UTS:ENGINEERING AND INFORMATION TECHNOLOGY

UTS
UNIVERSITY OF TECHNOLOGY, SYDNEY

lecture 10: SQL III

Correlated Subquery

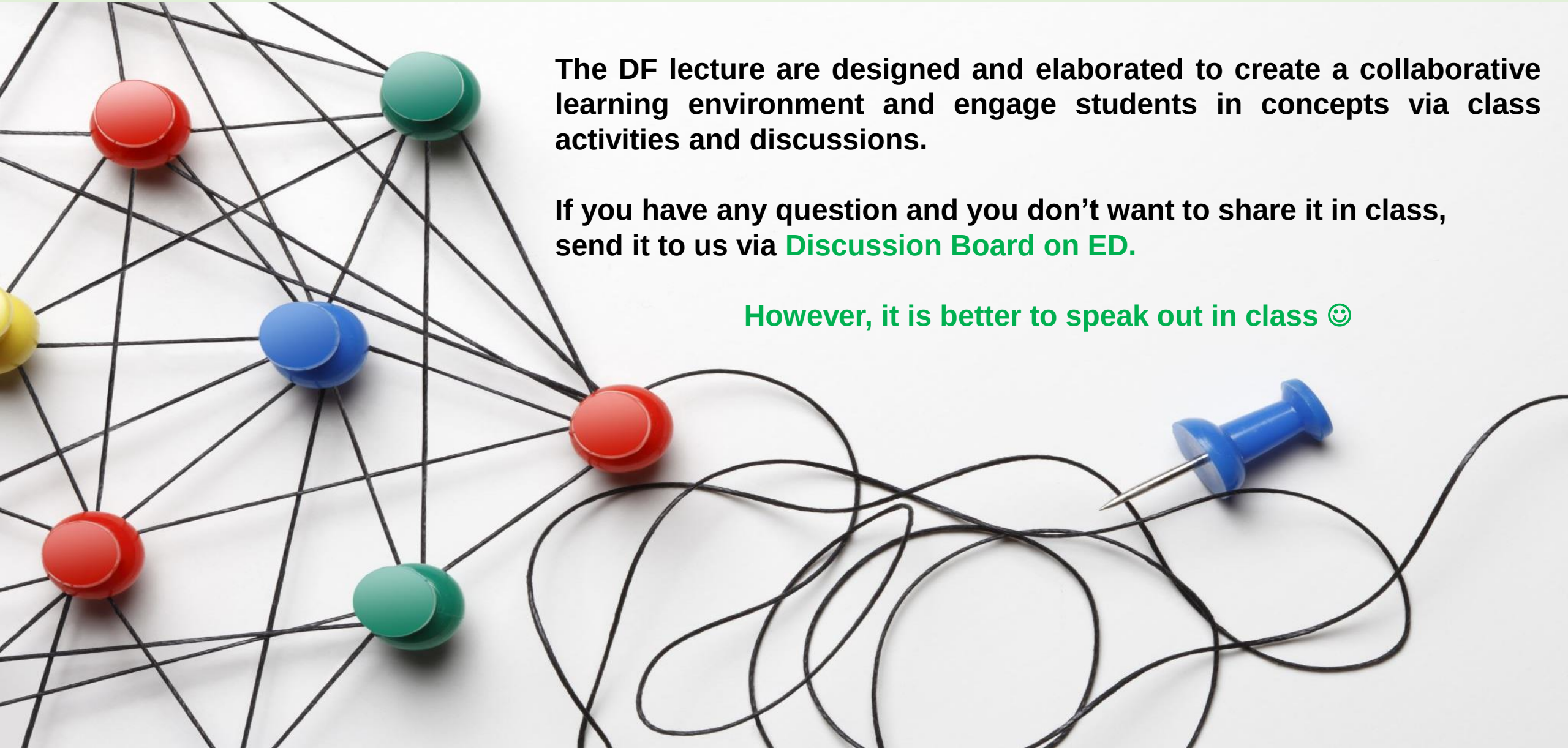
Main reference:

Modern Database Management, 11th Edition
Chapter 7: Advanced SQL

Subject Coordinator and Instructor:

Dr. Danna (Fahimeh) Ramezani

Participations and Discussions

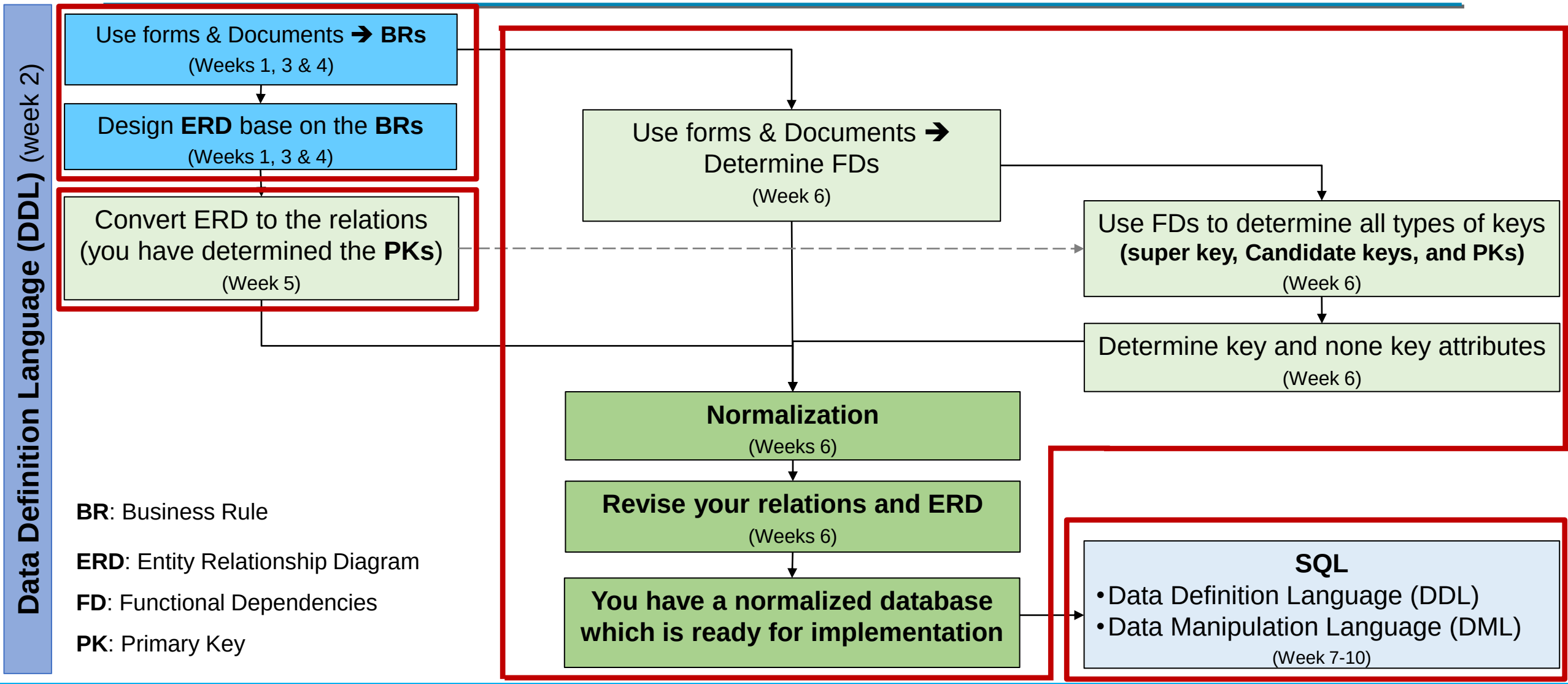


The DF lecture are designed and elaborated to create a collaborative learning environment and engage students in concepts via class activities and discussions.

