Chapter 5

Chapter 5

Algebraic and Logical Query Languages

We now switch our att
databases. We start in
guages, one algebraic ar

A	B
1	2
3	4
1	2
1	2

Figure 5.1: A bag

5.1.1 Why Bags?

Bag Semantics

- Bags (or Multisets):
  - Generalization of the notion of a set.
  - Members are allowed to appear more than once.

- Commercial DB's implement relations as Bags.
- Some relational operations are much more efficient when done on bags.

Projection w/ Bags

R1

K	A	B	C
4	2	0	6
5	2	0	5
1	1	3	8
2	1	3	7
3	2	3	3

- $\pi_{A,B}$(R1) -- efficiency

A	B
2	0
2	0
1	3
1	3
2	3

Bag Semantics

- Bags let us calculate some types of values more naturally.
  - Find the average B value
  - Projection onto B with sets yields (0,3)
    - Average = 3/2 = 1.5
  - Projection onto B with bags yields (0, 0, 3, 3, 3)
    - Average = 9/5 = 1.8

A	B
2	0
2	0
1	3
1	3
2	3

Bag Semantics
Selection

- Selection – Applied to each tuple independently.

Bag Semantics
Set Operations

- Union – Tuples from each relation added to bag.
- Intersection – Tuples in output are the min of the number of times they appear in each relation.
- Difference – Tuples are removed one for one with a minimum value of 0.

Bag Semantics
Joins

- Product – Tuples treated independently
- Joins – With multiple tuples having same values, we get multiple join possibilities.

Bag Semantics Example

- create table t1 (x char(1), y int);
- create table t2 (x char(1), z int);
- insert into t1 values (‘A’,2), (‘B’,3), (‘C’,4), (‘B’,7);
- insert into t2 values (‘B’, 0), (‘C’,1), (‘B’, 4);

x	y
A	2
B	3
C	4
B	7

t1

x	z
B	0
C	1
B	4

t2

Bag Semantics
Joins

- select * from t1 natural join t2;

x	y	z
B	3	0
B	3	4
C	4	1
B	7	0
B	7	4

x	y
A	2
B	3
C	4
B	7

t1

x	z
B	0
C	1
B	4

t2

Bag Semantics – Slightly Modified Joins

- select * from t1 natural join t2;

x	y	z
B	3	0
B	3	4
C	4	1
B	**3**	**0**
B	**3**	**4**

x	y
A	2
B	**3**
C	4
B	**3**

t1

x	z
B	0
C	1
B	4

t2

Relational Algebra – Advanced (Extended)

δ = eliminate duplicates from bags.

τ = sort tuples.

γ = grouping and aggregation.

Outerjoin :

avoids "dangling tuples" = tuples that do not join with anything.

Duplicate Elimination

- R1 = δ(R2)
  - Out relation R1 with a single copy of each tuple in R2.

Duplicate Elimination

- $R1 = \delta(R2)$

R2

x	y	z
B	3	0
B	3	4
C	4	1
B	3	0
B	3	4

$R1 = \delta(R2)$

x	y	z
B	3	0
C	4	1
B	3	4

Sort Tuples

- τ = sort tuples
- R1 := τ_L (R2).
  - *L* is a list of attributes from R2.
- R1:
  - List of tuples of R2
  - sorted first on the value of the first attribute on *L*
  - Sorted second on the second attribute of *L*
  - ….

Sort Tuples

- R1 := $\tau_{X,Z}$ (R2).

R2

x	y	z
B	3	0
C	4	1
B	3	4

$\tau_{X,Z}$ (R2)

x	y	z
B	3	0
B	3	4
C	4	1

Aggregation Operators

- Aggregation Operators:
  - are applied to a entire column of data.
  - Return a single value.
  - i.e., SUM, AVG, COUNT, MIN, and MAX.

Aggregation Operators

- Aggregation Operators:
  - SUM(K) = 15
  - AVG(K) = 3
  - COUNT(K) = 5
  - MIN(K) = 1
  - MAX(K) = 5

R1

K	A	B	C
4	2	0	6
5	2	0	5
1	1	3	8
2	1	3	7
3	2	3	3

γ – Grouping

- γ_L: (lowercase gamma)
  - Grouping Operator
- L is list of:
  - Individual Attributes (used for grouping)
  - Grouping Operators
    - (i.e., COUNT(), SUM(), …)

$\gamma_L(R)$ – Grouping

- Group tuples in R:
  - Form one group for every set of values of attributes from L in R.
  - For each aggregation operator in AGG() in L, apply AGG() to each group formed.
- Outputs:
  - One tuple for each set of values of attributes from L in R.
  - A single value for each AGG() applied to that group.

Grouping: $\gamma_{B,count(*)}$(R1)

R1

K	A	B	C
4	2	0	6
5	2	0	5
1	1	3	8
2	1	3	7
3	2	3	3

- $\gamma_{B,count(*)}$(R1)
- Group tuples in R1:
  - Form one group for every set of values of attribute ‘B’ in R1 (0, 3).
  - Apply Count(*) to each group formed.

Grouping: $\gamma_{B,count(*)}(R1)$

R1

K	A	B	C
4	2	0	6
5	2	0	5
1	1	3	8
2	1	3	7
3	2	3	3

- $\gamma_B(R1)$
- **Group tuples in R1:**
  - **Form one group for every set of values of attribute ‘B’ in R1 (0, 3).**
  - Apply Count(*) to each group formed.

Grouping: $\gamma_{B,count(*)}$(R1)

R1

B	Count(*)
0	2
3	3

- $\gamma_{B,count(*)}$(R1)
- **Group tuples in R1:**
  - **Form one group for every set of values of attribute 'B' in R1 (0, 3).**
  - **Apply Count(*) to each group formed.**

Outerjoins

- R1 $\bowtie$ R2
- Dangling Tuple:
  - A tuple of R1 with no corresponding tuple from R2 is said to be dangling.
  - A tuple of R2 with no corresponding tuple from R1 is also said to be dangling
  - Outerjoin preserves these tuples by padding them with null.

