Today’s Topics

• SQL Statements (Continue)
SQL SELECT Statement (Second Part)

- Queries with aggregate functions
- Queries with GROUP BY/HAVING
- Queries with ORDER BY
Aggregates

- AVG, COUNT, SUM, VARIANCE, MIN/MAX, and STDEV

```sql
SELECT [DISTINCT] AVG(S.gpa)
FROM Students S
WHERE S.dept = 'CS'
```

- Before producing output, compute a summary (a.k.a. an aggregate) of some arithmetic expression

- Produces one row of output
  - with one column in this case
### Queries with Aggregate Functions

<table>
<thead>
<tr>
<th>SUPNR</th>
<th>PRODNR</th>
<th>PURCHASE_PRICE</th>
<th>DELIV_PERIOD</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>21</td>
<td>0178</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>37</td>
<td>0178</td>
<td><strong>16.99</strong></td>
<td>4</td>
</tr>
<tr>
<td>68</td>
<td>0178</td>
<td>17.99</td>
<td>5</td>
</tr>
<tr>
<td>69</td>
<td>0178</td>
<td><strong>16.99</strong></td>
<td>NULL</td>
</tr>
<tr>
<td>94</td>
<td>0178</td>
<td>18.00</td>
<td>6</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Queries with Aggregate Functions

Q12: SELECT COUNT(*)
    FROM SUPPLIES
    WHERE PRODNR = '0178'

Q13: SELECT COUNT(PURCHASE_PRICE)
    FROM SUPPLIES
    WHERE PRODNR = '0178'

Q14: SELECT COUNT(DISTINCT PURCHASE_PRICE)
    FROM SUPPLIES
    WHERE PRODNR = '0178'
Group By

SELECT [DISTINCT] AVG(S.gpa), S.dept
FROM Students S
GROUP BY S.dept

- Partition table into groups with same GROUP BY column values
  - Can group by a list of columns
- Produce an aggregate result per group
  - Cardinality of output = # of distinct group values
- Always follows the WHERE Clause
- Always precedes the ORDER BY
- Note: can put grouping columns in SELECT list
Queries with Aggregate Functions

Q15: SELECT PRODNR, SUM(PURCHASE_PRICE) AS SUM_PURCHASE_PRICE
     FROM SUPPLIES
     WHERE PRODNR = '0178'
     GROUP BY PRODNR;

<table>
<thead>
<tr>
<th>prodnr</th>
<th>sum_purchase_price</th>
</tr>
</thead>
<tbody>
<tr>
<td>0178</td>
<td>69.97</td>
</tr>
</tbody>
</table>
Queries with Aggregate Functions

Q16: SELECT SUM(PURCHASE_PRICE) AS TOTAL_ORDERS FROM SUPPLIES;

Q17: SELECT PRODNR, AVG(PURCHASE_PRICE) AS WEIGHTED_AVG_PRICE FROM SUPPLIES WHERE PRODNR = '0178' GROUP BY PRODNR

<table>
<thead>
<tr>
<th>SUPNR</th>
<th>PRODNR</th>
<th>PURCHASE_PRICE</th>
<th>DELIV_PERIOD</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>21</td>
<td>0178</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>37</td>
<td>0178</td>
<td>16.99</td>
<td>4</td>
</tr>
<tr>
<td>68</td>
<td>0178</td>
<td>17.99</td>
<td>5</td>
</tr>
<tr>
<td>69</td>
<td>0178</td>
<td>16.99</td>
<td>NULL</td>
</tr>
<tr>
<td>94</td>
<td>0178</td>
<td>18.00</td>
<td>6</td>
</tr>
</tbody>
</table>

0178, (16.99 + 17.99 + 16.99 + 18.00) / 4 = 17.4925
Queries with Aggregate Functions

Q18: SELECT PRODNR, AVG(DISTINCT PURCHASE_PRICE) AS UNWEIGHTED_AVG_PRICE
FROM SUPPLIES WHERE PRODNR = '0178'
GROUP BY PRODNR