If you have any question and you don't want to share it in class, send it to us via Discussion Board on ED.

However, it is better to speak out in class ☺

Subject Flowchart



Subject Overview

- **Design Entity Relationship Diagram (ERD)**
  - **Week 1: Data Modelling I (Conceptual Level):** Entity, Attributes, PK, FK, …
  - **Week 2: Data Definition Language (DDL):** Create tables, constraints, insert, …
  - **Week 3: Data Modelling II (Conceptual Level):** Associative, Weak, …
  - **Week 4: Data Modelling III (Conceptual Level):** Subtype/Supertype
  - **Week 5: Convert ERD to Relations (Logical Level)**
  - **Week 6: Functional Dependencies, and Normalization**
- **Data manipulation**
  - **Week 7: Simple Query**
  - **Week 8: Multiple Table Queries**
  - **Week 9: Subquery**
  - **Week 10: Correlated Subquery**

Lecture Objectives:

1. Correlated Subquery
2. Examples

Processing Multiple Tables Using Subqueries

Subquery is an **inner** query (SELECT statement) inside an **outer** query.

Options:

- In a condition of the WHERE clause
- As a "**table**" of the FROM clause
- Within the HAVING clause

Subqueries (Nested queries) can be:

- Noncorrelated (Simple or Type 1)– executed once for the entire outer query
- **Correlated– executed once for each row returned by the outer query**

Subquery Example

Question 1: List all products whose price is above average price of products with ‘Oak’ finished.

```
SELECT productdescription, productstandardprice, productfinish
FROM product_t Table_a
WHERE productstandardprice >
      (SELECT avg(productstandardprice)
       FROM product_t Table_b
       WHERE Table_b.productfinish= 'Oak’)
ORDER BY productfinish;
```

avg
592.50

productdescription	productstandardprice	productfinish
8-Drawer Dresser	750.00	Birch
Entertainment Center	1650.00	Cherry
6' Grandfather Clock	890.00	Oak
7' Grandfather Clock	1100.00	Oak
8-Drawer Dresser	800.00	Oak
Oak Computer Desk	750.00	Oak
Amoire	1200.00	Walnut

7 rows

Correlated Subquery Example

Question 2: List all products with a standard price above the average price of products with the same finish.

product_t Table_a

productid	productlineid	productdescription	productfinish	productstandardprice
1	1	Cherry End Table	Cherry	175.00
2	1	Birch Coffee Tables	Birch	200.00
3	1	Oak Computer Desk	Oak	750.00
4	1	Entertainment Center	Cherry	1650.00
5	2	Writer's Desk	Oak	325.00
6	1	8-Drawer Dresser	Birch	750.00
7	3	48 Bookcase	Walnut	150.00
...	...	...	...	...

```
SELECT  Table_a.productdescription, Table_a .productstandardprice, Table_a .productfinish
FROM product_t Table_a
WHERE productstandardprice >
    (SELECT avg(productstandardprice)
     FROM product_t Table_b
     WHERE Table_b.productfinish =  Table_a.productfinish )
ORDER BY productfinish;
```

avg
658.33

productdescription	productstandardprice	productfinish
Writer's Desk	512.00	Birch
8-Drawer Dresser	750.00	Birch
Entertainment Center	1650.00	Cherry
7' Grandfather Clock	1100.00	Oak
Oak Computer Desk	750.00	Oak
8-Drawer Dresser	800.00	Oak
6' Grandfather Clock	890.00	Oak
Amoire	1200.00	Walnut

Please run the ppt and follow the animations to understand the process of executing the query.

Another way to explain the process of executing the query.

```
SELECT Table_a.productdescription, Table_a .productstandardprice, Table_a .productfinish
FROM product_t Table_a
WHERE productstandardprice >
        (SELECT avg(productstandardprice)
         FROM product_t Table_b
         WHERE Table_b.productfinish = Table_a.productfinish )
ORDER BY productfinish;
```

avg
912.50

175.00 < 912.5 so "Cherry End Table" will not be in the result table of the outer query … and the story continues for every row of Table_a …

Product_t Table_a

ProductID	ProductDescription	ProductFinish	ProductStandardPrice	ProductLineID
1	Cherry End Table	Cherry	175.00	1
2	Birch Coffee Tables	Birch	200.00	1
3	Oak Computer Desk	Oak	750.00	1
4	Entertainment Center	Cherry	1650.00	1
5	Writer's Desk	Oak	325.00	2
6	8-Drawer Dresser	Birch	750.00	1
7	48 Bookcase	Walnut	150.00	3
8	48 Bookcase	Oak	175.00	3
…	…	…	…	…

Table_a.productfinish

Product_t Table _b

ProductID	ProductDescription	ProductFinish	ProductStandardPrice	ProductLineID
1	Cherry End Table	Cherry	175.00	1
2	Birch Coffee Tables	Birch_S	200.00	1
3	Oak Computer Desk	Oak	750.00	1
4	Entertainment Center	Cherry	1650.00	1
5	Writer's Desk	Oak	325.00	2
6	8-Drawer Dresser	Birch	750.00	1
7	48 Bookcase	Walnut	150.00	3
8	48 Bookcase	Oak	175.00	3
…	…	…	…	…

Then … Average 175.00 and 1650.00 which is (175.00+ 1650.00)/2=912.5 will be passed to the outer query to be used.

Processing a correlated subquery:

- **List all products whose price is above average price of products with same finished.**

In the outer query,

- SQL engine goes to **Product-t table (Table_a)**,
- then starts from the first row of this table, and
- then take the value of ProductFinish (**Table_a.ProductFinish** that is cherry in the first row)
- then pass this value of ProductFinish (i.e. **Table_a.ProductFinish**) to the subquery.

In the subquery,

- SQL engine goes to **Product_t table (Table_b)**,
- then find the **rows** where **Table_b.productfinish=Table_a.productfinish,**
- then calculate the average standard price of the products related to these rows (i.e. the products that finish in the current value of **Table_a.productfinish**)
- then pass this average value to the outer query to show in the result table of the outer query.

The SQL engine **then starts again from the outer query**, goes to the second row of the **Product-t table (Table_a)**, and the process will be repeated for each row of the table in the outer query (that is **Product-t table (Table_a)** in this example).

Therefore, the subquery executes once for each row of the table in the outer query.

