SQL Intermediate

SQL Intermediate 3-Day Course,
Day 2
An anthology of practical, illustrative SQL Query examples

By: Sarah Barnard

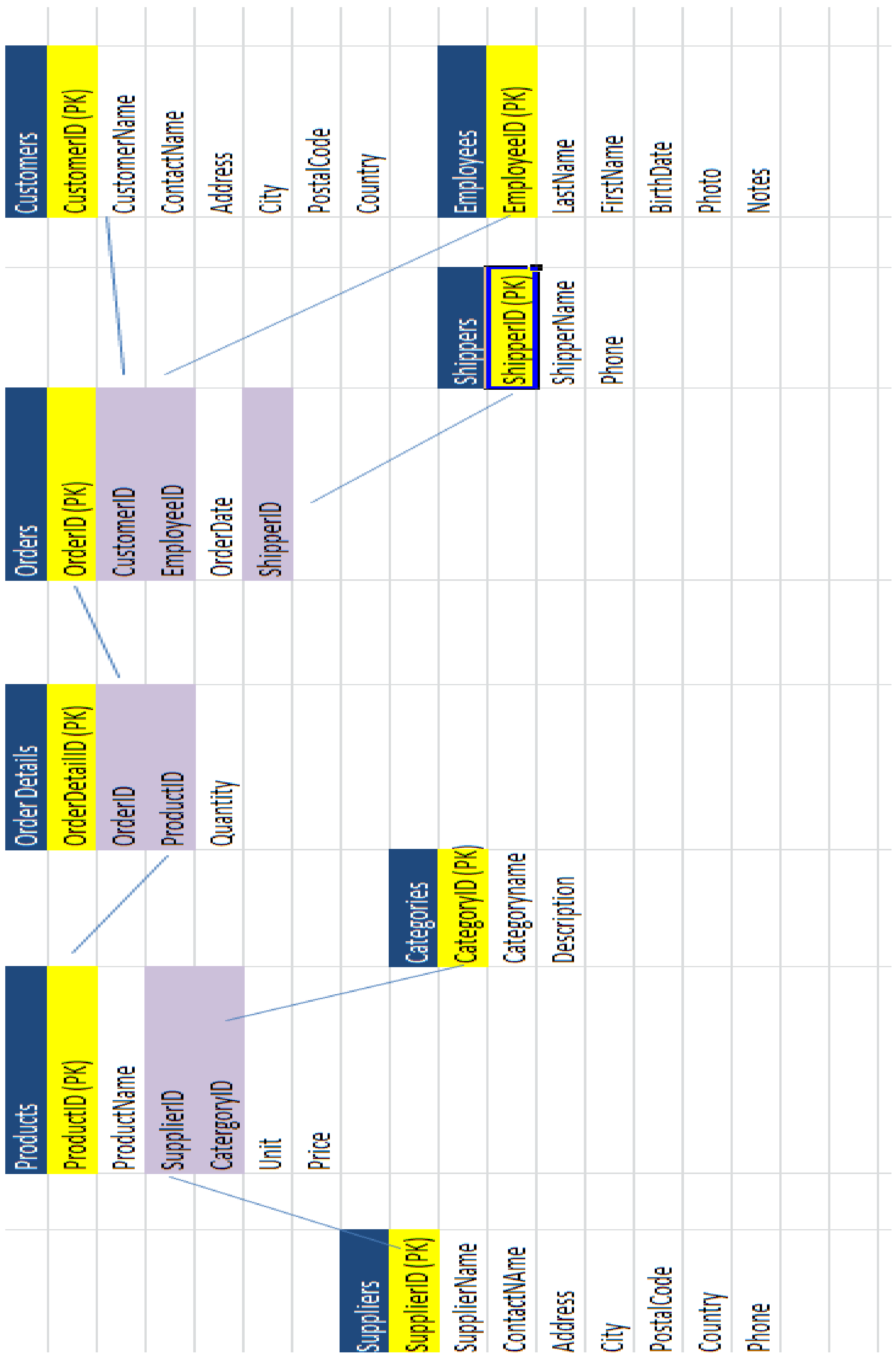
SQL Intermediate Course: Day 2

Thank you to my assistant, Dr Mary Smith., for assisting with formatting and editing this document.