Outerjoins

- movie_list (mid, myear, mname);
- movie_ratings(pid, mid, rating);
- movie_list ⋈ movie_ratings

mid	myear	mname
979	1979	Movie 1
1079	1979	Movie 2
393	1993	Movie 3
1293	1993	Movie 4
300	2004	Movie 5

- movie_list

pid	mid	Rating
200	979	2
200	393	2
304	300	4

- movie_ratings

Outerjoins

- movie_list ⋈ movie_ratings

mid	myear	mname
979	1979	Movie 1
1079	1979	Movie 2
393	1993	Movie 3
1293	1993	Movie 4
300	2004	Movie 5

pid	mid	Rating
200	979	2
200	393	2
304	300	4

- movie_ratings

- movie_list

mid	myear	mname	pid	mid	Rating
979	1979	Movie 1	200	979	2
393	1993	Movie 3	200	393	2
300	2004	Movie 5	304	300	4

Outerjoins

- movie_list OUTERJOIN movie_ratings

mid	myear	mname
979	1979	Movie 1
1079	1979	Movie 2
393	1993	Movie 3
1293	1993	Movie 4
300	2004	Movie 5

pid	mid	Rating
200	979	2
200	393	2
304	300	4

- movie_ratings

- movie_list

mid	myear	mname	pid	mid	Rating
979	1979	Movie 1	200	979	2
1079	1979	Movie 2	null	null	null
393	1993	Movie 3	200	393	2
1293	1993	Movie 4	null	null	null
300	2004	Movie 5	304	300	4

SQL - Advanced

- Outer Joins
- Aggregations
- Eliminating Duplicates
- Grouping
- Having Clause
- Database Modifications

Joins - Revisited

- Join On (Theta Join)

SELECT *

FROM relation1 JOIN relation2 ON <condition1>

WHERE <condition2>;

Joins - Revisited

- Join Using

```
SELECT *
FROM relation1 JOIN relation2 USING (att1, …)
WHERE <condition>;
```

Outerjoins w/ SQL

- SELECT * from R1 NATURAL OUTER JOIN R2;
- Dangling Tuple:
  - A tuple of R1 with no corresponding tuple from R2 is said to be dangling.
  - A tuple of R2 with no corresponding tuple from R1 is also said to be dangling
  - Outer Join preserves these tuples by padding them with null.

Outerjoins

- Full outer join not available in MySQL - emulated
- SELECT * from R1 NATURAL LEFT JOIN R2 UNION SELECT * FROM R1 NATURAL RIGHT JOIN R2 ;
- Dangling Tuple:
  - A tuple of R1 with no corresponding tuple from R2 is said to be dangling. Left Join preserves these.
  - A tuple of R2 with no corresponding tuple from R1 is also said to be dangling. Right Join preserves these.
  - Tuples preserved by padding them with null.
  - NOTE: Second Query needs extra where clause.
    - (Reader Exercise)

Outerjoins

- movie_list (mid, myear, mname);
- movie_ratings(pid, mid, rating);
- Select * from
  movie_list natural join movie_ratings

mid	myear	mname
979	1979	Movie 1
1079	1979	Movie 2
393	1993	Movie 3
1293	1993	Movie 4
300	2004	Movie 5

pid	mid	Rating
200	979	2
200	393	2
304	300	4

- movie_ratings

- movie_list

Outerjoins

- Select * from movie_list natural join movie_ratings;

mid	myear	mname
979	1979	Movie 1
1079	1979	Movie 2
393	1993	Movie 3
1293	1993	Movie 4
300	2004	Movie 5

- movie_list

pid	mid	Rating
200	979	2
200	393	2
304	300	4

- movie_ratings

mid	myear	mname	pid	mid	Rating
979	1979	Movie 1	200	979	2
393	1993	Movie 3	200	393	2
300	2004	Movie 5	304	300	4

Outerjoins

- Select * from movie_list LEFT JOIN movie_ratings;

mid	myear	mname
979	1979	Movie 1
1079	1979	Movie 2
393	1993	Movie 3
1293	1993	Movie 4
300	2004	Movie 5

pid	mid	Rating
200	979	2
200	393	2
304	300	4

- movie_ratings

- movie_list

mid	myear	mname	pid	mid	Rating
979	1979	Movie 1	200	979	2
1079	1979	Movie 2	null	null	null
393	1993	Movie 3	200	393	2
1293	1993	Movie 4	null	null	null
300	2004	Movie 5	304	300	4

Relational Algebra – Extended Now In SQL

δ = eliminate duplicates from bags.

τ = sort tuples.

γ = grouping and aggregation.

Outerjoin : avoids "dangling tuples" = tuples that do not join with anything.

Bag Semantics Example

- create table t1 (x char(1), y int);
- create table t2 (x char(1), z int);
- insert into t1 values (‘A’,2), (‘B’,3), (‘C’,4), (‘B’,3);
- insert into t2 values (‘B’, 0), (‘C’,1), (‘B’, 4);

x	y
A	2
B	3
C	4
B	3

t1

x	z
B	0
C	1
B	4

t2

Bag Semantics
Joins

- select * from t1 natural join t2;

R2

x	y	z
B	3	0
B	3	4
C	4	1
B	3	0
B	3	4

x	y
A	2
B	3
C	4
B	7

t1

x	z
B	0
C	1
B	4

t2

Duplicate Elimination: Distinct

- SELECT * FROM R2;

R2

x	y	z
B	3	0
B	3	4
C	4	1
B	3	0
B	3	4

SELECT DISTINCT * FROM R2;

δ(R2)

x	y	z
B	3	0
C	4	1
B	3	4

Sort Tuples

- **τ** = sort tuples
- R1 := τ_L (R2).
  - *L* is a list of attributes from R2.
- R1:
  - List of tuples of R2
  - sorted first on the value of the first attribute on *L*
  - Sorted second on the second attribute of *L*
  - ….

Sort Tuples: Order By

SELECT DISTINCT * FROM R2;

x	y	z
B	3	0
C	4	1
B	3	4

SELECT DISTINCT * FROM R2 ORDER BY x,z;

$\tau_{x,z}$ (R2)

x	y	z
B	3	0
B	3	4
C	4	1

Aggregation Operators

- Aggregation Operators:
  - are applied to a entire column of data.
  - Return a single value.
  - i.e., SUM, AVG, COUNT, MIN, and MAX.