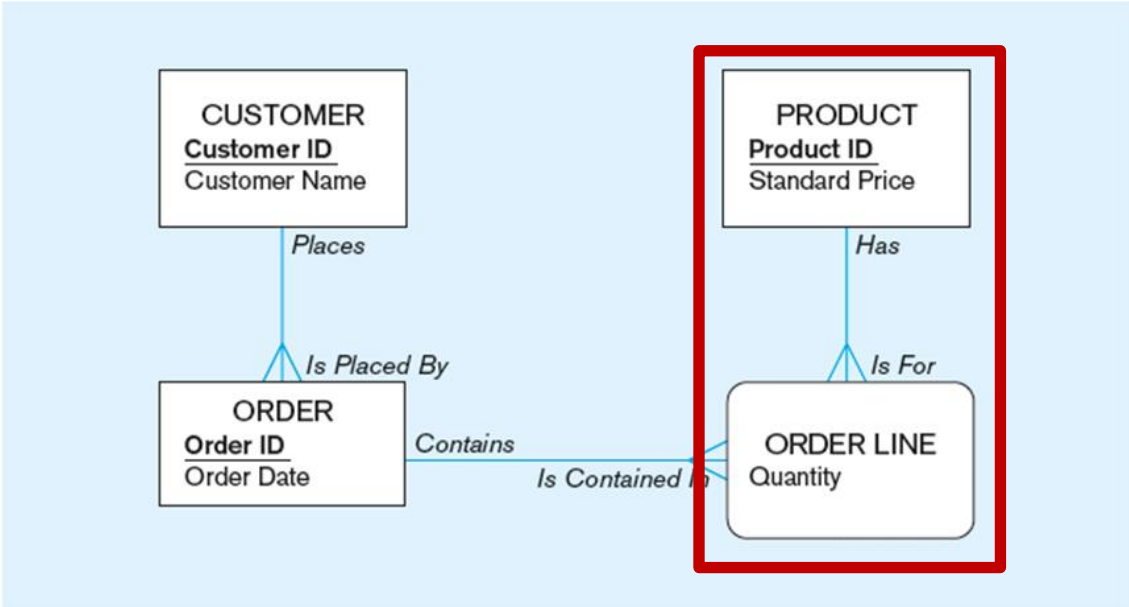
Correlated vs. Noncorrelated Subqueries

- **Noncorrelated subqueries:**
  - **Do not depend on** data from the outer query
  - Execute **once** for the **entire** outer query
- **Correlated subqueries:**
  - **Make use of** data from the outer query
  - Execute once **for each row of** the outer query
  - Can use the **EXISTS** operator

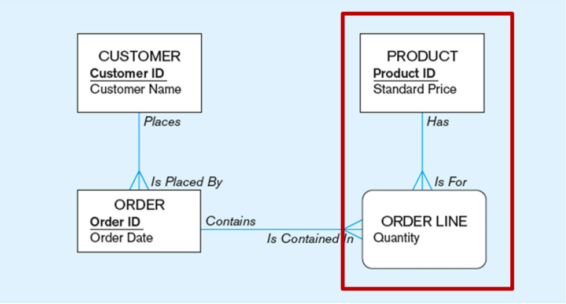
Questions 3, 4 and 5

Show all orders that include furniture finished in Oak using:

- **Join**
- **Simple Subquery**
- **Correlated Subquery**



Question 3: Show all orders that include furniture finished in Oak using Join



Note: Order_ID and **Product_ID** are FKs in Orderline_T

Let's run the following query first and check the result table:

```
SELECT orderid, product_t.productid, productdescription, productfinish
FROM orderline_t, product_t
WHERE product_t.productid = orderline_t.productid
    and productfinish='Oak';
```

orderid	productid	productdescription	productfinish
1	10	96 Bookcase	Oak
2	3	Oak Computer Desk	Oak
2	8	48 Bookcase	Oak
4	3	Oak Computer Desk	Oak
4	5	Writer's Desk	Oak
32	5	Writer's Desk	Oak
51	3	Oak Computer Desk	Oak
54	3	Oak Computer Desk	Oak
58	3	Oak Computer Desk	Oak
63	3	Oak Computer Desk	Oak
71	3	Oak Computer Desk	Oak

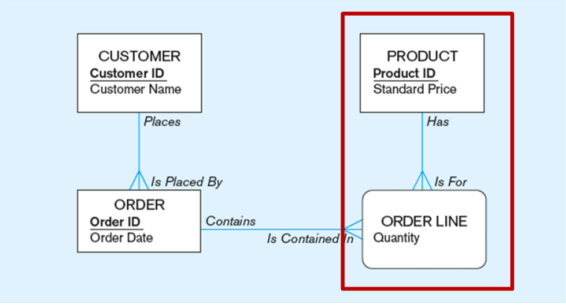
11 rows

Now, only show the order IDs.

```
SELECT distinct(orderid)
FROM orderline_t, product_t
WHERE product_t.productid = orderline_t.productid
    and productfinish='Oak';
```

```
orderid
----------
      1
      2
      4
     32
     51
     54
     58
     63
     71
(9 rows)
```

Question 4: Show all orders that include furniture finished in Oak using simple subquery.



Note: **Order_ID** and **Product_ID** are FKs in Orderline_T

```
SELECT distinct(orderid)
FROM orderline_t
WHERE productid in (SELECT productid
                    FROM product_t
                    WHERE productfinish='Oak');
```

```
productid
----------
         3
         5
         8
        10
        11
        12
        18
        19
(8 rows)
```