<table>
<thead>
<tr>
<th>SUPNR</th>
<th>PRODNR</th>
<th>PURCHASE_PRICE</th>
<th>DELIV_PERIOD</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>0178</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>37</td>
<td>0178</td>
<td>16.99</td>
<td>4</td>
</tr>
<tr>
<td>68</td>
<td>0178</td>
<td>17.99</td>
<td>5</td>
</tr>
<tr>
<td>69</td>
<td>0178</td>
<td>16.99</td>
<td>NULL</td>
</tr>
<tr>
<td>94</td>
<td>0178</td>
<td>18.00</td>
<td>6</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

0178, (16.99 + 17.99 + 18.00) / 3 = 17.66
Queries with Aggregate Functions

• Q19: SELECT PRODNR, VARIANCE(PURCHASE_PRICE) AS PRICE_VARIANCE FROM SUPPLIES WHERE PRODNR = '0178' GROUP BY PRODNR

<table>
<thead>
<tr>
<th>prodnr</th>
<th>price_variance</th>
</tr>
</thead>
<tbody>
<tr>
<td>0178</td>
<td>0.336691666666563</td>
</tr>
</tbody>
</table>
Queries with Aggregate Functions

Q20: SELECT PRODNR, MIN(PURCHASE_PRICE) AS LOWEST_PRICE, MAX(PURCHASE_PRICE) AS HIGHEST_PRICE
FROM SUPPLIES
WHERE PRODNR = '0178'
GROUP BY PRODNR

<table>
<thead>
<tr>
<th>PRODNR</th>
<th>LOWEST_PRICE</th>
<th>HIGHEST_PRICE</th>
</tr>
</thead>
<tbody>
<tr>
<td>0178</td>
<td>16.99</td>
<td>18.00</td>
</tr>
</tbody>
</table>
ORDER BY

- SELECT S.name, S.gpa, S.age*2 AS a2
  FROM Students S
  WHERE S.dept = 'CS'
  ORDER BY S.gpa, S.name, a2;

- ORDER BY clause specifies output to be sorted
  - Lexicographic ordering
- Obviously must refer to columns in the output
  - Note the AS clause for naming output columns!
Order By and ASC, DESC

- SELECT S.name, S.gpa, S.age*2 AS a2
  FROM Students S
  WHERE S.dept = 'CS'
  ORDER BY S.gpa DESC, S.name ASC, a2;

- Ascending order by default, but can be overridden
  - DESC flag for descending, ASC for ascending
- Can mix and match, lexicographically
Queries with ORDER BY

Q21: SELECT name, type, cost
    FROM basic_cards
    ORDER BY name DESC, cost ASC limit 5;

<table>
<thead>
<tr>
<th>name</th>
<th>type</th>
<th>cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Weasel Tunneler</td>
<td>MINION</td>
<td>1</td>
</tr>
<tr>
<td>Venture Co. Mercenary</td>
<td>MINION</td>
<td>5</td>
</tr>
<tr>
<td>Vaelastrasz the Corrupt</td>
<td>HERO</td>
<td>None</td>
</tr>
<tr>
<td>Trogg Hate Minions!</td>
<td>HERO_POWER</td>
<td>0</td>
</tr>
<tr>
<td>Tank Up!</td>
<td>HERO_POWER</td>
<td>2</td>
</tr>
</tbody>
</table>
Q22: SELECT PRODNR, SUPNR, PURCHASE_PRICE
FROM SUPPLIES
WHERE PRODNR = '0178'
ORDER BY PURCHASE_PRICE DESC

<table>
<thead>
<tr>
<th>prodnr</th>
<th>supnr</th>
<th>purchase_price</th>
</tr>
</thead>
<tbody>
<tr>
<td>0178</td>
<td>21</td>
<td>None</td>
</tr>
<tr>
<td>0178</td>
<td>94</td>
<td>18.0</td>
</tr>
<tr>
<td>0178</td>
<td>68</td>
<td>17.99</td>
</tr>
<tr>
<td>0178</td>
<td>37</td>
<td>16.99</td>
</tr>
<tr>
<td>0178</td>
<td>69</td>
<td>16.99</td>
</tr>
</tbody>
</table>
Having

SELECT [DISTINCT] AVG(S.gpa), S.dept
FROM Students S
GROUP BY S.dept
HAVING COUNT(*) > 2

