



Politecnico
di Torino

DBG
MG

Nested queries

SQL language: basics

Nested queries

- Introduction
- The IN operator
- The NOT IN operator
- The tuple constructor
- The EXISTS operator
- The NOT EXISTS operator
- Correlation among queries
- The division operation



Politecnico
di Torino



Introduction

Nested queries

Introduction

- A nested query is a **SELECT** statement contained within another query
 - query nesting allows decomposing a complex problem into simpler subproblems
- **SELECT** statements may be introduced
 - within a predicate in the **WHERE** clause
 - within a predicate in the **HAVING** clause
 - in the **FROM** clause

Example database: Supply-Product

P

<u>PId</u>	PName	Color	Size	Store
P1	Jumper	Red	40	London
P2	Jeans	Green	48	Paris
P3	Blouse	Blue	48	Rome
P4	Blouse	Blue	44	London
P5	Skirt	Blue	40	Paris

S

<u>SId</u>	SName	#Employees	City
S1	Smith	20	London
S2	Jones	10	Paris
S3	Blake	30	Paris
S4	Clark	20	London
S5	Adams	30	Athens

SP

<u>SId</u>	<u>PId</u>	Qty
S1	P1	300
S1	P2	200
S1	P3	400
S1	P4	200
S1	P5	100
S2	P1	300
S2	P2	400
S3	P2	200
S4	P3	200
S4	P4	300
S4	P5	400

Foreign key

Foreign key

Nested queries (no.1)

- Find the codes of the suppliers that are based in the same city as S1

- By using a formulation with nested queries, the problem may be decomposed into two subproblems
 - city of supplier S1
 - codes of the suppliers based in the same city

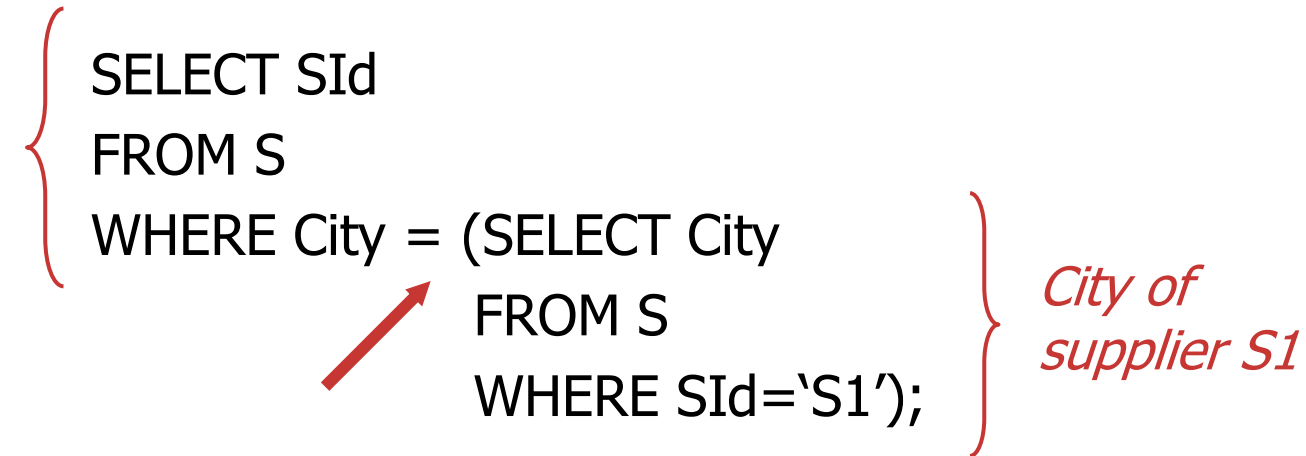
Nested queries (no.1)

- Find the codes of the suppliers that are based in the same city as S1

IDs of the suppliers based in the same city as S1

```
SELECT SId
FROM S
WHERE City = (SELECT City
              FROM S
              WHERE SId='S1');
```

City of supplier S1



- The '=' operator may be used only if it is known in advance that the inner SELECT statement always returns **a single** value
- An equivalent formulation may be defined using a join operation

Equivalent formulation

- The equivalent formulation with join is characterized by
 - a **FROM** clause including all the tables referenced by the **FROM** clauses of each **SELECT** statement
 - appropriate join conditions in the **WHERE** clause
 - if needed selection predicates added in the **WHERE** clause

FROM clause (no.1)

- Find the codes of the suppliers that are based in the same city as S1

```
SELECT SId
FROM (S) ← SX
WHERE City = (SELECT City
              FROM (S) ← SY
              WHERE SId='S1');
```

FROM clause (no.1)

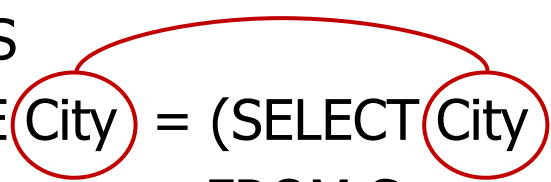
- Find the codes of the suppliers that are based in the same city as S1

```
SELECT ...  
FROM S AS SX, S AS SY  
...
```