Aggregation Operators

- Aggregation Operators:
  - SUM(K) = 15
  - AVG(K) = 3
  - COUNT(K) = 5
  - MIN(K) = 1
  - MAX(K) = 5

R1

K	A	B	C
4	2	0	6
5	2	0	5
1	1	3	8
2	1	3	7
3	2	3	3

```
SELECT sum(k), avg(k), count(k),
   min(k), max(k)
FROM R1;
```

sum(k)	avg(k)	count(k)	min(k)	max(k)
15	3	5	1	5

γ – Grouping

- γ_L: (lowercase gamma)
  - Grouping Operator
- L is list of:
  - Individual Attributes (used for grouping)
  - Grouping Operators
    - (i.e., COUNT(), SUM(), …)

$\gamma_L(R)$ – Grouping

- Group tuples in R:
  - Form one group for every set of values of attributes from L in R.
  - For each aggregation operator in AGG() in L, apply AGG() to each group formed.
- Outputs:
  - One tuple for each set of values of attributes from L in R.
  - A single value for each AGG() applied to that group.

Grouping: $\gamma_{B,count(K)}(R1)$

R1

K	A	B	C
4	2	0	6
5	2	0	5
1	1	3	8
2	1	3	7
3	2	3	3

- $\gamma_B(R1)$
- Group tuples in R1:
  - Form one group for every set of values of attribute ‘B’ in R1 (0, 3).
  - Apply Count(K) to each group formed.

Grouping: $\gamma_{B,count(K)}(R1)$

R1

K	A	B	C
4	2	0	6
5	2	0	5
1	1	3	8
2	1	3	7
3	2	3	3

- $\gamma_{B,count(*)}(R1)$
- **Group tuples in R1:**
  - **Form one group for every set of values of attribute 'B' in R1 (0, 3).**
  - Apply Count(K) to each group formed.

Grouping: $\gamma_{B,count(K)}(R1)$

R1

K	A	B	C
4	2	0	6
5	2	0	5
1	1	3	8
2	1	3	7
3	2	3	3

- $\gamma_{B,\ count(*)}(R1)$:
  - SELECT B, Count(K) FROM R1
    GROUP BY B;
    - **Group tuples in R1:**
    - **Form one group for every set of values of attribute 'B' in R1 (0, 3).**
    - Apply Count(K) to each group formed.

Grouping: $\gamma_{B,count(*)}(R1)$

R1

B	Count(K)
0	2
3	3

- $\gamma_{B,\ count(*)}(R1)$
  - SELECT B, Count(K) FROM R1
    GROUP BY B;
    - **Group tuples in R1:**
    - **Form one group for every set of values of attribute 'B' in R1 (0, 3).**
    - **Apply Count(K) to each group formed.**

