COMP 330: SQL 3

Chris Jermaine and Kia Teymourian

Rice University

HAVING

RATES (DRINKER, BEER, SCORE)

Example: What is the highest rated beer, on average, considering only beers that have at least 10 ratings?

HAVING

RATES (DRINKER, BEER, SCORE)

Example: What is the highest rated beer, on average, considering only beers that have at least 10 ratings?

- Change AGG to:

```
CREATE VIEW AGG AS
SELECT r.BEER, AVERAGE (r.SCORE) AS AVG_RATING
FROM RATES r
GROUP BY BEER
HAVING COUNT (*) >= 10
```

Relational Algebra in FROM

LIKES (DRINKER, BEER)

FREQUENTS (DRINKER, BAR)

SERVES (BAR, BEER)

Example: Who has gone to a bar serving 'Bud', but does not like 'PBR'?

Relational Algebra in FROM

LIKES (DRINKER, BEER)

FREQUENTS (DRINKER, BAR)

SERVES (BAR, BEER)

Example: Who has gone to a bar serving 'Bud', but does not like 'PBR'?

```
SELECT f.DRINKER
FROM FREQUENTS f JOIN SERVES s ON f.DRINKER = s.DRINKER
WHERE s.BEER = 'Bud' AND NOT EXISTS (
    SELECT *
    FROM LIKES l
    WHERE l.BEER = 'PBR' AND l.DRINKER = f.DRINKER)
```

Relational Algebra in FROM

In FROM, we allow joins of the form:

```
TABLE1 t1 JOIN TABLE2 t2 ON pred
TABLE1 t1 INNER JOIN TABLE2 t2 ON pred
TABLE1 t1 CROSS JOIN TABLE2 t2
TABLE1 t1 LEFT OUTER JOIN TABLE2 t2 ON pred
TABLE1 t1 RIGHT OUTER JOIN TABLE2 t2 ON pred
TABLE1 t1 FULL OUTER JOIN TABLE2 t2 ON pred
```

Relational Algebra in FROM

In FROM, we allow joins of the form:

```
TABLE1 t1 JOIN TABLE2 t2 ON pred
TABLE1 t1 INNER JOIN TABLE2 t2 ON pred
```

- These are exactly the same, just a good, old-fashioned join

Relational Algebra in FROM

In FROM, we allow joins of the form:

```
TABLE1 t1 CROSS JOIN TABLE2 t2
```

- has the obvious meaning: do a cross product

Relational Algebra in FROM

In FROM, we allow joins of the form:

```
TABLE1 t1 LEFT OUTER JOIN TABLE2 t2 ON pred
TABLE1 t1 RIGHT OUTER JOIN TABLE2 t2 ON pred
TABLE1 t1 FULL OUTER JOIN TABLE2 t2 ON pred
```

What is an outer join?

Outer Joins

LIKES (DRINKER, BEER)

RATES (DRINKER, BEER, SCORE)

Ex: for each drinker, give rating for 'PBR' and for 'SSTP'

Outer Joins

LIKES (DRINKER, BEER)

RATES (DRINKER, BEER, SCORE)

Ex: for each drinker, give rating for 'PBR' and for 'SSTP'

```
SELECT r1.DRINKER,
  'PBR rating: ' + CAST (r1.SCORE AS VARCHAR (30)),
  'SSTP rating: ' + CAST (r2.SCORE AS VARCHAR(30))
FROM RATES r1, RATES r2
WHERE r1.DRINKER = r2.DRINKER AND
  r1.BEER = 'PBR' AND R2.BEER = 'SSTP'
```

- What's the problem here?
- What if someone fails to rate either beer?
- Use an outer join instead!

Outer Joins

LIKES (DRINKER, BEER)

RATES (DRINKER, BEER, SCORE)

Ex: for each drinker, give rating for 'PBR' and for 'SSTP'

```
SELECT r1.DRINKER,
  'PBR rating: ' + CAST (r1.SCORE AS VARCHAR (30)),
  'SSTP rating: ' + CAST (r2.SCORE AS VARCHAR(30))
FROM LIKES l
  LEFT OUTER JOIN RATES r1 ON l.DRINKER = r1.DRINKER
  LEFT OUTER JOIN RATES r2 ON l.DRINKER = r2.DRINKER
WHERE r1.BEER = 'PBR' AND R2.BEER = 'SSTP'
```

- What's another problem here?
- Outer join pads with NULL values

Outer Joins

LIKES (DRINKER, BEER)

RATES (DRINKER, BEER, SCORE)