Join condition (no.1)

- Find the codes of the suppliers that are based in the same city as S1

```
SELECT SId
FROM S
WHERE City = (SELECT City
              FROM S
              WHERE SId='S1');
```



Join condition (no.1)

- Find the codes of the suppliers that are based in the same city as S1

```
SELECT ...  
FROM S AS SX, S AS SY  
WHERE SX.City=SY.City  
...
```

Selection predicate (no.1)

- Find the codes of the suppliers that are based in the same city as S1

```
SELECT SId
FROM S
WHERE City = (SELECT City
              FROM S
              WHERE SId='S1');
```

SELECT clause (no.1)

- Find the codes of the suppliers that are based in the same city as S1

```
SELECT SY.SId
FROM S AS SX, S AS SY
WHERE SX.City=SY.City AND
      SX.SId='S1';
```

Equivalent formulation (no.2)

- Find the codes of the suppliers whose number of employees is smaller than the maximum number of employees

```
SELECT SId
FROM S
WHERE #Employees < (SELECT MAX(#Employees)
                     FROM S);
```

- An equivalent formulation with join is not possible

IN OPERATOR

- It expresses the concept of membership to a set of values

AttributeName **IN** (*NestedQuery*)

- It allows to write a query by
 - breaking down the problem into subproblems
 - following a "bottom-up" process
- The nested query can be replaced with a list of values
- The equivalent formulation with the join is characterized by
 - **FROM** clause containing the tables referenced in the **FROM** of all **SELECTs**
 - appropriate join conditions in the **WHERE** clause
 - any selection predicates added in the **WHERE** clause

The IN operator (no.1)

- Find the names of the suppliers who supply product P2
- Decomposition of the problem into two subproblems
 - codes of the suppliers of product P2
 - names of the suppliers with such codes

The IN operator (no.1)

- Find the names of the suppliers who supply product P2

SP

<u>SId</u>	<u>PId</u>	Qty
S1	P1	300
S1	P2	200
S1	P3	400
S1	P4	200
S1	P5	100
S1	P6	100
S2	P1	300
S2	P2	400
S3	P2	200
S4	P3	200
S4	P4	300
S4	P5	400



<u>SId</u>
S1
S2
S3

```
(SELECT SId  
FROM SP  
WHERE PId='P2')
```

*Codes
of the
suppliers
of P2*

The IN operator (no.1)

- Find the names of the suppliers who supply product P2

S

<u>SId</u>	SName	#Employees	City
S1	Smith	20	London
S2	Jones	10	Paris
S3	Blake	30	Paris
S4	Clark	20	London
S5	Adams	30	Athens

```
SELECT SName
FROM S
WHERE SId IN (SELECT SId
              FROM SP
              WHERE PId='P2');
```

Set membership

Codes of the suppliers of P2

<u>SId</u>
S1
S2
S3

Example 1: Equivalent Formulation with Join

- Find the names of the suppliers who supply product P2

IN

```
SELECT SName
FROM S
WHERE SId IN (SELECT SId
              FROM SP
              WHERE PId='P2');
```

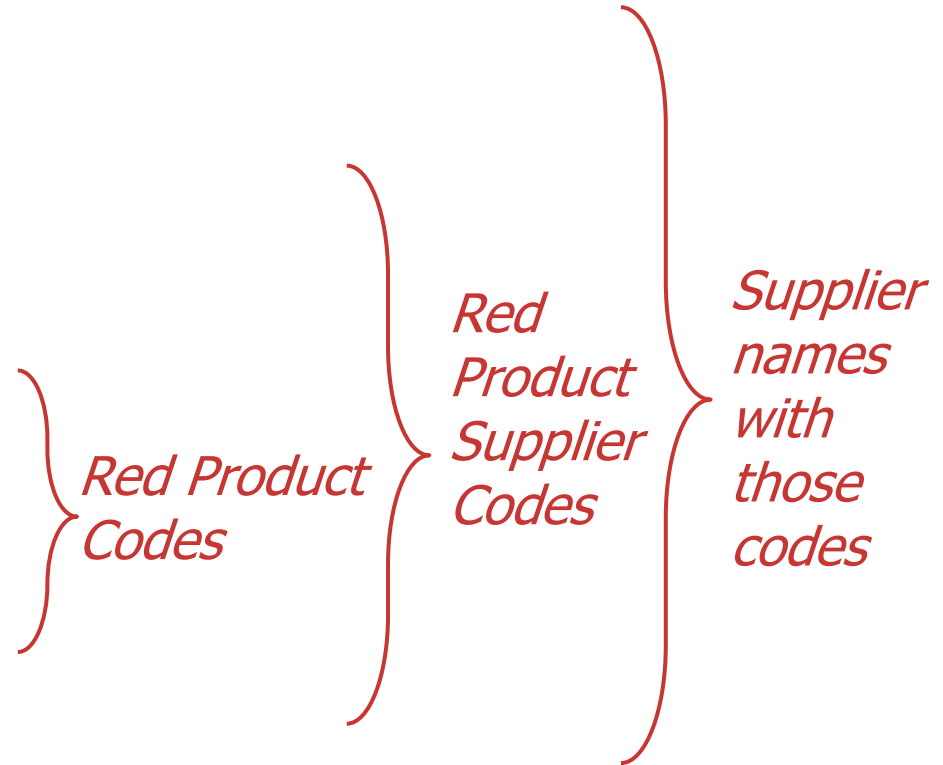
JOIN

```
SELECT SName
FROM S, SP
WHERE S.SId=SP.SId
      AND PId='P2';
```

