CS 4604: Introduction to Database Management Systems

SQL II

Virginia Tech CS 4604 Sprint 2021

Instructor: Yinlin Chen

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

```
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

SUPPLIES

SUPNR	PRODNR	PURCHASE_PRICE	DELIV_PERIOD
…			
21	0178	NULL	NULL
37	0178	**16.99**	4
68	0178	17.99	5
69	0178	**16.99**	NULL
94	0178	18.00	6
…			

Queries with Aggregate Functions

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

5

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

4

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

3

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;
```

prodnr	sum_purchase_price
0178	69.97

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
```

SUPPLIES

SUPNR	PRODNR	PURCHASE_PRICE	DELIV_PERIOD
…			
21	0178	NULL	NULL
37	0178	16.99	4
68	0178	17.99	5
69	0178	16.99	NULL
94	0178	18.00	6

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
```

SUPNR	PRODNR	PURCHASE_PRICE	DELIV_PERIOD
…			
21	0178	NULL	NULL
37	0178	16.99	4
68	0178	17.99	5
69	0178	16.99	NULL
94	0178	18.00	6
…			

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
```

prodnr	price_variance
0178	0.336691666666563

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
```

PRODNR	LOWEST_PRICE	HIGHEST_PRICE
0178	16.99	18.00

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;
```

name	type	cost
Weasel Tunneler	MINION	1
Venture Co. Mercenary	MINION	5
Vaelastrasz the Corrupt	HERO	None
Trogg Hate Minions!	HERO_POWER	0
Tank Up!	HERO_POWER	2

Queries with ORDER BY

```
Q22: SELECT PRODNR, SUPNR, PURCHASE_PRICE
     FROM SUPPLIES
     WHERE PRODNR = '0178'
     ORDER BY PURCHASE_PRICE DESC
```

prodnr	supnr	purchase_price
0178	21	None
0178	94	18.0
0178	68	17.99
0178	37	16.99
0178	69	16.99

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
```

type	quantity
ENCHANTMENT	4
MINION	12
SPELL	7
HERO	3
HERO_POWER	4

Queries with GROUP BY/HAVING

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

player_class	total
NEUTRAL	52
WARLOCK	8
DRUID	5
PALADIN	5

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
- 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)

PC	
p-id	**c-id**
Mary	Tom
Peter	Mary
John	Tom

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**

SQL Calculator!

```
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, ...)

SUPNR	SUPNAME	SUPADDRESS	SUPCITY	SUPSTATUS
32	Best wines			90
68	The Wine Depot			10
84	Wine Trade Logistics			92
:	:			:

SUPNR	PRODNR	PURCHASE_PRICE	DELIV_PERIOD
32	0474	40.00	1
32	0154	21.00	4
84	0494	15.99	2
:	:	:	

Joins

```
Q25: SELECT
R.SUPNR, R.SUPNAME,
R.SUPSTATUS,
S.SUPNR, S.PRODNR,
S.PURCHASE_PRICE

FROM SUPPLIER R,
SUPPLIES S
```

supnr	supname	supstatus	supnr_1	prodnr	purchase_price
21	Deliwines	20	21	0178	None
32	Best Wines	90	21	0178	None
37	Ad Fundum	95	21	0178	None
52	Spirits & co.	None	21	0178	None
68	The Wine Depot	10	21	0178	None
69	Vinos del Mundo	92	21	0178	None
94	The Wine Crate	75	21	0178	None
84	Wine Trade Logistics	92	21	0178	None
21	Deliwines	20	37	0178	16.99
32	Best Wines	90	37	0178	16.99

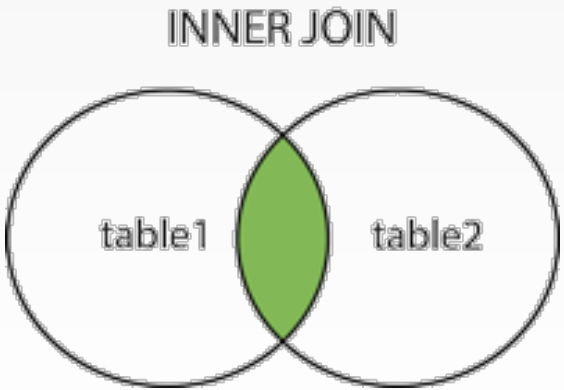
Inner Joins

```
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
```

supnr	supname	supstatus	supnr_1	prodnr	purchase_price
21	Deliwines	20	21	0178	None
37	Ad Fundum	95	37	0178	16.99
68	The Wine Depot	10	68	0178	17.99
69	Vinos del Mundo	92	69	0178	16.99
94	The Wine Crate	75	94	0178	18.0

Inner Joins

```
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)

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

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)
```

supname	supname_1	supcity
Best Wines	The Wine Depot	San Francisco
Ad Fundum	The Wine Crate	Chicago

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
```

PRODNR	PRODNAME	SUM(POL.QUANTITY)
0178	Meerdael, Methode Traditionnelle Chardonnay, 2014	9
0185	Chateau Petrus, 1975	2
0212	Billecart-Salmon, Brut Réserve, 2014	23
0295	Chateau Pape Clement, Pessac-Léognan, 2001	9
0306	Chateau Coupe Roses, Granaxa, 2011	11
…		

