Database Systems

COMP 3010E FALL 2025

LECTURE 8 ADVANCED SQL

Agenda

- Subquery (nested query)
- WHERE subquery
- FROM subquery
- Having Subquery

Sub-Queries

- ❑Include a query in a WHERE/HAVING or FROM clauses of another query

Query: Find the most expensive product. Show its description and price

```
SELECT ProductID, ProductDescription, ProductStandardPrice
FROM Product_T
WHERE ProductStandardPrice = (SELECT MAX(ProductStandardPrice)
                              FROM Product_T)
;
```

outer query

Subquery (inner query)

```
SELECT *
FROM (SELECT ProductID, ProductDescription, ProductStandardPrice
      FROM Product_T
     ORDER BY ProductStandardPrice DESC)
WHERE ROWNUM=1;
```

Subqueries can be nested multiple times

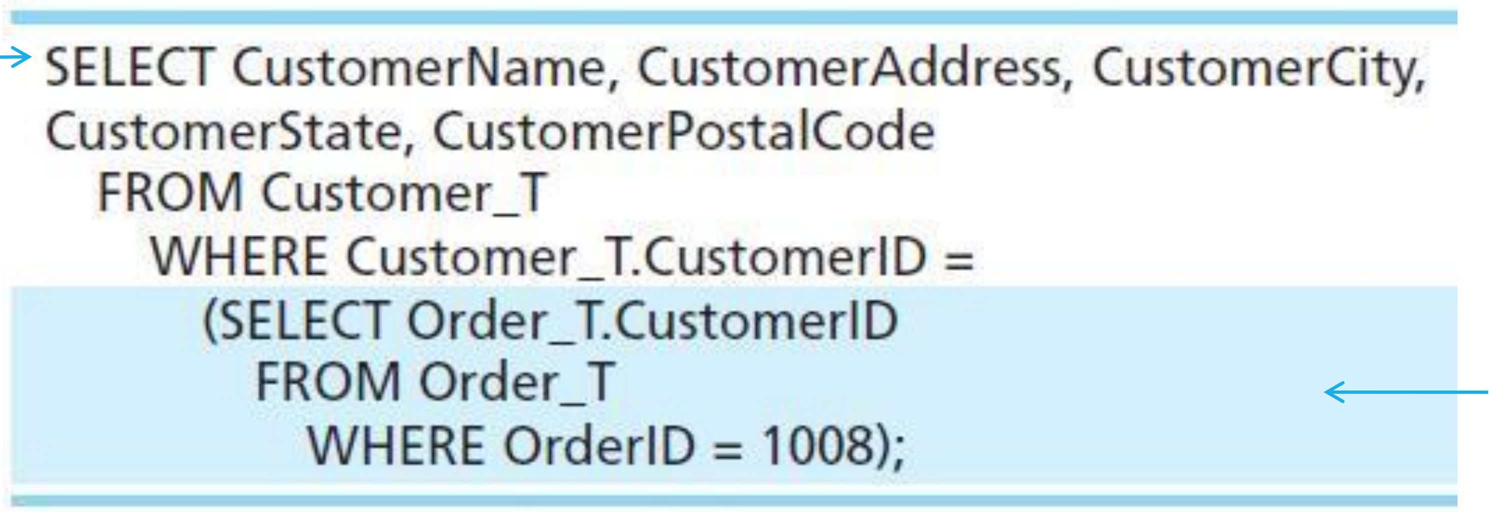
In WHERE clause (1)

```
SELECT CustomerName, CustomerAddress, CustomerCity,
       CustomerState, CustomerPostalCode
FROM Customer_T, Order_T
WHERE Customer_T.CustomerID = Order_T.CustomerID
AND OrderID = 1008;
```

- ❑ In WHERE clause: use subquery results as part of the conditions for row selection
  - ❖ Use comparison operator (e.g., >, =) when subquery returns a single value *(scalar subquery)*

Query: What are the name and address of the customer who placed order number 1008?

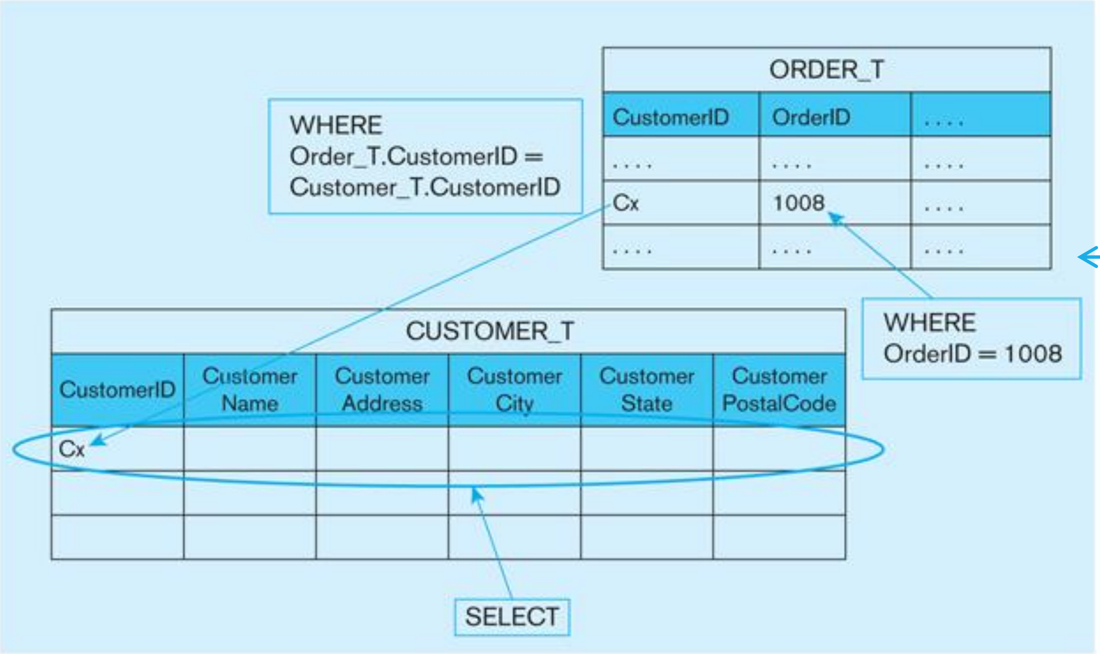
outer query



```
SELECT CustomerName, CustomerAddress, CustomerCity,
CustomerState, CustomerPostalCode
   FROM Customer_T
      WHERE Customer_T.CustomerID =
         (SELECT Order_T.CustomerID
            FROM Order_T
               WHERE OrderID = 1008);
```

Subquery (inner query)

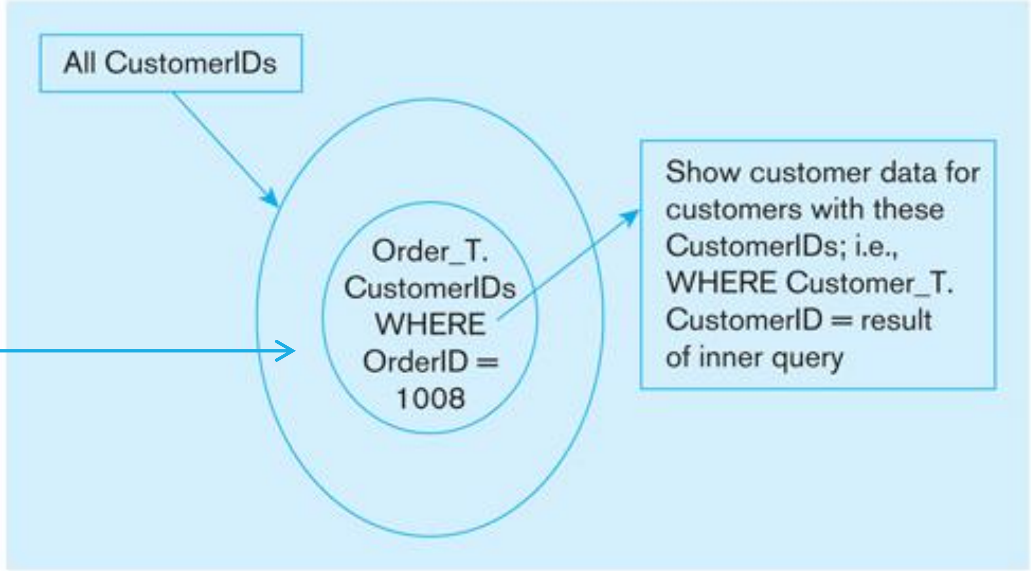
(a) Join query approach



Customer_T and Order_T are combined into a (flat) single table, which contains all columns of both tables