Example 2: IN Operator

- Find the name of suppliers who provide at least one red product

```
SELECT SName
FROM S
WHERE SId IN (SELECT SId
              FROM SP
              WHERE PId IN (SELECT PId
                           FROM P
                           WHERE Color='Red'));
```

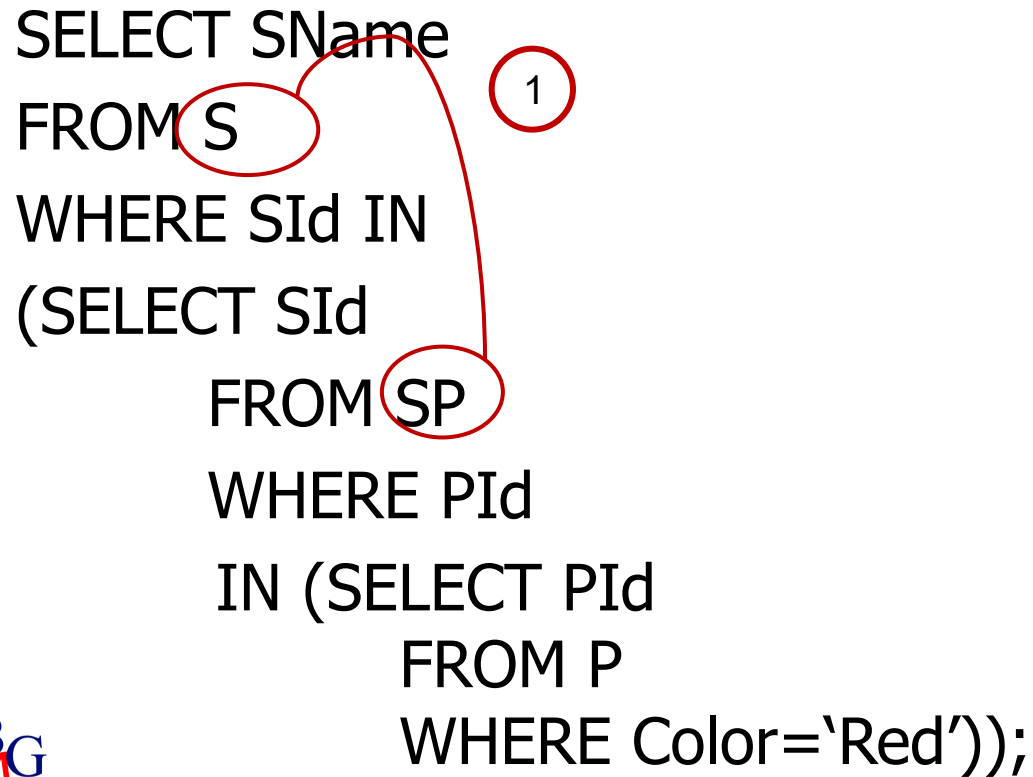


Example 2: Equivalent formulation

- Find the name of suppliers who provide at least one red product

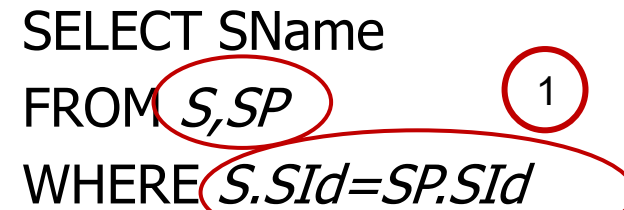
IN

```
SELECT SName
FROM S
WHERE SId IN
(SELECT SId
 FROM SP
 WHERE PId
 IN (SELECT PId
      FROM P
      WHERE Color='Red'));
```