Iris Domain

- Iris Setosa

- Iris Versicolor

- Iris Virginica

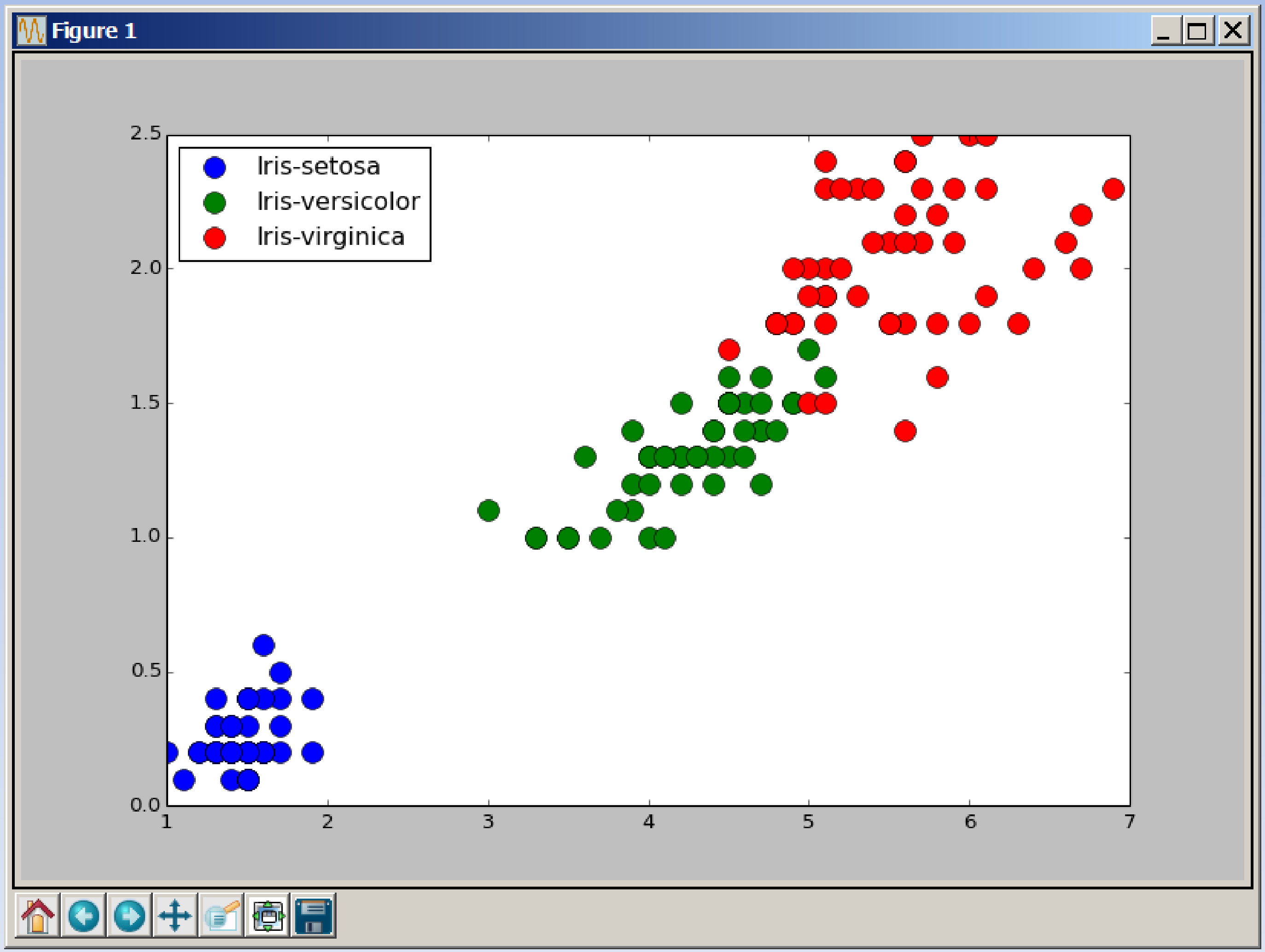
Iris_2d

petallength	petalwidth	class
1.4	0.2	Iris-setosa
1.4	0.2	Iris-setosa
1.3	0.2	Iris-setosa
1.5	0.2	Iris-setosa
1.4	0.2	Iris-setosa
3.9	1.1	Iris-versicolor
4.8	1.8	Iris-versicolor
4	1.3	Iris-versicolor
4.9	1.5	Iris-versicolor
4.7	1.2	Iris-versicolor
5.2	2.3	Iris-virginica
5	1.9	Iris-virginica
5.2	2	Iris-virginica
5.4	2.3	Iris-virginica
5.1	1.8	Iris-virginica

Iris_2d Domain

```
def test2():
   db = "iris2d"
   table = "iris_2d"
   x_item = 'petallength'
   y_item = 'petalwidth'
   plot(db, table, x_item, y_item, 'Iris-setosa')
   plot(db, table, x_item, y_item, 'Iris-versicolor')
   plot(db, table, x_item, y_item, 'Iris-virginica')
   matplotlib.pyplot.legend(loc='upper left', numpoints = 1)
   matplotlib.pyplot.show()
```

Iris_2d Domain



Iris_2d Domain

- iris_2d (petallength float, petalwidth float, class varchar(30));
- FIND: Average petal length and width for Iris Setosa, Iris Versicolor, Iris Virginica
  - SELECT class, avg(petallength), avg(petalwidth)
    FROM iris_2d
    GROUP BY class
    ORDER BY avg(petallength), avg(petalwidth);

class	avg(petallength)	avg(petalwidth)
Iris-setosa	1.463999996	0.244000005
Iris-versicolor	4.259999981	1.325999992
Iris-virginica	5.551999989	2.025999978

Having Clause

- HAVING <condition>
  - Can follow a GROUP BY clause
  - Condition is applied to each group
  - Groups not satisfying condition are not included in query.

Iris_2d Domain

- iris_2d (petallength float, petalwidth float, class varchar(30));
- Average petal length and width for Iris Setosa, Iris Versicolor, Iris Virginica
  - SELECT class, avg(petallength), avg(petalwidth)
    FROM iris_2d
    GROUP BY class
    HAVING min(petallength) > 1;

class	avg(petallength)	avg(petalwidth)
Iris-versicolor	4.259999981	1.325999992
Iris-virginica	5.551999989	2.025999978

Having Clause

- Same Constraints as Select w/ Group By
  - Anything goes w/ a subquery
  - Grouping Attributes w/ condition.
  - Grouping Operators w/ condition.

Iris Domain

- Iris Setosa

- Iris Versicolor

- Iris Virginica

Iris_2d

petallength	petalwidth	class
1.4	0.2	Iris-setosa
1.4	0.2	Iris-setosa
1.3	0.2	Iris-setosa
1.5	0.2	Iris-setosa
1.4	0.2	Iris-setosa
3.9	1.1	Iris-versicolor
4.8	1.8	Iris-versicolor
4	1.3	Iris-versicolor
4.9	1.5	Iris-versicolor
4.7	1.2	Iris-versicolor
5.2	2.3	Iris-virginica
5	1.9	Iris-virginica
5.2	2	Iris-virginica
5.4	2.3	Iris-virginica
5.1	1.8	Iris-virginica

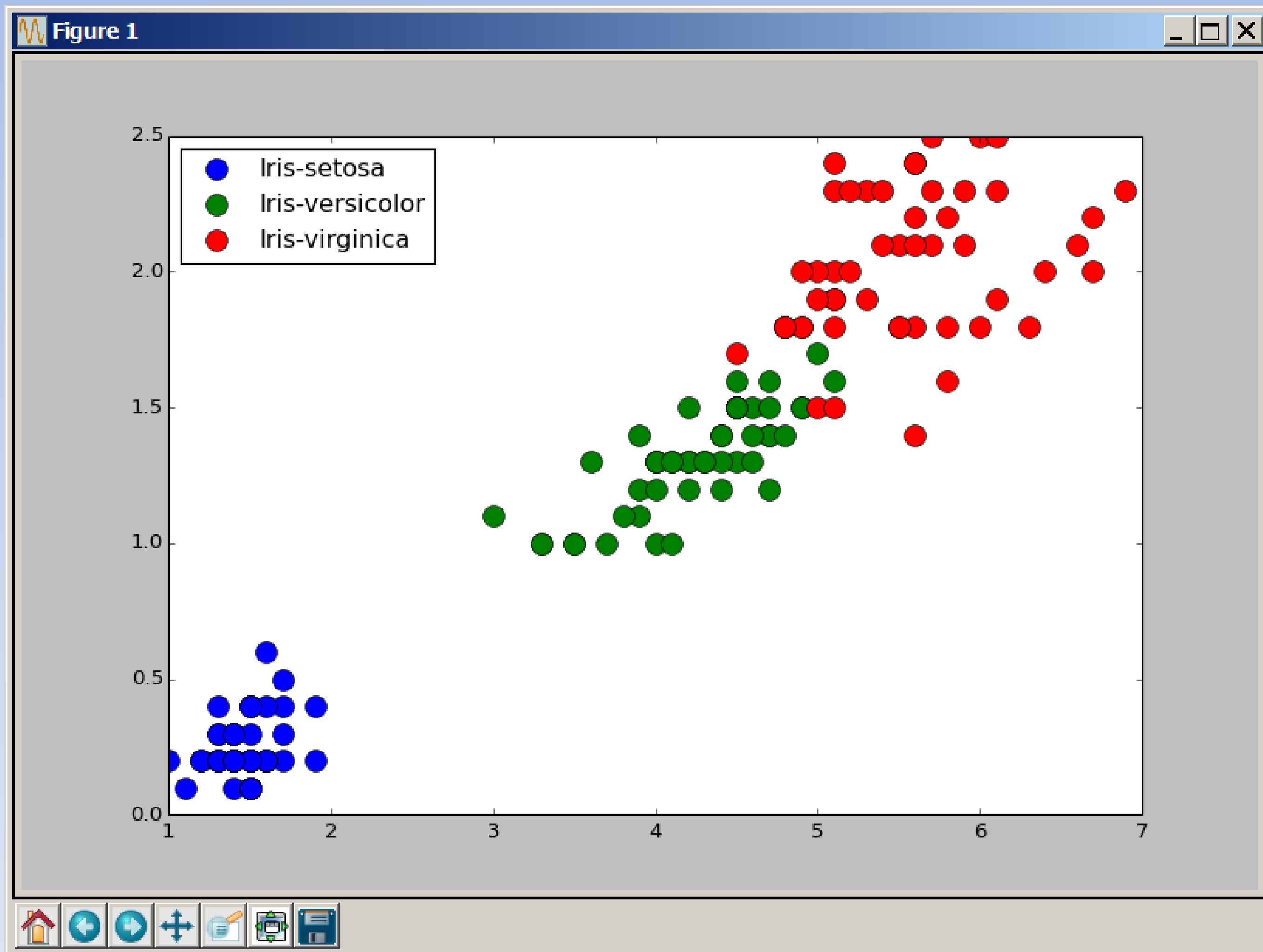
Iris_2d Domain

```
def plot(db, table, x_item, y_item, ciris):
  conn = mysql.connector.connect (host = "localhost",  user="root",  passwd = "cs126",  db = db)
  cursor = conn.cursor ()
  s = "select " +x_item+","+y_item+" from " + table \
              + " where class like '"+ciris+"%'"
  cursor.execute (s)
  row = cursor.fetchone ()
  X = y= []
while row != None:
    X.append(row[0])
    Y.append(row[1])
    row = cursor.fetchone ()
  cursor.close ()
  conn.close ()
  matplotlib.pyplot.plot(X,Y,'o', markersize=12, label= ciris)
```