Ex: for each drinker, give rating for 'PBR' and for 'SSTP'

```
SELECT r1.DRINKER,
  'PBR␣rating:␣' + CAST (r1.SCORE AS VARCHAR (30)),
  'SSTP␣rating:␣' + CAST (r2.SCORE AS VARCHAR(30))
FROM LIKES l
  LEFT OUTER JOIN RATES r1 ON l.DRINKER = r1.DRINKER
  LEFT OUTER JOIN RATES r2 ON l.DRINKER = r2.DRINKER
WHERE r1.BEER = 'PBR' AND R2.BEER = 'SSTP'
```

- What's another problem here?
- Outer join pads with NULL values
- Instead:

Outer Joins

LIKES (DRINKER, BEER)

RATES (DRINKER, BEER, SCORE)

Ex: for each drinker, give rating for 'PBR' and for 'SSTP'

```
SELECT r1.DRINKER,
  'PBR rating: ' +
     ISNULL (CAST (r1.SCORE AS VARCHAR (30)), 'unknown'),
  'SSTP rating: ' +
    ISNULL (CAST (r2.SCORE AS VARCHAR(30)), 'unknown')
FROM LIKES l
  LEFT OUTER JOIN RATES r1 ON l.DRINKER = r1.DRINKER
  LEFT OUTER JOIN RATES r2 ON l.DRINKER = r2.DRINKER
WHERE r1.BEER = 'PBR' AND R2.BEER = 'SSTP'
```

NULL Values

In SQL, every attribute type can take the value NULL

- NULL is a special value
- Used to signal a missing value
- Nearly all non-comparison ops taking NULL as input return NULL

Common SQL code used to handle NULL

```
SELECT ISNULL (exp, altexp)...
WHERE exp IS NULL...
```

Unknown Values

SQL actually uses a 3-value logic

- Values are true, false, unknown
- Truth tables generally make sense
- Ex: true and unknown gives unknown
- Ex: true or unknown gives true

Any comparison with NULL returns unknown

- For a WHERE to accept the tuple, must get a true

A bit on the DDL

Creating tables

```
CREATE TABLE RATES (
  DRINKER VARCHAR (30),
  BEER VARCHAR (30),
  SCORE INTEGER
}
```

Are many types!

- Do a Google search: tsql data types

Defining a Primary Key

```
CREATE TABLE RATES (
  DRINKER VARCHAR (30),
  BEER VARCHAR (30),
  SCORE INTEGER,
  PRIMARY KEY (DRINKER, BEER)
}
```

What about:

```
  UNIQUE (DRINKER, BEER)
```

Defining a Primary Key

Can also use:

```
CREATE TABLE RATES (
  DRINKER VARCHAR (30),
  BEER VARCHAR (30),
  SCORE INTEGER
}

ALTER TABLE RATES ADD CONSTRAINT PK
  PRIMARY KEY (DRINKER, BEER)
```

Why do it this way?

Defining a Foreign Key

```
CREATE TABLE RATES (
  DRINKER VARCHAR (30),
  BEER VARCHAR (30),
  SCORE INTEGER
}

ALTER TABLE RATES ADD CONSTRAINT FK
  FOREIGN KEY (DRINKER, BEER)
  REFERENCES LIKES (DRINKER, BEER)
```

Adding Data

```
INSERT INTO RATES VALUES ('Chris', 'SSTP', 10);
INSERT INTO RATES (BEER, DRINKER) VALUES ('SSTP', 'Chris');
```

- What happens to SCORE in the second case?

Adding Data

Data to add can be the result of a query

- Ex: Create a tuple giving Chris a NULL rating for each beer he's not actually rated.

Adding Data

Data to add can be the result of a query

- Create a tuple giving Chris a NULL rating for each beer he's not actually rated.

```
INSERT INTO RATES (BEER, DRINKER)
  SELECT l.BEER, 'Chris'
  FROM LIKES l
  WHERE NOT EXISTS (
    SELECT *
    FROM RATES r
    WHERE r.BEER = l.BEER AND r.DRINKER = 'Chris')
```

Deleting Data

Ex: delete all of the ratings with a NULL score, or one less than 1 or greater than 10.

Deleting Data

Ex: delete all of the ratings with a NULL score, or one less than 1 or greater than 10.

```
DELETE FROM RATES r
WHERE r.SCORE IS NULL OR r.SCORE NOT BETWEEN 1 AND 10
```

Modifying Data

Ex: Change every score that's bad (less than 1 or greater than 10) to NULL

Modifying Data

Ex: Change every score that's bad (less than 1 or greater than 10) to NULL

```
UPDATE RATES r
SET r.SCORE = NULL
WHERE r.SCORE NOT BETWEEN 1 AND 10
```

Questions?