COMP 330/543: SQL 3

Luis Guzman

Sinan Kockara

Chris Jermaine

Rice University

HAVING

HAVING allows us to check for conditions on aggregate functions

```
SELECT column_name(s)
FROM table_name
WHERE condition
GROUP BY column_name(s)
HAVING condition
ORDER BY column_name(s)
```

HAVING

RATES (DRINKER, BEER, SCORE)

Q: What is the highest-rated beer, on average, considering only beers that have at least 3 ratings?

HAVING

RATES (DRINKER, BEER, SCORE)

Q: What is the highest-rated beer, on average, considering only beers that have at least 3 ratings?

- From previous lecture:

```
CREATE VIEW AVG_RATES AS
SELECT r.BEER, AVERAGE (r.SCORE) AS AVG_RATING
FROM RATES r
GROUP BY r.BEER

SELECT a.BEER
FROM AVG_RATES a
WHERE a.AVG_RATING = (SELECT MAX (b.AVG_RATING)
                      FROM AVG_RATES b)
```

HAVING

RATES (DRINKER, BEER, SCORE)

Q: What is the highest-rated beer, on average, considering only beers that have at least 3 ratings?

▷ Change AGG to:

```
CREATE VIEW AVG_RATES AS
SELECT r.BEER, AVERAGE (r.SCORE) AS AVG_RATING
FROM RATES r
GROUP BY r.BEER
HAVING COUNT (*) >= 3
```

HAVING

```
CREATE VIEW AVG_RATES AS
SELECT r.BEER, AVERAGE (r.SCORE) AS AVG_RATING
FROM RATES r
GROUP BY r.BEER
HAVING COUNT (*) >= 3
```

```
(’Sinan’, ’PBR’, 4)
(’Chris’, ’PBR’, 3)
(’Chris’, ’SSTP’, 10)
(’Luis’, ’PBR’, 8)
(’Luis’, ’Modelo’, 9)
(’Sinan’, ’Modelo’, 6)
```

HAVING

```
CREATE VIEW AVG_RATES AS
SELECT r.BEER, AVERAGE (r.SCORE) AS AVG_RATING
FROM RATES r
GROUP BY r.BEER
HAVING COUNT (*) >= 3
```

1. Group by r.BEER

```
('Sinan', 'PBR', 4)
('Chris', 'PBR', 3)
('Luis', 'PBR', 8)

('Chris', 'SSTP', 10)

('Luis', 'Modelo', 9)
('Sinan', 'Modelo', 6)
```

HAVING

```
CREATE VIEW AVG_RATES AS
SELECT r.BEER, AVERAGE (r.SCORE) AS AVG_RATING
FROM RATES r
GROUP BY r.BEER
HAVING COUNT (*) >= 3
```

2. Having `COUNT(*) >= 3`

```
(’Sinan’, ’PBR’, 4)
(’Chris’, ’PBR’, 3)
(’Luis’, ’PBR’, 8)
```

3. Final output

```
(’PBR’, 5)
```

Joins in FROM

LIKES (DRINKER, BEER)

FREQUENTS (DRINKER, BAR)

SERVES (BAR, BEER)

Q: Who has gone to a bar serving ‘Bud’, but does not like ‘PBR’?

Joins in FROM

LIKES (DRINKER, BEER)

FREQUENTS (DRINKER, BAR)

SERVES (BAR, BEER)

Q: Who has gone to a bar serving ‘Bud’, but does not like ‘PBR’?

1. Linking in WHERE clause

```
SELECT f.DRINKER
FROM FREQUENTS f, SERVES s
WHERE
f.BAR = s.BAR AND s.BEER = 'Bud' AND
        NOT EXISTS (current DRINKER likes PBR)
```

Joins in FROM

LIKES (DRINKER, BEER)

FREQUENTS (DRINKER, BAR)

SERVES (BAR, BEER)

Q: Who has gone to a bar serving ‘Bud’, but does not like ‘PBR’?

1. Linking in WHERE clause

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

Joins in FROM

LIKES (DRINKER, BEER)

FREQUENTS (DRINKER, BAR)

SERVES (BAR, BEER)

Q: Who has gone to a bar serving ‘Bud’, but does not like ‘PBR’?

