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

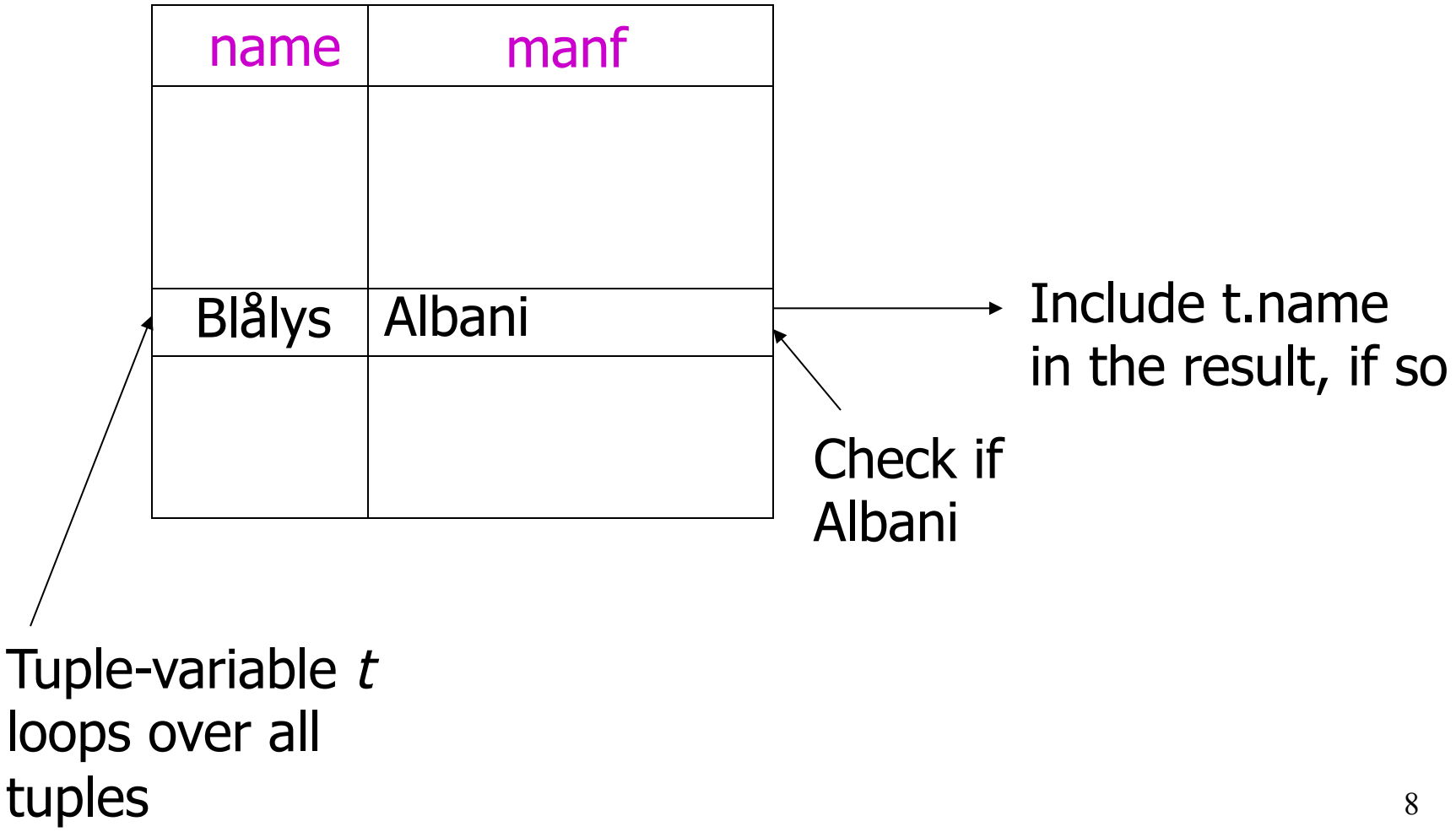
name
Od. Cl.
Eventyr
Blålys
. . .

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



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 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:

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

Surprising Example

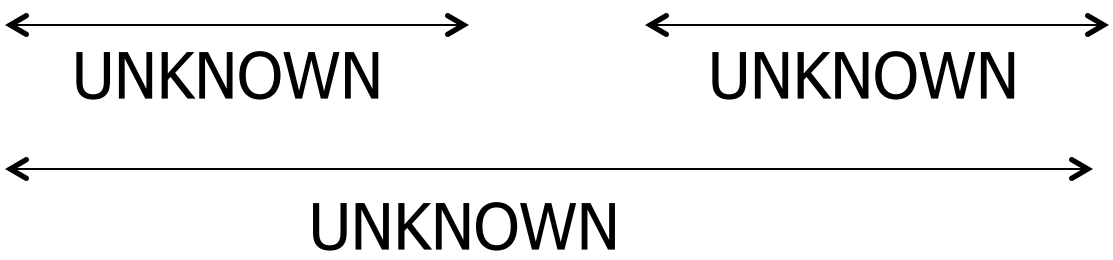
- From the following Sells relation:

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

SELECT bar

FROM Sells

WHERE price < 20 OR price >= 20;



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;
```

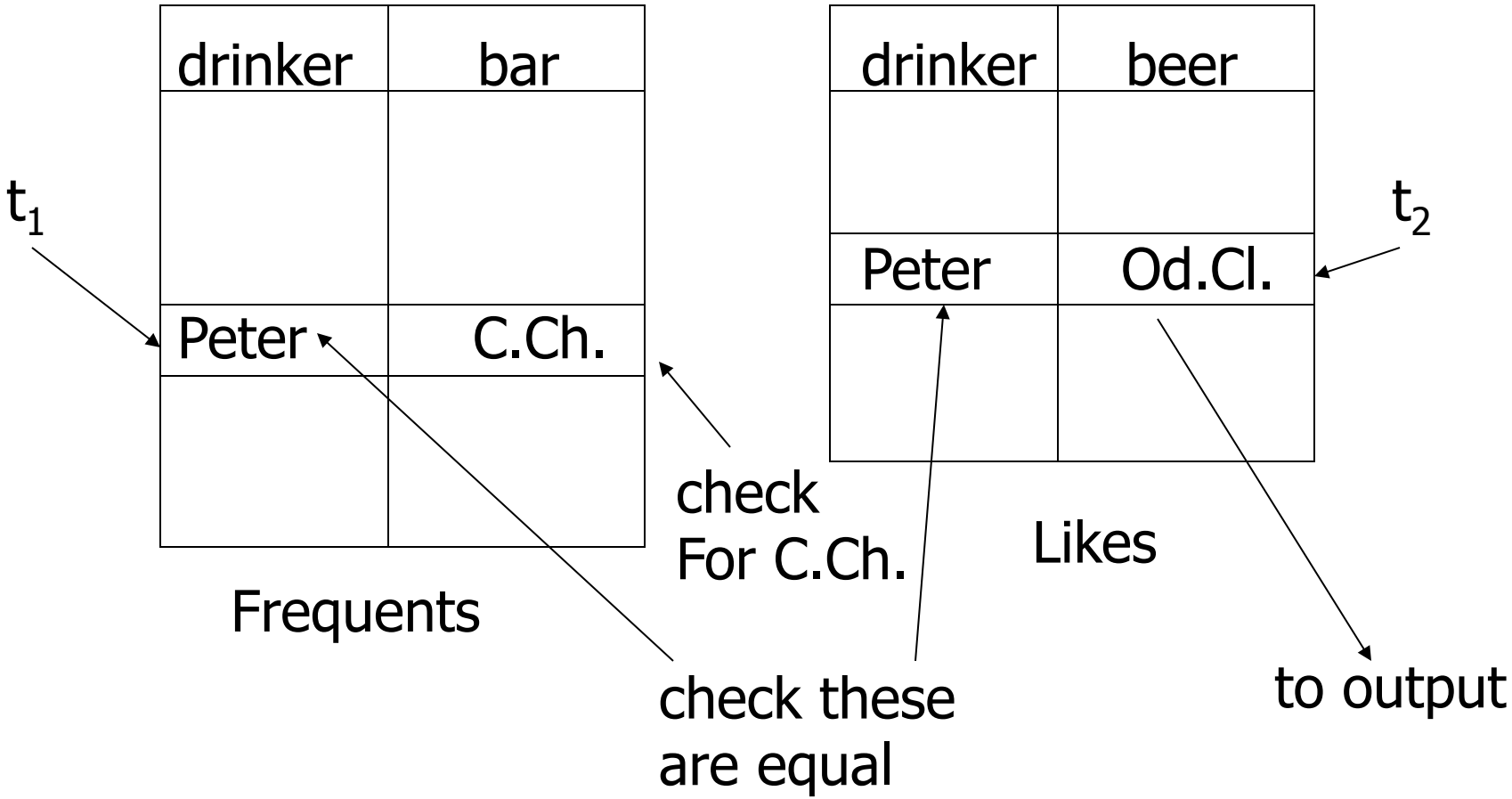
Formal Semantics

- 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
drinker
beer
Peter
Od.Cl.
Likes
t_1
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)

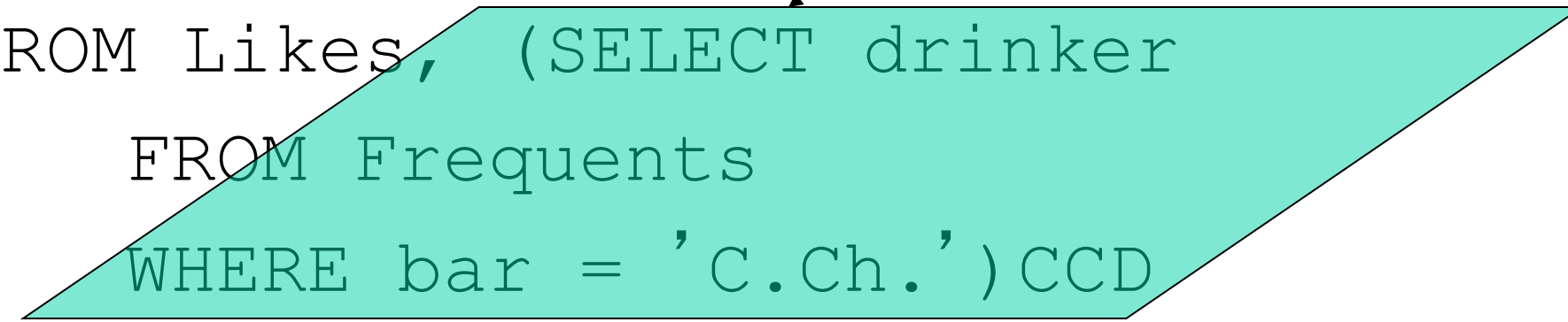
```
SELECT b1.name, b2.name
FROM Beers b1, Beers b2
WHERE b1.manf = b2.manf AND
   b1.name < b2.name;
```

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