Iris_2d Domain

```
def test2():
    db = "iris2d"
    table = "iris_2d"
    x_item = 'petallength'
    y_item = 'petalwidth'
    plot(db, table, x_item, y_item, 'Iris-setosa')
    plot(db, table, x_item, y_item, 'Iris-versicolor')
    plot(db, table, x_item, y_item, 'Iris-virginica')
    matplotlib.pyplot.legend(loc='upper left', numpoints = 1)
    matplotlib.pyplot.show()
```

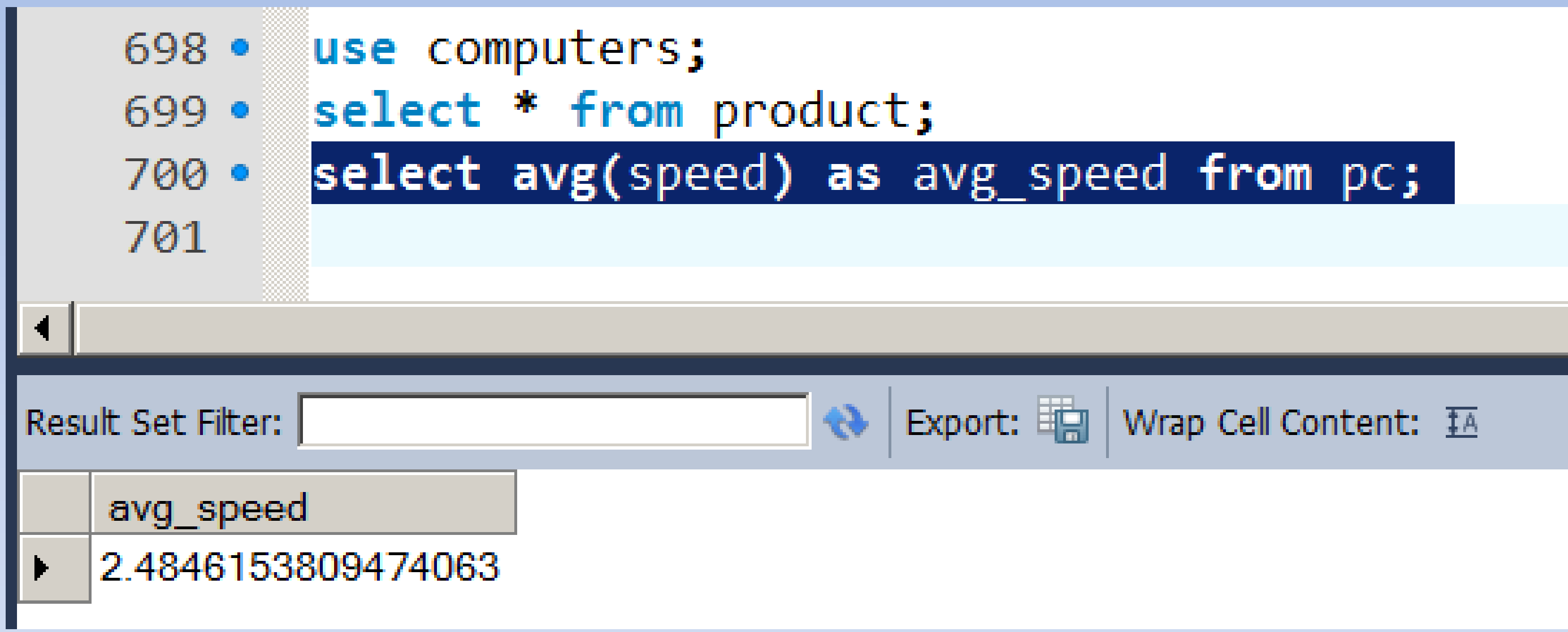
Iris_2d Domain



Questions 6.4.6

- Product(maker, model, type)
- PC(model, speed, ram, hd, price)
- Laptop(model, speed, ram, hd, screen, price)
- Printer(model, color, type, price)
- a) Find the average speed of PC's .