SQL Intermediate

Day 2

Table of Content


Customers
CustomerID (PK)
CustomerName
ContactName
Address
City
PostalCode
Country
Employees
EmployeeID (PK)
LastName
FirstName
BirthDate
Photo
Notes
Shippers
ShipperID (PK)
ShipperName
Phone
Orders
OrderID (PK)
CustomerID
EmployeeID
OrderDate
ShipperID
Order Details
OrderDetailID (PK)
OrderID
ProductID
Quantity
Categories
CategoryID (PK)
Categoryname
Description
Products
ProductID (PK)
ProductName
SupplierID
CatergoryID
Unit
Price
Suppliers
SupplierID (PK)
SupplierName
ContactNAme
Address
City
PostalCode
Country
Phone

Part 1: Revision

Queries: Inner Joins

1. Show the Orderdate, OrderId and CustomerName ordered by Orderdate, OrderID
   - *Tables : Orders , Customer*

2. Show the Orderdate, OrderId , CustomerName , **FirstName** ordered by Orderdate, OrderID
   - *Tables: Orders , Customer , **Employees***

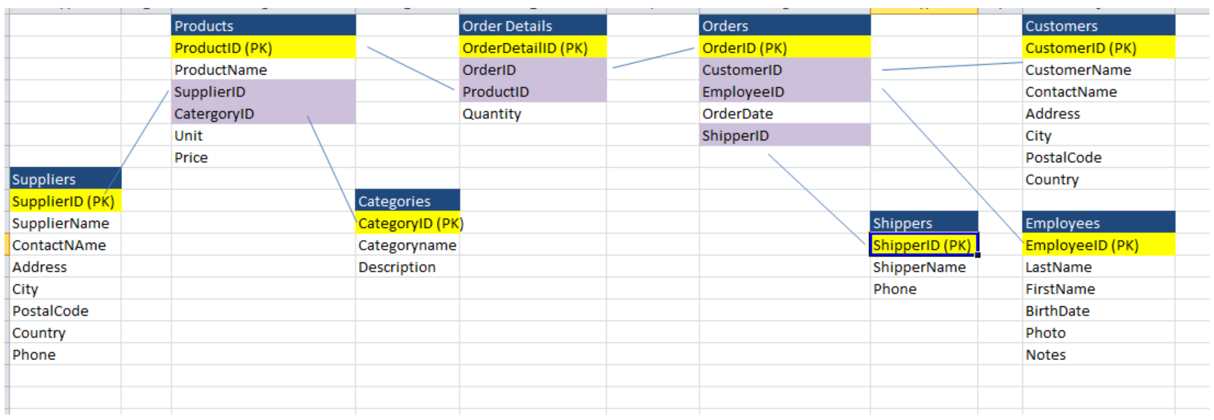
3. Show the Orderdate, OrderId , CustomerName , FirstName, **ShipperName** ordered by Orderdate, OrderID
   - *Tables: Orders , Customer , Employee, **Shippers***

4. Show the Orderdate, OrderId , CustomerName , FirstName, ShipperName , **Quantity** ordered by Orderdate, OrderID
   - *Tables: Orders , Customers , Employees, Shippers , **OrderDetails***

5. Show the Orderdate, OrderId , CustomerName , FirstName, ShipperName , Quantity, **Productname , Price** ordered by Orderdate, OrderID
   - *Tables: Orders , Customers , Employees, Shippers , OrderDetails , **Products***

6. Show the Orderdate, OrderId , CustomerName , FirstName, ShipperName , Quantity , Productname , Price, **CategoryName** ordered by Orderdate, OrderID
   - *Tables: Orders , Customers , Employees, Shippers , OrderDetails , Products, **Categories***

7. Show the Orderdate, OrderId , CustomerName , FirstName, ShipperName , Productname , Quantity , Price, CategoryName, **SupplierName** ordered by Orderdate, OrderID
   - *Tables: Orders , Customers , Employees, Shippers , OrderDetails , Products, Categories, **Suppliers***



1.

Query exercise – Join and Order By

8. Identify employees who are graduates, have a degree or went to college or university
9. Show the SupplierName, CategoryName, ProductName, Unit and Price for suppliers in the UK only, ordered by SupplierName, CategoryName, then ProductName
10. Select all columns from Customers Table for records where the CustomerName ends with iste
11. Select all columns from Customers Table for records where the CustomerName ends with essen
12. Select all columns from Customers Table for records where the CustomerName contains essen
13. Select all columns from Customers Table for records where the CustomerName contains grocer
14. Show all the columns from the Products Table where the Productname is Chais , Chang or Ikura (using IN)
15. Select all columns from Products Table for records where the CustomerID is between 10 and 15
16. Select all columns from Customers Table for records where the CustomerName is between a and b
17. Select all columns from Customers Table for records where the country is between a and m
18. Select all columns from Suppliers Table for suppliers in the UK and the suppliername contains ltd or the contactname contains pete
19. Show all the products on the Products Table where the price is null
20. Show the CategoryName, SupplierName, ProductName , Unit and Price
    - for beverages only and suppliers in the UK,
    - ordered by CatgoryName, then Suppliername, then ProductName
21. Show the SupplierName, ProductName, OrderDetailID, Price, Quantity
    - ordered by SupplierName, then ProductName
22. Show the SupplierName, OrderID, ProductName, Price, Quantity
    - ordered by SupplierName, then OrderID
23. Show EmployeeID, Productname
    - Where Categoryname = Beverages
    - ordered by EmployeeID, then Product