→ Have access to each attribute of either table (such as CustomerID, OrderID, OrderDate)

(b) Subquery approach

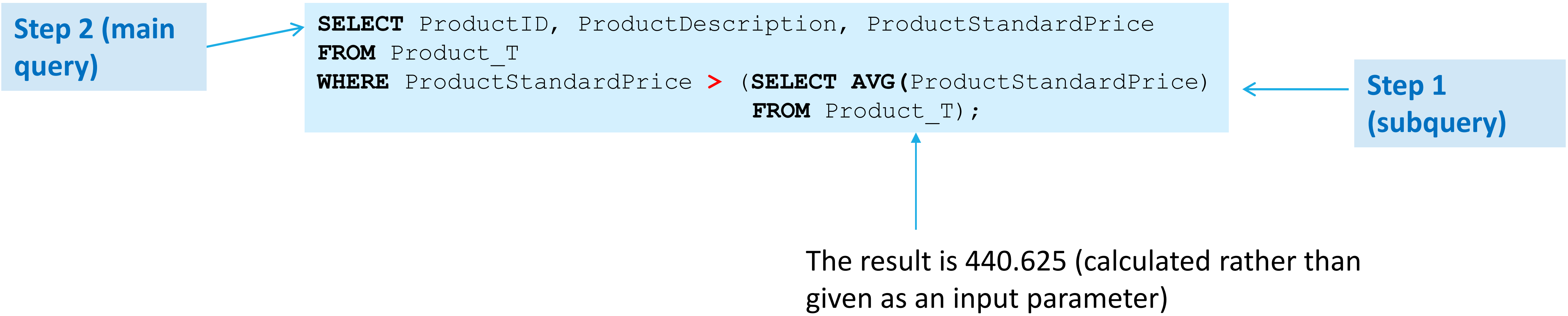


The inner query returns a set of values (with zero, one, or many values) for search condition

→ Only have access to the attributes of the table in the outer query

In WHERE clause

- *List products that are more expensive than the average price of all the product.*
- *Step 1: calculate the average price*
- *Step 2: compare every product’s price with this average price*



Step 2 (main query)

```
SELECT ProductID, ProductDescription, ProductStandardPrice
FROM Product_T
WHERE ProductStandardPrice > (SELECT AVG(ProductStandardPrice)
                              FROM Product_T);
```

Step 1 (subquery)

The result is 440.625 (calculated rather than given as an input parameter)

The “Max-Row" Problem!

In WHERE clause (2)

□ Query: Find the most expensive product and its description and price. If multiple products share the highest price, show all of them.

□ Previous solution (is it always working?):

```
SELECT ProductID, ProductDescription, ProductStandardPrice
FROM Product_T
ORDER BY ProductStandardPrice DESC
FETCH FIRST 1 ROWS ONLY;
```

□ Alternative solution

□ Step 1: find the highest price (a number)

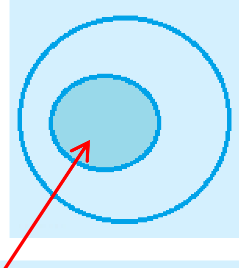
□ Step 2: compare every product’s price with This highest price

□If a product’s price = the highest price, it must be (one of) the most expensive product

```
SELECT ProductDescription, ProductStandardPrice
FROM Product_T
WHERE ProductStandardPrice =
                (SELECT MAX(ProductStandardPrice)
                 FROM Product_T);
```

In WHERE clause (2)

- In WHERE clause: use subquery results as part of the conditions for row selection
  - Use (IN, NOT IN) or (ANY, ALL with =, > , <) when subquery returns a set of values



Query: Show the names of the customers who have placed orders.

```
SELECT DISTINCT CustomerName
FROM Customer_T C, Order_T O
WHERE C.CustomerID = O.CustomerID;
```

```
SELECT CustomerName
FROM Customer_T
WHERE CustomerID IN
        (SELECT DISTINCT CustomerID
         FROM Order_T);
```

Query: Show the names of the customers who have NOT placed any orders.

If the subquery returns a **set of values** (more than one)
you must not use =, <, > to connect the subquery.
Is 2 = {1,2,3} true? This comparison is meaningless.

```
SELECT CustomerName
FROM Customer_T
WHERE CustomerID NOT IN
        (SELECT DISTINCT CustomerID
         FROM Order_T);
```

In WHERE clause (3)

- In WHERE clause: use subquery results as part of the conditions for row selection
  - Use (IN, NOT IN) or (ANY, ALL with =, > , <) when subquery returns a set of values

Query: Show the names of the customers who have NOT placed orders for “Computer Desk”.

```
SELECT CustomerName
FROM Customer_T
WHERE CustomerID NOT IN
        (SELECT DISTINCT CustomerID
         FROM Order_T O, OrderLine_T L, Product_T P
         WHERE O.OrderID = L.OrderID
         AND L.ProductID = P.ProductID
         AND ProductDescription = 'Computer Desk');
```

Show the names of the customers whose IDs are not on this list

A list of CustomerID’s on the orders which contain “Computer Desk”

In WHERE clause (3)

- In WHERE clause: use subquery results as part of the conditions for row selection
  - Use (IN, NOT IN) or (ANY, ALL with =, > , <) when subquery returns a set of values

Query: Find out the most expensive product and its price.

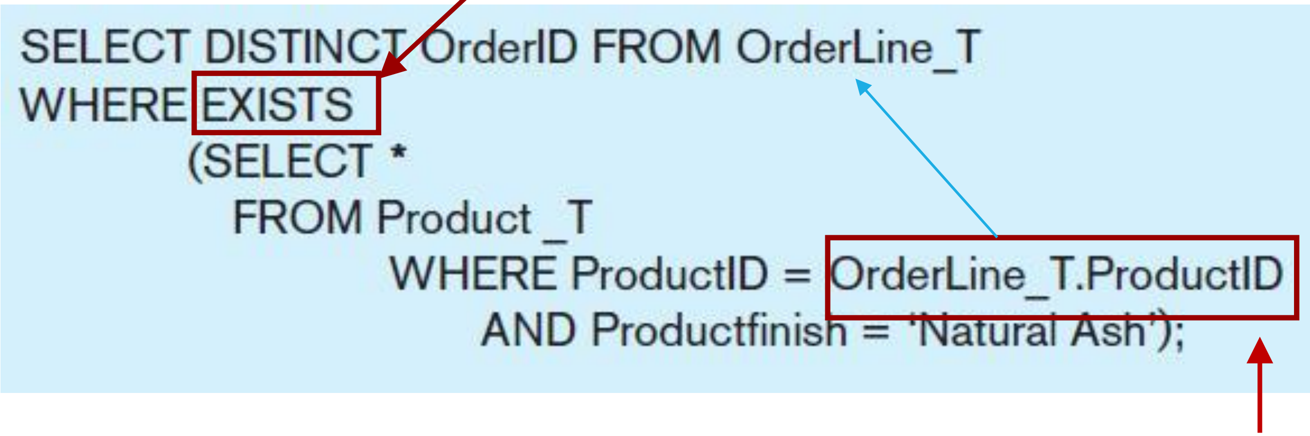
```
SELECT ProductDescription, ProductStandardPrice
FROM Product_T
WHERE ProductStandardPrice =
                (SELECT MAX(ProductStandardPrice)
                 FROM Product_T);
```

The “Max-Row" Problem!

```
SELECT ProductDescription, ProductStandardPrice
FROM Product_T
WHERE ProductStandardPrice >= ALL
                (SELECT ProductStandardPrice
                 FROM Product_T);
```

