DM505 Database Design and Programming
DM576 Database Systems

Panagiotis Tampakis

ptampakis@imada.sdu.dk

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

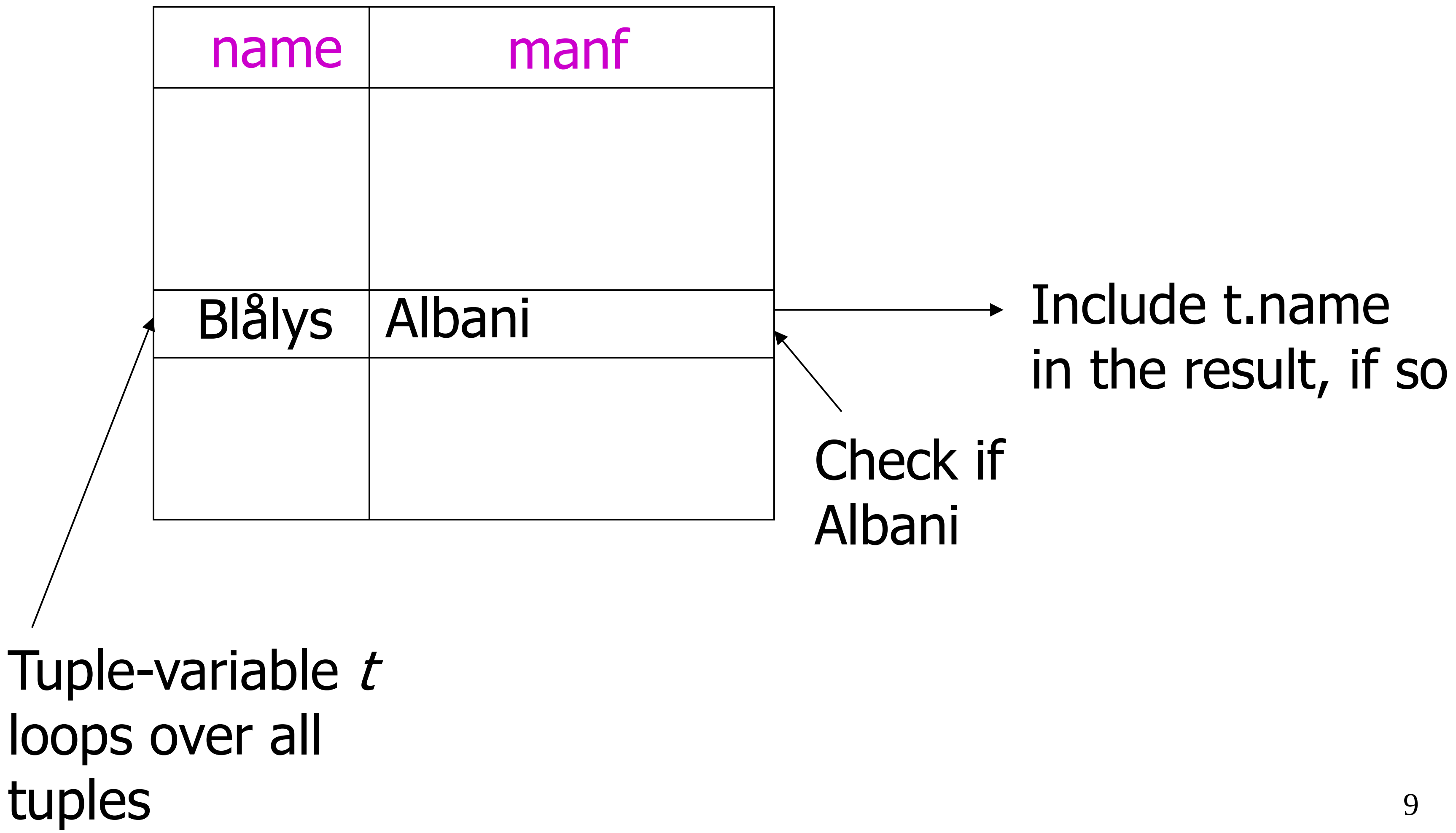
name
Od. Cl.
Eventyr
Blålys
. . .

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
…	…	…

Expressions in SELECT Clauses

- A new attribute can also be added.
- Example: Using Sells(bar, beer, price):

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

Result of Query

bar	beer	price	priceInEuro
C.Ch.	Od.Cl.	20	2.68
C.Ch.	Er.Wei.	35	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' AS bar, 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 address 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

- Let's say, we additionally have the attributes type and isRegularPrice
- Now we want to find those beers which are "Pilsener" and cost less than 20 DKK or are on sale.