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;
```

Reserves

sid	bid	day
1	102	9/12
2	102	9/13
1	101	10/01

Sailors

sid	sname	rating	age
1	Popeye	10	22
2	OliveOyl	11	39
3	Garfield	1	27
4	Bob	5	19

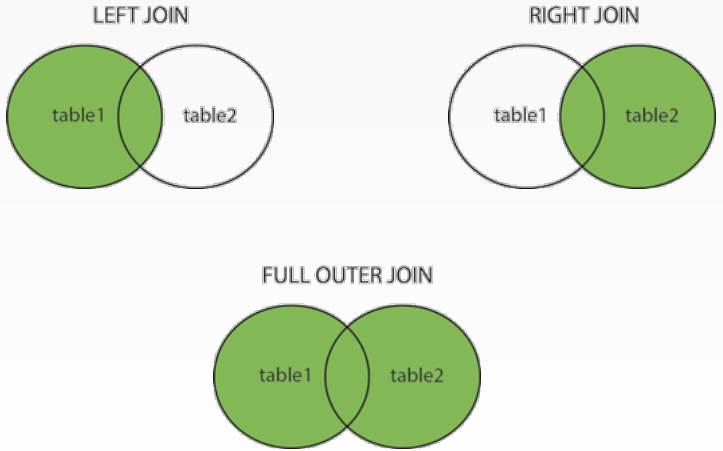
Boats

bid	bname	color
101	Nina	red
102	Pinta	blue
103	Santa Maria	red

- **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;
```

sid	sname	bid
1	Popeye	102
1	Popeye	101
2	OliveOyl	102
3	Garfield	None
4	Bob	None

- 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
```

sid	bid	bname
1	102	Pinta
2	102	Pinta
1	101	Nina
None	103	Santa Maria

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

Full Outer Join

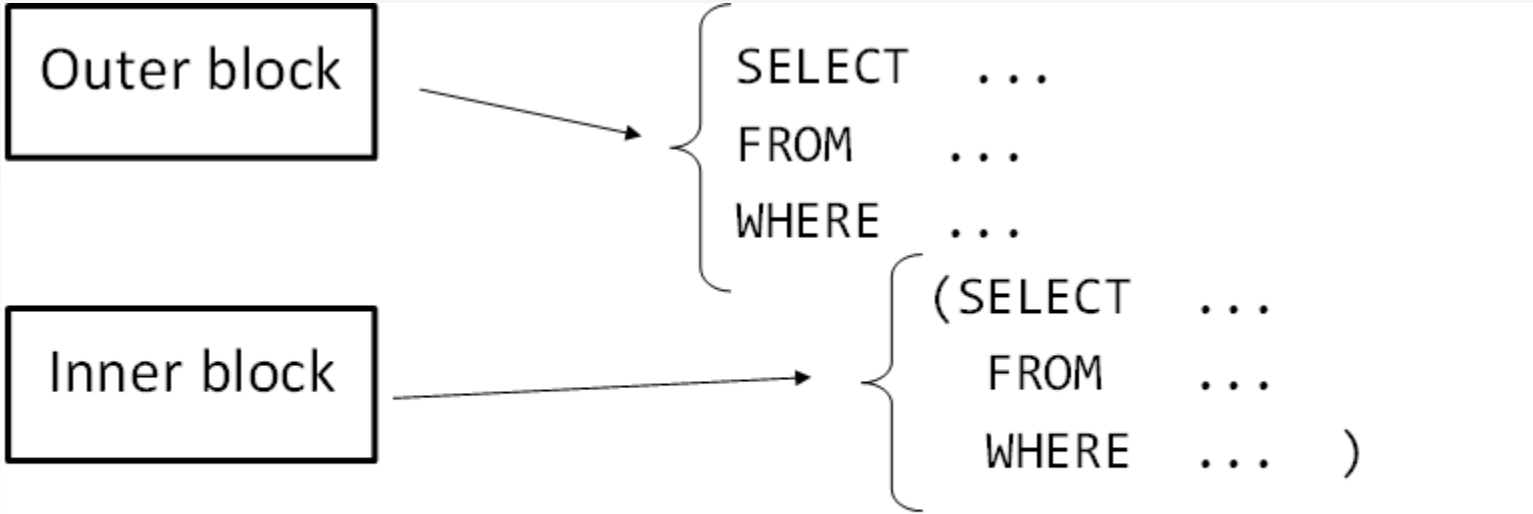
- Returns all (matched or unmatched) rows from the tables on both sides of the join clause

```
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!

sid	bid	bname
1	102	Pinta
2	102	Pinta
1	101	Nina
None	103	Santa Maria

Nested Queries



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

sid	sname	rating	age
1	Popeye	10	22
2	OliveOyl	11	39
3	Garfield	1	27
4	Bob	5	19

Reserves

sid	bid	day
1	102	9/12
2	102	9/13
1	101	10/01

sname
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)
```

sname
Popeye
OliveOyl

- Correlated subquery is recomputed for each Sailors tuple.

Nested Queries: Not Exists

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

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

sname
Garfield
Bob

- 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)
```

```
Q41: SELECT P1.PRODNR
    FROM PRODUCT P1
    WHERE 3 >
        (SELECT COUNT(*)
         FROM PRODUCT P2
         WHERE P1.PRODNR <
             P2.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)
```

Reading and Next Class

- SQL II: Ch 5
- Next: SQL III: Ch5