Correlated Query

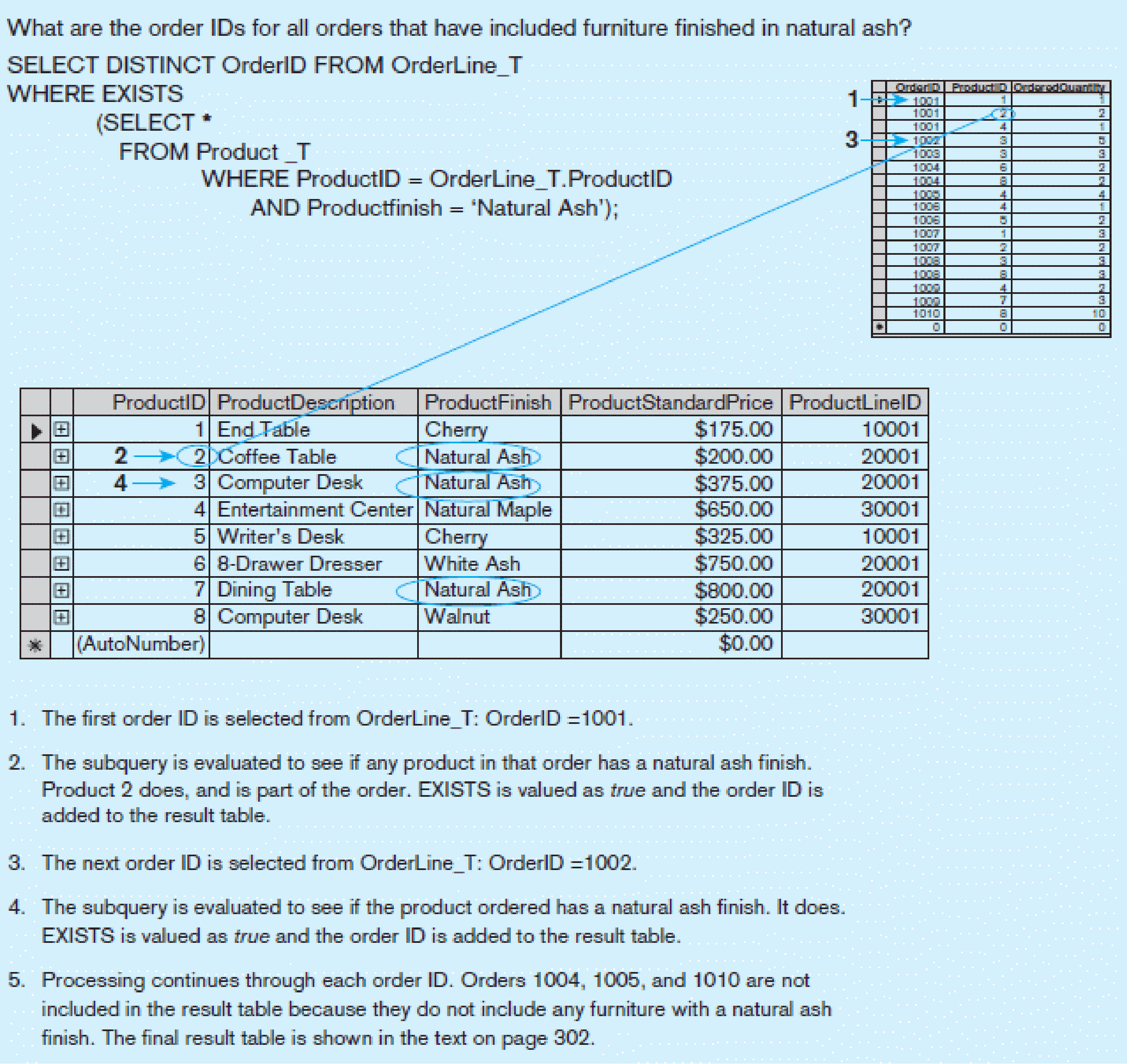
The EXISTS operator will return a TRUE value if the subquery resulted in a non-empty set, otherwise it returns a FALSE



```
SELECT DISTINCT OrderID FROM OrderLine_T
WHERE EXISTS
        (SELECT *
           FROM Product _T
                WHERE ProductID = OrderLine_T.ProductID
                    AND Productfinish = 'Natural Ash');
```

➔ A correlated subquery always refers to an attribute from a table referenced in the outer query

The subquery is testing for a value that comes from the outer query



What are the order IDs for all orders that have included furniture finished in natural ash?

```
SELECT DISTINCT OrderID FROM OrderLine_T
WHERE EXISTS
        (SELECT *
          FROM Product _T
                 WHERE ProductID = OrderLine_T.ProductID
                      AND Productfinish = 'Natural Ash');
```

OrderID	ProductID	OrderedQuantity
1001	1	1
1001	2	2
1001	4	1
1002	3	5
1003	3	3
1004	6	2
1004	8	2
1005	4	4
1006	4	1
1006	5	2
1007	1	3
1007	2	2
1008	3	3
1008	8	3
1009	4	2
1009	7	3
1010	8	10
0	0	0

ProductID	ProductDescription	ProductFinish	ProductStandardPrice	ProductLineID
1	End Table	Cherry	$175.00	10001
2	Coffee Table	Natural Ash	$200.00	20001
3	Computer Desk	Natural Ash	$375.00	20001
4	Entertainment Center	Natural Maple	$650.00	30001
5	Writer's Desk	Cherry	$325.00	10001
6	8-Drawer Dresser	White Ash	$750.00	20001
7	Dining Table	Natural Ash	$800.00	20001
8	Computer Desk	Walnut	$250.00	30001
(AutoNumber)			$0.00	

1. The first order ID is selected from OrderLine_T: OrderID =1001.

2. The subquery is evaluated to see if any product in that order has a natural ash finish. Product 2 does, and is part of the order. EXISTS is valued as *true* and the order ID is added to the result table.

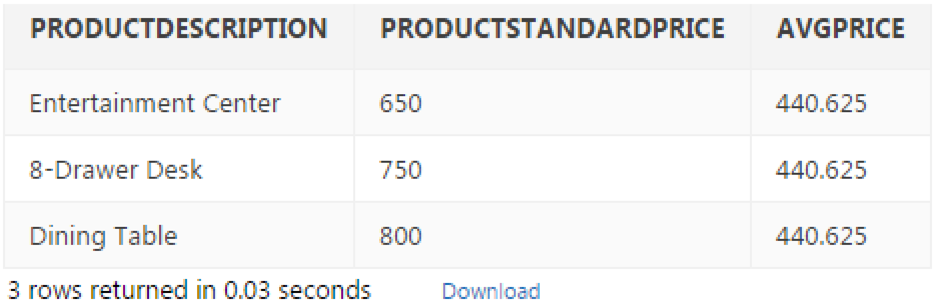
3. The next order ID is selected from OrderLine_T: OrderID =1002.

4. The subquery is evaluated to see if the product ordered has a natural ash finish. It does. EXISTS is valued as *true* and the order ID is added to the result table.

5. Processing continues through each order ID. Orders 1004, 1005, and 1010 are not included in the result table because they do not include any furniture with a natural ash finish. The final result table is shown in the text on page 302.

Note: Only the orders that involve products with Natural Ash will be included in the final results.

In FROM clause (1)



PRODUCTDESCRIPTION	PRODUCTSTANDARDPRICE	AVGPRICE
Entertainment Center	650	440.625
8-Drawer Desk	750	440.625
Dining Table	800	440.625

3 rows returned in 0.03 seconds

- ❑ In FROM clause: create a temporary derived table (*in order to have access to the attributes in the subquery*)

Query: Show all the products that have a standard price higher than the average standard price.

```
SELECT ProductDescription, ProductStandardPrice
FROM Product_T
WHERE ProductStandardPrice > (SELECT AVG(ProductStandardPrice)
                              FROM Product_T);
```