- The HAVING predicate filters groups
- HAVING is applied after grouping and aggregation
  - Hence can contain anything that could go in the SELECT list
  - I.e. aggs or GROUP BY columns
- HAVING can only be used in aggregate queries
- It’s an optional clause
Queries with GROUP BY/HAVING

Q23: SELECT type, count(*) as quantity
    FROM basic_cards
    GROUP BY type
    HAVING COUNT(*) >= 3

<table>
<thead>
<tr>
<th></th>
<th>quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENCHANTMENT</td>
<td>4</td>
</tr>
<tr>
<td>MINION</td>
<td>12</td>
</tr>
<tr>
<td>SPELL</td>
<td>7</td>
</tr>
<tr>
<td>HERO</td>
<td>3</td>
</tr>
<tr>
<td>HERO_POWER</td>
<td>4</td>
</tr>
</tbody>
</table>
Queries with GROUP BY/HAVING

Q22: SELECT player_class, sum(cost) as total
    FROM basic_cards
    GROUP BY player_class
    HAVING sum(cost) >= 5

<table>
<thead>
<tr>
<th>player_class</th>
<th>total</th>
</tr>
</thead>
<tbody>
<tr>
<td>NEUTRAL</td>
<td>52</td>
</tr>
<tr>
<td>WARLOCK</td>
<td>8</td>
</tr>
<tr>
<td>DRUID</td>
<td>5</td>
</tr>
<tr>
<td>PALADIN</td>
<td>5</td>
</tr>
</tbody>
</table>
LIMIT

- SELECT S.name, S.gpa, S.age*2 AS a2
  FROM Students S
  WHERE S.dept = 'CS'
  ORDER BY S.gpa DESC, S.name ASC, a2;
  LIMIT 3;

- Only produces the first <integer> output rows
- Typically used with ORDER BY
  - Otherwise the output is non-deterministic
  - Not a “pure” declarative construct in that case – output set depends on algorithm for query processing
Put it All Together

- SELECT S.dept, AVG(S.gpa), COUNT(*)
  FROM Students S
  WHERE S.gender = 'F'
  GROUP BY S.dept
  HAVING COUNT(*) >= 2
  ORDER BY S.dept DESC;
DISTINCT Aggregates

1. SELECT COUNT(DISTINCT S.name) 
   FROM Students S 
   WHERE S.dept = 'CS';

2. SELECT DISTINCT COUNT(S.name) 
   FROM Students S 
   WHERE S.dept = 'CS';
Let’s Do Labs

- [https://github.com/VTCourses/CS4604_Labs](https://github.com/VTCourses/CS4604_Labs)
- Lab2: 2.select
SQL SELECT Statement

- Join queries
- Nested queries
- Correlated queries
- Queries with ALL/ANY
- Queries with EXISTS
- Queries with subqueries in FROM/WHERE
- Queries with set operations
Renaming – Self-join

• Find Tom’s grandparent(s)

<table>
<thead>
<tr>
<th>PC</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>p-id</td>
<td>c-id</td>
</tr>
<tr>
<td>Mary</td>
<td>Tom</td>
</tr>
<tr>
<td>Peter</td>
<td>Mary</td>
</tr>
<tr>
<td>John</td>
<td>Tom</td>
</tr>
</tbody>
</table>

```
Select gp.p-id
from PC as gp, PC
where gp.c-id = PC.p-id
and PC.c-id = "Tom"
```
Arithmetic Expressions

- SELECT S.age, S.age-5 AS age1, 2*S.age AS age2
  FROM Sailors AS S
  WHERE S.sname = 'Popeye'

- SELECT S1.sname AS name1, S2.sname AS name2
  FROM Sailors AS S1, Sailors AS S2
  WHERE 2*S1.rating = S2.rating - 1
SELECT
    log(1000) as three,
    exp(ln(2)) as two,
    cos(0) as one,
    ln(2*3) = ln(2) + ln(3) as sanity;

three  two  one  sanity
---  ----  ---  ------
3.0   2.0   1.0  True
Join Queries

• Inner joins
• Outer joins

SELECT [DISTINCT] <column expression list>
FROM <table1 [AS t1], ... , tableN [AS tn]>
[WHERE <predicate>]
[GROUP BY <column list>][HAVING <predicate>]
[ORDER BY <column list>];
Inner Joins