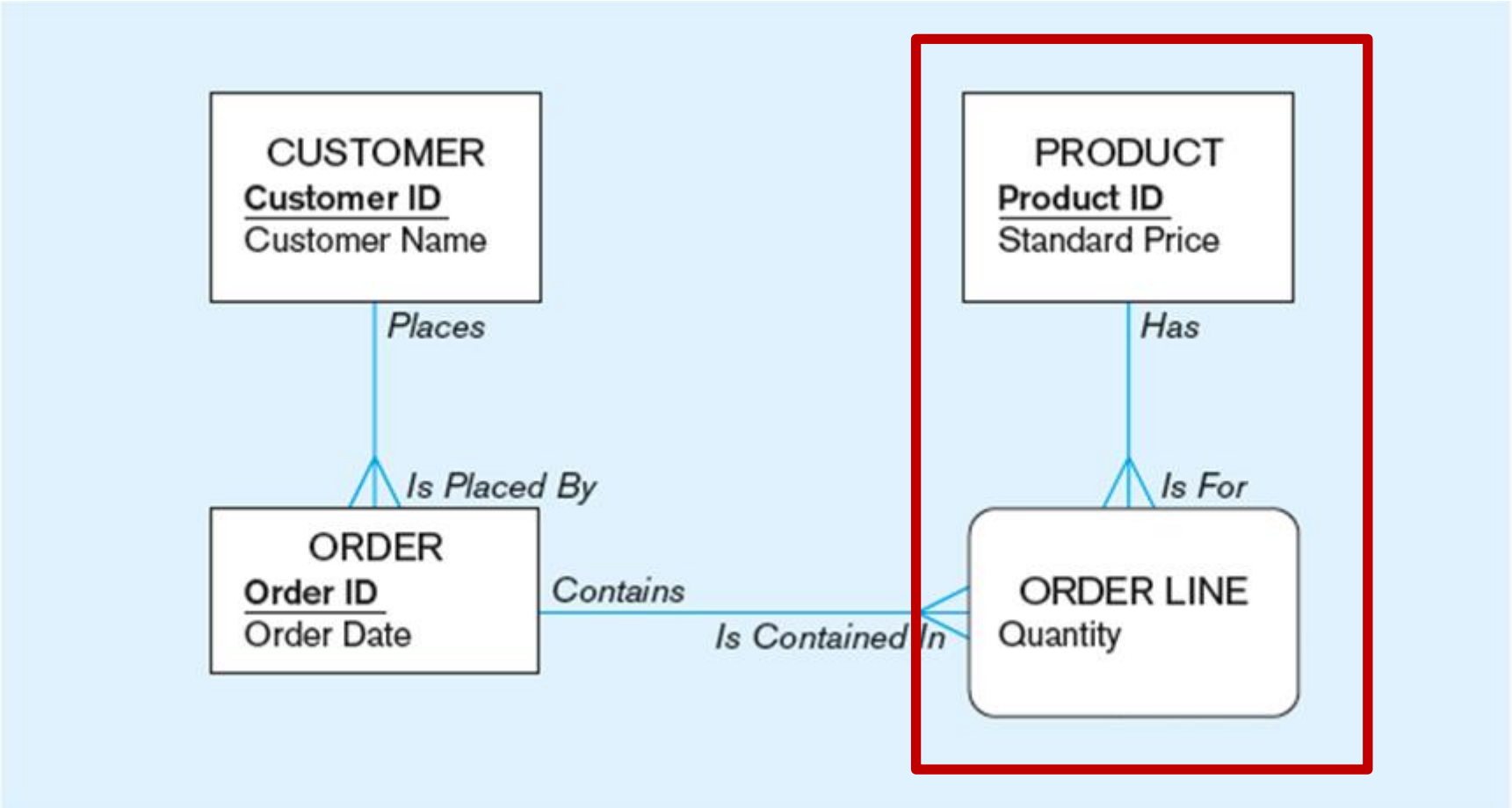
```
orderid
---------
      1
      2
      4
     32
     51
     54
     58
     63
     71
(9 rows)
```

Non-correlated subqueries:

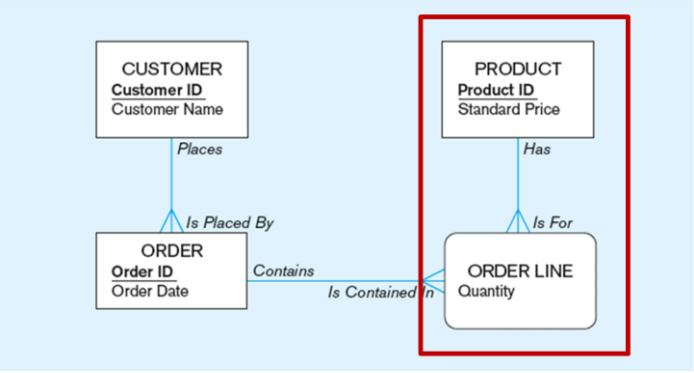
* Do not depend on data from the outer query

* Execute once for the entire outer query

Question 5: Show all orders that include furniture finished in Oak using Correlated Subquery.



Question 5: Show all orders that include furniture finished in Oak using Correlated Subquery.



Note: **Order_ID** and **Product_ID** are FKs in Orderline_T

```
SELECT distinct(orderid)
FROM orderline_t
WHERE EXISTS (SELECT *
              FROM product_t
              WHERE productid=orderline_t.productid
                  and productfinish='Oak');
```

```
orderid
---------
      1
      2
      4
     32
     51
     54
     58
     63
     71
(9 rows)
```

Processing a correlated subquery:

- Show all orders that include 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');
```

orderline_t.productid

Orderline_t

	OrderID	ProductID	OrderQuantity
1	1001	1	1
2 ✓	1001	2	2
3	1001	4	1
4 ✓	1002	3	5
	1003	3	3
	1004	6	2
	1004	8	2
	1005	4	4
	1006	4	1
	0	0	0

Product_t

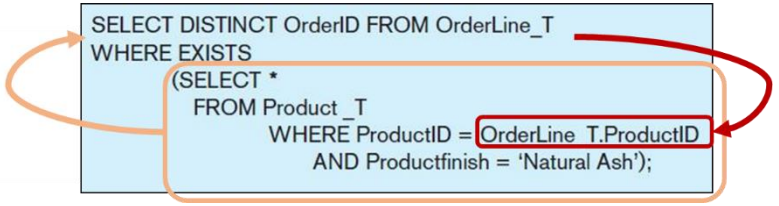
ProductID	ProductDescription	ProductFinish	ProductStandardPrice	ProductLineID
1	End Table	Cherry	175	10001
2	Coffee Table	Natural Ash	200	20001
3	Computer Desk	Natural Ash	375	20001
4	Entertainment Center	Natural Maple	650	30001
5	Writer's Desk	Cherry	325	10001
6	8-Drawer Dresser	White Ash	750	20001
7	Dining Table	Natural Ash	800	20001
8	Computer Desk	Walnut	250	30001

Subquery refers to outer-query data, so executes once for each row of the table in the outer query.

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

Processing a correlated subquery:

- **Show all orders that include furniture finished in natural ash.**



In the outer query,

- SQL engine goes to **Orderline-t table**,
- then starts from the first row of this table which is related to the OrderID 1001, and
- then take the related ProductID ProductID (i.e. **Orderline_t. ProductID** which equals to 1 in the first row of the table),
- then pass this ProductID (i.e. **Orderline_t. ProductID**) to the subquery.

In the subquery,

- SQL engine goes to **Product_t table**,
- then find the row where **ProductID= Orderline_t. ProductID**,
  then check if this product finish in 'Natural Ash'.
- If yes, then pass the related **OrderID** to the outer query to show in the result table of the outer query.

The SQL engine **then starts again from the outer query**, goes to the second row of the **Orderline-t table,** and the process will be repeated for each row of the table in the outer query (that is **Orderline-t table** in this example).

Therefore, the subquery executes once for each row of the table in the outer query.

Correlated Subquery (more Description for the example)

Question 5: Show all orders that include furniture finished in natural ash.

The EXISTS operator returns 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');
```

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

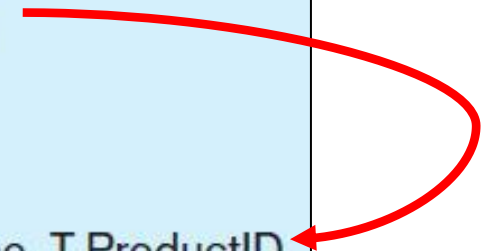
➔ **A correlated subquery always refers to an attribute from a table referenced in the outer query**

Correlated subqueries:
* Make use of data from the outer query
* Execute once for each row of the outer query
* Can use the EXISTS operator

Processing a correlated subquery (Figure 7-8b):

Show all orders that include 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');
```



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.

Subquery in FROM clause (Questions 6 and 7)

Questions 6: Calculate the average price of three groups of products with finishing in *Oak*, *Pine* and *Walnut.*

Question 7: Show all products whose standard price is higher than the average price.

Answer these questions using Subquery in from clause

Subquery in FROM clause

Question 6: Calculate the average price of three groups of products with finishing in *Oak*, *Pine* and *Walnut*.

First: Calculate the average price for each group of products that have same finishing .