24. Which customers buy from suppliers in the same country
25. Which employee made the most orders
26. Which 5 employee made the most revenue
27. Which employee sold the most beverages

Query exercise – Aggregate

28. Show the number of orders per customer
29. Show the number of orders per employee
30. Show the number of orders per shipper
31. Show the number of orders per product
32. Show the number of orders per category (join orderdetails and products)

Query exercise – Join and aggregate and Order By

33. Show the OrderID, sum of Quantity , sum(Price * Quantity) ordered by OrderID
34. Show the number of orders for beverages
35. Show the number of orders for beverages from suppliers in the US

36. Show the Number of Orders per Customer, per Product
37. Show the Number of Orders per Customer, per Supplier, per Product
38. Show the Number of Orders per Customer, per Category, per Product
39. Show Customers where the sum of quantity on an order is more than 30

Left Joins

- List the customers with no orders
- List the employees with no orders
- List the suppliers with no products
- List the categories with no products

SQL - Wildcard Operators

We have already discussed about the SQL LIKE operator, which is used to compare a value to similar values using the wildcard operators.
SQL supports two wildcard operators in conjunction with the LIKE operator which are explained in detail in the following table.

Sr.No.	**Wildcard & Description**
1	**The percent sign (%)** Matches one or more characters. **Note** − MS Access uses the asterisk (*) wildcard character instead of the percent sign (%) wildcard character.
2	**The underscore (_)** Matches one character. **Note** − MS Access uses a question mark (?) instead of the underscore (_) to match any one character.

The percent sign represents zero, one or multiple characters. The underscore represents a single number or a character. These symbols can be used in combinations.

Syntax

The basic syntax of a '%' and a '_' operator is as follows.

```
SELECT FROM table_name
WHERE column LIKE 'XXXX%'

or

SELECT FROM table_name
WHERE column LIKE '%XXXX%'

or

SELECT FROM table_name
WHERE column LIKE 'XXXX_'

or

SELECT FROM table_name
WHERE column LIKE '_XXXX'

or

SELECT FROM table_name
WHERE column LIKE '_XXXX_'
```

You can combine N number of conditions using the AND or the OR operators. Here, XXXX could be any numeric or string value.

Part 2:

Expressions, Combine Queries and Sub-Queries

Expression Queries

Use a query to perform arithmetic calculations on columns, and display the result when the query is run without adding new columns to the table

Example:
```
SELECT ProductID, ProductName, Price,  Price * 1.2 as PricePlusVat
from PRODUCTS
```

Expressions Exercise

For each question, first decide which table (s) to use

- From the products table:
  Show the ProductID, ProductName , Price,   ___ as NewPrice
  -- calculate a new price that is 10% more than Price --

- From the products table:
  Show the ProductID,   ProductName , Price,   ___ as Vat ,   ___ as PriceInclVat
  -- calculate a VAT on the price as 20% of the Price -- ,
  -- calculate PRICE + 20% VAT on the price --

- What was the revenue (Price * Quantity) made from beverages?
- How much money (Price * Quantity) was paid to each supplier?
- What is the total order value (Price * Quantity) of all orders per customer ?
- What is the total revenue generated by customers in European countries for the category beverages

ISNULL(ColumnName,Replacement)

ISNULL(ColumnName, Replacement)
What should be displayed in this case when a value is NULL.

Isnull Example:

```
SELECT Productname, Price,  ISNULL(Price,0)
FROM products
```

MySQL COALESCE()

```
SELECT Products.ProductID,Price, Coalesce(Price,0)
FROM products
```

IIF(condition, true value, false value)

IIF(condition, true value, false value)

```
Select colukm(s)
          , iif(column operator va;ue, value_if_true, value_if_false )
FROM   products
```

Examples

```
Select productname
          , price
          , iif(price < 10, price*1.1, price)
FROM   products
```

```
Select productname
          , price
          , iif(price < 10, price,
                    iif (price > 20 , price*1.2, price)
                    )
FROM   products
```

Exercise:

- If the Orderdate is After DEC 1997 amd the employeeID = 2 Then change the employeeID on the orders table to 5
- If the category is beverages, then apply a discount of 10% to the price

MySQL
Select Productname, price   , if(price < 10, price*1.1, price) FROM products