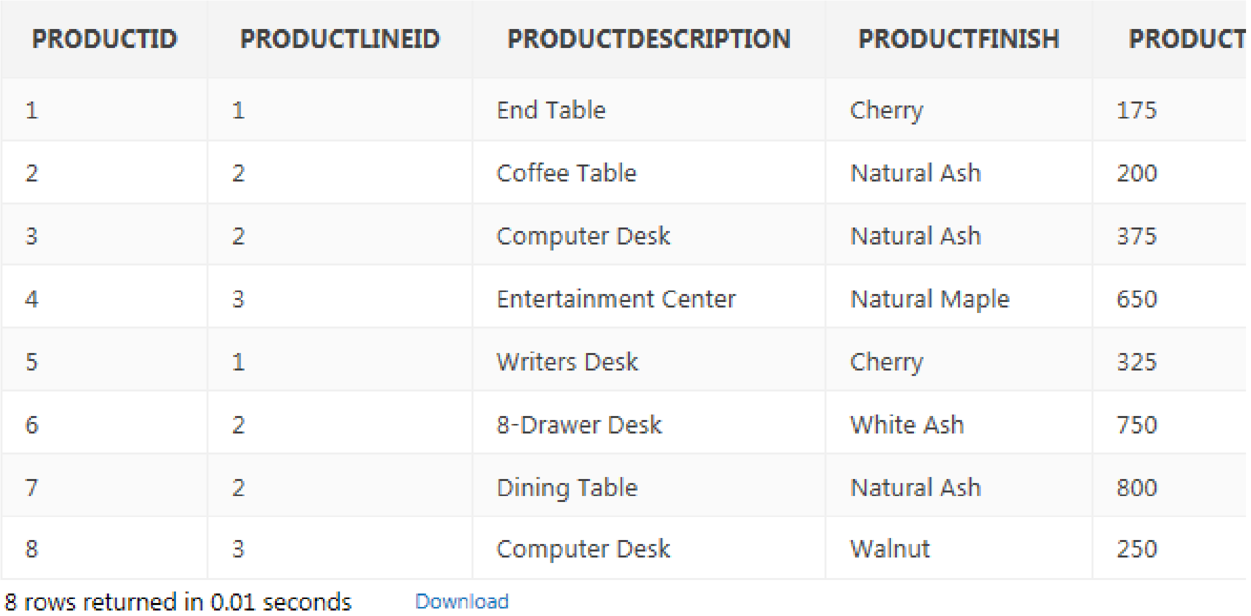
Query: Show all the products that have a standard price higher than the average standard price.

```
SELECT ProductDescription, ProductStandardPrice
FROM Product_T, (SELECT AVG(ProductStandardPrice) AS AvgPrice
                 FROM Product_T)
WHERE ProductStandardPrice > AvgPrice;
```

Cross-join between the two tables

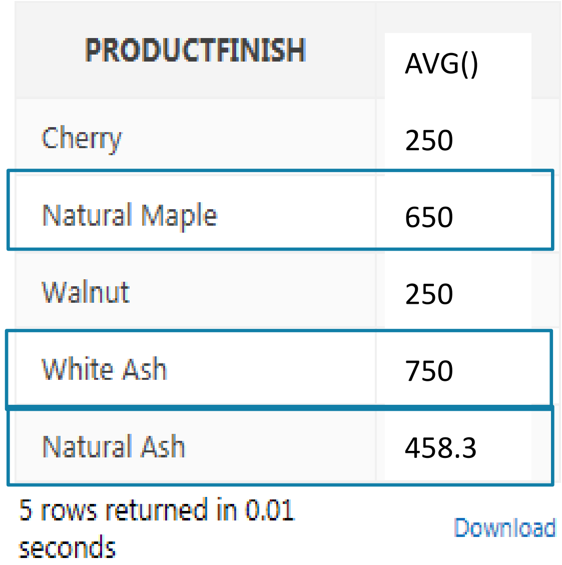
HAVING subquery

Find the finish types with average price higher than that of all the product.



PRODUCTID	PRODUCTLINEID	PRODUCTDESCRIPTION	PRODUCTFINISH	PRODUCT
1	1	End Table	Cherry	175
2	2	Coffee Table	Natural Ash	200
3	2	Computer Desk	Natural Ash	375
4	3	Entertainment Center	Natural Maple	650
5	1	Writers Desk	Cherry	325
6	2	8-Drawer Desk	White Ash	750
7	2	Dining Table	Natural Ash	800
8	3	Computer Desk	Walnut	250

8 rows returned in 0.01 seconds Download



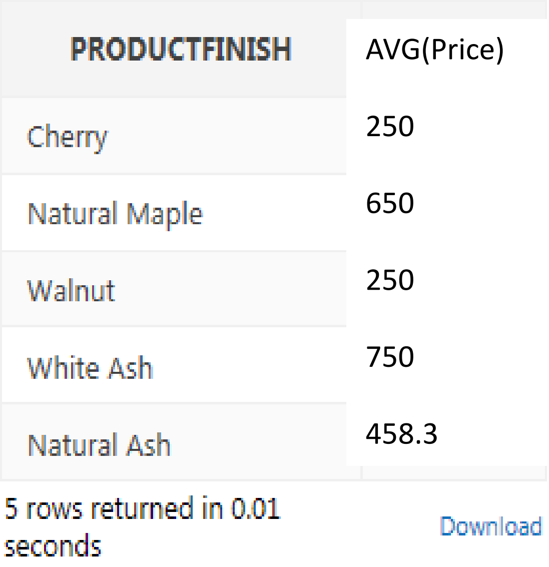
PRODUCTFINISH	AVG()
Cherry	250
Natural Maple	650
Walnut	250
White Ash	750
Natural Ash	458.3

5 rows returned in 0.01 seconds Download

AvgPrice
400.625

HAVING subquery

Find the finish types with average price higher than that of all the product.



PRODUCTFINISH	AVG(Price)
Cherry	250
Natural Maple	650
Walnut	250
White Ash	750
Natural Ash	458.3

5 rows returned in 0.01 seconds

Download

AvgPrice
400.625

```
SELECT ProductFinish
FROM Product_T
GROUP BY ProductFinish
HAVING AVG(ProductStandardPrice) >=
(SELECT AVG(ProductStandardPrice)
 FROM Product_T
 );
```

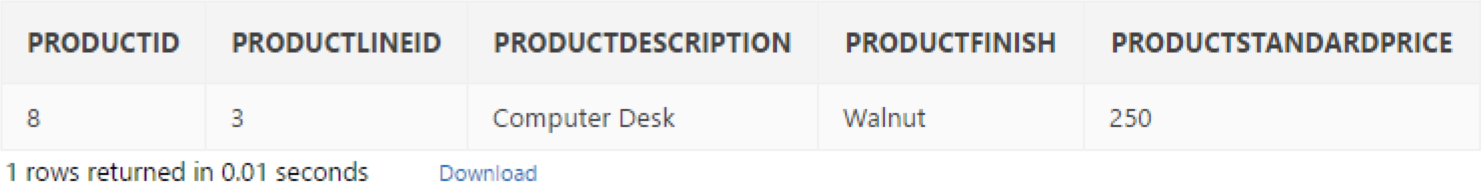
Multi-layer subqueries

❑ Multi-layer (nested) subqueries in the WHERE/HAVING clause

Query: Display all the attributes of the product(s) that has been sold in the largest quantity.

```
SELECT *
FROM Product_T
WHERE ProductID IN (SELECT ProductID
                    FROM OrderLine_T
                    GROUP BY ProductID
                    HAVING SUM(OrderedQuantity) = (SELECT MAX(SUM(OrderedQuantity))
                                                   FROM OrderLine_T
                                                   GROUP BY ProductID));
```

Divide-and-conquer approach



PRODUCTID	PRODUCTLINEID	PRODUCTDESCRIPTION	PRODUCTFINISH	PRODUCTSTANDARDPRICE
8	3	Computer Desk	Walnut	250

1 rows returned in 0.01 seconds Download

Sub-Query Summary

- Where to “nest” a subquery?
  - FROM : Add the subquery as an input table.
  - WHERE: Connect main and subquery using keywords or operators
    - WHERE A = (subquery Q), Q must return a single value (1 column, 1 row).
    - WHERE A IN/NOT IN (subquery Q), Q can return a list of values (1 col, multiple rows)
    - WHERE A >= ALL/SOME (subquery Q), Q can return a list of values (1 col, multiple rows)
    - WHERE (NOT) EXISTS (subquery Q), Q uses main query table (correlated query)
  - HAVING
    - Same as WHERE subqueries