```
SELECT productfinish, round(avg(productstandardprice)) as AveragePrice
FROM product_t
GROUP BY productfinish;
```

productfinish	averageprice
	0
Pine	256
Birch	487
Cherry	658
Walnut	525
Oak	593
Leather	362

Subquery in FROM clause

Question 6: Calculate the average price of three groups of products with finishing in *Oak*, *Pine* and *Walnut*.

Then we use the query in previous slide in FROM clause to calculate the average price of specific groups, e.g. 'Pine', 'Walnut', 'Oak'.

```
SELECT avg(AveragePrice)
FROM
(SELECT productfinish, round(avg(productstandardprice)) as AveragePrice
 FROM product_t
 GROUP BY productfinish) MyTable
WHERE productfinish in ('Oak', 'Pine', 'Walnut');
```

```
avg
--------
 458
(1 row)
```

Productfinish	averageprice
	0
Pine	256
Birch	487
Cherry	658
Walnut	525
Oak	593
Leather	362

Question 7: Show all products whose standard price is higher than the average price using subquery in From Clause.

```
SELECT productdescription, productstandardprice, AvgPrice
FROM
    (SELECT round(avg(productstandardprice)) as AvgPrice
     FROM product_t) Tavg , product_t
WHERE productstandardprice> AvgPrice;
```

avgprice
484

avgprice	productid	productlineid	productdescription	productfinish	productstandardprice
484	1	1	Cherry End Table	Cherry	175.00
484	2	1	Birch Coffee Tables	Birch	200.00
484	3	1	Oak Computer Desk	Oak	750.00
484	4	1	Entertainment Center	Cherry	1650.00
484	5	2	Writer's Desk	Oak	325.00
484	6	1	8-Drawer Dresser	Birch	750.00
484	7	3	48 Bookcase	Walnut	150.00
484	8	3	48 Bookcase	Oak	175.00
484	9	3	96 Bookcase	Walnut	225.00
484	10	3	96 Bookcase	Oak	200.00
484	11	1	4-Drawer Dresser	Oak	500.00
484	12	1	8-Drawer Dresser	Oak	800.00
484	13	1	Nightstand	Cherry	150.00
484	14	2	Writer's Desk	Birch	300.00
484	17	3	High Back Leather Chair	Leather	362.00
484	18	4	6' Grandfather Clock	Oak	890.00
484	19	4	7' Grandfather Clock	Oak	1100.00
484	20	2	Amoire	Walnut	1200.00
484	21	1	Pine End Table	Pine	256.00
484	24	5			0.00
484	25	2			0.00

(21 rows)

Subquery in From Clause

Question 7: Show all products whose standard price is higher than the average price.

Subquery forms the **derived table** used in the FROM clause of the outer query. This derived table should have an alias name (Tavg).

One column of the subquery is an aggregate function that has an alias name (AvgPrice). That alias can then be referred to in the outer query.

```
SELECT productdescription, productstandardprice, AvgPrice
FROM
    (SELECT round(avg(productstandardprice)) as AvgPrice
     FROM product_t) Tavg , product_t
WHERE productstandardprice> AvgPrice;
```

The WHERE clause normally **cannot include aggregate functions**, **but** because the aggregate is performed in the subquery its result can be used in the outer query's WHERE clause.

Video

What are Sub-Query and Correlated Sub-Query?

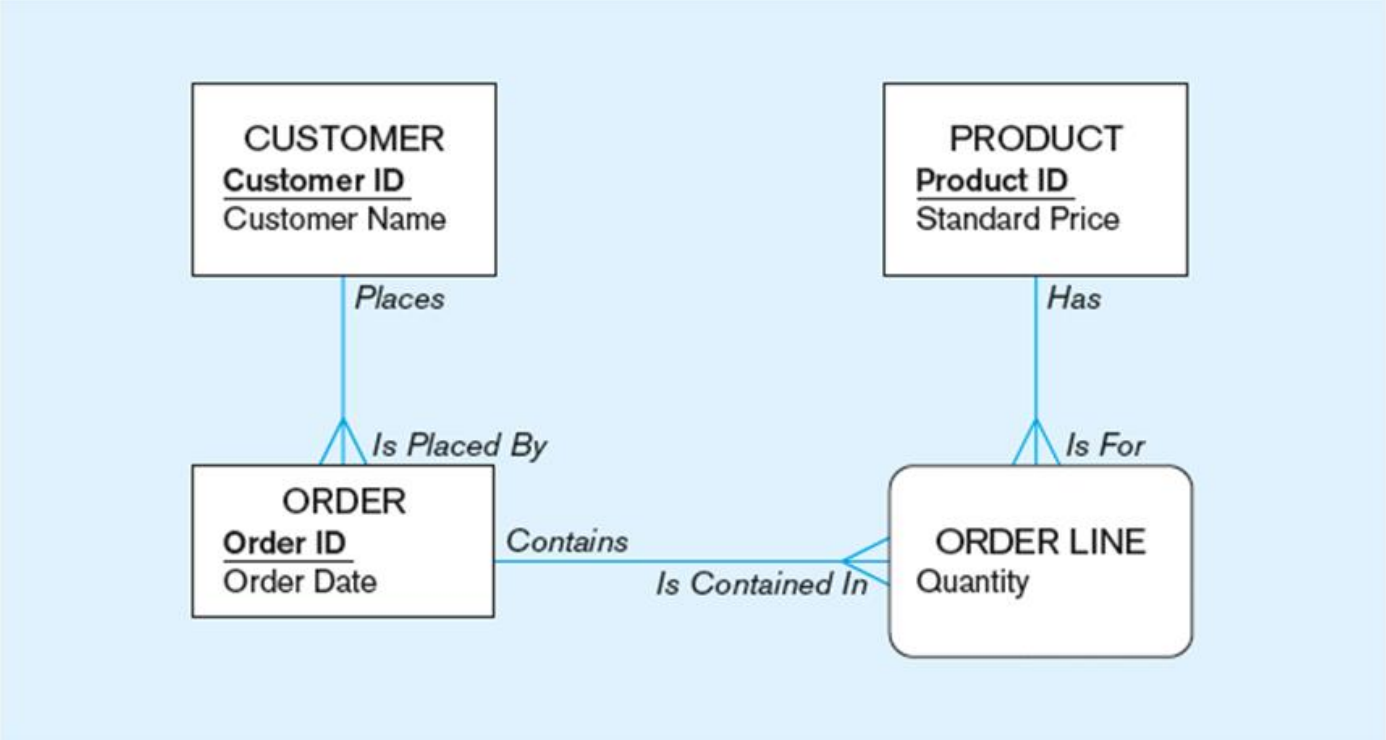
Link: 

https://www.youtube.com/watch?v=00Vxnod-6iE&index=8&list=PLJKaNMxPrhJTsly8opxBCAW2Lon8gJCge

Examples

Example 1

Produce a list of all products (product description) and the number of times each product has been ordered.



Solution 1 to Example 1: Produce a list of all products (product description) and the number of times each product has been ordered.