JOIN

```
SELECT SName
FROM S,SP
WHERE S.SId=SP.SId
```

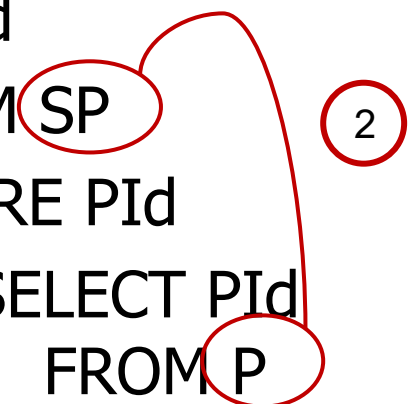


Example 2: Equivalent formulation

- Find the name of suppliers who provide at least one red product

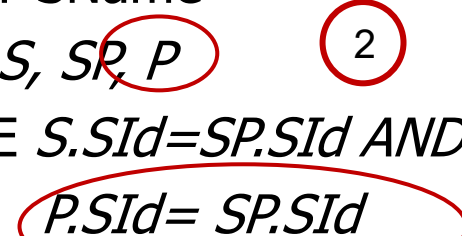
IN

```
SELECT SName
FROM S
WHERE SId IN
(SELECT SId
FROM SP
WHERE PId
IN (SELECT PId
FROM P
WHERE Color='Red'));
```



JOIN

```
SELECT SName
FROM S, SP, P
WHERE S.SId=SP.SId AND
P.SId= SP.SId
```



Example 2: Equivalent formulation

- Find the name of suppliers who provide at least one red product

IN

```
SELECT SName
FROM S
WHERE SId IN
(SELECT SId
 FROM SP
 WHERE PId
 IN (SELECT PId
     FROM P
     WHERE Color='Red'));
```

JOIN

```
SELECT SName
FROM S, SP, P
WHERE S.SId=SP.SId AND
      P.SId= SP.SId AND
      Color = 'Red'
```

3

3

NOT IN OPERATOR

- It expresses the concept of exclusion from a set of values

AttributeName **NOT IN** (*NestedQuery*)

- It requires the identification of an appropriate *set to be excluded* defined by
 - a nested query
 - a list of values
- There is no equivalent formulation with join

Example 1: Concept of exclusion

- Find the names of the suppliers who *do not* supply product P2
 - it is not possible to express the query with a join operation

```
SELECT SName  
FROM S, SP  
WHERE S.SId = SP.SId  
AND PId <>'P2';
```

Wrong solution

- The query matches the request:
 - Find the name of suppliers who provide at least one product other than P2


Wrong solution (no.1)

- Find the names of the suppliers who *do not* supply product P2

S

<u>SId</u>	SName	#Employees	City
S1	Smith	20	London
S2	Jones	10	Paris
S3	Blake	30	Paris
S4	Clark	20	London
S5	Adams	30	Athens

R



SName
Smith
Jones
Clark

SP

<u>SId</u>	<u>PId</u>	Qty
S1	P1	300
S1	P2	200
S1	P3	400
S1	P4	200
S1	P5	100
S2	P1	300
S2	P2	400
S3	P2	200
S4	P3	200
S4	P4	300
S4	P5	400

The NOT IN operator (no.1)

- Find the names of the suppliers who *do not* supply product P2
- Set to be excluded
 - suppliers of product P2