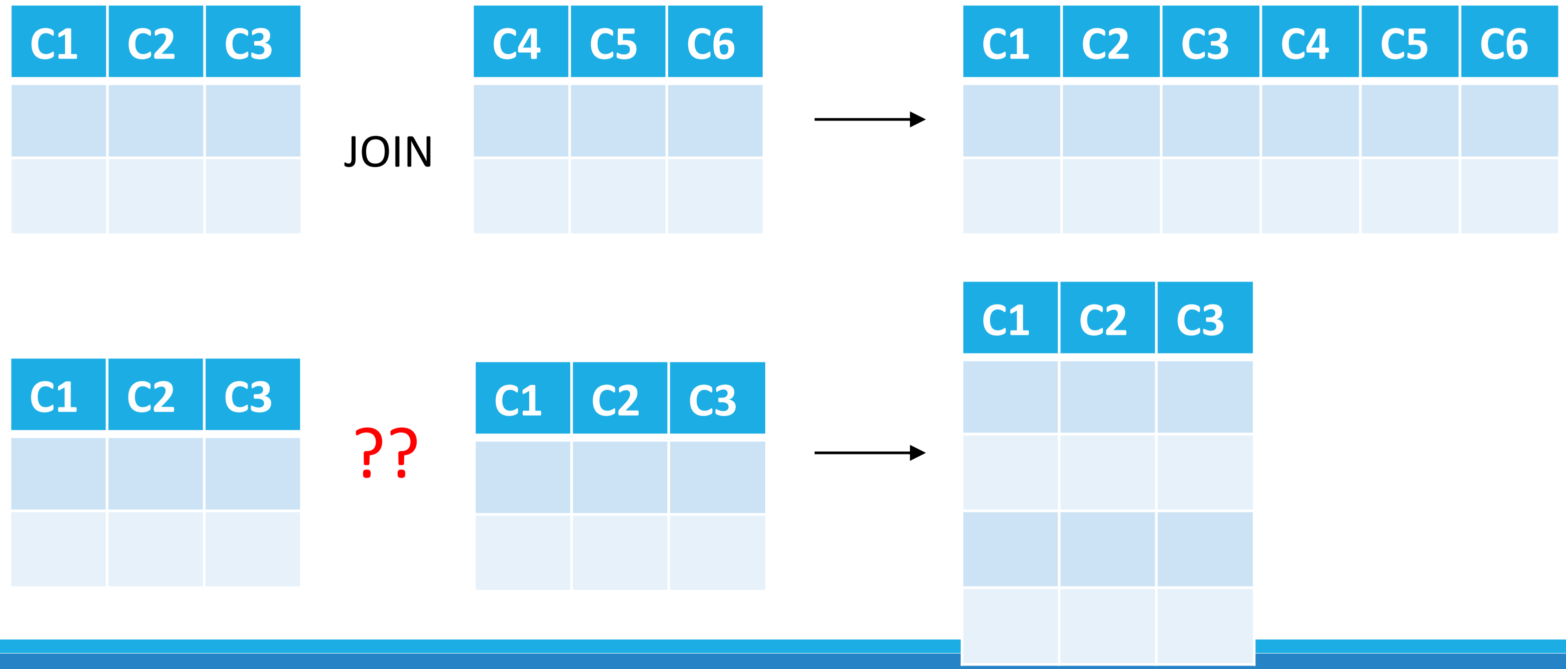
Combine Queries

- ❑ JOIN operation -- combine columns
- ❑ How to combine rows in two tables with the same schema?
- ❑ Set-based operations
  - UNION
  - INTERSECT
  - MINUS

1. Combine the results of two component queries into a single result;
2. Queries containing set operators are called compound queries.

Combining Rows in Results

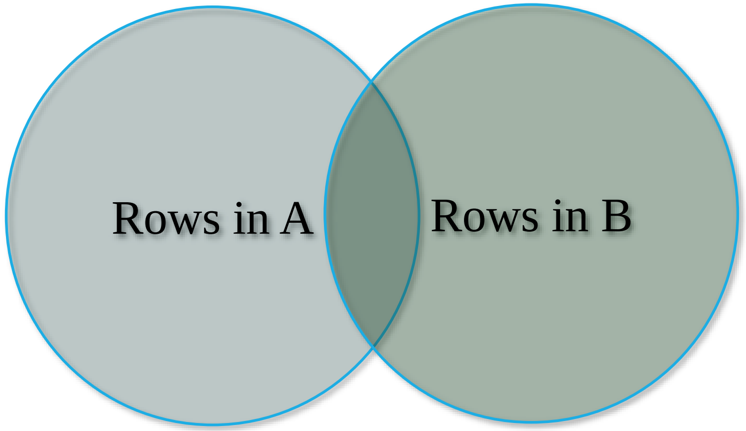
- ❑ We talked about combining columns (join operation)
- ❑ How to combine rows in two tables with the same schema?
- ❑ Set-based operations
  - ❑ UNION
  - ❑ INTERSECT
  - ❑ MINUS



UNION

- **Syntax**

```
SELECT * FROM A
UNION
SELECT * FROM B;
```



- **Output**

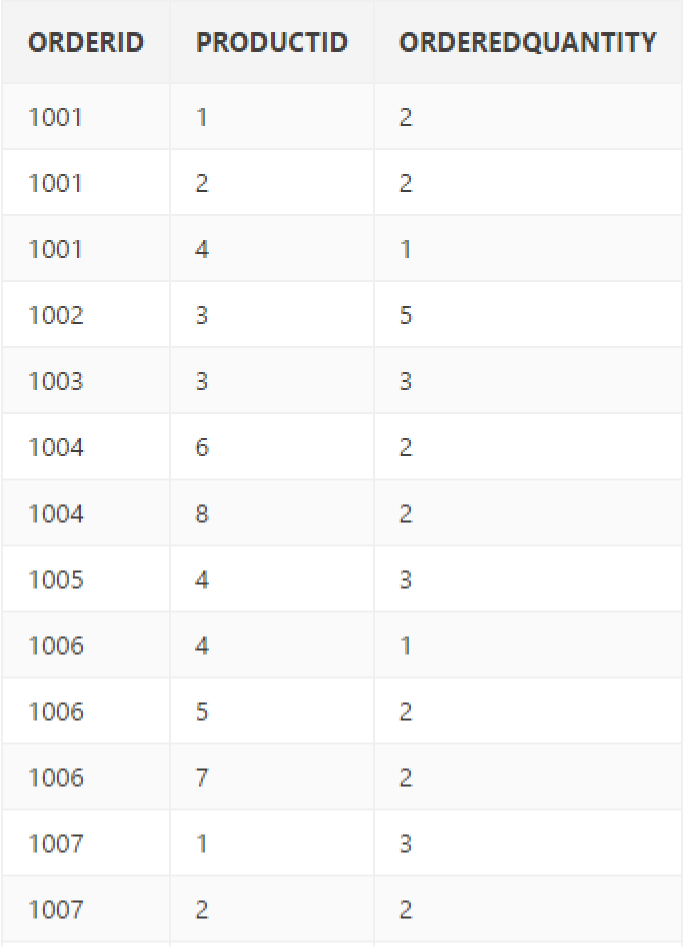
  The result table will combine rows in both input tables

- **Features**
  - The two sub-queries must generate the same columns (name and data type)

    i.e., “Union Compatibility”
  - Duplicate rows (if any) will be eliminated
  - To include duplicate rows, use UNION ALL

A Table for Examples

- The ORDERLINE table in the PVFC database



ORDERID	PRODUCTID	ORDEREDQUANTITY
1001	1	2
1001	2	2
1001	4	1
1002	3	5
1003	3	3
1004	6	2
1004	8	2
1005	4	3
1006	4	1
1006	5	2
1006	7	2
1007	1	3
1007	2	2

UNION – example (1)

Query: Show the Orders that contain either Product 4 or Product 7

```
SELECT OrderID
FROM OrderLine_T
WHERE ProductID = 4
UNION
SELECT OrderID
FROM OrderLine_T
WHERE ProductID = 7;
```

