COMP 543: Tools & Models for Data Science

SQL 3

Chris Jermaine & Risa Myers

Rice University

Relational Algebra in `FROM`

LIKES (DRINKER, COFFEE)
FREQUENTS (DRINKER, CAFE)
SERVES (CAFE, COFFEE)

? Who has gone to a cafe serving ‘Cold Brew’, but does not like ‘Chai Latte’?

Relational Algebra in FROM

LIKES (DRINKER, COFFEE)
FREQUENTS (DRINKER, CAFE)
SERVES (CAFE, COFFEE)

- Who has gone to a cafe serving ‘Cold Brew’, but does not like ‘Chai Latte’?

```
SELECT f.DRINKER
FROM FREQUENTS f
   JOIN SERVES s ON f.CAFE = s.CAFE
WHERE s.COFFEE = 'Cold␣Brew' AND NOT EXISTS (
   SELECT *
   FROM LIKES l
   WHERE l.COFFEE = 'Chai␣Latte' AND l.DRINKER = f.DRINKER)
```

Relational Algebra in FROM

LIKES (DRINKER, COFFEE)
FREQUENTS (DRINKER, CAFE)
SERVES (CAFE, COFFEE)

- Who has gone to a cafe serving ‘Cold Brew’, but does not like ‘Chai Latte’?

```
SELECT f.DRINKER
FROM FREQUENTS f
   JOIN SERVES s ON f.CAFE = s.CAFE
WHERE s.COFFEE = 'Cold␣Brew' AND NOT EXISTS (
   SELECT *
   FROM LIKES l
   WHERE l.COFFEE = 'Chai␣Latte' AND l.DRINKER = f.DRINKER)
```

$$\pi_{f.DRINKER}\left(\sigma_{s.COFFEE='CB'}\left(\text{FREQUENTS} * \text{SERVES}\right)\right)$$
$$-\left(\pi_{l.DRINKER}\left(\sigma_{s.COFFEE='ChaiLatte'}\left(\text{LIKES}\right)\right)\right)$$

Relational Algebra in FROM

- In the FROM clause, 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 NATURAL JOIN TABLE2 t2
TABLE1 t1 FULL OUTER JOIN TABLE2 t2 ON pred
```

Relational Algebra in `FROM`

- In the `FROM` clause, 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 the `FROM` clause, 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, COFFEE)
RATES (DRINKER, COFFEE, SCORE)

? For each drinker, give the rating for ‘Cold Brew’ and for ‘Chai Latte’

Outer Joins

LIKES (DRINKER, COFFEE)
RATES (DRINKER, COFFEE, SCORE)

- For each drinker, give the rating for ‘Cold Brew’ and for ‘Chai Latte’

```
SELECT r1.DRINKER,
  CONCAT('Cold␣Brew␣rating:␣', CAST(r1.SCORE AS CHAR),
  '␣Chai␣Latte␣rating:␣',
CAST(r2.SCORE AS CHAR))
FROM RATES r1, RATES r2
WHERE r1.DRINKER = r2.DRINKER
        AND r1.COFFEE = 'Cold␣Brew'
        AND r2.COFFEE = 'Chai␣Latte'
```

? What’s the problem here?

Outer Joins

LIKES (DRINKER, COFFEE)
RATES (DRINKER, COFFEE, SCORE)

- For each drinker, give the rating for ‘Cold Brew’ and for ‘Chai Latte’

```
SELECT r1.DRINKER,
  CONCAT('Cold␣Brew␣rating:␣', CAST(r1.SCORE AS CHAR),
  '␣Chai␣Latte␣rating:␣',
CAST(r2.SCORE AS CHAR)) AS ratings
FROM RATES r1, RATES r2
WHERE r1.DRINKER = r2.DRINKER
        AND r1.COFFEE = 'Cold␣Brew'
        AND r2.COFFEE = 'Chai␣Latte'
```

- What’s the problem here?
  - What if someone fails to rate either coffee?

Outer Joins

LIKES (DRINKER, COFFEE)
RATES (DRINKER, COFFEE, SCORE)

- For each drinker, give the rating for ‘Cold Brew’ and for ‘Chai Latte’

```
SELECT r1.DRINKER,
  CONCAT('Cold␣Brew␣rating:␣', CAST(r1.SCORE AS CHAR),
  '␣Chai␣Latte␣rating:␣',
CAST(r2.SCORE AS CHAR))
AS ratings
FROM RATES r1, RATES r2
WHERE r1.DRINKER = r2.DRINKER
        AND r1.COFFEE = 'Cold␣Brew'
        AND r2.COFFEE = 'Chai␣Latte'
```

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

Outer Joins

LIKES (DRINKER, COFFEE)
RATES (DRINKER, COFFEE, SCORE)

- For each drinker, give the rating for ‘Cold Brew’ and for ‘Chai Latte’

```
SELECT r1.DRINKER,
  CONCAT('Cold␣Brew␣rating:␣', CAST(r1.SCORE AS CHAR),
  '␣Chai␣Latte␣rating:␣',
CAST(r2.SCORE AS CHAR))
AS ratings
FROM LIKES l
  LEFT OUTER JOIN RATES r1 ON l.DRINKER = r1.DRINKER
    AND r1.COFFEE = 'Cold␣Brew'
  LEFT OUTER JOIN RATES r2 ON l.DRINKER = r2.DRINKER
    AND r2.COFFEE = 'Chai␣Latte'
```

? What’s another problem here?

Outer Joins

- For each drinker, give the rating for ‘Cold Brew’ and for ‘Chai Latte’

