

# Structured Query Language – Continued

---

Rose-Hulman Institute of Technology  
Curt Clifton



# The Story Thus Far

---

- ❑ SELECT ... FROM ... WHERE
- ❑ SELECT \* ...
- ❑ SELECT Foo AS Bar ...
- ❑ SELECT *expression* ...
- ❑ SELECT ... FROM ... WHERE ... LIKE ...
- ❑ SELECT ... FROM Foo, Bar ...
- ❑ SELECT ... FROM Foo f1, Foo f2 ...



# Next Up: Subqueries

---

- As values
- As relations



# Subqueries as Values

---

- Only allowed when subquery evaluates to **single** value
  - Run-time error otherwise
- Example: Find the restaurants that sell Slice for the price the Joe's charges for Pepsi

# Subqueries as Relations – in FROM

---

```
□ SELECT Likes.customer,  
           mix.soda1, mix.soda2  
   FROM Likes,  
        (SELECT s1.name AS soda1,  
             s2.name AS soda2  
        FROM Soda s1, Soda s2  
        WHERE s1.manf = s2.manf  
             AND s1.name < s2.name)  
        AS mix  
  WHERE Likes.soda = mix.soda1
```



## Subqueries as Relations – in WHERE

---

- *value IN relation*
- Evaluates to **true** if relation contains value
- SELECT \*  
FROM Soda  
WHERE name IN (SELECT soda  
FROM Likes  
WHERE  
customer = 'Fred')

# Subqueries as Relations – in WHERE

---

- ❑ EXISTS *relation*
- ❑ Evaluates to **true** if relation is non-empty
- ❑ Find every soda where its manufacturer does not make anything else
- ❑ SELECT name  
FROM Soda s1  
WHERE NOT EXISTS (  
SELECT \*  
FROM Soda s2  
WHERE s2.manf = s1.manf  
AND s2.name <> s1.name)



# Subqueries as Relations – in WHERE

---

## □ ANY

- $x \text{ comp ANY}(\text{relation})$
- $\text{comp}$  can be  $<$ ,  $>$ ,  $=$ ,  $<>$ ,  $>=$ ,  $<=$
- Evaluates to **true** if comparison holds for any tuple in relation

## □ ALL

- $x \text{ comp ALL}(\text{relation})$
- $\text{comp}$  can be  $<$ ,  $>$ ,  $=$ ,  $<>$ ,  $>=$ ,  $<=$
- Evaluates to **true** if comparison holds for every tuple in relation





# Example

---

□ SELECT soda  
FROM Sells  
WHERE price  $\geq$  ALL(SELECT price  
FROM Sells)



# Subqueries Summary

---

- As values
- As relations in FROM clause
- As relations in WHERE clause
  - IN
  - EXISTS
  - ANY
  - ALL



# Combining Relations

---

- Union, Intersection, Difference
- Joins



# Union, Intersection, and Difference

---

- Union
  - (subquery) UNION (subquery )
- Intersection
  - (subquery) INTERSECT (subquery)
- Difference
  - (subquery) EXCEPT (subquery)



# SQL Goofiness – Sets vs. Bags

---

- Bags by default
  - SELECT
- Sets by default
  - UNION
  - INTERSECT
  - EXCEPT
- Overriding defaults
  - SELECT DISTINCT
  - UNION ALL
  - *Cannot override*
  - *Cannot override*



# Example

---

- Find all the different prices charged for sodas



# Example

---

- Find all the different prices charged for sodas
  - `SELECT DISTINCT price`  
`FROM Sells`



# Theta Join

---

- Syntax:
- SELECT ...  
    FROM *table1* JOIN *table2* ON *condition*  
    ...





# Example

---

- Give name and phone number of all customers that frequent Joe's Sushi



# Example

---

- SELECT name, phone  
FROM Customer JOIN Frequent  
ON name = customer  
WHERE rest = 'Joe"s Sushi'
- Compare:
  - SELECT name, phone  
FROM Customer, Frequent  
WHERE name = customer  
AND rest = 'Joe"s Sushi'



# Natural Join

---

- Not in SQL Server
- But some DBMS allow:
  - SELECT ...  
FROM *table1* NATURAL JOIN *table2*



# Outer Joins

---

- Recall: solution to dangling tuple problem
- Make sure every tuple shows up, even if no “mate”, by inserting nulls if needed
- Three basic forms:
  - SELECT ... FROM *t1* LEFT OUTER JOIN *t2*
  - SELECT ... FROM *t1* RIGHT OUTER JOIN *t2*
  - SELECT ... FROM *t1* OUTER JOIN *t2*



# Cross Product

---

- Possible, though less common
- SELECT ...  
    FROM *table1* CROSS JOIN *table2*
- Or just write:
  - SELECT ...  
    FROM *table1, table2*



# Reporting

---

- Aggregation
- Grouping

# Aggregation

---

- Calculations over rows
- Example:
  - `SELECT AVG(price)`  
`FROM Sells`  
`WHERE soda = 'Pepsi'`
- Other aggregations:
  - SUM
  - AVG
  - COUNT, COUNT(\*)
  - MIN, MAX



“Let me explain. No, would take too long. Let me sum up.”



# Aggregation and Duplicates

---

- Can use DISTINCT inside an aggregation
- Example – Find the number of different prices charged for Pepsi





# Aggregation and Duplicates

---

- Can use DISTINCT inside an aggregation
- Example – Find the number of different prices charged for Pepsi
  - `SELECT COUNT(DISTINCT price)`  
`FROM Sells`  
`WHERE soda = 'Pepsi'`

# Grouping

---

- For aggregating subsections of result
- **SELECT ...**  
**FROM ...**  
**WHERE ...**  
**GROUP BY** *attr,...*





# Example: Grouping

---

- Find the average price for each soda



# Example: Grouping

---

- Find the average price for each soda
- `SELECT soda, AVG(price)`  
    `FROM Sells`  
    `GROUP BY soda`



# Having

---

- Like a WHERE clause for groups
- SELECT ...  
FROM ...  
WHERE ... -- *filter rows*  
GROUP BY ... -- *group rows*  
HAVING ... -- *filter groups*



# Example: Having

---

- Find the average price of those sodas that are served by at least three restaurants



## Example: Having

---

- Find the average price of those sodas that are served by at least three restaurants
- `SELECT soda, AVG(price)`  
    `FROM Sells`  
    `GROUP BY soda`  
    `HAVING COUNT(rest) >= 3`



# Modifying the Database

---

- Insert
- Delete
- Update





# Insertion

---

- Single tuple, quick and dirty:
  - INSERT INTO *table*  
VALUES (*value1*, ...)
- Single tuple, more robust:
  - INSERT INTO *table(attr1, ...)*  
VALUES (*value1*, ...)
- Many tuples:
  - INSERT INTO *table (subquery)*



# Deletion

---

- Single tuple:
  - DELETE FROM *table* WHERE *condition*
- All tuples (zoinks!):
  - DELETE FROM *table*



# Updates

---

□ Syntax:

■ UPDATE *table*

SET *attr1 = expr1, ...*      -- *attributes, new values*

WHERE *condition*              -- *rows to change*



# Example

---

- Change Fred's phone number to 555-1212



# Example

---

- Change Fred's phone number to 555-1212
- UPDATE Customer  
    SET phone = '555-1212'  
    WHERE name = 'Fred'



# Example

---

- Raise all prices by 10%



# Example

---

- Raise all prices by 10%
- UPDATE Sells  
SET price = (price \* 1.10)