2. Linking in FROM clause with JOIN

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

Joins 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 NATURAL JOIN TABLE2 t2  (Not supported by MS SQL Server)
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
```

Joins 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

What is the difference with a NATURAL JOIN?

```
TABLE1 t1 NATURAL JOIN TABLE2 t2
```

Joins in FROM

LIKES(DRINKER, BEER)

FREQUENTS(DRINKER, BAR)

```
('Sinan', 'SSTP')
('Chris', 'PBR')
('Luis', 'Modelo')

('Sinan', 'Beer␣Garden')
('Chris', 'Valhalla')
('Luis', 'Wild␣Duck')
```

Joins in FROM

Using INNER JOIN

```
(’Sinan’, ’SSTP’)
(’Chris’, ’PBR’)
(’Luis’, ’Modelo’)
```

```
(’Sinan’, ’Beer␣Garden’)
(’Chris’, ’Valhalla’)
(’Luis’, ’Wild␣Duck’)
```

```
SELECT *
FROM LIKES l INNER JOIN FREQUENTS f ON l.DRINKER = f.DRINKER
```

```
(’Sinan’, ’SSTP’, ’Sinan’, ’Beer␣Garden’)
(’Chris’, ’PBR’, ’Chris’, ’Valhalla’)
(’Luis’, ’Modelo’, ’Luis’, ’Wild␣Duck’)
```

Joins in FROM

Using NATURAL JOIN

```
(’Sinan’, ’SSTP’)
(’Chris’, ’PBR’)
(’Luis’, ’Modelo’)
```

```
(’Sinan’, ’Beer␣Garden’)
(’Chris’, ’Valhalla’)
(’Luis’, ’Wild␣Duck’)
```

```
SELECT *
FROM LIKES l NATURAL JOIN FREQUENTS f
```

```
(’Sinan’, ’SSTP’, ’Beer␣Garden’)
(’Chris’, ’PBR’, ’Valhalla’)
(’Luis’, ’Modelo’, ’Wild␣Duck’)
```

Joins in FROM

In FROM, we allow joins of the form:

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

- ▷ Has the obvious meaning: do a cross product

Same as:

```
TABLE1 t1, TABLE2 t2
```

Joins 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?

- Includes all the tuples from the "outer" side
- Assigns NULLs if there is no matching tuple
- Pick one and stick with it!

Outer Joins

LIKES (DRINKER, BEER)

RATES (DRINKER, BEER, SCORE)

Q: for each drinker, give a rating for ‘PBR’ and for ‘SSTP’

```
('Luis', 'Modelo')
('Chris', 'SSTP')
('Sinan', 'Blue␣Moon')

('Luis', 'PBR', 6)
('Luis', 'SSTP', 8)
('Chris', 'SSTP', 10)
('Sinan', 'PBR', 4)
```

Outer Joins

LIKES (DRINKER, BEER)

RATES (DRINKER, BEER, SCORE)

Q: for each drinker, give a 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?

Outer Joins

LIKES (DRINKER, BEER)

RATES (DRINKER, BEER, SCORE)

Q: for each drinker, give a 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

Q: for each drinker, give a rating for ‘PBR’ and for ‘SSTP’

```
(’Luis’, ’Modelo’)
(’Chris’, ’SSTP’)
(’Sinan’, ’Blue␣Moon’)

(’Luis’, ’PBR’, 6)
(’Luis’, ’SSTP’, 8)
(’Chris’, ’SSTP’, 10)
(’Sinan’, ’PBR’, 4)

(’Luis’, ’PBR␣rating:␣6’,’SSTP␣rating:␣8’)
```

Outer Joins

LIKES (DRINKER, BEER)

RATES (DRINKER, BEER, SCORE)

Q: for each drinker, give a rating for ‘PBR’ and for ‘SSTP’

```
SELECT l.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 a problem here?