CASE

Syntax:

```
CASE ValueForTheNewColumn
        WHEN ValueInColumnName THEN ValueForTheNewColumn
        WHEN ValueInColumnName THEN ValueForTheNewColumn
        WHEN ValueInColumnName THEN ValueForTheNewColumn
        ELSE ValueForTheNewColumn
END as ValueForTheNewColumn
```

Example--Using IN

```
select shipperid
        , phone
        , shippername
        , CASE
         WHEN shippername IN ('mr shipper', 'mr delivery', 'deliveroo')
                        THEN 'Uber Clients'
         ELSE 'independent shippers'
        END as 'Alternative ShipperName '
from shippers
```

Examples:

```
Select ProductName
          , Price
         , CASE
             WHEN Price =  0 THEN 'Item not for resale'
             WHEN Price < 10 THEN 'Under $10'
             WHEN Price >= 10 and Price < 20 THEN 'Under $20'
             WHEN Price >= 20 and Price < 100 THEN 'Under $100'
             ELSE 'Over $100'
   END as Classification
From Products
```

Examples:

```
select shipperid
        , phone
        , shippername
        , CASE
         WHEN shippername = 'deliveroo' THEN 'Owner Mr Smith'
         WHEN shippername = 'mr delivery' THEN ' Owner Mr Jones'
         WHEN shippername = 'mr shipper' THEN 'Owner Mr Bloggs'
         ELSE 'Owned by Uber'
        END as Alternative_ShipperName
from shippers
```

Examples:

```
select shippername
        , CASE shippername
                WHEN 'deliveroo' THEN 'Owner Mr Smith'
                WHEN 'mr delivery' THEN ' Owner Mr Jones'
                WHEN 'mr shipper' THEN 'Owner Mr Bloggs'
                ELSE 'Owned by Uber'   END as Alternative_ShipperName
        from shipper
```

Exercise

- List the customername , city and country , and Europe , UK or ‘Rest of the World’

Example – Using the case-expression in the where clause

```
select
    customername, contactname,  country, city, orders.OrderID, orderdate
  from
    customers join orders on customers.CustomerID= orders.customerID
  where
                (case
                        when orderdate < '1996-12-01' then  CustomerName
                        ELSE contactname
                        end ) like 'm%'
        order by CustomerName, contactname
```

Union query

```
SELECT City FROM Customers
UNION ALL
SELECT City FROM Suppliers
ORDER BY City
```

Or

```
SELECT City FROM Customers
UNION
SELECT City FROM Suppliers
ORDER BY City
```

Union ALL – All records will show – Union- unique records only like when using DISTINCT

- Both queries must have the same number of columns of related information of the same datatype

Union Exercise

- Show all cities involved in either customers or suppliers
- Select the Customer Name as Addressee or the SupplierName as Addressee, address, city , postalcode, country from the Customers and Suppliers table using a Union Query
- Create a union query that will list all company names from shippers, suppliers and customers tables

Union Example

```
select
        Null as 'Full Average'
        , CategoryId as 'Category id'
        , avg(price) as 'Avg Price per Category'
from Products
where CategoryID is not null
group by CategoryID

union

select
        avg(price) as 'full avg'
        , NULL as 'Category id'
        ,  Null as 'cat avg'
from products
where CategoryID is not null
```

Intersect

```
[SQL Statement 1]
INTERSECT
[SQL Statement 2];
```

--the INTERSECT command acts as an AND operator
(value is selected only if it appears in both statements).

--The syntax is as follows:

```
SELECT City FROM Customers
Intersect
SELECT City FROM Suppliers
ORDER BY City

(this will give cities that appear in both queries)
```

Intersect Exercise

- Which cities in the Germany has Suppliers and Customers
- Which cities in the Canada has Suppliers and Customers
- Which cities are not in the Canada or Germany for both customers and suppliers

```
MySQL
select categoryid from categories ;
intersect
select  categoryid from categories
where categoryid = 1 ;
```

Except

```
[SQL Statement 1]
EXCEPT
[SQL Statement 2];
```

Note:

- There must be same number of expressions/ columns in both SELECT statements.
- The corresponding expressions must have the same data type in the SELECT statements. For example: *expression1* must be the same data type in both the first and second SELECT statement.

Example 1 - Customers NOT on the Orders Table

```
SELECT CustomerID  FROM Customers
EXCEPT
SELECT CustomerID  FROM Orders
```

-Example 2 – Employees NOT on the Orders Table

```
SELECT EmployeeID  FROM Employees
EXCEPT
SELECT EmployeeID  FROM Orders
```

Except Exercise

1. Show the Products with NO orders
2. Show Shippers with no orders
3. Show Employees with No orders
4. Show Suppliers with NO Products
5. Find out which categories have no products

```
MYSQL :
select categoryid from categories ;
Minus
select  categoryid from categories
where categoryid = 1 ;
```

Cartesian Product / Cross Join

```
SYNTAX:
SELECT table1.column1, table2.column2...
FROM  table1, table2 [, table3 ]
```

SQL Cartesian Product Tips. The Cartesian product, also referred to as a **cross-join**, returns all the rows in all the tables listed in the query. Each row in the first table is paired with all the rows in the second table. This happens when there is no relationship defined between the two tables.