```
SELECT r1.DRINKER,
  CONCAT('Cold␣Brew␣rating:␣', CAST(r1.SCORE AS CHAR),
  '␣Chai␣Latte␣rating:␣',
CAST(r2.SCORE AS CHAR))
AS ratings
FROM LIKES l
  LEFT OUTER JOIN RATES r1 ON l.DRINKER = r1.DRINKER
    AND r1.COFFEE = 'Cold␣Brew'
  LEFT OUTER JOIN RATES r2 ON l.DRINKER = r2.DRINKER
    AND r2.COFFEE = 'Chai␣Latte'
```

- What’s another problem here?
- Outer join pads with NULL values
- We can see duplicate rows when there are NULL values

Outer Joins

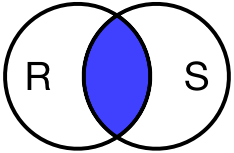
LIKES (DRINKER, COFFEE)
RATES (DRINKER, COFFEE, SCORE)

- Ex: for each drinker, give rating for ‘Cold Brew’ and for ‘Chai Latte’

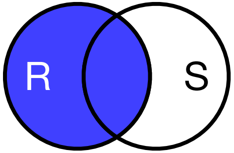
```
SELECT r1.DRINKER,
  CONCAT('Cold␣Brew␣rating:␣' ,
  COALESCE(CAST(r1.SCORE AS CHAR), 'unknown'),
  '␣Chai␣Latte␣rating:␣',
    COALESCE(CAST(r2.SCORE AS CHAR), 'unknown')) AS ratings
FROM LIKES l
  LEFT OUTER JOIN RATES r1 ON l.DRINKER = r1.DRINKER
    AND r1.COFFEE = 'Cold␣Brew'
  LEFT OUTER JOIN RATES r2 ON l.DRINKER = r2.DRINKER
    AND r2.COFFEE = 'Chai␣Latte'
```

Joins

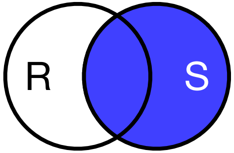
Inner/Theta / Natural Join



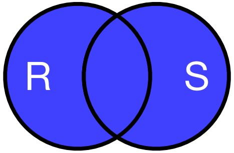
Left Outer Join



Right Outer Join



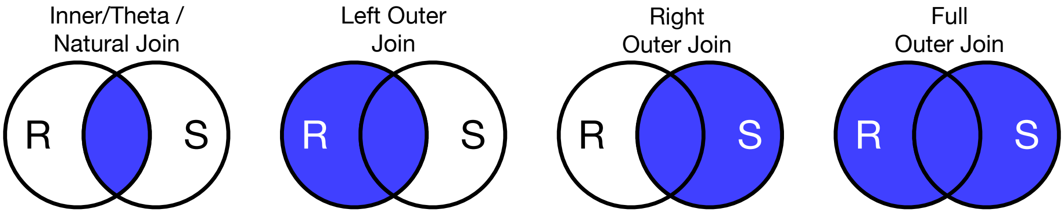
Full Outer Join



- Relation R has set of attributes A
- Relation S has set of attributes B

? What attributes are in the resulting relations?

Joins



- Relation R has set of attributes A
- Relation S has set of attributes B
- What attributes are in the resulting relations?
  - $R_A \bullet S_B$
  - ? Are there any exceptions?

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 COALESCE (exp, altexp1, altexp2)...
```

or

```
WHERE exp IS NULL...
WHERE exp IS NOT 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	B	A AND B
true	unknown	unknown
false	unknown	false
unknown	unknown	unknown

A	B	A OR B
true	unknown	true
false	unknown	unknown
unknown	unknown	unknown

A bit on the DDL

- Creating tables

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

- There are many data types!
  - Do an Internet search: ‘Postgres data types’

Defining a Primary Key

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

? What about

```
  UNIQUE (DRINKER, COFFEE)
```

Defining a Primary Key

- Can also use:

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

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

? Why do it this way?

Defining a Foreign Key

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

ALTER TABLE RATES ADD CONSTRAINT LIKES_FK
  FOREIGN KEY (DRINKER, COFFEE)
  REFERENCES LIKES (DRINKER, COFFEE)
```

Adding Data

```
INSERT INTO RATES VALUES ('Risa', 'Eiskaffe', 5);
INSERT INTO RATES (COFFEE, DRINKER) VALUES ('Eiskaffe', 'Risa');
```

? What happens to SCORE in the second case?

? Why might you prefer the second case?

Adding Data

- Data to add can be the result of a query

? Create a tuple giving Risa a NULL rating for each coffee she’s not yet rated.

Adding Data

- Data to add can be the result of a query
- Create a tuple giving Risa a NULL rating for each coffee she's not yet rated.

```
INSERT INTO RATES (COFFEE, DRINKER)
  SELECT DISTINCT s.COFFEE, 'Risa'
  FROM SERVES s
  WHERE NOT EXISTS (
    SELECT *
    FROM RATES r
    WHERE r.COFFEE = s.COFFEE AND r.DRINKER = 'Risa')
```

  - ? Why 'Risa' in the first SELECT?
  - ? Why DISTINCT?

Deleting Data

? Delete all of the ratings with a score that is NULL, or less than 1 or greater than 5

Deleting Data

- Delete all of the ratings with a NULL score, or less than 1 or greater than 5

```
DELETE FROM RATES
WHERE SCORE IS NULL OR SCORE NOT BETWEEN 1 AND 5
```

Modifying Data

? Change every score that's bad (less than 1 or greater than 5) to NULL

Modifying Data

- Change every score that's bad (less than 1 or greater than 5) to NULL

```
UPDATE RATES r
SET SCORE = NULL
WHERE r.SCORE NOT BETWEEN 1 AND 5
```

Questions?