Questions 6.4.6



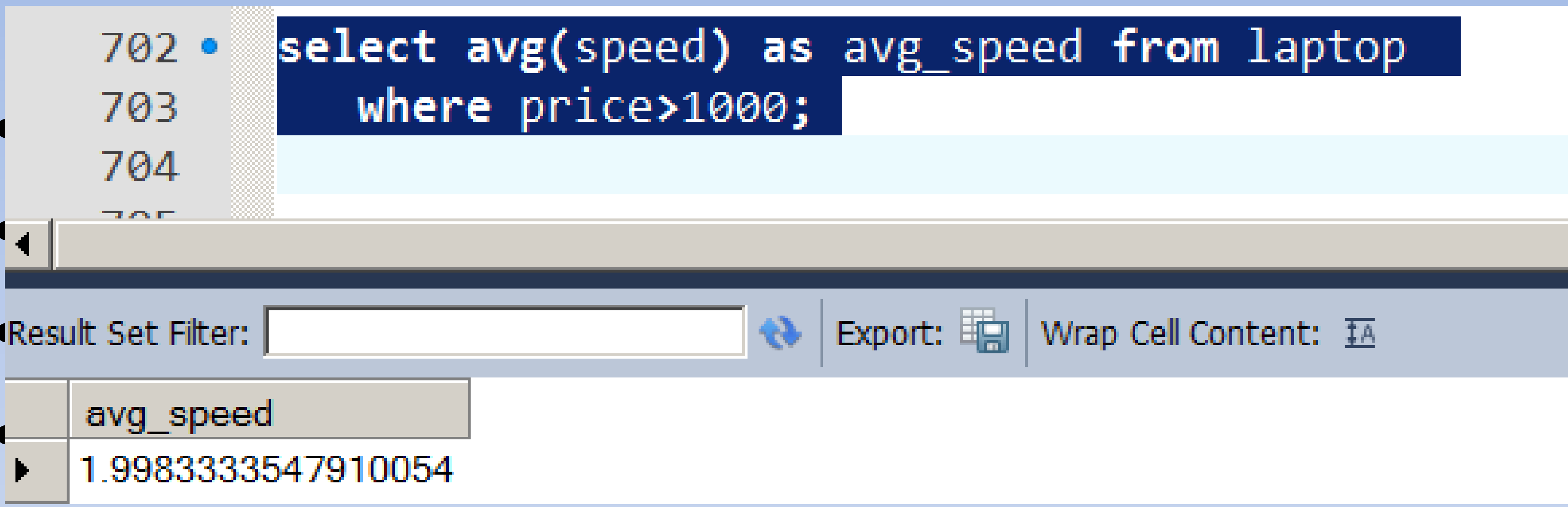
)

select avg(speed) as avg_speed from pc;

Questions 6.4.6

- Product(maker, model, type)
- PC(model, speed, ram, hd, price)
- Laptop(model, speed, ram, hd, screen, price)
- Printer(model, color, type, price)
- b) Find the average speed of laptops costing over $1000.

Questions 6.4.6



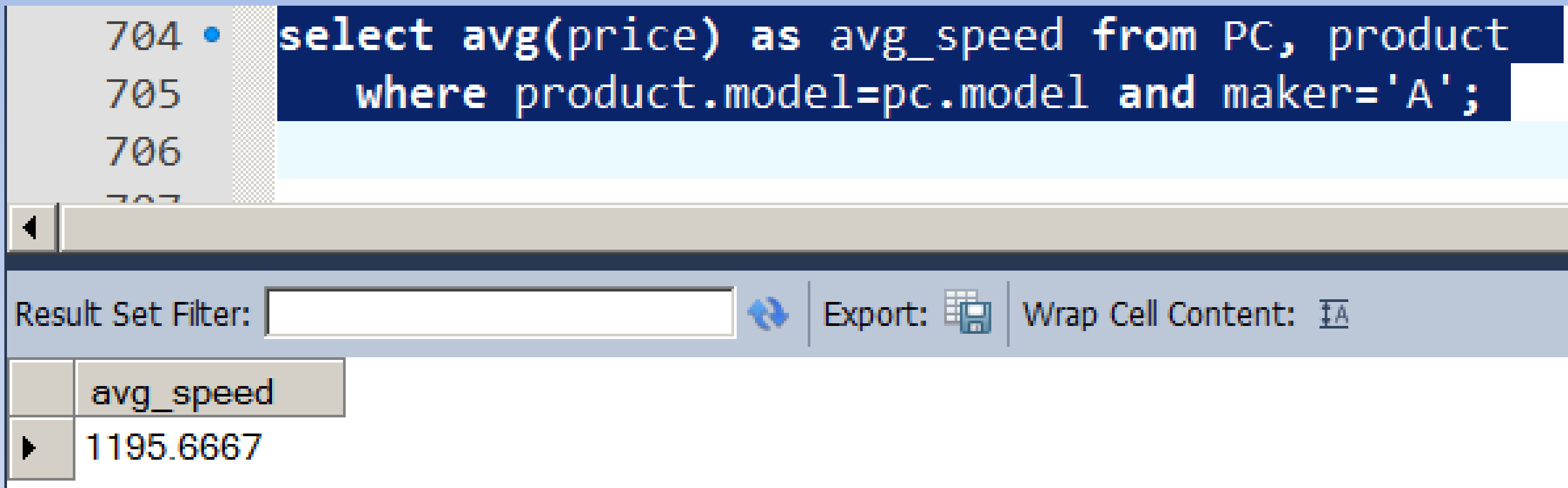
- b) Find the average speed of laptops costing over $1000.

select avg(speed) as avg_speed from laptop
where price>1000;

Questions 6.4.6

- Product(maker, model, type)
- PC(model, speed, ram, hd, price)
- Laptop(model, speed, ram, hd, screen, price)
- Printer(model, color, type, price)
- c) Find the average price of PC's made by manufacturer “A ."

Questions 6.4.6



c) Find the average price of PC's made by manufacturer "A ."