Eg

Table1

EmployeeId	Firstname
1	Paul
2	Peter

Table2

Memberships

MID	Membership description
1	TableTennis
2	BookClub
3	Running

Cartesian Product

Paul	TableTennis
Paul	Bookclub
Paul	Running
Peter	TableTennis
Peter	Bookclub
Peter	Running

Select Firstname, 'Membership Description' from Table1, Table2

Cartesian Product Example

```
SELECT categories.CategoryID, products.productName
FROM  Categories, Products

SELECT categories.CategoryID, shippers.shippername
FROM  Categories, shippers
```

Cartesian Product Example

Eg

Table 1

EmplyeeId	Firstname
i.	Paul
ii.	Peter

Table 2

Memberships

MID	Membership description
1	Tabletennis
2	BookClub
3	Running

Cartesian Product

Paul	TableTennis
Paul	Bookclub
Paul	Running
Peter	TableTennis
Peter	Bookclub
Peter	Running

Exercise

Part 3

Formatting Numbers

Rounding Number

Date Functions

1. GetDate()
2. Datepart()
3. Datediff()
4. Dateadd()

Date queries

Convert

Numeric Functions

- RAND
- SQRT

String Functions

- Upper and Lowercase
- Concat
- Substring
- Replace
- Length
- Trim
- Ltrim
- Rtrim
- Right

Formatting Numbers

Same for SQL Server and MySQL

Syntax:

SELECT FORMAT(column_name, format) FROM table_name;

:

The has represents a Suppressed leading zero , i.e. if in this position there is a number, show the number, however if the number is 0 then don’t show it.
E.g. if my number is 1:
#0 will show 1 and 00 will show 01
##0 will show 1 and 000 will show 001.
#,##0 will show 1 and 0,000 will show 0,001

Examples: Format Numbers

```
SELECT Format(1222.4, '#,##0.00')           -- Returns "1,222.40".
SELECT Format(345.9, '#0.00')               -- Returns "345.90".
SELECT Format(15, '0.00%')                  -- Returns "1500.00%".
select Format(Price, '€#,##0.0000')          Returns currency
as ProductPrice
from Products order by price desc
```

Example:

```
select Format(Price, '€#,##0.0000')  as ProductPrice from Products order by price desc
```

Exercise

1. Select productName and Price from Products, formatting the price column
2. What is the average price per supplier, formatting the price ?
3. What is the average price per Category, formatting the price ?
4. What is the total order amount per order for customers in France, formatting the price ?

Round

Same for SQL Server and MySQL

Syntax:

```
SELECT ROUND(column_name,decimals) FROM table_name;
```

e.g.

```
SELECT ProductId, ROUND(Price,0) AS RoundedPrice
FROM Products;
```

Try the Examples:

1. Select round (price*1.333, 2), price*1.333, price from products
2. Select round (price*1.333, 1), price*1.333, price from products
3. Select round (price*1.333, 0), price*1.333, price from products
4. Select round (price*1.333, -1), price*1.333, price from products

FORMAT function to format dates

SQL Server

Symbol	Meaning	Example
D	Day in one digit	1 or 12
dd	Day in 2 digits	01 0r 12
ddd	Day name in 3 letters	Mon
dddd	Day name in full	Monday
M	Month in one digit	1 or 12
MM	Month in 2 digits	01 or 12
MMM	Month name in 3 letters	Jan
MMMM	Month name in full	January
yy	Year in 2 digits	96
yyyy	Year in 4 digits	1996
tt	Am/pm	Am/pm
zzz	Difference from GMT	+5
H	Hours in one digit	
hh	Hours in 2 digits	
m	Minutes in one digit	
mm	Minutes in 2 digits	
s	Seconds in one digit	
ss	Seconds in 2 digits	

Examples

Example	Result
SELECT Format(Orderdate, 'dd/MM/yy') from Orders	Returns "1/12/91"
SELECT Format(OrderDate, 'dddd, MMM d yyyy') from Orders	Returns "Thursday, Dec 22 2011"
SELECT Format(getdate(), 'M/d/yyyy H:mm tt zzz')	12/22/2011 1:15 AM -05:00
SELECT Format(getdate(), 'M/d/yyyy H:mm tt')	Returns 12/22/2011 1:14 AM
SELECT Format(OrderDate , 'h:m:s') from Orders	Returns "1:10:47".
SELECT Format(OrderDate, 'hh:mm:ss tt') from Orders	Returns "01:10:47 AM".