**SUPPLIER**(*SUPNR, SUPNAME, ..., SUPSTATUS*)

**SUPPLIES**(*SUPNR, PRODNR, PURCHASE_PRICE, ...*)

<table>
<thead>
<tr>
<th>SUPNR</th>
<th>SUPNAME</th>
<th>SUPADDRESS</th>
<th>SUPCITY</th>
<th>SUPSTATUS</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>Best wines</td>
<td></td>
<td></td>
<td>90</td>
</tr>
<tr>
<td>68</td>
<td>The Wine Depot</td>
<td></td>
<td></td>
<td>10</td>
</tr>
<tr>
<td>84</td>
<td>Wine Trade Logistics</td>
<td></td>
<td></td>
<td>92</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SUPNR</th>
<th>PRODNR</th>
<th>PURCHASE_PRICE</th>
<th>DELIV_PERIOD</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>0474</td>
<td>40.00</td>
<td>1</td>
</tr>
<tr>
<td>32</td>
<td>0154</td>
<td>21.00</td>
<td>4</td>
</tr>
<tr>
<td>84</td>
<td>0494</td>
<td>15.99</td>
<td>2</td>
</tr>
</tbody>
</table>
## Joins

**Q25: SELECT**  
\[ \text{R.SUPNR}, \text{R.SUPNAME}, \text{R.SUPSTATUS}, \text{S.SUPNR}, \text{S.PRODNR}, \text{S.PURCHASE\_PRICE} \]  

**FROM SUPPLIER R, SUPPLIES S**

<table>
<thead>
<tr>
<th>supnr</th>
<th>supname</th>
<th>supstatus</th>
<th>supnr_1</th>
<th>prodnr</th>
<th>purchase_price</th>
</tr>
</thead>
<tbody>
<tr>
<td>21</td>
<td>Deliwines</td>
<td>20</td>
<td>21</td>
<td>0178</td>
<td>None</td>
</tr>
<tr>
<td>32</td>
<td>Best Wines</td>
<td>90</td>
<td>21</td>
<td>0178</td>
<td>None</td>
</tr>
<tr>
<td>37</td>
<td>Ad Fundum</td>
<td>95</td>
<td>21</td>
<td>0178</td>
<td>None</td>
</tr>
<tr>
<td>52</td>
<td>Spirits &amp; co.</td>
<td>None</td>
<td>21</td>
<td>0178</td>
<td>None</td>
</tr>
<tr>
<td>68</td>
<td>The Wine Depot</td>
<td>10</td>
<td>21</td>
<td>0178</td>
<td>None</td>
</tr>
<tr>
<td>69</td>
<td>Vinos del Mundo</td>
<td>92</td>
<td>21</td>
<td>0178</td>
<td>None</td>
</tr>
<tr>
<td>94</td>
<td>The Wine Crate</td>
<td>75</td>
<td>21</td>
<td>0178</td>
<td>None</td>
</tr>
<tr>
<td>84</td>
<td>Wine Trade Logistics</td>
<td>92</td>
<td>21</td>
<td>0178</td>
<td>None</td>
</tr>
<tr>
<td>21</td>
<td>Deliwines</td>
<td>20</td>
<td>37</td>
<td>0178</td>
<td>16.99</td>
</tr>
<tr>
<td>32</td>
<td>Best Wines</td>
<td>90</td>
<td>37</td>
<td>0178</td>
<td>16.99</td>
</tr>
</tbody>
</table>
Inner Joins

Q26: SELECT R.SUPNR, R.SUPNAME, R.SUPSTATUS, S.SUPNR, S.PRODNR, S.PURCHASE_PRICE
FROM SUPPLIER R, SUPPLIES S
WHERE R.SUPNR = S.SUPNR

