Patterns

22

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

22

Example: LIKE

23

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

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

23

NULL Values

24

- 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.

24

Comparing NULL's to Values

25

- 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).

25

Three-Valued Logic

26

- To understand how AND, OR, and NOT work in 3-valued logic
- For TRUE result
  - OR: at least one operand must be TRUE
  - AND: both operands must be TRUE
  - NOT: operand must be FALSE
- For FALSE result
  - OR: both operands must be FALSE
  - AND: at least one operand must be FALSE
  - NOT: operand must be TRUE
- Otherwise, result is UNKNOWN

26

Example

27

- From the following Sells relation:

bar	beer	price
Joe's Bar	Bud	NULL

SELECT bar

FROM Sells

WHERE price < 2.00 OR price >= 5.00;

UNKNOWN UNKNOWN

UNKNOWN

27

Multi-Relation Queries

28

- 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>" .

28

Example: Joining Two Relations

29

- 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;
```

29

Example: Joining Two Relations

30

- Alternatively can use explicit (named) tuple variables

```
SELECT beer
 FROM Likes l, Frequents f
 WHERE bar = 'Joe''s Bar' AND
    f.drinker = l.drinker;
```

30

Formal Semantics

31

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

31

Operational Semantics

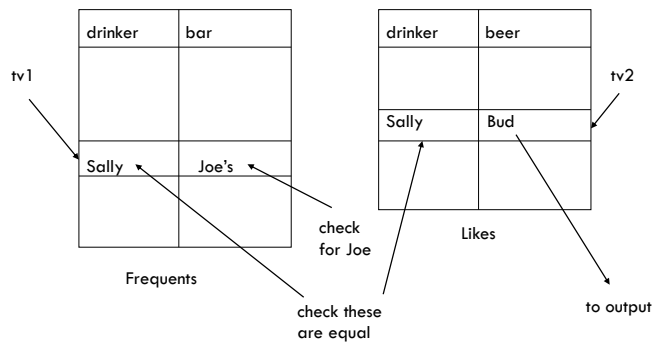
32

- 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.

32

Example

33



33

Explicit Tuple-Variables

34

- 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.

34

Example: Self-Join

35

- From Beers(name, manf), find all pairs of beers by the same manufacturer.
  - Do not produce pairs like (Bud, Bud).
  - Do not produce the same pair twice like (Bud, Miller) and (Miller, Bud).

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

35

Subqueries

36

- 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.

36

Example: Subquery in FROM

37

- Find the beers liked by at least one person who frequents Joe's Bar.

Drinkers who frequent Joe's Bar

```
SELECT beer
FROM Likes, (SELECT drinker
             FROM Frequents
             WHERE bar = 'Joe''s Bar') JD
WHERE Likes.drinker = JD.drinker;
```

37

Subqueries often obscure queries

38

- Find the beers liked by at least one person who frequents Joe's Bar.

```
SELECT beer
FROM Likes l, Frequents f
WHERE l.drinker = f.drinker AND
   bar = 'Joe''s Bar';
```

Simple join query

38

Subqueries That Return One Tuple

39

- If a subquery is guaranteed to produce one tuple, then the subquery can be used as a value.
  - Usually, the tuple has one component.
  - Remember SQL's 3-valued logic.

39

Example: Single-Tuple Subquery

40

- Using Sells(bar, beer, price), find the bars that serve Miller for the same price Joe charges for Bud.

Two queries would work:

- Find the price Joe charges for Bud.
- Find the bars that serve Miller at that price.

40

Query + Subquery Solution

41

- Find the price Joe charges for Bud.
- Find the bars that serve Miller at that price.

Sells(bar, beer, price)

```
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

What if price of Bud is NULL?

41

Query + Subquery Solution

42

```
SELECT bar
FROM Sells
WHERE beer = 'Miller' AND
    price = (SELECT price
             FROM Sells
             WHERE beer = 'Bud');
```

What if subquery returns multiple values?

42