- … WHERE type = 'Pilsener' AND (price < 20 OR NOT(isRegularPrice))

Example

- Now let’s say we have a Pilsener costing 30 DKK but the bar doesn’t provide Information whether they have a sale or not.

- type = ‘Pilsener’ AND (price < 20 OR NOT(isRegularPrice))

Evaluates to …

- Now let’s say we have a Pilsener costing 30 DKK but the bar doesn’t provide Information whether they have a sale or not.
- TRUE AND (FALSE OR NOT(UNKNOWN))

Replace with values

- Now let’s say we have a Pilsener costing 30 DKK but the bar doesn’t provide Information whether they have a sale or not.
- 1 AND (0 OR NOT(½))

Evaluates to …

- Now let’s say we have a Pilsener costing 30 DKK but the bar doesn’t provide Information whether they have a sale or not.

- 1 AND (0 OR NOT(½))

Evaluates to …

- Now let's say we have a Pilsener costing 30 DKK but the bar doesn't provide Information whether they have a sale or not.

- MIN(1, 0 OR NOT(½))

Evaluates to …

- Now let’s say we have a Pilsener costing 30 DKK but the bar doesn’t provide Information whether they have a sale or not.

- MIN(1, 0 OR NOT(½))

Evaluates to …

- Now let's say we have a Pilsener costing 30 DKK but the bar doesn't provide Information whether they have a sale or not.

- MIN(1, MAX(0, NOT(½)))

Evaluates to …

- Now let's say we have a Pilsener costing 30 DKK but the bar doesn't provide Information whether they have a sale or not.

- MIN(1, MAX(0, NOT(½)))

Evaluates to …

- Now let’s say we have a Pilsener costing 30 DKK but the bar doesn’t provide Information whether they have a sale or not.

- MIN(1, MAX(0, (1 - ½)))

Evaluates to …

- Now let' s say we have a Pilsener costing 30 DKK but the bar doesn' t provide Information whether they have a sale or not.

- MIN(1, MAX(0, (1 - ½)))

Evaluates to …

- Now let's say we have a Pilsener costing 30 DKK but the bar doesn't provide Information whether they have a sale or not.

- MIN(1, MAX(0, ½))

Evaluates to …

- Now let’s say we have a Pilsener costing 30 DKK but the bar doesn’t provide Information whether they have a sale or not.

- MIN(1, MAX(0, ½))

Evaluates to …

- Now let's say we have a Pilsener costing 30 DKK but the bar doesn't provide Information whether they have a sale or not.

- MIN(1, ½)

Evaluates to …

- Now let's say we have a Pilsener costing 30 DKK but the bar doesn't provide Information whether they have a sale or not.

- MIN(1, ½)

Evaluates to …

- Now let’s say we have a Pilsener costing 30 DKK but the bar doesn’t provide Information whether they have a sale or not.

- ½ = UNKOWN

Surprising Example

- From the following Sells relation:

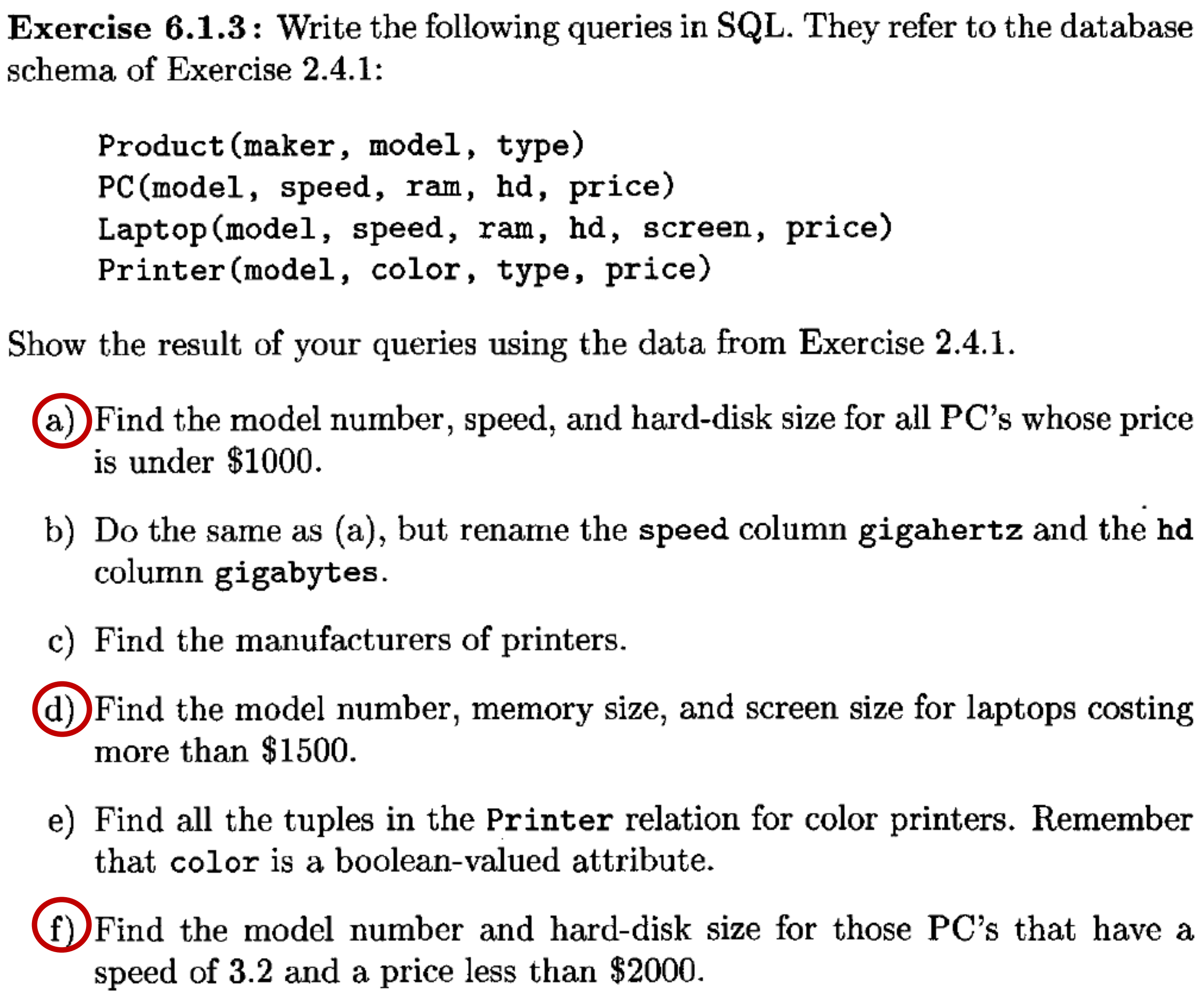
bar	beer	price
C.Ch.	Od.Cl.	NULL

```
SELECT bar
FROM Sells
WHERE price < 20 OR price >= 20;
```

$\xleftrightarrow[\text{UNKNOWN}]{\qquad\qquad}$ $\qquad$ $\xleftrightarrow[\text{UNKNOWN}]{\qquad\qquad}$

$\xleftrightarrow[\text{UNKNOWN}]{\qquad\qquad\qquad\qquad\qquad\qquad}$

Exercise



Exercise 6.1.3: Write the following queries in SQL. They refer to the database schema of Exercise 2.4.1:

```
Product(maker, model, type)
PC(model, speed, ram, hd, price)
Laptop(model, speed, ram, hd, screen, price)
Printer(model, color, type, price)
```

Show the result of your queries using the data from Exercise 2.4.1.

a) Find the model number, speed, and hard-disk size for all PC's whose price is under $1000.

b) Do the same as (a), but rename the `speed` column `gigahertz` and the `hd` column `gigabytes`.

c) Find the manufacturers of printers.

d) Find the model number, memory size, and screen size for laptops costing more than $1500.

e) Find all the tuples in the `Printer` relation for color printers. Remember that `color` is a boolean-valued attribute.

f) Find the model number and hard-disk size for those PC's that have a speed of 3.2 and a price less than $2000.