<table>
<thead>
<tr>
<th>supnr</th>
<th>supname</th>
<th>supstatus</th>
<th>supnr_1</th>
<th>prodnr</th>
<th>purchase_price</th>
</tr>
</thead>
<tbody>
<tr>
<td>21</td>
<td>Deliwines</td>
<td>20</td>
<td>21</td>
<td>0178</td>
<td>None</td>
</tr>
<tr>
<td>37</td>
<td>Ad Fundum</td>
<td>95</td>
<td>37</td>
<td>0178</td>
<td>16.99</td>
</tr>
<tr>
<td>68</td>
<td>The Wine Depot</td>
<td>10</td>
<td>68</td>
<td>0178</td>
<td>17.99</td>
</tr>
<tr>
<td>69</td>
<td>Vinos del Mundo</td>
<td>92</td>
<td>69</td>
<td>0178</td>
<td>16.99</td>
</tr>
<tr>
<td>94</td>
<td>The Wine Crate</td>
<td>75</td>
<td>94</td>
<td>0178</td>
<td>18.0</td>
</tr>
</tbody>
</table>
Q27: SELECT R.SUPNR, R.SUPNAME, R.SUPSTATUS, S.PRODNR, S.PURCHASE_PRICE
FROM SUPPLIER AS R INNER JOIN SUPPLIES AS S
ON (R.SUPNR = S.SUPNR)
### Inner Joins

**Q28: SELECT** R.SUPNR, R.SUPNAME, PO.PONR, PO.PODATE, P.PRODNR, P.PRODNAME, POL.QUANTITY

**FROM** SUPPLIER R, PURCHASE_ORDER PO, PO_LINE POL, PRODUCT P

**WHERE** (R.SUPNR = PO.SUPNR) AND (PO.PONR = POL.PONR) AND (POL.PRODNR = P.PRODNR)

<table>
<thead>
<tr>
<th>R.SUPNR</th>
<th>R.SUPNAME</th>
<th>PO.PONR</th>
<th>PO.PODATE</th>
<th>POL.PRODNR</th>
<th>P.PRODNAME</th>
<th>POL.QUANTITY</th>
</tr>
</thead>
<tbody>
<tr>
<td>37</td>
<td>Ad Fundum</td>
<td>1511</td>
<td>2015-03-24</td>
<td>0212</td>
<td>Billecart-Salmon, Brut Réserve, 2014</td>
<td>2</td>
</tr>
<tr>
<td>37</td>
<td>Ad Fundum</td>
<td>1511</td>
<td>2015-03-24</td>
<td>0345</td>
<td>Vascosassetti, Brunello di Montalcino, 2004</td>
<td>4</td>
</tr>
<tr>
<td>37</td>
<td>Ad Fundum</td>
<td>1511</td>
<td>2015-03-24</td>
<td>0783</td>
<td>Clos D’Opleeuw, Chardonnay, 2012</td>
<td>1</td>
</tr>
<tr>
<td>37</td>
<td>Ad Fundum</td>
<td>1511</td>
<td>2015-03-24</td>
<td>0856</td>
<td>Domaine Chandon de Briailles, Savigny-Les-Beaune, 2006</td>
<td>9</td>
</tr>
<tr>
<td>94</td>
<td>The Wine Crate</td>
<td>1512</td>
<td>2015-04-10</td>
<td>0178</td>
<td>Meerdael, Methode Traditionnelle Chardonnay, 2014</td>
<td>3</td>
</tr>
</tbody>
</table>
Inner Joins

Q29: SELECT R1.SUPNAME, R2.SUPNAME, R1.SUPCITY
    FROM SUPPLIER R1, SUPPLIER R2
    WHERE R1.SUPCITY = R2.SUPCITY
    AND (R1.SUPNR < R2.SUPNR)

<table>
<thead>
<tr>
<th>supname</th>
<th>supname_1</th>
<th>supcity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Best Wines</td>
<td>The Wine Depot</td>
<td>San Francisco</td>
</tr>
<tr>
<td>Ad Fundum</td>
<td>The Wine Crate</td>
<td>Chicago</td>
</tr>
</tbody>
</table>
Inner Joins

Q30: SELECT R.SUPNAME
    FROM SUPPLIER R, SUPPLIES S
    WHERE R.SUPNR = S.SUPNR
    AND S.PRODNR = '0899'

Q31: SELECT DISTINCT R.SUPNAME
    FROM SUPPLIER R, SUPPLIES S, PRODUCT P
    WHERE S.SUPNR = R.SUPNR
    AND S.PRODNR = P.PRODNR
    AND P.PRODTYPE = 'ROSE'
Inner Joins

Q32: SELECT P.PRODNR, P.PRODNAME, SUM(POL.QUANTITY) FROM PRODUCT P, PO_LINE POL WHERE P.PRODNR = POL.PRODNR GROUP BY P.PRODNR