```
SELECT productdescription,productid,
(SELECT count(*)
 FROM orderline_t
 GROUP BY productid
 HAVING productid=p.productid) as number_of_orders
FROM product_t p;
```

productdescription	productid	number_of_orders
Cherry End Table	1	7
Birch Coffee Tables	2	5
Oak Computer Desk	3	7
Entertainment Center	4	7
Writer's Desk	5	2
8-Drawer Dresser	6	3
48 Bookcase	7	1
48 Bookcase	8	1
96 Bookcase	9	<null>
96 Bookcase	10	1
4-Drawer Dresser	11	<null>
8-Drawer Dresser	12	<null>
Nightstand	13	1
Writer's Desk	14	2
High Back Leather Chair	17	1
6' Grandfather Clock	18	<null>
7' Grandfather Clock	19	<null>
Amoire	20	1
Pine End Table	21	<null>
<null>	24	<null>
<null>	25	<null>

Solution 2 to Example 1: Produce a list of all products (product description) and the number of times each product has been ordered.

```
Select productdescription, p.productid, mycount
From
(select productid, count(*) as mycount
 from orderline_t
 group by productid)MT
 full outer join product_t p on MT.productid= p.productid;
```

productdescription	productid	mycount
Cherry End Table	1	7
Birch Coffee Tables	2	5
Oak Computer Desk	3	7
Entertainment Center	4	7
Writer's Desk	5	2
8-Drawer Dresser	6	3
48 Bookcase	7	1
48 Bookcase	8	1
96 Bookcase	<null>	<null>
96 Bookcase	10	1
4-Drawer Dresser	<null>	<null>
8-Drawer Dresser	<null>	<null>
Nightstand	13	1
Writer's Desk	14	2
High Back Leather Chair	17	1
6' Grandfather Clock	<null>	<null>
7' Grandfather Clock	<null>	<null>
Amoire	20	1
Pine End Table	<null>	<null>
<null>	<null>	<null>
<null>	<null>	<null>

Question: Change this query to produce a result table like the result table of the first solution.

productdescription	productid	number_of_orders
Cherry End Table	1	7
Birch Coffee Tables	2	5
Oak Computer Desk	3	7
Entertainment Center	4	7
Writer's Desk	5	2
8-Drawer Dresser	6	3
48 Bookcase	7	1
48 Bookcase	8	1
96 Bookcase	9	<null>
96 Bookcase	10	1
4-Drawer Dresser	11	<null>
8-Drawer Dresser	12	<null>
Nightstand	13	1
Writer's Desk	14	2
High Back Leather Chair	17	1
6' Grandfather Clock	18	<null>
7' Grandfather Clock	19	<null>
Amoire	20	1
Pine End Table	21	<null>
<null>	24	<null>
<null>	25	<null>

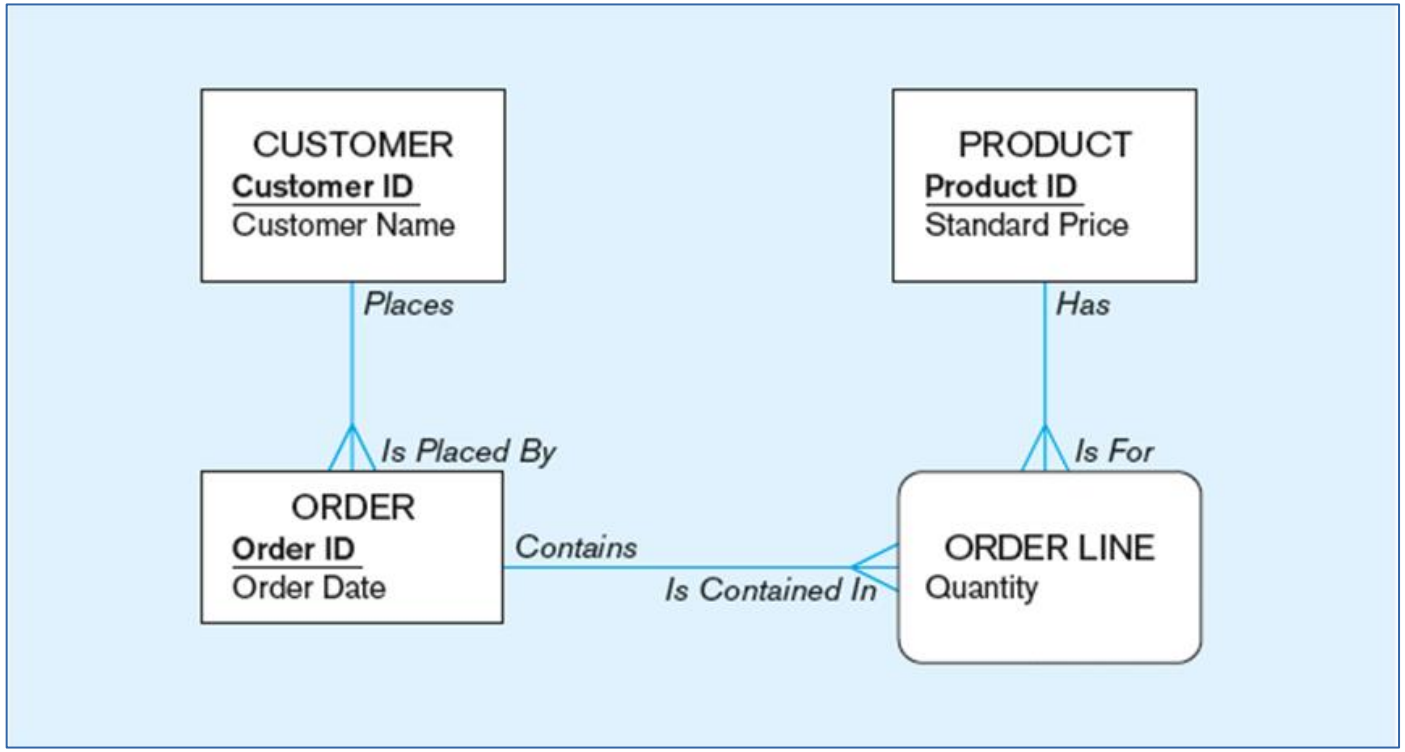
Solution 3 to Example 1: Produce a list of all products (product description) and the number of times each product has been ordered.