FORMAT function to format dates

mySQL

Syntax

DATE_FORMAT(date,format)

Where date is a valid date and format specifies the output format for the date/time. The formats that can be used are:

Format	Description
%a	Abbreviated weekday name (Sun-Sat)
%b	Abbreviated month name (Jan-Dec)
%c	Month, numeric (0-12)
%D	Day of month with English suffix (0th, 1st, 2nd, 3rd, �)
%d	Day of month, numeric (00-31)
%e	Day of month, numeric (0-31)
%f	Microseconds (000000-999999)
%H	Hour (00-23)
%h	Hour (01-12)
%I	Hour (01-12)
%i	Minutes, numeric (00-59)
%j	Day of year (001-366)
%k	Hour (0-23)
%l	Hour (1-12)
%M	Month name (January-December)
%m	Month, numeric (00-12)
%p	AM or PM
%r	Time, 12-hour (hh:mm:ss followed by AM or PM)
%S	Seconds (00-59)
%s	Seconds (00-59)
%T	Time, 24-hour (hh:mm:ss)
%U	Week (00-53) where Sunday is the first day of week
%u	Week (00-53) where Monday is the first day of week
%V	Week (01-53) where Sunday is the first day of week, used with %X
%v	Week (01-53) where Monday is the first day of week, used with %x
%W	Weekday name (Sunday-Saturday)
%w	Day of the week (0=Sunday, 6=Saturday)
%X	Year for the week where Sunday is the first day of week, four digits, used with %V
%x	Year for the week where Monday is the first day of week, four digits, used with %v
%Y	Year, numeric, four digits
%y	Year, numeric, two digits

Example

The following script uses the DATE_FORMAT() function to display different formats. \
We will use the NOW() function to get the current date/time:

```
DATE_FORMAT(NOW(),'%b %d %Y %h:%i %p')

DATE_FORMAT(NOW(),'%m-%d-%Y')

DATE_FORMAT(NOW(),'%d %b %y')

DATE_FORMAT(NOW(),'%d %b %Y %T:%f')
```

CONVERT Function to format dates

(MS SQL Server only)

```
SELECT CONVERT(nvarchar, orderdate, 101) from orders
Result: '05/02/2014'
```

Value (without century)	**Value (with century)**	**Explanation**
0	100	mon dd yyyy hh:miAM/PM (Default)
1	101	mm/dd/yyyy (US standard)
2	102	yy.mm.dd (ANSI standard)
3	103	dd/mm/yy (British/French standard)
4	104	dd.mm.yy (German standard)
5	105	dd-mm-yy (Italian standard)
6	106	dd mon yy
7	107	Mon dd, yy
8	108	hh:mi:ss
9	109	mon dd yyyy hh:mi:ss:mmmAM/PM
10	110	mm-dd-yy (USA standard)
11	111	yy/mm/dd (Japan standard)
12	112	yymmdd (ISO standard)

Exercise: Use the above formatting table

1. Using Us Standards to format the orderdate on the orders table
2. Using ISO Standard to format the orderdate on the orders table
3. Using British / French to format the orderdate on the orders table
4. Using German Standard to format the orderdate on the orders table