<table>
<thead>
<tr>
<th>PRODNR</th>
<th>PRODNAME</th>
<th>SUM(POL.QUANTITY)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0178</td>
<td>Meerdael, Methode Traditionnelle Chardonnay, 2014</td>
<td>9</td>
</tr>
<tr>
<td>0185</td>
<td>Chateau Petrus, 1975</td>
<td>2</td>
</tr>
<tr>
<td>0212</td>
<td>Billecart-Salmon, Brut Réserve, 2014</td>
<td>23</td>
</tr>
<tr>
<td>0295</td>
<td>Chateau Pape Clement, Pessac-Léognan, 2001</td>
<td>9</td>
</tr>
<tr>
<td>0306</td>
<td>Chateau Coupe Roses, Granaxa, 2011</td>
<td>11</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Join Variants

SELECT  <column expression list>
FROM   table_name
  [INNER  |  NATURAL
   |  {LEFT  |  RIGHT  |  FULL  }  {OUTER} ] JOIN table_name
ON     <qualification_list>
WHERE   ...

• INNER is default
• Inner join what we’ve learned so far
  – Same thing, just with different syntax.
Inner/Natural Joins

SELECT s.sid, s.sname, r.bid
FROM Sailors s, Reserves r
WHERE s.sid = r.sid
     AND s.age > 20;

SELECT s.sid, s.sname, r.bid
FROM Sailors s
     INNER JOIN
     Reserves r
     ON s.sid = r.sid
WHERE s.age > 20;

SELECT s.sid, s.sname, r.bid
FROM Sailors s
     NATURAL JOIN
     Reserves r
WHERE s.age > 20;

• ALL 3 ARE EQUIVALENT!
• “NATURAL” means equi-join for pairs of attributes with the same name
Outer Join

- Outer join can be used when we want to keep all the tuples of **one** or **both** tables in the result of the JOIN, regardless of whether or not they have matching tuples in the other table
- Left outer join
- Right outer join
- Full outer join
Left Outer Joins

- Returns all matched rows, and *preserves* all unmatched rows from the table on the left of the join clause.
- Use nulls in fields of non-matching tuples.

Q33: SELECT s.sid, s.sname, r.bid
FROM Sailors s LEFT OUTER JOIN Reserves r
ON s.sid = r.sid;

- Returns all sailors & bid for boat in any of their reservations.
- Note: no match for s.sid? r.bid IS NULL!
Right Outer Joins

- Returns all matched rows, and preserves all unmatched rows from the table on the right of the join clause
- Use nulls in fields of non-matching tuples

Q34: SELECT r.sid, b.bid, b.bname
FROM Reserves r RIGHT OUTER JOIN Boats b
ON r.bid = b.bid

- Returns all boats and sid for any sailor associated with the reservation.
- Note: no match for b.bid? r.sid IS NULL!

<table>
<thead>
<tr>
<th>sid</th>
<th>bid</th>
<th>bname</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>102</td>
<td>Pinta</td>
</tr>
<tr>
<td>2</td>
<td>102</td>
<td>Pinta</td>
</tr>
<tr>
<td>1</td>
<td>101</td>
<td>Nina</td>
</tr>
<tr>
<td>None</td>
<td>103</td>
<td>Santa Maria</td>
</tr>
</tbody>
</table>
**Full Outer Join**

- **Returns all (matched or unmatched) rows from the tables on both sides of the join clause**

  ```sql
  SELECT r.sid, b.bid, b.bname 
  FROM Reserves r FULL OUTER JOIN Boats b 
  ON r.bid = b.bid
  ```

- Returns all boats & all information on reservations
- No match for r.bid?
  - b.bid IS NULL AND b.bname IS NULL!
- No match for b.bid?
  - r.sid IS NULL!
Nested Queries

Outer block

Inner block

SELECT ... 
FROM ... 
WHERE ... 

