Introduction to SQL

Select-From-Where Statements

Meaning of queries

Subqueries

Why SQL?

- SQL is a very-high-level language, in which the programmer is able to avoid specifying a lot of data-manipulation details that would be necessary in languages like C++.
- What makes SQL viable is that its queries are “optimized” quite well, yielding efficient query executions.

Select-From-Where Statements

- The principal form of a query is:

  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 Anheuser-Busch?

```
SELECT name
FROM Beers
WHERE manf = ‘Anheuser-Busch’;
```

Result of Query

name
‘Bud’
‘Bud Lite’
‘Michelob’

The answer is a relation with a single attribu
name, and tuples with the name of each bee
by Anheuser-Busch, such as Bud.

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

- To implement this algorithm think of a *tuple variable* ranging over 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 = 'Anheuser-Busch';
```

Result of Query:

name	manf
'Bud'	'Anheuser-Busch'
'Bud Lite'	'Anheuser-Busch'
'Michelob'	'Anheuser-Busch'

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 based on Beers(name, manf):

```
SELECT name AS beer, manf
FROM Beers
WHERE manf = 'Anheuser-Busch'
```

Result of Query:

beer	manf
‘Bud’	‘Anheuser-Busch’
‘Bud Lite’	‘Anheuser-Busch’
‘Michelob’	‘Anheuser-Busch’

Expressions in SELECT Clauses

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

```
SELECT bar, beer,
       price * 120 AS priceInYen
FROM Sells;
```

Result of Query

bar	beer	priceInYen
Joe’s	Bud	300
Sue’s	Miller	360
…	…	…

Another Example: Constant Expressions

- From Likes(drinker, beer):

```
SELECT drinker,
       ‘likes Bud’ AS whoLikesBud
FROM Likes
WHERE beer = ‘Bud’;
```

Result of Query

drinker	whoLikesBud
‘Sally’	‘likes Bud’
‘Fred’	‘likes Bud’
…	…

Complex Conditions in WHERE Clause

- From Sells(bar, beer, price), find the price Joe’s Bar charges for Bud:

```
SELECT price
FROM Sells
WHERE bar = ‘Joe’’s Bar’ AND
      beer = ‘Bud’;
```

Important Points

- Two single quotes inside a string represent the single-quote (apostrophe).
- Conditions in the WHERE clause can use AND, OR, NOT, and parentheses in the usual way boolean conditions are built.
- SQL is *case-insensitive*. In general, upper and lower case characters are the same, except inside quoted strings.

Patterns

- WHERE clauses can have conditions in which a string is compared with a pattern, to see if it matches.
- General form: <Attribute> LIKE <pattern> or <Attribute> NOT LIKE <pattern>
- Pattern is a quoted string with % = "any string"; _ = "any character."

Example

- From Drinkers(name, addr, phone) find the drinkers with exchange 555:

```
SELECT name
FROM Drinkers
WHERE phone LIKE ‘%555-_ _ _ _’;
```

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 Joe's Bar 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.
- When any value is compared with NULL, the truth value is UNKNOWN.
- But a query only produces a tuple in the answer if its truth value for 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
Joe’s Bar	Bud	NULL

```
SELECT bar
FROM Sells
WHERE price < 2.00 OR price >= 2.00;
```

$$\underbrace{\underbrace{\text{price < 2.00}}_{\text{UNKNOWN}} \text{ OR } \underbrace{\text{price >= 2.00}}_{\text{UNKNOWN}}}_{\text{UNKNOWN}}$$

Reason: 2-Valued Laws != 3-Valued Laws

- Some common laws, like the commutativity of AND, hold in 3-valued logic.
- But others do not; example: the "law of 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

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