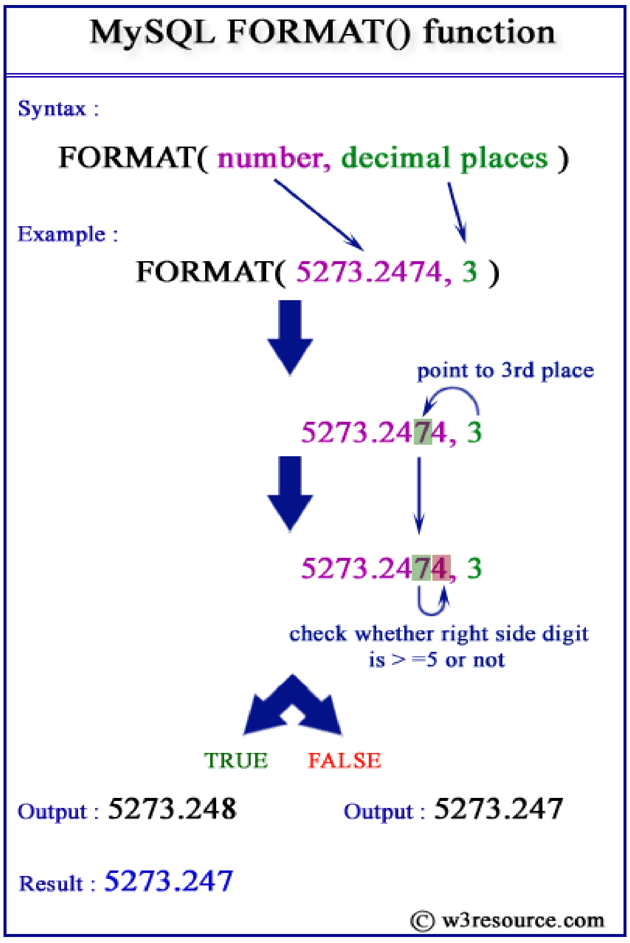
Date Queries

Exercises: : Refer to date as ‘yyyy-mm-dd’ and Format All the DATES

1. select * from orders where OrderDate between '1996-07-01' and '1996-07-31'
2. select * from orders where OrderDate > '1996-10-01'
3. select * from orders where OrderDate = '1996-07-08'
4. select * from orders where OrderDate < '1996-10-01'

mySQL

```
SELECT book_name,FORMAT(book_price,4)
FROM book_mast  WHERE book_price>150
```



Date Functions

Getdate –SQL Server

GETDATE() - Returns the current date and time

Example: Select getdate();

Examples:

```
SELECT
    OrderDate
    , FORMAT (  getdate()    ,  'YYYY-MM-DD'  ) AS   TodaysDate
FROM Orders;
```

Now()–mySQL

Select now();

Examples:

```
    SELECT OrderDate, now() AS TodaysDate
    FROM Orders;
```

DATEPART and DATENAME , SQL Server

SQL Sqerver

A date has parts: **day, month, year, dw (**DAYofWEEK)
DATEPART returns the requested part of a date as an **integer**
Syntax:

```
Datepart ( part, date)
```

Examples:

```
DATEPART (year, getdate()); - shows the year of the system date
DATEPART (month, getdate()); - shows the month of the system date
DATEPART (day, getdate()); - shows the day of the system date

SELECT CustomerID, DATEPART (year, OrderDate) from Orders;
SELECT CustomerID, DATEPART (month, OrderDate) from Orders;
SELECT CustomerID, DATEPART (day, OrderDate) as Day from Orders;

SELECT OrderID, DATEPart(dw, OrderDate) as DAYofWEEK from Orders;
```

- Which orders were made out of working hours and from which countries were they made?
- What is the count of orders per year?
- What is the count of orders in 1997 only?
- What is the count of orders in 1997 , January, February and March using the DATEPART and AND in the WHERE clause? (Use later on stored procedure)
- Use expressions to show the year, count of orders, sum of quantity * price for all orders, ordered by year, then month
- Use expressions to show the year, month, count of orders, sum of quantity * price for all orders, ordered by year, then month (same as above but also with a month column)
    - Filter the above where month in (1,2,3)
    - Filter the above where month is BETWEEN 1 and 6

Exercise:

Show: `-- today : Thursday 9 September 2021`

```
-- today : Thursday 9 September 2021
select datename(dw, getdate()) as Day
        , day(getdate()) as Date
        ,  datename(month, getdate()) as Month
        , year(getdate()) as Yr
```

Datename **(part, date)**

```
SELECT OrderID, DATEName(month, OrderDate) as MonthName from Orders;
SELECT OrderID, DATENAME(dw, OrderDate) as DAYofWEEK from Orders;
```

Exercise:

- Filter the above for month 'january' only
- Filter the above where month in ('march','april','may')
- Filter the above where monthname starts with s
- Filter the above to show all orders made on a Monday

- Show how many orders were made on Monday, Tuesday, Wednesday, Thursday, Friday

DATEPART mySQL

MySQL

A date has 3 parts: day, month, year
DATEPART returns the requested part of a date as an integer
Syntax:

    Month(date)

    YEAR(date)
    Day(date)

Exercise:

Year(), Month(), Day()

Examples:

```
SELECT CustomerID, YEAR (OrderDate) from Orders;
SELECT CustomerID, MONTH (OrderDate) from Orders;
SELECT CustomerID, DAY (OrderDate) as Day from Orders;
```

- What is the count of orders per year?
- What is the count of orders in 1997 only?
- What is the count of orders in 1997 , January, February and March using the DATEPART and AND in the WHERE clause? (Use later on stored procedure)
- Use expressions to show the year, count of orders, sum of quantity * price for all orders, ordered by year, then month
- Use expressions to show the year, month, count of orders, sum of quantity * price for all orders, ordered by year, then month (same as above but also with a month column)
- Filter the above for month 1 only
- Filter the above where month in (1,2,3)
- Filter the above where month is BETWEEN 1 and 6

DATEDIFF - SQL Server

DATEDIFF returns the number of days, months or years between 2 dates as an integer
Syntax
DATEDIFF (datepart, date1, date2)