Outer Joins

Q: for each drinker, give a rating for ‘PBR’ and for ‘SSTP’

```
(’Luis’, ’Modelo’)
(’Chris’, ’SSTP’)
(’Sinan’, ’Blue␣Moon’)

(’Luis’, ’PBR’, 6)
(’Luis’, ’SSTP’, 8)
(’Chris’, ’SSTP’, 10)
(’Sinan’, ’PBR’, 4)
```

After first LEFT JOIN

```
(’Luis’, ’Modelo’, ’Luis’, ’PBR’, 6)
(’Luis’, ’Modelo’, ’Luis’, ’SSTP’, 8)
(’Chris’, ’SSTP’, ’Chris’, ’SSTP’, 10)
(’Sinan’, ’Blue␣Moon’, ’Sinan’, ’PBR’, 4)
```

Outer Joins

```
('Luis', 'PBR', 6)
('Luis', 'SSTP', 8)
('Chris', 'SSTP', 10)
('Sinan', 'PBR', 4)

('Luis', 'Modelo', 'Luis', 'PBR', 6)
('Luis', 'Modelo', 'Luis', 'SSTP', 8)
('Chris', 'SSTP', 'Chris', 'SSTP', 10)
('Sinan', 'Blue␣Moon', 'Sinan', 'PBR', 4)
```

After second LEFT JOIN

```
l.DR,    l.BR,     r1.DR,  r1.BR, r1.S, r2.DR,  r2.BR,  r2.S
('Luis', 'Modelo', 'Luis', 'PBR', 6, 'Luis', 'PBR', 6)
('Luis', 'Modelo', 'Luis', 'SSTP', 8, 'Luis', 'PBR', 6)
('Luis', 'Modelo', 'Luis', 'PBR', 6, 'Luis', 'SSTP', 8)
('Luis', 'Modelo', 'Luis', 'SSTP', 8, 'Luis', 'SSTP', 8)
('Chris', 'SSTP', 'Chris', 'SSTP', 10, 'Chris', 'SSTP', 10)
('Sinan', 'Blue␣Moon', 'Sinan', 'PBR', 4, 'Sinan', 'PBR', 4)
```

Outer Joins

LIKES (DRINKER, BEER)

RATES (DRINKER, BEER, SCORE)

Q: for each drinker, give a rating for ‘PBR’ and for ‘SSTP’

```
SELECT l.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 a problem here?

- We need the outer join to happen AFTER the selection on PBR, SSTP

Outer Joins

LIKES (DRINKER, BEER)

RATES (DRINKER, BEER, SCORE)

Q: for each drinker, give a rating for ‘PBR’ and for ‘SSTP’

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

▷ What’s another problem here?

Outer Joins

Q: for each drinker, give a rating for ‘PBR’ and for ‘SSTP’

```
(’Luis’, ’Modelo’)
(’Chris’, ’SSTP’)
(’Sinan’, ’Blue␣Moon’)
```

```
(’Luis’, ’PBR’, 6)
(’Sinan’, ’PBR’, 4)
```

After first left JOIN

```
(’Luis’, ’Modelo’, ’Luis’, ’PBR’, 6)
(’Chris’, ’SSTP’, NULL, NULL, NULL)
(’Sinan’, ’Blue␣Moon’, ’Sinan’, ’PBR’, 4)
```

Outer Joins

```
('Luis', 'SSTP', 8)
('Chris', 'SSTP', 10)
```

```
('Luis', 'Modelo', 'Luis', 'PBR', 6)
('Chris', 'SSTP', NULL, NULL, NULL)
('Sinan', 'Blue␣Moon', 'Sinan', 'PBR', 4)
```

After second LEFT JOIN

l.DR, l.BR, r1.DR, r1.BR, r1.S, r2.DR, r2.BR, r2.S

```
('Luis', 'Modelo', 'Luis', 'PBR', 6, 'Luis', 'SSTP', 8)
('Chris', 'SSTP', NULL, NULL, NULL, 'Chris', 'SSTP', 10)
('Sinan', 'Blue␣Moon', 'Sinan', 'PBR', 4, NULL, NULL, NULL)
```

Outer Joins

LIKES (DRINKER, BEER)

RATES (DRINKER, BEER, SCORE)

Q: for each drinker, give a rating for ‘PBR’ and for ‘SSTP’

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

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

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...
```

Outer Joins

LIKES (DRINKER, BEER)

RATES (DRINKER, BEER, SCORE)

Q: for each drinker, give a rating for ‘PBR’ and for ‘SSTP’

```
SELECT l.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 (SELECT * FROM RATES
       WHERE BEER = ’PBR’) r1 ON l.DRINKER = r1.DRINKER
  LEFT OUTER JOIN (SELECT * FROM RATES
       WHERE BEER = ’SSTP’) r2 ON l.DRINKER = r2.DRINKER
```