```
SELECT beer
FROM Likes, Frequents
WHERE bar = ‘Joe’’s Bar’ AND
   Frequents.drinker = Likes.drinker;
```

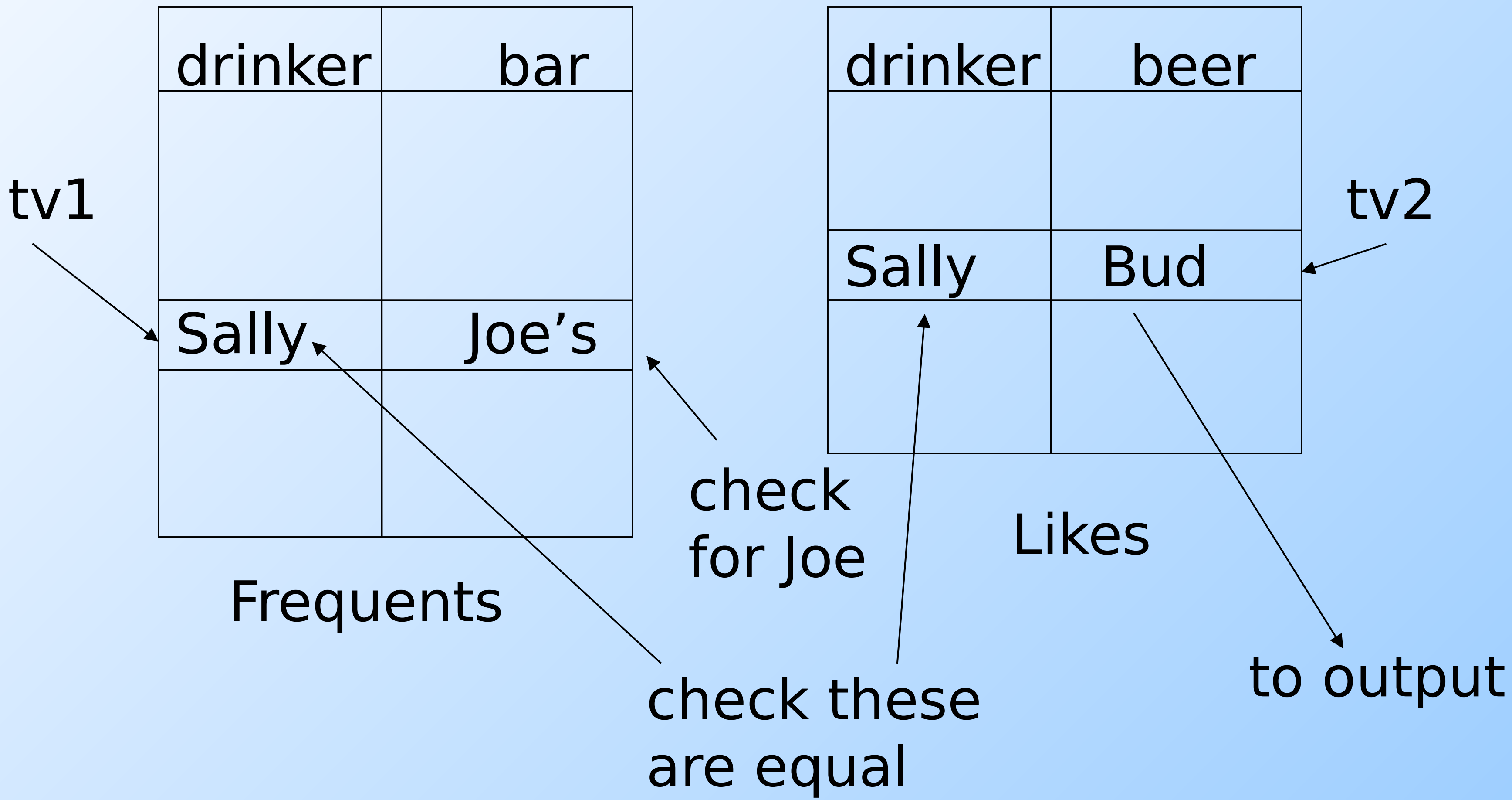
Formal Semantics

- 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
Sally
Joe's
Frequents
tv1
check for Joe
drinker
beer
Sally
Bud
Likes
tv2
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

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

```
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 place another query, and then query its result.
  - Better use a tuple-variable to name tuples of the result.

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.
  - Also typically, a single tuple is guaranteed by keyness of attributes.
  - A run-time error occurs if there is no tuple or more than one tuple.

Example

- From Sells(<u>bar</u>, <u>beer</u>, price), find the bars that serve Miller for the same price Joe charges for Bud.
- Two queries would surely work:
  1. Find the price Joe charges for Bud.
  2. Find the bars that serve Miller at that price.

Query + Subquery Solution

```
SELECT bar
FROM Sells
WHERE beer = ‘Miller’ AND
    price = (SELECT price
             FROM Sells
             WHERE bar = ‘Joe’’s Bar’
               AND beer = ‘Bud’);
```

The price at which Joe sells Bud

The IN Operator

- <tuple> IN <relation> is true if and only if the tuple is a member of the relation.
  - <tuple> NOT IN <relation> means the opposite.
- IN-expressions can appear in WHERE clauses.
- The <relation> is often a subquery.

Example

- From Beers(name, manf) and Likes(drinker, beer), find the name and manufacturer of each beer that Fred likes.

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

The set of beers Fred likes

The Exists Operator

- EXISTS(<relation>) is true if and only if the <relation> is not empty.
- Being a boolean-valued operator, EXISTS can appear in WHERE clauses.
- Example: From Beers(name, manf), find those beers that are the unique beer by their manufacturer.

Example Query with EXISTS

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

Notice scope rule: manf refe
to closest nested FROM with
a relation having that attribu

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(<relation>) is a boolean condition meaning that *x* equals at least one tuple in the relation.
- Similarly, = can be replaced by any of the comparison operators.
- Example: *x* >= ANY(<relation>) means *x* is not smaller than all tuples in the relation.
  - Note tuples must have one component only.

The Operator ALL

- Similarly, *x* <> ALL(<relation>) is true if and only if for every tuple *t* in the relation, *x* is not equal to *t*.
  - That is, *x* is not a member of the relation.
- The <> can be replaced by any comparison operator.
- Example: *x* >= ALL(<relation>) means there is no tuple larger than *x* in the relation.

Example

- From Sells(bar, beer, price), find the beer(s) sold for the highest price.

```
SELECT beer
FROM Sells
WHERE price >= ALL(
    SELECT price
    FROM Sells);
```

price from the outer Sells must not be less than any price.