```
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

Example: Single-Tuple Subquery

- Using Sells(bar, beer, price), find the bars that serve Pilsener for the same price Cafe Chino charges for Od.Cl.
- Two queries would surely work:
  1. Find the price Cafe Chino charges for Od.Cl.
  2. Find the bars that serve Pilsener at that price

Query + Subquery Solution

```
SELECT bar
FROM Sells
WHERE beer = 'Pilsener' AND
    price = (SELECT price
             FROM Sells
             WHERE bar = 'Cafe Chino'
               AND beer = 'Od.Cl.');
```

The price at Which C.Ch. sells Od.Cl.

The IN Operator

- <tuple> IN (<subquery>) is true if and only if the tuple is a member of the relation produced by the subquery
  - Opposite: <tuple> NOT IN (<subquery>)
- IN-expressions can appear in WHERE clauses

Example: IN

- Using Beers(name, manf) and Likes(drinker, beer), find the name and manufacturer of each beer that Peter likes

```
SELECT *
FROM Beers
WHERE name IN (SELECT beer
               FROM Likes
               WHERE drinker =
'Peter');
```

The set of Beers Peter likes

What is the difference?

```
R(a,b); S(b,c)

SELECT a
FROM R, S
WHERE R.b = S.b;

SELECT a
FROM R
WHERE b IN (SELECT b FROM S);
```

IN is a Predicate About R's Tuples

```
SELECT a
FROM R
WHERE b IN (SELECT b FROM S);
```

Two 2's

One loop, over the tuples of R

a	b
1	2
3	4

R

b	c
2	5
2	6

S

(1,2) satisfies the condition; 1 is output once

This Query Pairs Tuples from R, S

```
SELECT a
FROM R, S
WHERE R.b = S.b;
```

Double loop, over the tuples of R and S

a	b
1	2
3	4

R

b	c
2	5
2	6

S

(1,2) with (2,5) and (1,2) with (2,6) both satisfy the condition;
1 is output twice

The Exists Operator

- EXISTS(<subquery>) is true if and only if the subquery result is not empty
- Example: From Beers(name, manf), find those beers that are the unique beer by their manufacturer

Example: EXISTS

```
SELECT name
FROM Beers b1
WHERE NOT EXISTS (
    SELECT *
    FROM Beers
    WHERE manf = b1.manf AND
          name <> b1.name);
```

Notice scope rule: manf refers to closest nested FROM with a relation having that attribute

Set of beers with the same manf as b1, but not the same beer

Notice the SQL "not equals" operator

The Operator ANY

- *x* = ANY(<subquery>) is a boolean condition that is true iff *x* equals at least one tuple in the subquery result
  - = could be any comparison operator.
- Example: *x* >= ANY(<subquery>) means *x* is not the uniquely smallest tuple produced by the subquery
  - Note tuples must have one component only

The Operator ALL

- *x* <> ALL(<subquery>) is true iff for every tuple *t* in the relation, *x* is not equal to *t*
  - That is, *x* is not in the subquery result
- <> can be any comparison operator
- Example: *x* >= ALL(<subquery>) means there is no tuple larger than *x* in the subquery result