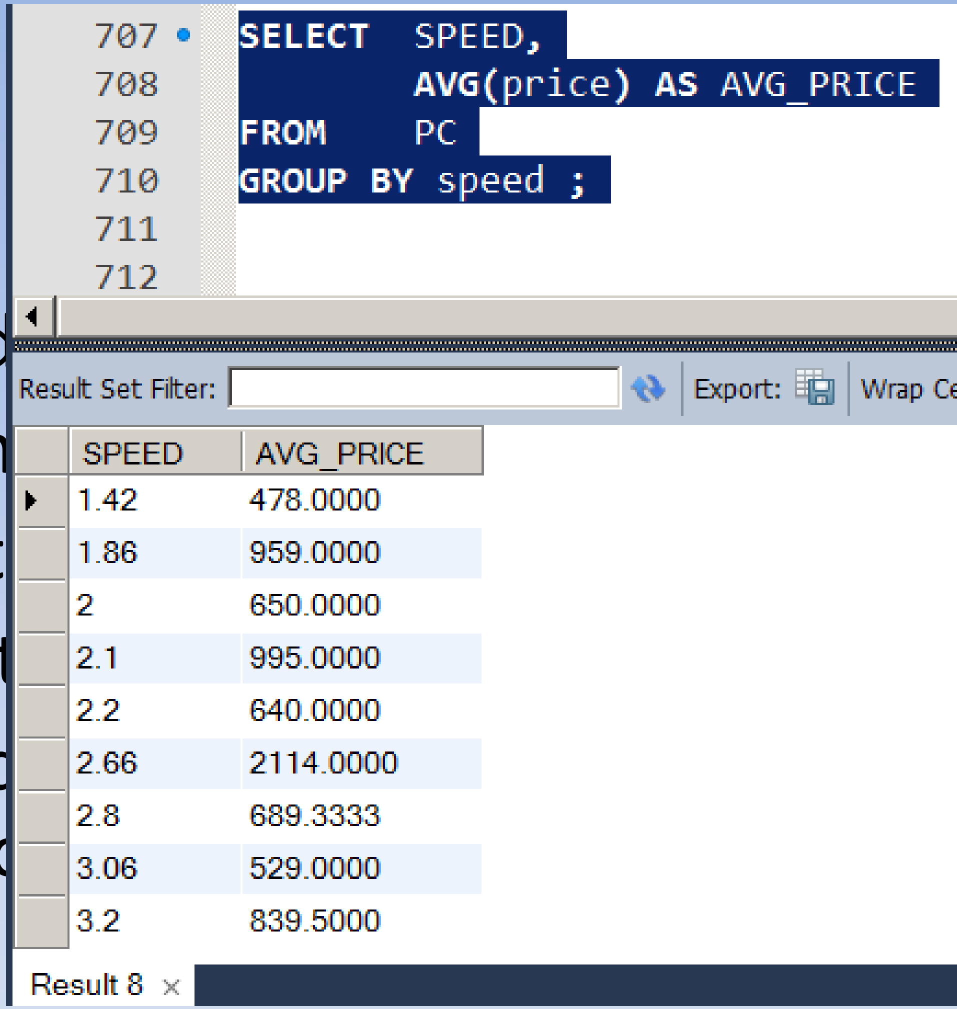
select avg(price) as avg_price from PC, product

where product.model=pc.model and maker='A';

Questions 6.4.6

- Product(maker, model, type)
- PC(model, speed, ram, hd, price)
- Laptop(model, speed, ram, hd, screen, price)
- Printer(model, color, type, price)
- c) Find , for each different speed, the average price of a PC .

- Prod
- PC(n
- Lapt reen, price)
- Print
- c) Find e average price c



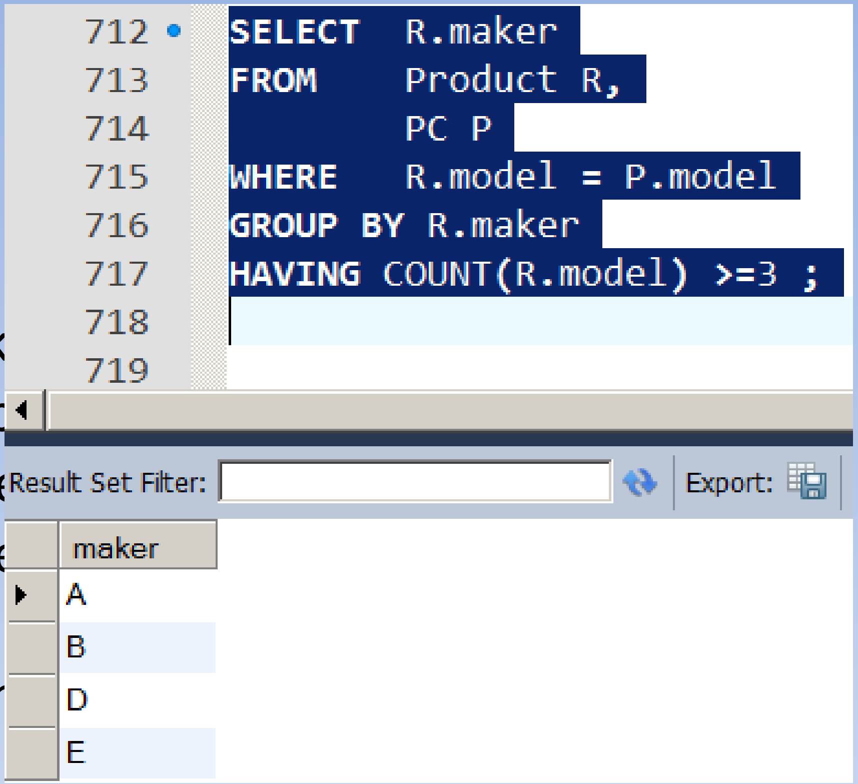
select speed, avg(price) as avg_price from PC group by speed ;

Questions 6.4.6

- Product(maker, model, type)
- PC(model, speed, ram, hd, price)
- Laptop(model, speed, ram, hd, screen, price)
- Printer(model, color, type, price)

- g) Find the manufacturers that make at least three different models of PC .

- Product(mak
- PC(model, sp
- Laptop(mode
- Printer(mode



- g) Find the mar different models of PC.