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;
```

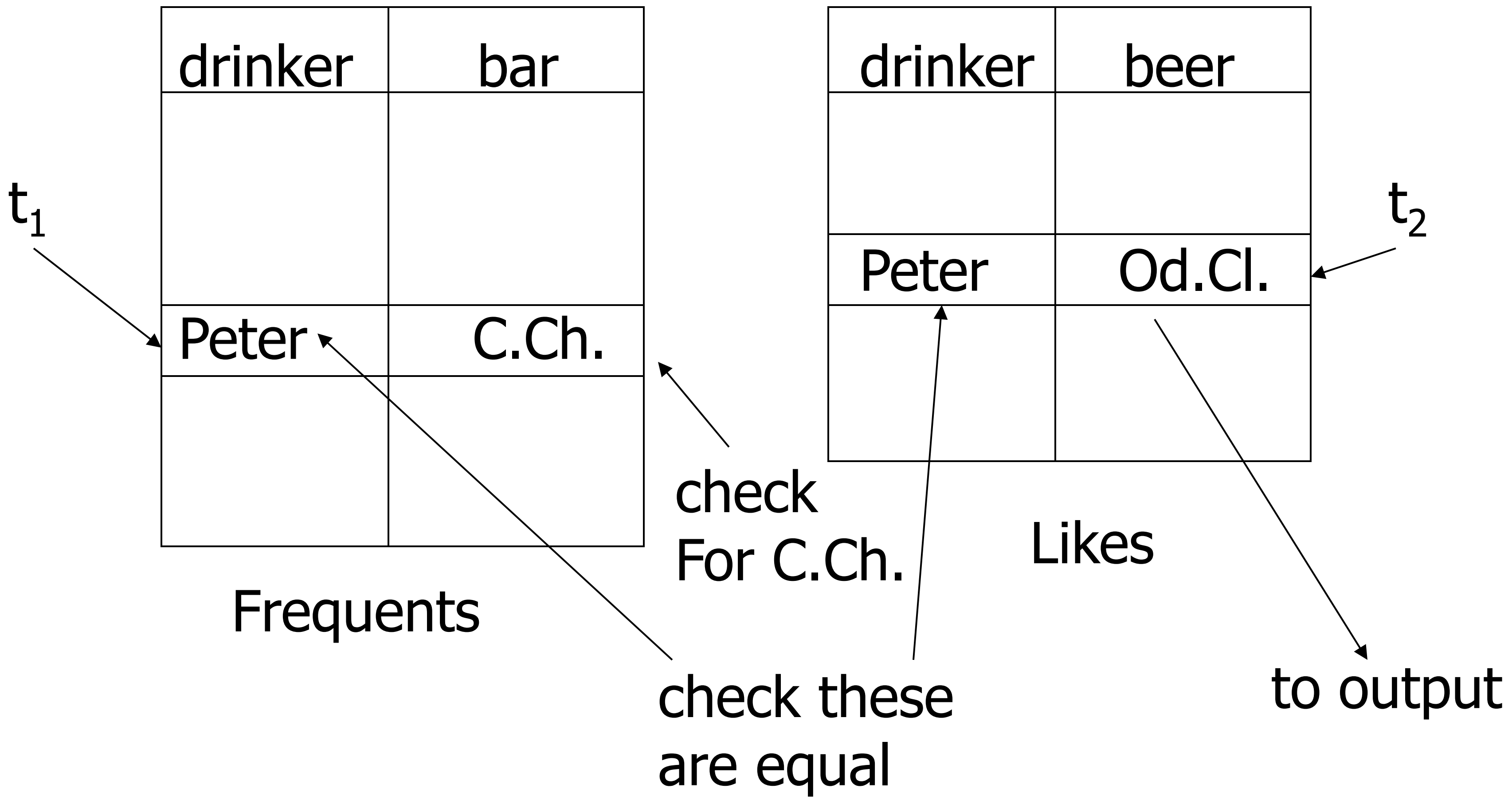
Formal Semantics

- 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
drinker
beer
Peter
Od.Cl.
Likes
t_2
check
For C.Ch.
check these
are equal
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;
```

Products and Natural Joins

- Natural join:

  R NATURAL JOIN S;

- Product:

  R CROSS JOIN S;

- Example:

  ```
  Likes NATURAL JOIN Sells;
  ```

- Relations can be parenthesized subqueries, as well

Theta Join

- R JOIN S ON <condition>
- Example: using Drinkers(name, addr) and Frequents(drinker, bar):

```
    Drinkers JOIN Frequents ON
         name = drinker;
```

gives us all (*d*, *a*, *d*, *b*) quadruples such that drinker *d* lives at address *a* and frequents bar *b*

Union, Intersection, and Difference

- Union, intersection, and difference of relations are expressed by the following forms, each involving subqueries:
  - *R* UNION *S*
  - *R* INTERSECT *S*
  - *R* EXCEPT *S*

where *R* and *S* are two relations that satisfy the requirements of Set Operations (Union, Intersectionand Difference).