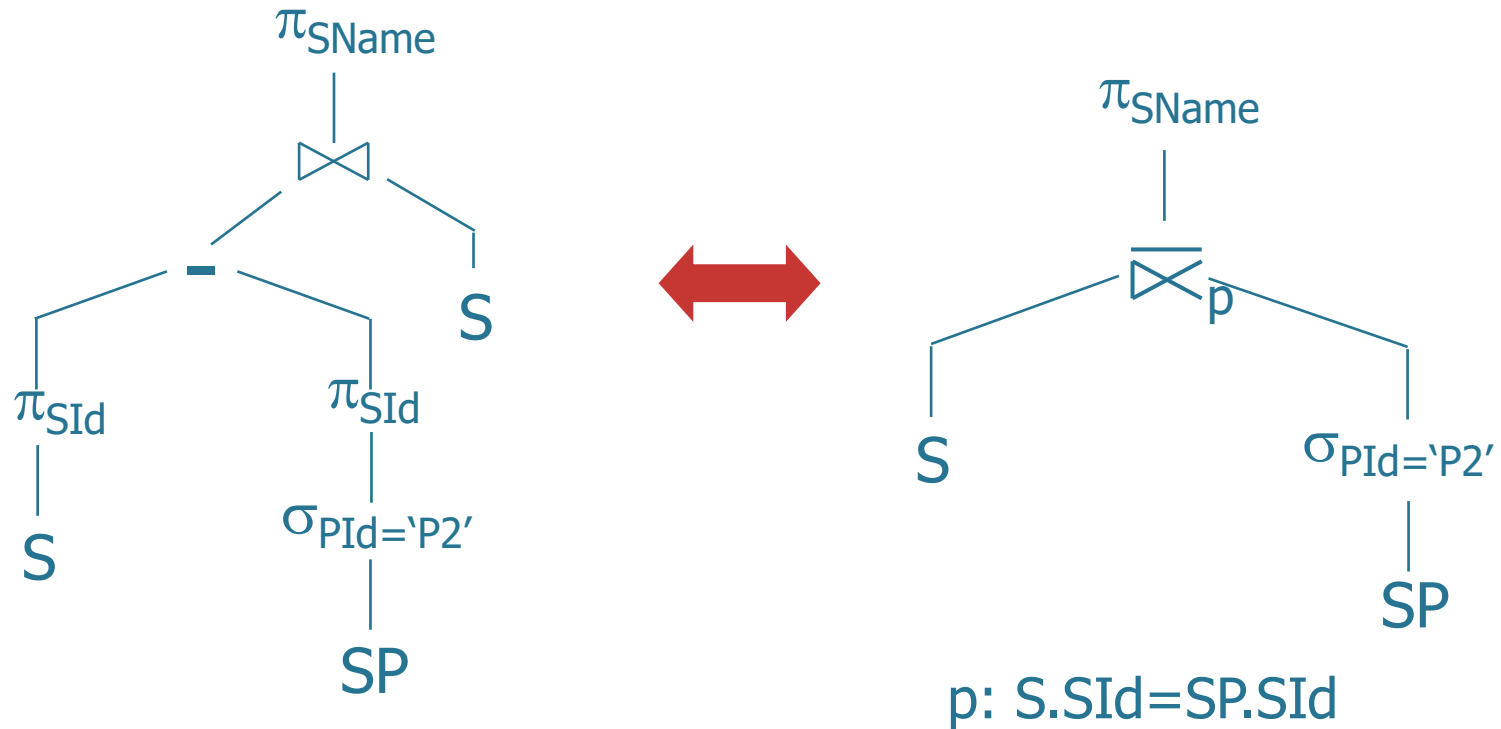
```
SELECT SName
FROM S
WHERE SId NOT IN (SELECT SId
                  FROM SP
                  WHERE PId='P2');
```

does not belong to

*Codes of the suppliers
who supply P2*

NOT IN and relational algebra (no.1)

- Find the names of the suppliers who *do not* supply product P2



The NOT IN operator (no.2)

- Find the names of the suppliers who **only** supply product P2



*Find the names of the suppliers of P2 who have **never** supplied products **other than** P2*

- Set to be excluded
 - suppliers of products other than P2

The NOT IN operator (no.2)

- Find the names of the suppliers who only supply product P2

```
SELECT SName
FROM S, SP
WHERE S.SId NOT IN (SELECT SId
                    FROM SP
                    WHERE PId<>'P2')
AND S.SId=SP.SId;
```

*Codes of the suppliers
who supply
at least one product
other than P2*

Alternative solution (no.2)

- Find the names of the suppliers who only supply product P2

```
SELECT SName
FROM S
WHERE S.SId NOT IN (SELECT SId
                    FROM SP
                    WHERE PId<>'P2')
AND S.SId IN (SELECT SId
              FROM SP);
```

*Codes of the suppliers
who supply
at least one product
other than P2*

The NOT IN operator (no.3)

- Find the names of the suppliers who *do not* supply any red products
- Set to be excluded:
 - suppliers of red products, identified by their codes

```
SELECT SName  
FROM S  
WHERE SId NOT IN
```

*Codes of the
suppliers of at least one
red product*

```
(SELECT SId  
FROM SP  
WHERE PId IN (SELECT PId  
FROM P  
WHERE Color='Red'));
```

Wrong alternative (no.3)

- Find the names of the suppliers who *do not* supply any red products

```
SELECT SName
FROM S
WHERE SId IN
  (SELECT SId
   FROM SP
   WHERE PId NOT IN (SELECT PId
                     FROM P
                     WHERE Color='Red'));
```

Codes of the suppliers of non-red products

- The set of elements to be excluded is incorrect

Wrong alternative (no.3)

- Find the names of the suppliers who *do not* supply any red products

```
SELECT SName  
FROM S  
WHERE SId IN  
    (SELECT SId  
     FROM SP  
     WHERE PId NOT IN (SELECT PId  
                       FROM P  
                       WHERE Color='Red'));
```

*Codes of the
suppliers of
non-red
products*

- The set of elements to be excluded is incorrect

Wrong alternative (no.3)

- Find the names of the suppliers who *do not* supply any red products

P

<u>PId</u>	PName	Color	Size	Store
P1	Jumper	Red	40	London
P2	Jeans	Green	48	Paris
P3	Blouse	Blue	48	Rome
P4	Blouse	Blue	44	London
P5	Skirt	Blue	40	Paris

S

<u>SId</u>	SName	#Employees	City
S1	Smith	20	London
S2	Jones	10	Paris
S3	Blake	30	Paris
S4	Clark	20	London
S5	Adams	30	Athens

SP

<u>SId</u>	<u>PId</u>	Qty
S1	P1	300
S1	P2	200
S1	P3	400
S1	P4	200
S1	P5	100
S2	P1	300
S2	P2	400
S3	P2	200
S4	P3	200
S4	P4	300
S4	P5	400

TUPLE CONSTRUCTOR

- It allows defining a temporary structure for a tuple
 - the attributes belonging to it must be listed within ()

(AttributeName₁, AttributeName₂, ...)