Outer Joins

Q: for each drinker, give a rating for ‘PBR’ and for ‘SSTP’

l.DR, l.BR, r1.DR, r1.BR, r1.S, r2.DR, r2.BR, r2.S

('Luis', 'Modelo', 'Luis', 'PBR', 6, 'Luis', 'SSTP', 8)
('Chris', 'SSTP', **NULL**, **NULL**, **NULL**, 'Chris', 'SSTP', 10)
('Sinan', 'Blue␣Moon', 'Sinan', 'PBR', 4, **NULL**, **NULL**, **NULL**)

Query result:

```
('Luis', 'PBR␣rating:␣6','SSTP␣rating:␣8')
('Chris', 'PBR␣rating:␣unknown','SSTP␣rating:␣10')
('Sinan', 'PBR␣rating:␣4','SSTP␣rating:␣unknown')
```

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

DML & DDL

Data Manipulation Language

- Data retrieval (SELECT)
- Data insertion (INSERT)
- Data deletion (DELETE)
- Data modification (UPDATE)

Data Definition Language

- Relation definition (CREATE TABLE)
- Relation schema update (ALTER TABLE)
- Relation deletion (DROP TABLE)

A bit on the DDL

Creating tables

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

There are many types!

- Do a Google search: SQL data types

Defining a Primary Key

```
CREATE TABLE RATES (
  DRINKER VARCHAR (30) NOT NULL,
  BEER VARCHAR (30) NOT NULL,
  SCORE INTEGER,
  PRIMARY KEY (DRINKER, BEER)
)
```

What about:

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

Defining a Primary Key

```
CREATE TABLE RATES (
  DRINKER VARCHAR (30) NOT NULL,
  BEER VARCHAR (30) NOT NULL,
  SCORE INTEGER,
  PRIMARY KEY (DRINKER, BEER)
)
```

What about:

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

- UNIQUE can accept NULL values
- There can only be one PK but multiple unique fields/combinations

Defining a Primary Key

Can also use:

```
CREATE TABLE RATES (
  DRINKER VARCHAR (30) NOT NULL,
  BEER VARCHAR (30) NOT NULL,
  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)
```

Why use a FK constraint?

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)
```

Why use a FK constraint?

- Makes sure the FK values exist in the parent table

ALTER TABLE

- Add/delete columns
- Rename table/columns
- Add/delete constraints
- Change column data types

DROP/TRUNCATE TABLE

DROP

- Removes contents of the table and its definition

```
DROP TABLE [IF EXISTS] <tableName>
```

TRUNCATE

- Only removes table contents
- Faster and more efficient than DELETE without WHERE

```
TRUNCATE TABLE <tableName>
```

Adding Data

Manually adding tuples

```
INSERT INTO <tableName> [(columnName [, columnName] ...)]
        VALUES (value) [, (value) ...]
```

```
INSERT INTO RATES VALUES (’Chris’, ’SSTP’, 10);
INSERT INTO RATES (SCORE, BEER, DRINKER)
                    VALUES (8, ’Modelo’, ’Luis’);
INSERT INTO RATES (BEER, DRINKER) VALUES (’SSTP’, ’Chris’);
```

▷ What happens to SCORE in the third case?

Adding Data

Data to add can be the result of a query

```
INSERT INTO <tableName> [(columnName [, columnName] ...)]
        <SELECT statement>
```

- ▷ 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

```
INSERT INTO <tableName> [(columnName [, columnName] ...)]
        <SELECT statement>
```

▷ Ex: 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 (Rates given by Chris to l.BEER)
```

Adding Data

Data to add can be the result of a query

```
INSERT INTO <tableName> [(columnName [, columnName] ...)]
        <SELECT statement>
```

▷ Ex: 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

```
DELETE FROM <tableName>
        [WHERE predicate>]
```

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

Deleting Data

```
DELETE FROM <tableName>
        [WHERE predicate>]
```

- ▷ Ex: delete all of the ratings with a NULL score, or 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

```
UPDATE <tableName>
SET <set clause>
    [WHERE predicate>]
```

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

Modifying Data

```
UPDATE <tableName>
SET <set clause>
    [WHERE predicate>]
```

- ▷ 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?