Example: Intersection

- Using Likes(drinker, beer), Sells(bar, beer, price), and Frequents(drinker, bar), find the drinkers and beers such that:
    1. The drinker likes the beer, and
    2. The drinker frequents at least one bar that sells the beer

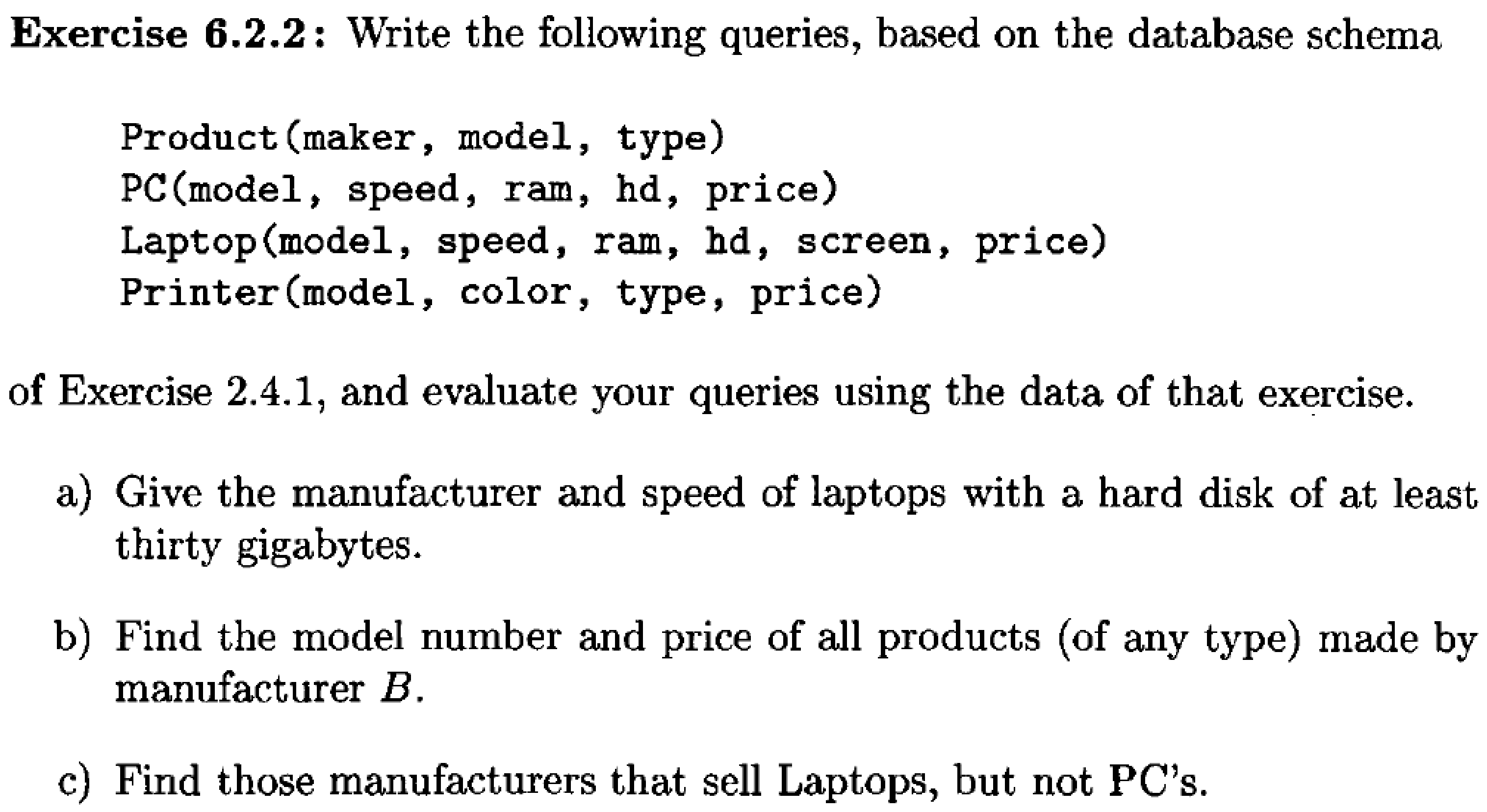
Notice trick: subquery is actually a stored table.

Solution

The drinker frequents a bar that sells the beer.

```
(SELECT * FROM Likes)
   INTERSECT
(SELECT drinker, beer
 FROM Sells, Frequents
 WHERE Frequents.bar = Sells.bar
);
```

Exercise



Exercise 6.2.2: Write the following queries, based on the database schema

```
Product(maker, model, type)
PC(model, speed, ram, hd, price)
Laptop(model, speed, ram, hd, screen, price)
Printer(model, color, type, price)
```

of Exercise 2.4.1, and evaluate your queries using the data of that exercise.

a) Give the manufacturer and speed of laptops with a hard disk of at least thirty gigabytes.

b) Find the model number and price of all products (of any type) made by manufacturer *B*.

c) Find those manufacturers that sell Laptops, but not PC's.

Summary 3

More things you should know:

- SELECT FROM WHERE statements with one or more tables
- Complex conditions, pattern matching
- Natural joins, theta joins