- It enhances the expressive power of the **IN** and **NOT IN** operators

Example (no.1)

TRIP (TId, StartingPlace, Destination,
DepartureTime, ArrivalTime)

- Find the pairs of starting places and destinations for which none of the trips lasts more than 2 hours

Example (no.1)

TRIP (TId, StartingPlace, Destination,
DepartureTime, ArrivalTime)

- Find the pairs of starting places and destinations for which none of the trips lasts more than 2 hours

```
SELECT StartingPlace, Destination
FROM TRIP
WHERE (StartingPlace, Destination) NOT IN
      (SELECT StartingPlace, Destination
       FROM TRIP
       WHERE ArrivalTime-DepartureTime>2);
```

Tuple constructor →

EXISTS OPERATOR

- The EXISTS operator admits **a nested query as a parameter** and **returns**
 - **true** if the nested query returns a **non-empty** set (that is, it returns at least one tuple)
 - **false** if the internal query returns the **empty** set (i.e., no tuple)
- In the internal query of EXISTS, the SELECT clause is mandatory, but irrelevant, because the attributes are not displayed
- The **correlation condition** ties the execution of the internal query to the values of the attributes of the current tuple in the external query

The EXISTS operator (no.1)

- Find the names of the suppliers of product P2



*Find the names of the suppliers **for which there exists** a product supply for P2*

Correlation condition (no.1)

- Find the names of the suppliers of product P2

```
SELECT SName
FROM S
WHERE EXISTS (SELECT *
              FROM SP
              WHERE PId='P2'
              AND SP.SId=S.SId );
```

Correlation condition

- The correlation condition **ties the execution of the internal query** to the values of the attributes of the **current tuple** in the external query

How EXISTS works (no.1)

- Find the names of the suppliers of product P2

S

SId	SName	#Employees	City
S1	Smith	20	London
S2	Jones	10	Paris
S3	Blake	30	Paris
S4	Clark	20	London
S5	Adams	30	Athens

SP

SId	PId	Qty
S1	P1	300
S1	P2	200
S1	P3	400
S1	P4	200
S1	P5	100
S2	P1	300
S2	P2	400
S3	P2	200
S4	P3	200
S4	P4	300
S4	P5	400

```
SELECT *  
FROM SP  
WHERE PId='P2'  
AND SP.SId='S1'
```

↑
*Value of SId in the
current line of table S*

How EXISTS works (no.1)

- Find the names of the suppliers of product P2

S

SId	SName	#Employees	City
S1	Smith	20	London
S2	Jones	10	Paris
S3	Blake	30	Paris
S4	Clark	20	London
S5	Adams	30	Athens

SP

SId	PId	Qty
S1	P1	300
S1	P2	200
S1	P3	400
S1	P4	200
S1	P5	100
S2	P1	300
S2	P2	400
S3	P2	200
S4	P3	200
S4	P4	300
S4	P5	400

- The predicate including **EXISTS** is true for S1 since there exists a supply for P2 by S1
 - S1 belongs to the result of the query

How EXISTS works (no.1)

- Find the names of the suppliers of product P2

S

<u>SId</u>	SName	#Employees	City
S1	Smith	20	London
S2	Jones	10	Paris
S3	Blake	30	Paris
S4	Clark	20	London
S5	Adams	30	Athens

SP

<u>SId</u>	<u>PId</u>	Qty
S1	P1	300
S1	P2	200
S1	P3	400
S1	P4	200
S1	P5	100
S2	P1	300
S2	P2	400
S3	P2	200
S4	P3	200
S4	P4	300
S4	P5	400

- The predicate including **EXISTS** is false for S4 since there does not exist a supply for P2 by S4
 - S4 does not belong to the result of the query

Result of the query (no.1)

- Find the names of the suppliers of product P2

R

SName
Smith
Jones
Blake

Scope of attributes

- A nested query may reference attributes defined within outer queries
- A query may not reference attributes defined
 - within a nested query at an inner level
 - within a different query at the same level

NOT EXISTS OPERATOR

- The EXISTS operator admits **a nested query as a parameter** and **returns**
 - **true** if the nested query returns an **empty** set (i.e., no tuple)
 - **false** if the nested query returns a **non-empty** set (that is, it returns at least one tuple)
- In the internal query of EXISTS, the SELECT clause is mandatory, but irrelevant, because the attributes are not displayed
- The **correlation condition** ties the execution of the internal query to the values of the attributes of the current tuple in the external query

The NOT EXISTS operator (no.1)