```
Select productdescription, p.productid, mycount
From
(select productid, count(*) as mycount
 from orderline_t
 group by productid)MT
 Right outer join product_t p on MT.productid= p.productid;
```

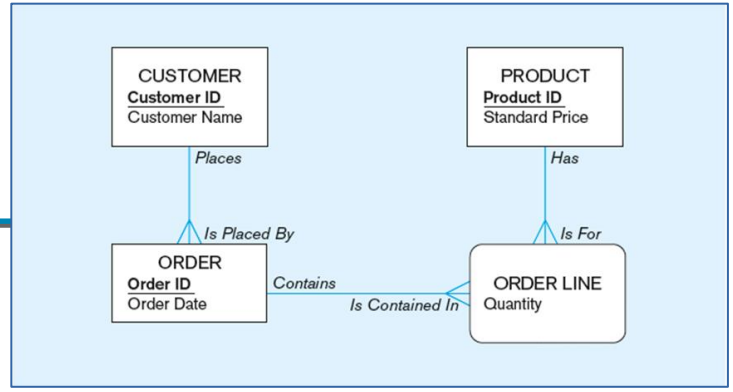
productdescription	productid	number_of_orders
Cherry End Table	1	7
Birch Coffee Tables	2	5
Oak Computer Desk	3	7
Entertainment Center	4	7
Writer's Desk	5	2
8-Drawer Dresser	6	3
48 Bookcase	7	1
48 Bookcase	8	1
96 Bookcase	9	<null>
96 Bookcase	10	1
4-Drawer Dresser	11	<null>
8-Drawer Dresser	12	<null>
Nightstand	13	1
Writer's Desk	14	2
High Back Leather Chair	17	1
6' Grandfather Clock	18	<null>
7' Grandfather Clock	19	<null>
Amoire	20	1
Pine End Table	21	<null>
<null>	24	<null>
<null>	25	<null>

Example 2

Show customers ID and name for all the customers who have ordered both product IDs 3 and 4 on the same order.



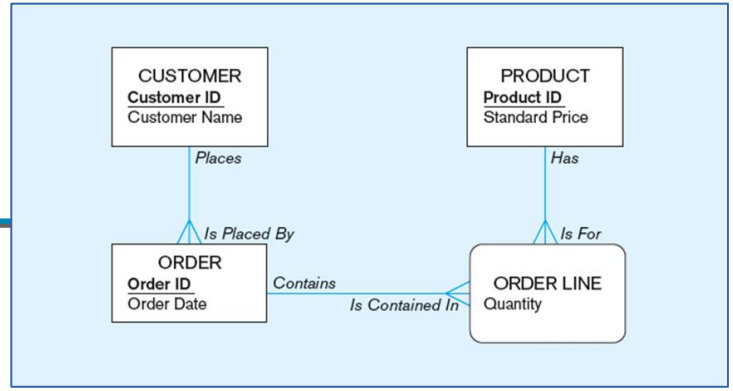
Solution 1 to Example 2: Show customers ID and name for all the customers who have ordered both product IDs 3 and 4 on the same order




```
select customerid, customername from customer_t
where customerid in
(select customerid from order_t where orderid in
   (select orderid from order_t O where
        3 in
      (select productid from orderline_t OL where OL.orderid=O.orderid)
       and 4 in
      (select productid from orderline_t OL where OL.orderid=O.orderid)));
```

```
 customerid |   customername
------------+------------------
          6 | Furniture Gallery
         16 | ABC Furniture Co.
(2 rows)
```

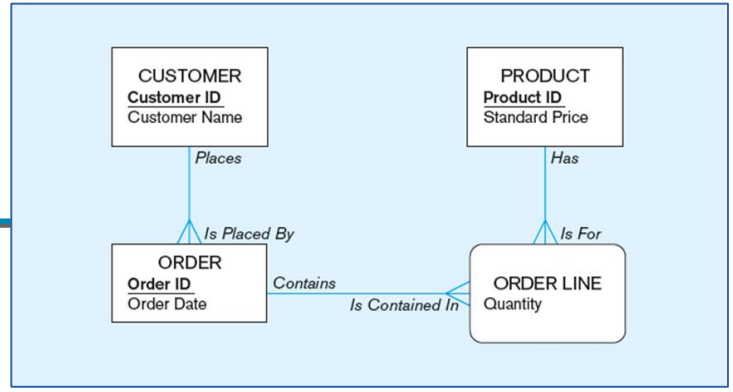
Solution 2 to Example 2: Show customers ID and name for all the customers who have ordered both product IDs 3 and 4 on the same order




```
Select customerid, customername from customer_t
where customerid in
  (select customerid from order_t where orderid in
     (select orderid from order_t O where
          exists
        (select * from orderline_t where productid=3 and orderid=O.orderid)
         and exists
        (select * from orderline_t where productid=4 and orderid=O.orderid)));
```

```
 customerid |   customername
------------+------------------
          6 | Furniture Gallery
         16 | ABC Furniture Co.
(2 rows)
```

Solution 3 to Example 2: Show customers ID and name for all the customers who have ordered both product IDs 3 and 4 on the same order



Using simple sub-query:

```
Select customerid, customername
from customer_t
where customerid in
  (select customerid from order_t
   where orderid in (select orderid from orderline_t where productid=3)
            and
              orderid in (select orderid from orderline_t where productid=4)
  )
Order by customerid;
```

```
 customerid |   customername
------------+------------------
          6 | Furniture Gallery
         16 | ABC Furniture Co.
(2 rows)
```

More Information

Conditional Expressions Using Case Syntax (Figure 7-10)

This is available with newer versions of SQL, previously not part of the standard

```
{CASE expression
{WHEN expression
THEN {expression  |  NULL}} . . .
| {WHEN predicate
THEN {expression  |  NULL}} . . .
[ELSE {expression  NULL}]
END }
| ( NULLIF (expression, expression) }
| ( COALESCE (expression . . .) }
```

```
SELECT CASE
      WHEN ProductLine = 1 THEN ProductDescription
      ELSE '####'
END AS ProductDescription
FROM Product_T;
```

```
SELECT CASE
WHEN productid between 1 and 10 THEN productdescription
WHEN productid between 11 and 15 THEN '**'
ELSE '####'
END AS productdescription
FROM product_t;
```

```
productdescription
----------------------
 Cherry End Table
 Birch Coffee Tables
 Oak Computer Desk
 Entertainment Center
 Writer's Desk
 8-Drawer Dresser
 48 Bookcase
 48 Bookcase
 96 Bookcase
 96 Bookcase
 **
 **
 **
 **
 ####
 ####
 ####
 ####
 ####
 ####
 ####
 ####
(21 rows)
```

You can create a View of the queries that are frequently required:

```
create view V1 as
select productid from product_t;
```

```
Select * from V1;
```

```
productid
-----------
          1
          2
          3
          4
          5
          6
          7
          8
          9
         10
         11
         12
         13
         14
         17
         18
         19
         20
         21
         24
         25
(21 rows)
```

Tips for Developing Queries

- Be familiar with the data model (entities and relationships)
- Understand the desired results
- Know the attributes desired in results
- Identify the entities that contain desired attributes
- Review ERD
- Construct a WHERE equality for each link
- Fine tune with GROUP BY and HAVING clauses if needed
- Consider the effect on unusual data

Query Efficiency Considerations

- Instead of SELECT *, identify the specific attributes in the SELECT clause; this helps reduce network traffic of result set
- Limit the number of subqueries; try to make everything done in a single query if possible
- If data is to be used many times, make a separate query and store it as a view

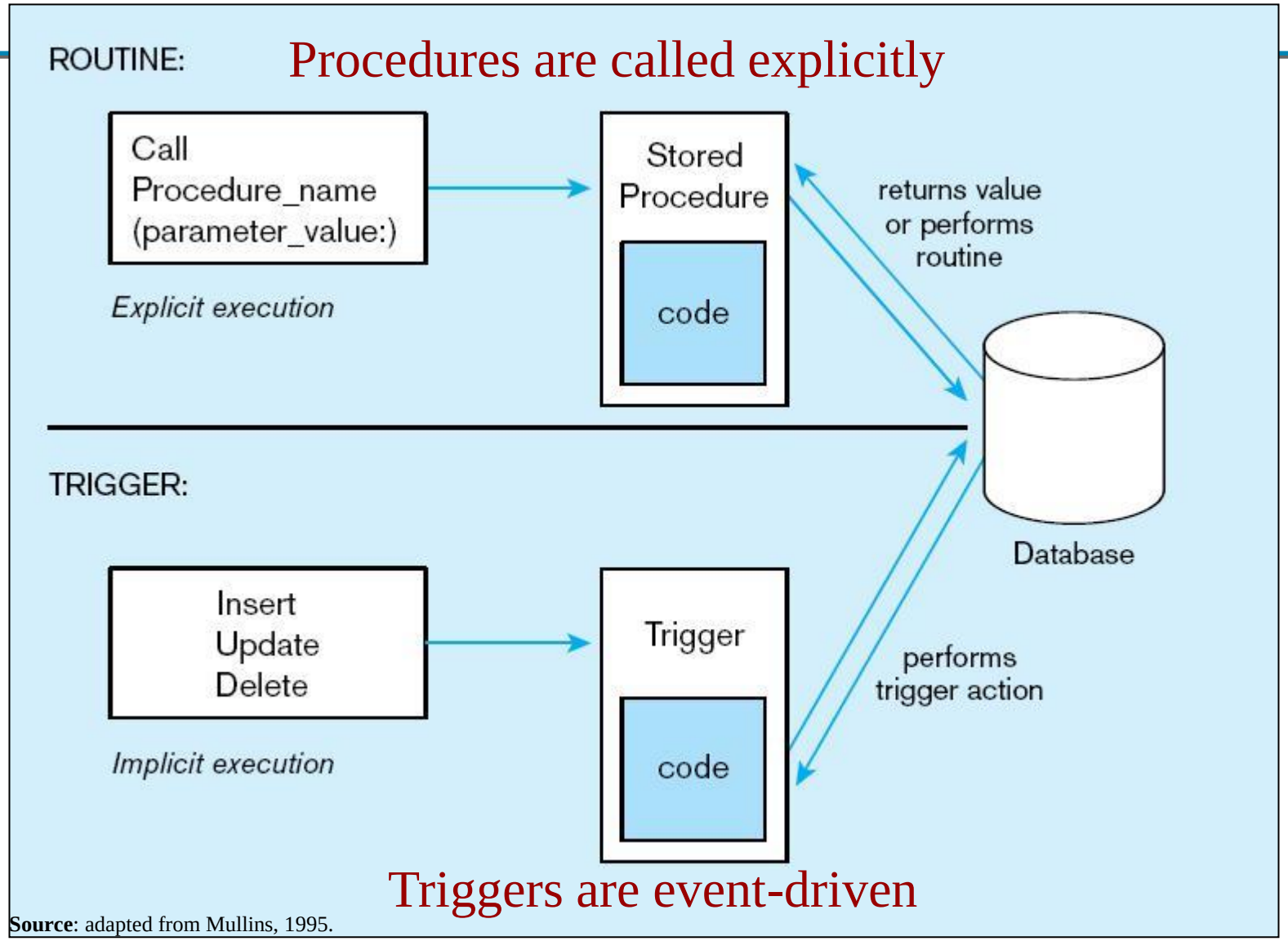
Guidelines for Better Query Design

- Understand how indexes are used in query processing
- Write simple queries
- Break complex queries into multiple simple parts
- Don’t nest one query inside another query
- Don’t combine a query with itself (if possible avoid self-joins)
- Retrieve only the data you need

Routines and Triggers

- **Routines:** Program modules that execute on demand
  - **Functions**–routines that return values and take input parameters
  - **Procedures**–routines that do not return values and can take input or output parameters
  - **Triggers**–routines that execute in response to a database event (INSERT, UPDATE, or DELETE)

Figure7-13 Triggers contrasted with stored procedures (based on Mullins 1995)



Source: adapted from Mullins, 1995.

TABLE 7-2 Comparison of Vendor Syntax Differences in Stored Procedures

The vendors' syntaxes differ in stored procedures more than in ordinary SQL. For an illustration, here is a chart that shows what CREATE PROCEDURE looks like in three dialects. We use one line for each significant part, so you can compare dialects by reading across the line.

SQL:1999/IBM	MICROSOFT/SYBASE	ORACLE
CREATE PROCEDURE	CREATE PROCEDURE	CREATE PROCEDURE
Sp_proc1	Sp_proc1	Sp_proc1
(param1 INT)	@param1 INT	(param1 IN OUT INT)
MODIFIES SQL DATA BEGIN DECLARE num1 INT;	AS DECLARE @num1 INT	AS num1 INT; BEGIN
IF param1 <> 0	IF @param1 <> 0	IF param1 <> 0
THEN SET param1 = 1;	SELECT @param1 = 1;	THEN param1 :=1;
END IF		END IF;
UPDATE Table1 SET column1 = param1;	UPDATE Table1 SET column1 = @param1	UPDATE Table1 SET column1 = param1;
END		END

Source: Data from *SQL Performance Tuning* (Gulutzan and Pelzer, Addison-Wesley, 2002). Viewed at **www.tdan.com/i023fe03.htm**, June 6, 2007 (no longer available from this site).

Embedded and Dynamic SQL

- Embedded SQL
  - Including hard-coded SQL statements in a program written in another language such as C or Java
- Dynamic SQL
  - Ability for an application program to generate SQL code on the fly, as the application is running

Reasons to Embed SQL in 3GL

- Can create a more flexible, accessible interface for the user
- Possible performance improvement
- Database security improvement; grant access only to the application instead of users

Message from previous students ☺
Angelo Athanasiou (DF Grade HD)

- **Why read the test book:**
  The modern database management textbook covers everything more in-depth than the lectures and will greatly help with understanding any areas that are unclear, the textbook is also available from the UTS library so students don't have to pay to access it. Older editions of the textbook can also be obtained for free and contain the same relevant information.

- **What to learn:**
  Learn how a relational database uses relations, cardinality, etc. because if you don't understand those concepts early on the subject won't be as clear as it progresses.

  Learn how SQL statements affect a database and what they do, as it is important to understand **how they work** instead of just understanding what they do, such as knowing why a certain output is given instead of just knowing what to do to get a certain output.

- **To aid with the transition from ERD to SQL,**
  Microsoft Access can be used to understand how things work as you can view the ERD, as well as use SQL to gain output. What I like about using Microsoft access to help people visualize is because you can use QbE to compare how a query would be undertaken in SQL.
  Links: How to use the Query By Example (QBE) grid | lynda.com tutorial:
  https://www.youtube.com/watch?v=X9vyzpdUWHs