Orders contain Product #4

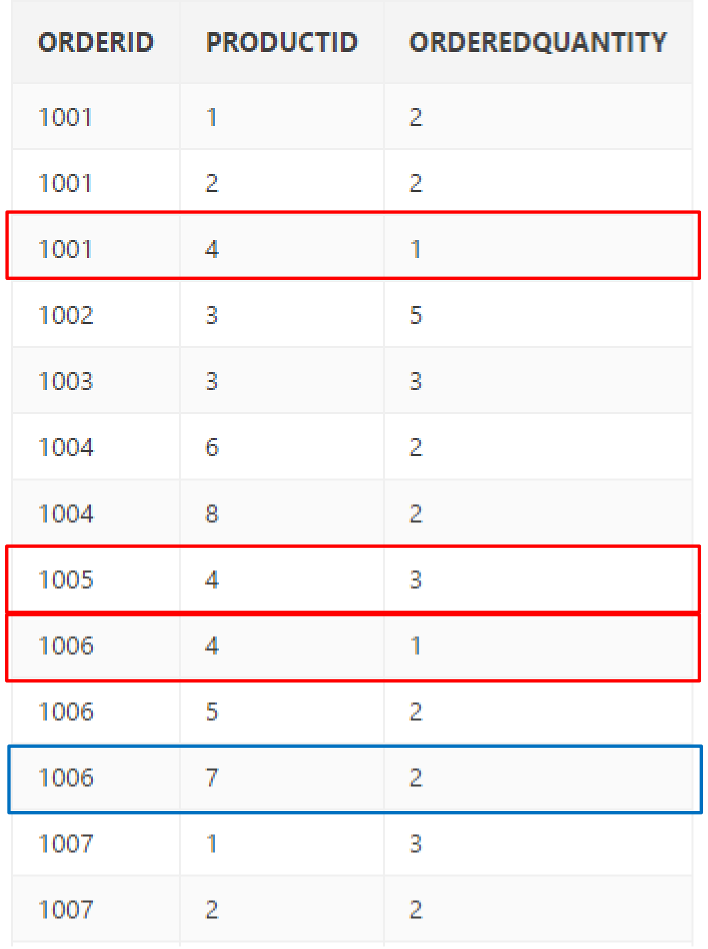
1001
1005
1006

Orders contain Product #7

1006

Final results (duplicate results removed)

1001
1005
1006



ORDERID	PRODUCTID	ORDEREDQUANTITY
1001	1	2
1001	2	2
1001	4	1
1002	3	5
1003	3	3
1004	6	2
1004	8	2
1005	4	3
1006	4	1
1006	5	2
1006	7	2
1007	1	3
1007	2	2

Intersect – example (1)

Query: Show the Orders that contain either Product 4 or Product 7

```
SELECT OrderID
FROM OrderLine_T
WHERE ProductID = 4
INTERSECT
SELECT OrderID
FROM OrderLine_T
WHERE ProductID = 7;
```

Orders contain Product #4

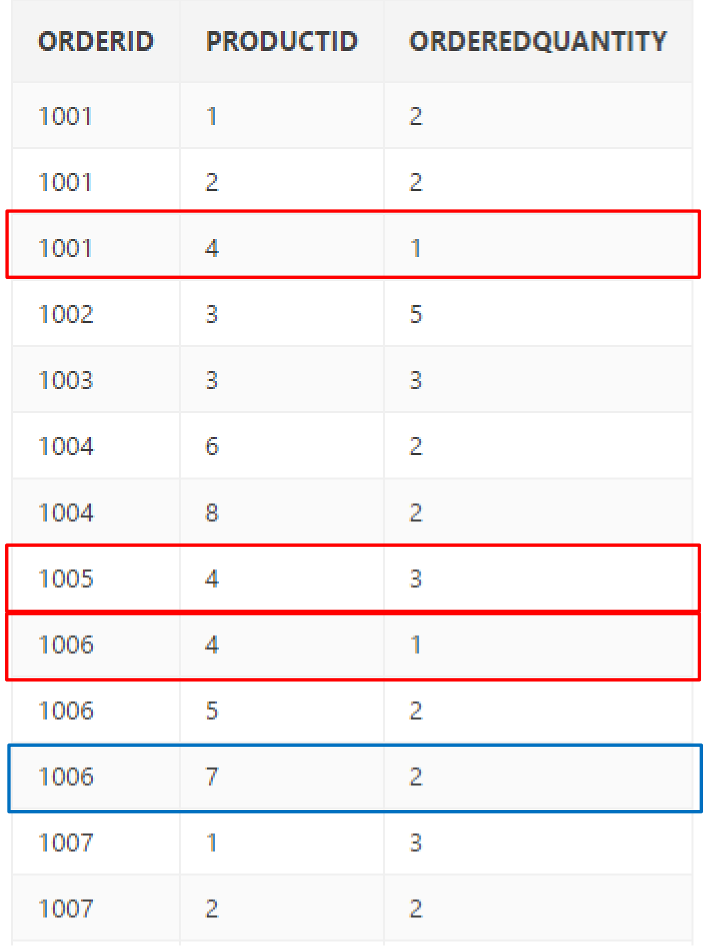
1001
1005
1006

Orders contain Product #7

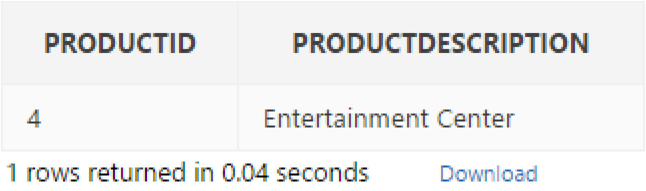
1006

Final results (duplicate results removed)

1006



ORDERID	PRODUCTID	ORDEREDQUANTITY
1001	1	2
1001	2	2
1001	4	1
1002	3	5
1003	3	3
1004	6	2
1004	8	2
1005	4	3
1006	4	1
1006	5	2
1006	7	2
1007	1	3
1007	2	2



PRODUCTID	PRODUCTDESCRIPTION
4	Entertainment Center

1 rows returned in 0.04 seconds Download

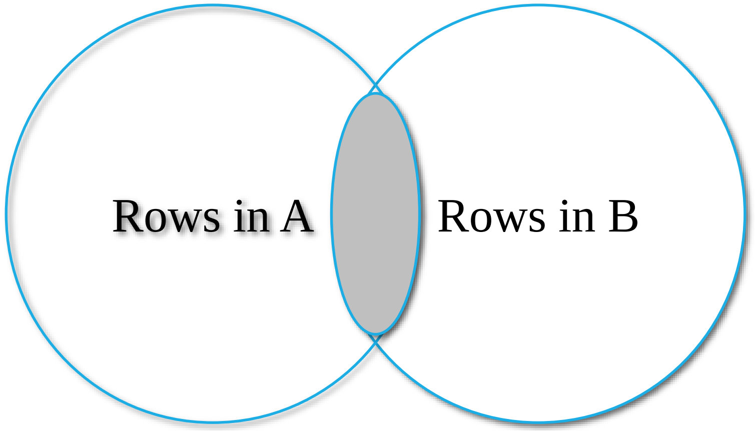
INTERSECT

- **Syntax**

  SELECT * FROM A
  INTERSECT
  SELECT * FROM B;

- **Output**

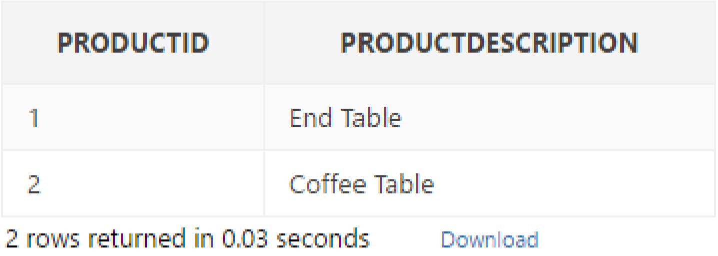
  Rows in common of the two input tables



Query: Show the ProductID and ProductDescription of the product(s) on BOTH OrderID = 1001 AND OrderID = 1006.

```
SELECT ProductID,ProductDescription
FROM Product_T
WHERE ProductID IN (SELECT ProductID FROM OrderLine_T WHERE OrderID = 1001)
INTERSECT
SELECT ProductID,ProductDescription
FROM Product_T
WHERE ProductID IN (SELECT ProductID FROM OrderLine_T WHERE OrderID = 1006);
```

MINUS



PRODUCTID	PRODUCTDESCRIPTION
1	End Table
2	Coffee Table

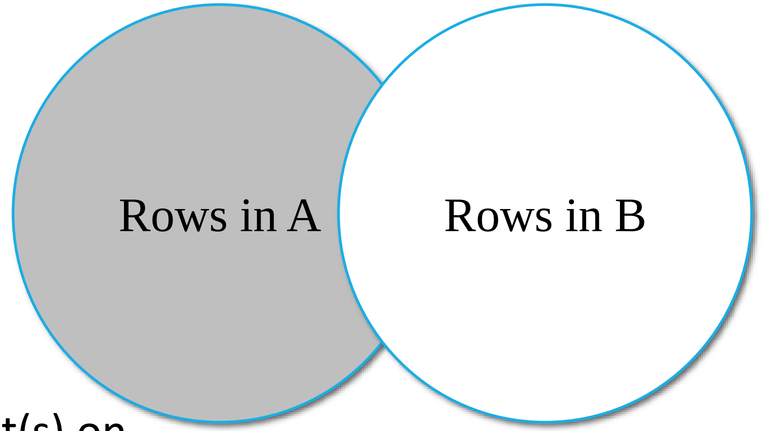
2 rows returned in 0.03 seconds   Download