SELECT R.maker

FROM Product R,

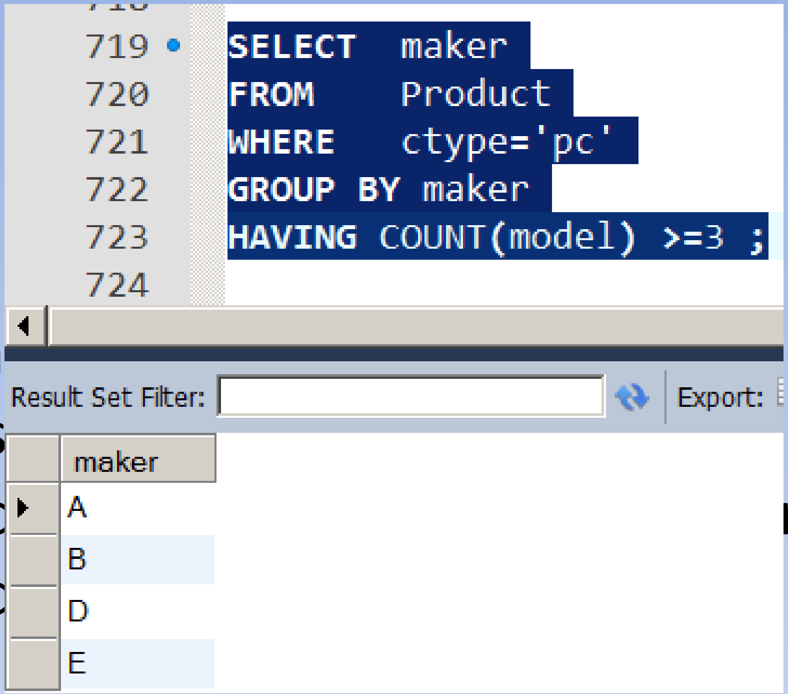
PC P

WHERE R.model = P.model

GROUP BY R.maker

HAVING COUNT(R.model) >=3 ;

- Product(ma
- PC(model, s
- Laptop(moc rice)
- Printer(moc



- g) Find the manufacturers that make at least three different models of PC.

SELECT maker

FROM Product

WHERE ctype='pc'

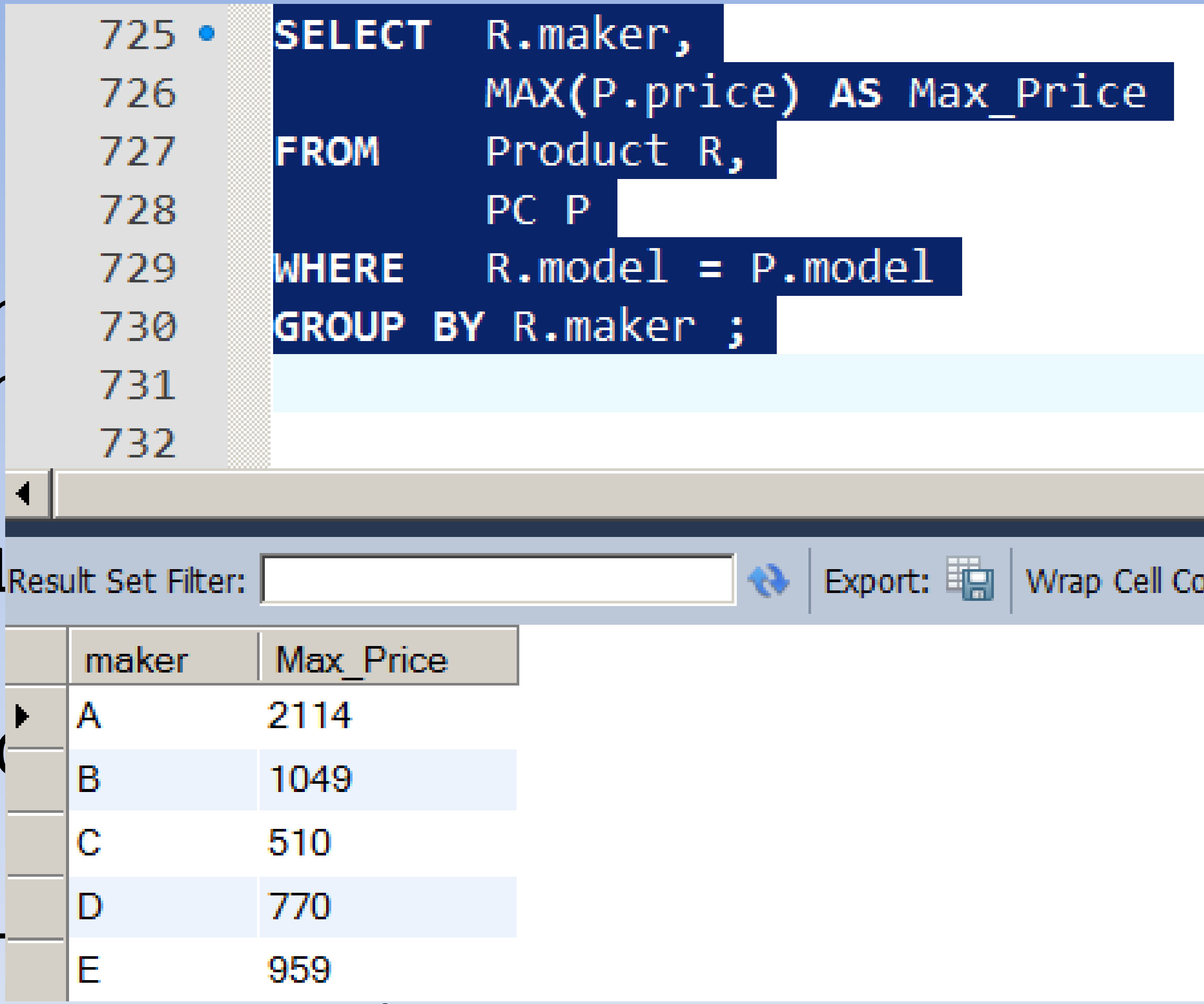
GROUP BY maker

HAVING COUNT(model) >=3 ;

In-Class

- h) Find for each manufacturer who sells PC's the maximum price of a PC.

• I … s the
r



SEL

FR

WH

GROUP BY R.maker ;