- Find the names of the suppliers that *do not* supply product P2



*Find the names of the suppliers **for which**
there does not exist a product supply for P2*

The NOT EXISTS operator (no.1)

- Find the names of the suppliers who *do not* supply product P2

```
SELECT SName
FROM S
WHERE NOT EXISTS (SELECT *
                  FROM SP
                  WHERE PId='P2'
                  AND SP.SId=S.SId );
```

Correlation condition

- The correlation condition *ties the execution of the internal query* to the values of the attributes of the *current tuple* in the external query

How NOT EXISTS works (no.1)

- Find the names of the suppliers who *do not* supply product P2

S

SId	SName	#Employees	City
S1	Smith	20	London
S2	Jones	10	Paris
S3	Blake	30	Paris
S4	Clark	20	London
S5	Adams	30	Athens

SP

SId	PId	Qty
S1	P1	300
S1	P2	200
S1	P3	400
S1	P4	200
S1	P5	100
S2	P1	300
S2	P2	400
S3	P2	200
S4	P3	200
S4	P4	300
S4	P5	400

```
SELECT *  
FROM SP  
WHERE PId='P2'  
AND SP.SId='S1'
```

↑
*Value of SId in the
current line of table S*

How NOT EXISTS works (no.1)

- Find the names of the suppliers who *do not* supply product P2

S

SId	SName	#Employees	City
S1	Smith	20	London
S2	Jones	10	Paris
S3	Blake	30	Paris
S4	Clark	20	London
S5	Adams	30	Athens

SP

SId	PId	Qty
S1	P1	300
S1	P2	200
S1	P3	400
S1	P4	200
S1	P5	100
S2	P1	300
S2	P2	400
S3	P2	200
S4	P3	200
S4	P4	300
S4	P5	400

- The predicate including **NOT EXISTS** is false for S1 since there exists a supply for P2 by S1
 - S1 *does not* belong to the result of the query

How NOT EXISTS works (no.1)

- Find the names of the suppliers who *do not* supply product P2

S

<u>SId</u>	SName	#Employees	City
S1	Smith	20	London
S2	Jones	10	Paris
S3	Blake	30	Paris
S4	Clark	20	London
S5	Adams	30	Athens

SP

<u>SId</u>	<u>PId</u>	Qty
S1	P1	300
S1	P2	200
S1	P3	400
S1	P4	200
S1	P5	100
S2	P1	300
S2	P2	400
S3	P2	200
S4	P3	200
S4	P4	300
S4	P5	400

- The predicate including **NOT EXISTS** is true for S4 since there does not exist a supply for P2 by S4
 - S4 *does* belong to the result of the query

How NOT EXISTS works (no.1)

- Find the names of the suppliers who *do not* supply product P2

S

<u>SId</u>	SName	#Employees	City
S1	Smith	20	London
S2	Jones	10	Paris
S3	Blake	30	Paris
S4	Clark	20	London
S5	Adams	30	Athens

SP

<u>SId</u>	<u>PId</u>	Qty
S1	P1	300
S1	P2	200
S1	P3	400
S1	P4	200
S1	P5	100
S2	P1	300
S2	P2	400
S3	P2	200
S4	P3	200
S4	P4	300
S4	P5	400

- The predicate including **NOT EXISTS** is true for S5 since there exists a supply for P2 by S5
 - S5 *does* belong to the result of the query

Result of the query (no.1)

- Find the names of the suppliers who *do not* supply product P2

R

SName
Clark
Adams

CORRELATION BETWEEN QUERIES

- It may be required to **bind the computation of a nested query** to the value(s) of one or more attributes in an outer query
 - the binding is expressed by one or more correlation conditions
- A **correlation condition**
 - must be specified in the **WHERE** clause of the nested query that requires it
 - is a predicate that binds some attributes of tables appearing **in the nested query's FROM** clause to attributes of tables appearing **in the FROM clause of outer queries**
- Correlation conditions may not be expressed
 - within queries at the same nesting level
 - with references to attributes of a table appearing in the **FROM** clause of a nested query

Correlation among queries (no.1)

- For each product, find the code of the supplier who supplies the highest quantity

```
SELECT PId, SId  
FROM SP AS SPX  
WHERE Qty = (...
```

```
)
```

*Maximum
quantity
for the current
product*

Correlation among queries (no.1)

- For each product, find the code of the supplier who supplies the highest quantity

```
SELECT PId, SId
FROM SP AS SPX
WHERE Qty = (SELECT MAX(Qty)
             FROM SP AS SPY
             ... )
```

*Maximum
quantity*

Correlation among queries (no.1)

- For each product, find the code of the supplier who supplies the highest quantity

```
SELECT PId, SId
FROM SP AS SPX
WHERE Qty = (SELECT MAX(Qty)
             FROM SP AS SPY
             WHERE SPY.PId=SPX.PId);
```