(SELECT ... 
FROM ... 
WHERE ... )
Nested Queries

**Q34:** Names of sailors who’ve reserved boat #102:

```
SELECT S.sname
FROM   Sailors S
WHERE  S.sid IN
       (SELECT R.sid
        FROM    Reserves R
        WHERE   R.bid=102)
```

---

**Sailors**

<table>
<thead>
<tr>
<th>sid</th>
<th>sname</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Popeye</td>
<td>10</td>
<td>22</td>
</tr>
<tr>
<td>2</td>
<td>OliveOyl</td>
<td>11</td>
<td>39</td>
</tr>
<tr>
<td>3</td>
<td>Garfield</td>
<td>1</td>
<td>27</td>
</tr>
<tr>
<td>4</td>
<td>Bob</td>
<td>5</td>
<td>19</td>
</tr>
</tbody>
</table>

**Reserves**

<table>
<thead>
<tr>
<th>sid</th>
<th>bid</th>
<th>day</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>102</td>
<td>9/12</td>
</tr>
<tr>
<td>2</td>
<td>102</td>
<td>9/13</td>
</tr>
<tr>
<td>1</td>
<td>101</td>
<td>10/01</td>
</tr>
</tbody>
</table>

---

Names: Popeye, OliveOyl
Queries with EXISTS

• The EXISTS function checks whether the result of a correlated nested query is empty or not
• EXISTS returns TRUE if there is at least one tuple in the result of the nested query, or otherwise returns FALSE
• Vice versa, the NOT EXISTS function returns TRUE if there are no tuples in the result of the nested query, or otherwise returns FALSE
Nested Queries: Exists with Correlation

Q35: Names of sailors who’ve reserved boat #102

```
SELECT  S.sname
FROM    Sailors S
WHERE   EXISTS
        (SELECT  *
         FROM   Reserves R
         WHERE  R.bid=102 AND
                S.sid=R.sid)
```

- Correlated subquery is recomputed for each Sailors tuple.
Nested Queries: Not Exists

Q35: Names of sailors who have not reserved boat #102

```sql
SELECT   S.sname  
FROM      Sailors S  
WHERE     NOT EXISTS  
            (SELECT *  
             FROM  Reserves R  
             WHERE R.bid=102 AND  
                 S.sid=R.sid)
```

- Correlated subquery is recomputed for each Sailors tuple.
Nested Queries

Q36: SELECT SUPNAME
FROM SUPPLIER
WHERE SUPNR IN
  (SELECT SUPNR
   FROM SUPPLIES
   WHERE PRODNR = '0178')

Q37: SELECT SUPNAME
FROM SUPPLIER
WHERE SUPNR IN
  (SELECT SUPNR
   FROM SUPPLIES
   WHERE PRODNR IN
     (SELECT PRODNR
      FROM PRODUCT
      WHERE PRODTYPE = 'ROSE')))
Nested Queries

Q38: SELECT PRODNAME
    FROM PRODUCT
    WHERE PRODNR IN
    (SELECT PRODNR
        FROM SUPPLIES
        WHERE SUPNR = '32')
    AND PRODNR IN
    (SELECT PRODNR
        FROM SUPPLIES
        WHERE SUPNR = '84')
Correlated Queries

- Whenever a condition in the WHERE clause of a nested query references some column of a table declared in the outer query, the two queries are said to be correlated.
- The nested query is then evaluated once for each tuple (or combination of tuples) in the outer query.
Examples: Correlated Queries

Q39: SELECT P.PRODNR
FROM PRODUCT P
WHERE 1 <
(SELECT COUNT(*)
FROM PO_LINE POL
WHERE P.PRODNR = POL.PRODNR)

Q40: SELECT R.SUPNR, R.SUPNAME,
P.PRODNR, P.PRODNAME,
S1.PURCHASE_PRICE,
S1.DELIV_PERIOD
FROM SUPPLIER R, SUPPLIES S1,
PRODUCT P
WHERE R.SUPNR = S1.SUPNR
AND S1.PRODNR = P.PRODNR
AND S1.PURCHASE_PRICE <
(SELECT AVG(PURCHASE_PRICE)
FROM SUPPLIES S2
WHERE P.PRODNR = S2.PRODNR)

Q41: SELECT P1.PRODNR
FROM PRODUCT P1
WHERE 3 >
(SELECT COUNT(*)
FROM PRODUCT P2
WHERE P1.PRODNR <
P2.PRODNR)
Reading and Next Class

• SQL II: Ch 5
• Next: SQL III: Ch5