❑ Syntax

```
SELECT * FROM A
MINUS
SELECT  * FROM B;
```

❑ Output

Rows in table A but not in table B

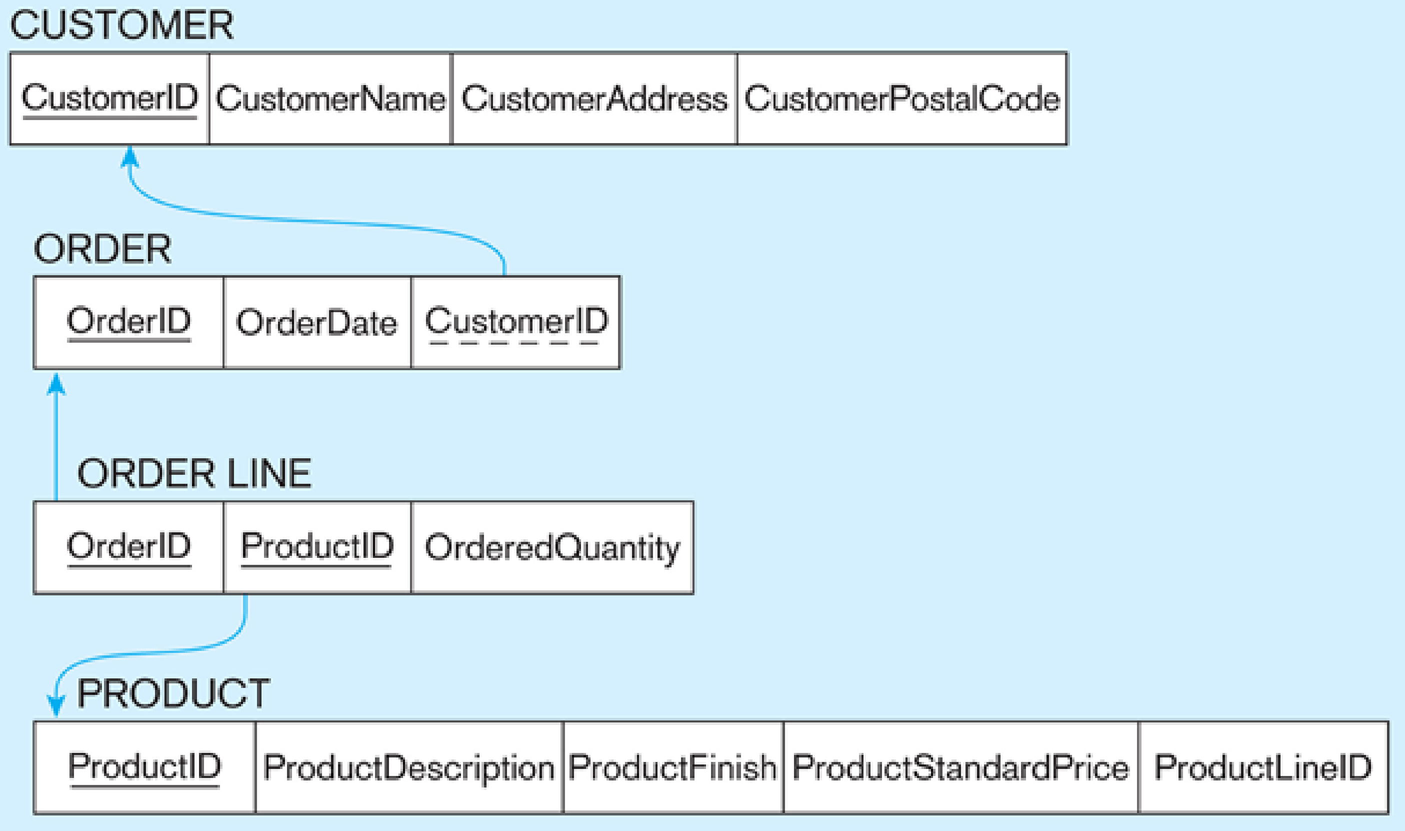


Query: Show the ProductID and ProductDescription of the product(s) on OrderID = 1001 but not on OrderID = 1006.

```
SELECT ProductID,ProductDescription
FROM Product_T
WHERE ProductID IN (SELECT ProductID FROM OrderLine_T WHERE OrderID = 1001)
MINUS
SELECT ProductID,ProductDescription
FROM Product_T
WHERE ProductID IN (SELECT ProductID FROM OrderLine_T WHERE OrderID = 1006);
```

Overall Exercises

Database given:



CUSTOMER

CustomerID	CustomerName	CustomerAddress	CustomerPostalCode

ORDER

OrderID	OrderDate	CustomerID

ORDER LINE

OrderID	ProductID	OrderedQuantity

PRODUCT

ProductID	ProductDescription	ProductFinish	ProductStandardPrice	ProductLineID

Advanced SQL Overall Exercise

1. Find customers whose total due amounts are above $3000. Show the customer names and their corresponding total amount dues. Rank the customers based on the due amount in descending order. (How to calculate total due for each customer?)

OrderID	ProductID	Quantity	ProductStandardprice	CustomerName
1001	1	2 ✖	200 = 400 +	ABC Furniture
1001	2	3 ✖	100 = 300 +	ABC Furniture
1002	4	2 ✖	50 = 100	ABC Furnature

‖

800

```
SELECT CustomerName, SUM(OrderedQuantity*ProductStandardPrice) as total_due
FROM Product_T p, Order_t o, Orderline_t l, Customer_t c
WHERE p.Productid = l.Productid and l.Orderid = o.Orderid and o.Customerid = c.Customerid
GROUP BY CustomerName
HAVING SUM(OrderedQuantity*ProductStandardPrice) > 3000
ORDER BY total_due DESC;
```

Advanced SQL Overall Exercise

2. Find customers who never ordered any product with a price higher than 400 dollars. (hint: customers who never ordered anything should also be included in your output). Show the names of the customers.

```
SELECT CustomerName
FROM Customer_T
WHERE CustomerName NOT IN
(
  SELECT DISTINCT CustomerName
  FROM Product_T p, Order_t o, Orderline_t l, Customer_t c
  WHERE p.Productid = l.Productid and l.Orderid = o.Orderid and o.Customerid =
c.Customerid and ProductStandardPrice > 400
);
```

Advanced SQL Overall Exercise

3. How many orders contain both Cherry and Natural Maple (ProductFinish) products?

```
SELECT COUNT(*) FROM
(
    SELECT DISTINCT o.OrderID
    FROM Product_T p, Order_T o, OrderLine_T l
    WHERE p.Productid = l.Productid AND l.Orderid = o.OrderID AND ProductFinish =
    'Cherry'

    INTERSECT

    SELECT DISTINCT o.OrderID
    FROM Product_T p, Order_T o, OrderLine_T l
    WHERE p.Productid = l.Productid AND l.Orderid = o.OrderID AND ProductFinish =
    'Natural Maple'
);
```

Conditional Control

Case Statement

- Enable us to control the flow of the execution based on a condition
- Logical processing; conditional branching
- Two forms

  1. CASE (with selector)
  2. searched CASE

CASE Statement (1)

❑ Syntax

```
CASE selector
     WHEN expression-1 THEN statement-1
     WHEN expression-2 THEN statement-2
      … …
     WHEN expression-N THEN statement-N
     ELSE statement-N+1
END
```

❑ How it works

- ❖ the selector is first evaluated
- ❖ WHEN clauses are then evaluated sequentially
- ❖ if expression_value = selector_value, execute the associated statement(s); and then jump to END
- ❖ if no expression matches the selector_value, execute the ELSE clause

CASE Statement (2)

Query: Display a column “Comments” based on the “finish type” of the product.