Correlation condition

*Maximum
quantity
for the current
product*

Correlation among queries (no.2)

TRIP (TId, StartingPlace, Destination,
DepartureTime, ArrivalTime)

- Find the codes of the trips whose duration is lower than the average duration of the trips on the same route (i.e., same starting place and destination)

```
SELECT TId
FROM TRIP AS TA
WHERE ArrivalTime-DepartureTime < (...
)
```

*Average
duration
of trips
on the current
route*

Correlation among queries (no.2)

TRIP (TId, StartingPlace, Destination,
DepartureTime, ArrivalTime)

- Find the codes of the trips whose duration is lower than the average duration of the trips on the same route (i.e., same starting place and destination)

```
SELECT TId
FROM TRIP AS TA
WHERE ArrivalTime-DepartureTime <
  (SELECT AVG(ArrivalTime-DepartureTime)
   FROM TRIP AS TB
   ... )
```

*Average
duration
of trips*

Correlation among queries (no.2)

TRIP (TId, StartingPlace, Destination,
DepartureTime, ArrivalTime)

- Find the codes of the trips whose duration is lower than the average duration of the trips on the same route (i.e., same starting place and destination)

```
SELECT TId
FROM TRIP AS TA
WHERE ArrivalTime-DepartureTime <
  (SELECT AVG(ArrivalTime-DepartureTime)
   FROM TRIP AS TB
   WHERE TB.StartingPlace=TA.StartingPlace
         AND TB.Destination=TA.Destination);
```

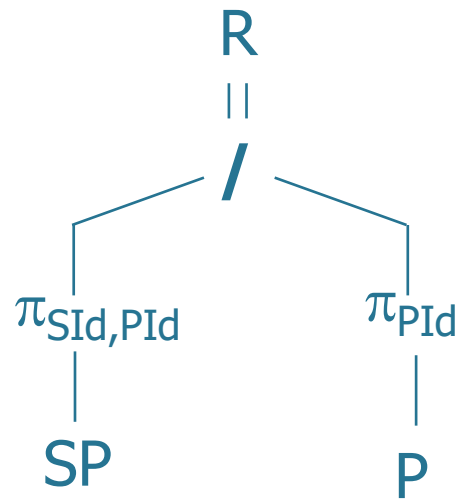
Correlation conditions

DIVISION OPERATOR

- In SQL, the division operation can be performed using the **COUNT** operator, to verify that **all** the elements of interest belong to the reference set

The division operation (no.1)

- Find the codes of the suppliers who supply *all* products
- In relational algebra we must use the division operator



Division in SQL (no.1)

- Find the codes of the suppliers who supply all products
- Remark
 - all products that may be supplied are listed in table P



- a supplier is supplying all products if he or she is supplying a number of distinct products equal to the cardinality of P

Division in SQL (no.1)

- Find the codes of the suppliers who supply all products

```
SELECT COUNT(*)  
FROM P
```

} *Total
number
of products*

Division in SQL (no.1)

- Find the codes of the suppliers who supply all products

*For each supplier,
total number of
products supplied* {
SELECT SId
FROM SP
GROUP BY SId
HAVING COUNT(*) = (SELECT COUNT(*)
FROM P) } *Total
number
of products*

Division in SQL (no.1)

- Find the codes of the suppliers who supply all products

SP(SID, PID, Qty)

*For each supplier,
total number of
products supplied*

```
SELECT SId
FROM SP
GROUP BY SId
HAVING COUNT(*) = (SELECT COUNT(*)
                    FROM P)
```

*Total
number
of products*

Division in SQL (no.1) – Different SP TABLE

- Find the codes of the suppliers who supply all products

SP(SID, PID, Date, Qty)

SELECT SId

FROM SP

GROUP BY SId

HAVING COUNT(**DISTINCT PID**)= (SELECT COUNT(*)
FROM P);

*For each supplier,
total number of
products supplied*

*Total
number
of products*

Division in SQL: procedure (no.2)

- Find the codes of the suppliers who supply at least all of the products supplied by supplier S2
- We must count
 - the number of products supplied by S2
 - the number of products supplied both by an arbitrary supplier and by S2
- The two counts must be equal

Division in SQL (no.2)

- Find the codes of the suppliers who supply at least all of the products supplied by supplier S2

```
SELECT COUNT(*)  
FROM SP  
WHERE SId='S2'
```

*Number of
products
supplied by S2*

Division in SQL (no.2)

- Find the codes of the suppliers who supply at least all of the products supplied by supplier S2

```
SELECT SId
FROM SP
WHERE PId IN (SELECT PId
              FROM SP
              WHERE SId='S2')
GROUP BY SId
HAVING COUNT(*)=(SELECT COUNT(*)
                  FROM SP
                  WHERE SId='S2');
```

*For each supplier,
the total number of
products supplied,
including only the
products supplied by
S2*

*Products
supplied by S2*

*Number
of products
supplied by S2*