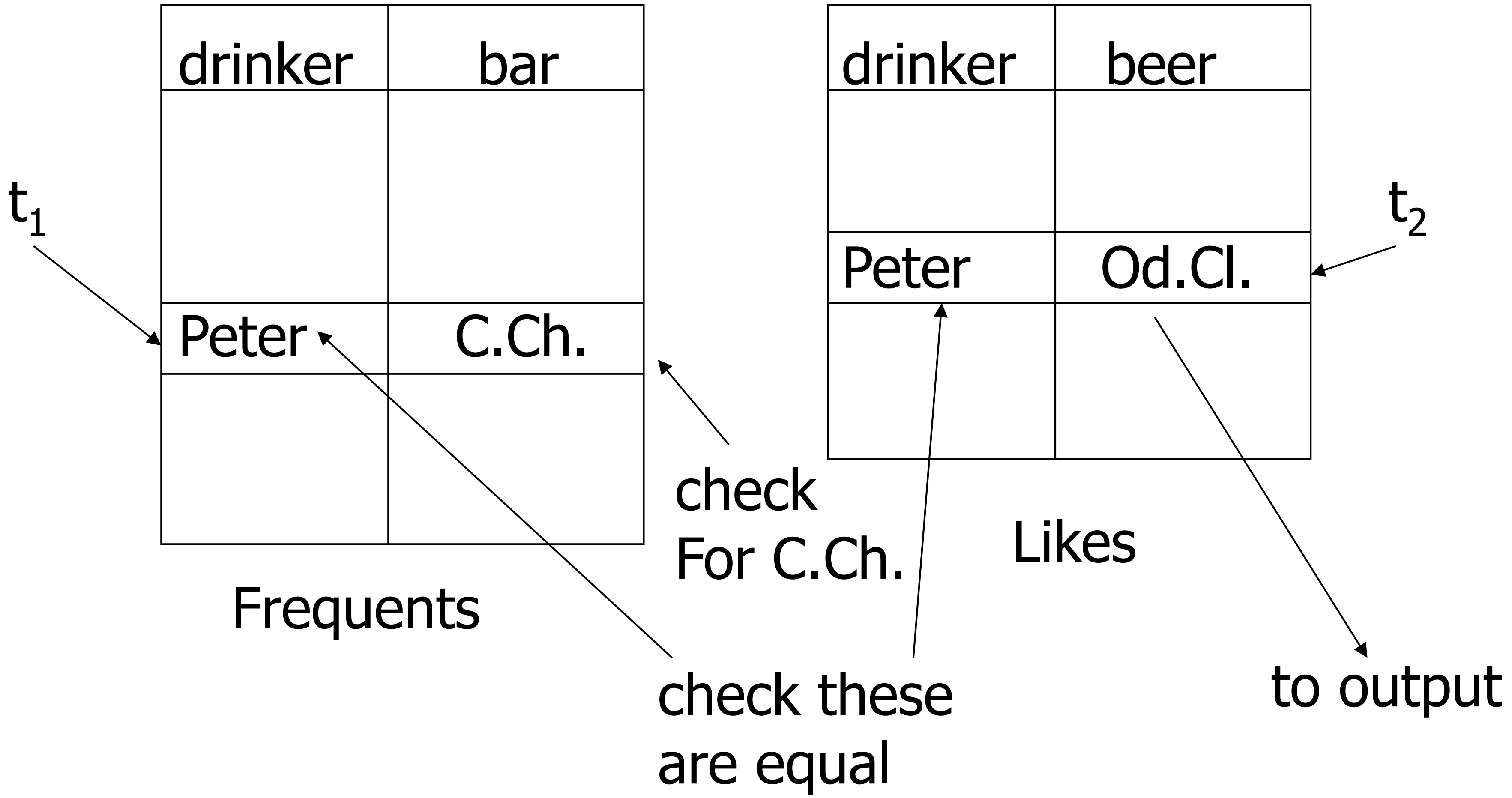
Formal Semantics

- Almost the same as for single-relation queries:
  1. Start with the product of all the relations in the FROM clause
  2. Apply the selection condition from the WHERE clause
  3. Project onto the list of attributes and expressions in the SELECT clause

Operational Semantics

- Imagine one tuple-variable for each relation in the FROM clause
  - These tuple-variables visit each combination of tuples, one from each relation
- If the tuple-variables are pointing to tuples that satisfy the WHERE clause, send these tuples to the SELECT clause

Example



drinker
bar
Peter
C.Ch.
Frequents
t_1
check
For C.Ch.
check these
are equal
drinker
beer
Peter
Od.Cl.
Likes
t_2
to output

Explicit Tuple-Variables

- Sometimes, a query needs to use two copies of the same relation
- Distinguish copies by following the relation name by the name of a tuple-variable, in the FROM clause
- It's always an option to rename relations this way, even when not essential

Example: Self-Join

- From Beers(name, manf), find all pairs of beers by the same manufacturer
  - Do not produce pairs like (Od.Cl., Od.Cl.)
  - Produce pairs in alphabetic order, e.g., (Blålys, Eventyr), not (Eventyr, Blålys)

```
SELECT b1.name, b2.name
FROM Beers b1, Beers b2
WHERE b1.manf = b2.manf AND
   b1.name < b2.name;
```

Subqueries

- A parenthesized SELECT-FROM-WHERE statement (*subquery*) can be used as a value in a number of places, including FROM and WHERE clauses
- Example: in place of a relation in the FROM clause, we can use a subquery and then query its result
  - Must use a tuple-variable to name tuples of the result

Example: Subquery in FROM

- Find the beers liked by at least one person who frequents Cafe Chino

Drinkers who frequent C.Ch.

```
SELECT beer
FROM Likes, (SELECT drinker
     FROM Frequents
     WHERE bar = 'C.Ch.')CCD
WHERE Likes.drinker = CCD.drinker;
```

Subqueries That Return One Tuple

- If a subquery is guaranteed to produce one tuple, then the subquery can be used as a value
  - Usually, the tuple has one component
  - A run-time error occurs if there is no tuple or more than one tuple