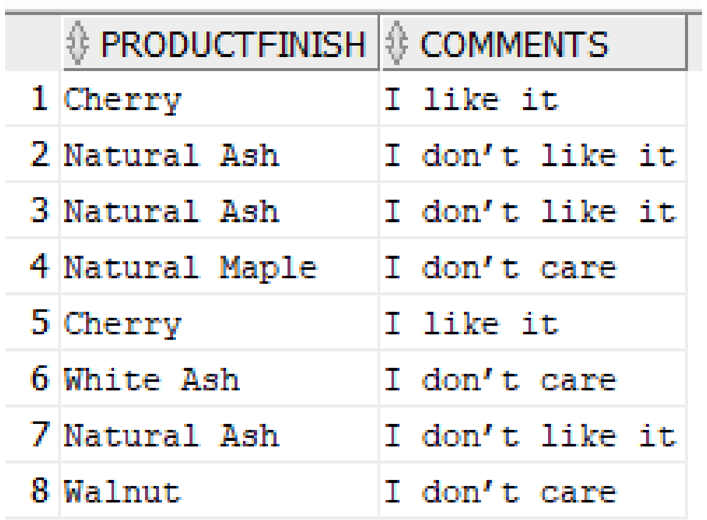
If the finish is “Cherry”, display ”I like it”.

If the finish is “Natural Ash”, display “I don’t like it”.

Otherwise, display “I don’t care”.

```
SELECT ProductFinish,
       CASE ProductFinish
        WHEN 'Cherry' THEN 'I like it'
        WHEN 'Natural Ash' THEN 'I don’t like it'
        ELSE 'I don’t care'
        END AS Comments
FROM Product_T;
```

One column with 3 possible values



	PRODUCTFINISH	COMMENTS
1	Cherry	I like it
2	Natural Ash	I don’t like it
3	Natural Ash	I don’t like it
4	Natural Maple	I don’t care
5	Cherry	I like it
6	White Ash	I don’t care
7	Natural Ash	I don’t like it
8	Walnut	I don’t care

Searched CASE Statement (1)

❑ Syntax

```
CASE
    WHEN search-condition-1 THEN statement-1
    WHEN search-condition-2 THEN statement-2
     … …
    WHEN search-condition-N THEN statement-N
    ELSE statement-N+1
END
```

❑ How it works

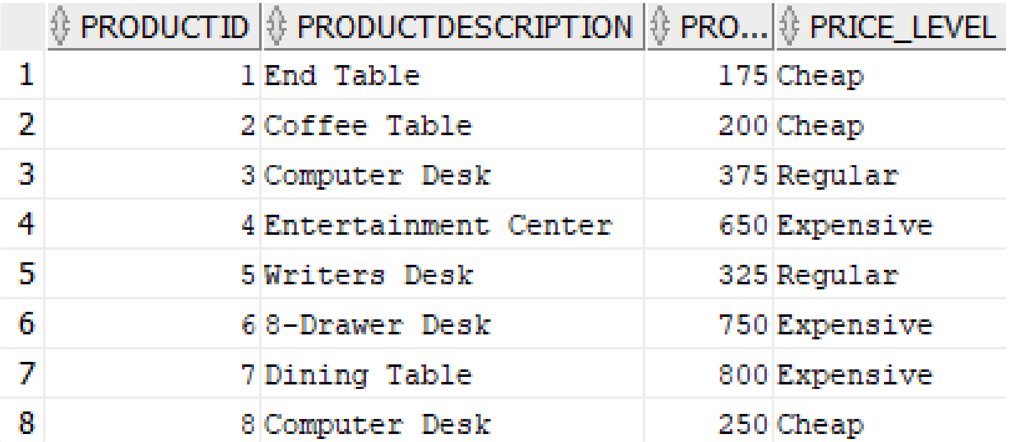
- A search condition yields a Boolean value: TRUE, FALSE, or NULL
- WHEN clauses are evaluated sequentially
- if search_condition = TRUE, execute the associated statement(s); and then jump to END
- if no search_condition = TRUE, execute the ELSE clause

Searched CASE Statement (2)

Query: Categorize all products into 3 price levels:

1) price >= $600 → ‘Expensive’;
2) $600 > price >= 300 → ‘Regular’;
3) price < 300 → ‘Cheap’.

Show the price level of each product.



	PRODUCTID	PRODUCTDESCRIPTION	PRO...	PRICE_LEVEL
1	1	End Table	175	Cheap
2	2	Coffee Table	200	Cheap
3	3	Computer Desk	375	Regular
4	4	Entertainment Center	650	Expensive
5	5	Writers Desk	325	Regular
6	6	8-Drawer Desk	750	Expensive
7	7	Dining Table	800	Expensive
8	8	Computer Desk	250	Cheap

```
SELECT ProductID, ProductDescription, ProductStandardPrice,
    CASE
    WHEN ProductStandardPrice >= 600 THEN 'Expensive'
    WHEN ProductStandardPrice >= 300 THEN  'Regular'
    ELSE 'Cheap'
    END AS price_level
FROM product_T;
```

NO column after CASE

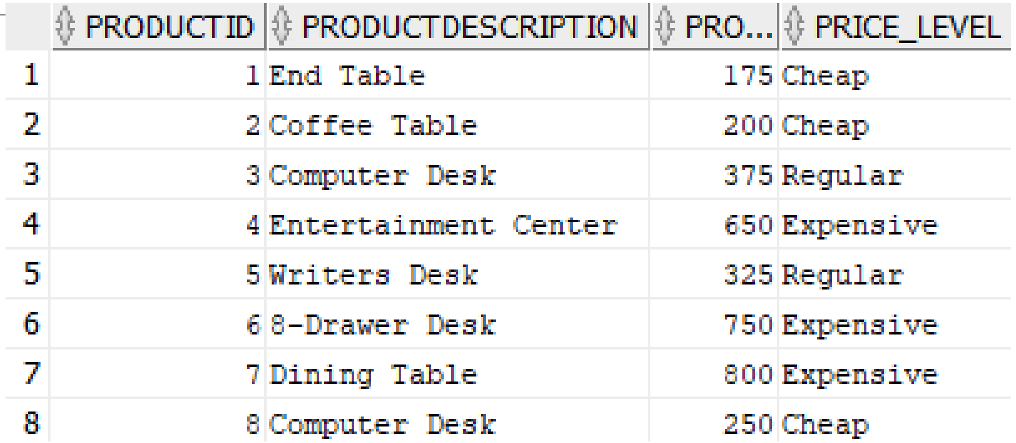
WHEN + Test Conditions

Searched CASE Statement (3)

Query: Categorize all products into 3 price levels:

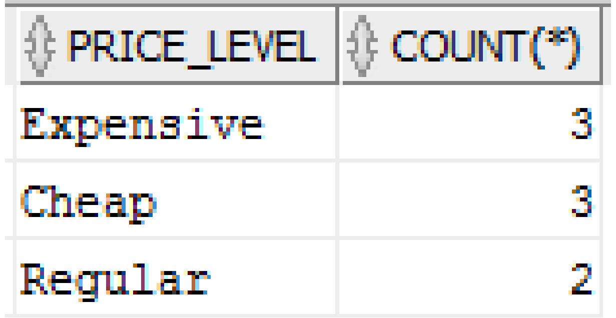
1) price >= $600 → ‘Expensive’;
2) $600 > price >= 300 → ‘Regular’;
3) price < 300 → ‘Cheap’.

Show the **number of products** in **each** price level.



	PRODUCTID	PRODUCTDESCRIPTION	PRO...	PRICE_LEVEL
1	1	End Table	175	Cheap
2	2	Coffee Table	200	Cheap
3	3	Computer Desk	375	Regular
4	4	Entertainment Center	650	Expensive
5	5	Writers Desk	325	Regular
6	6	8-Drawer Desk	750	Expensive
7	7	Dining Table	800	Expensive
8	8	Computer Desk	250	Cheap

```
SELECT price_level, count(*)
FROM (SELECT CASE
                    WHEN ProductStandardPrice >= 600 THEN 'Expensive'
                    WHEN ProductStandardPrice >= 300 THEN 'Regular'
                    ELSE 'Cheap'
                    END AS price_level
          FROM product_T)
GROUP BY price_level;
```



PRICE_LEVEL	COUNT(*)
Expensive	3
Cheap	3
Regular	2

Searched CASE Statement (5)

Q. Find the total number of Tables and Desks. Show them in the same result table.

```
SELECT
sum(case when productdescription like '%Table%' then 1 else 0 end) as Tables,
sum(case when productdescription like '%Desk%' then 1 else 0 end) as Desks
FROM product_t;
```

TABLES	DESKS
3	4