Example:

```
select CustomerID , datediff(day, OrdersDate, '2015-10-31')  from Orders
select CustomerID, datediff(month, OrdersDate, '2015-10-31')  from Orders
select CustomerID, datediff(year, OrdersDate, '2015-10-31')  from Orders
```

Exercise

- How many days do we have until it is Christmas Day.
- How many months and how many days do we have until it is Christmas Day.
- For every employee, list their Firstname, Birthdate and age.
- For every customer , how many days are between the Orderdate and today
- For CustomerID = 60 , how many YEARS has elapsed since his last order and today?
- For CustomerID = 60 , how many YEARS has elapsed since his last order and the last day 0f 1997?
- For CustomerID = 60 , how many MONTHS has elapsed since his last order and the last day 0f 1997?
- For CustomerID = 60 , how many DAYS has elapsed since his last order and the last day 0f 1997?
- For every customer who ordered in 1997 and in 1996 , how many days are between the Orderdate and the last day of 1996, how many days are between the Orderdate and the last day of 1997, how many MONTHS are between the Orderdate and the last day of 1996, how many MONTHS are between the Orderdate and the last day of 1997

MySQL: DATEDIFF (date1, date2)

DATEADD SQL Server

SQL Server

Syntax

```
DATEADD ( units, how many , date)
```

Example:

```
SELECT OrderID , DATEADD (day, 10, Orderdate) , OrderDate from Orders
```

Exercise:

For every employee, firstname, lastname

show their retirement date assuming they would retire at 67

and their current age,

and how many years they still have to work.

For every customer who made orders in 1996,

- Show the Month of the orderdate, the Orderdate, CustomerName, add 30 days to the orderdate:
- order by datepart MONTH

For every customer

- Show datepart YEAR of the orderdate, datepart Month of the orderdate, Orderdate. CustomerName, add 1 year to the orderdate,
- order by datepart YEAR, datepart MONTH, orderdate

For every customer,:

add 12 MONTHS to the orders,

and show only the customers who have to pay in December

TIMESTAMPADD - MySQL

MySQL

```
SELECT DATE_ADD("2017-06-15", INTERVAL 10 DAY);
```

Syntax

Timestampadd (units, how many , date)

Units in Mysql:

MINUTE	MINUTES
HOUR	HOURS
DAY	DAYS
WEEK	WEEKS
MONTH	MONTHS
QUARTER	QUARTERS
YEAR	YEARS

Example:

SELECT OrderID , Timestampadd (day, 10, Orderdate) , OrderDate from Orders

Exercise:

- For every customer who made orders in 1996,
- Show datepart Month of the orderdate, Orderdate. CustomerName, add 30 days to the orderdate,
- order by datepart MONTH
- For every customer
- Show datepart YEAR of the orderdate, datepart Month of the orderdate, Orderdate. CustomerName, add 1 year to the orderdate,
- order by datepart YEAR, datepart MONTH, orderdate
- For every customer, add 12 MONTHS to the orders, order by date part MONTH

Exercise: Date and Number Formatting queries

- Show the SupplierName , ProductName , OrderDetailsID, Price, Quantity

  for orders in December of 1996,

  order by SupplierName (Display the Price with Pounds in currency format)

- Show the OrderID, sum of Price, sum of Quantity

  for orders from September 1996 to March 1997

  ordered by OrderID. (Display the Price with Pounds in currency format)

- Show the OrderID, EmployeeID, EmployeeFirstName and orderdate for orders in July – Dec 1996. Display the date in US format

Exercise: Dates with joins and aggregates

- Show the count of orders per customer, per year. Format numbers and dates
- Show the count of orders per employee, per year. Format numbers and dates Show the number of orders per shipper
- Show the sum of (quantity * price) of orders per product, per year. Format numbers and dates
- Show the sum of quantity of orders per supplier, per year. Format numbers and dates
- Show the sum of quantity of orders per category, per year. Format numbers and dates

  e.g. :

```
select CustomerID, datepart(year, orderdate) as years , count (orderid)
 from orders
 group by CustomerID, datepart(year, orderdate)
 order by CustomerID, datepart(year, orderdate)
```

MySQL Date format

```
SELECT DATE_FORMAT('1997-10-04 22:23:00', '%W %M %Y');

SELECT DATE_FORMAT('1997-10-04 22:23:00', '%H %k %I %r %T %S %w');
```

MySQL Date format codes

S. No.	Specifier & Description
1	**%a** Abbreviated weekday name (Sun..Sat)
2	**%b** Abbreviated month name (Jan..Dec)
3	**%c** Month, numeric (0..12)
4	**%D** Day of the month with English suffix (0th, 1st, 2nd, 3rd, .)
5	**%d** Day of the month, numeric (00..31)
8	**%H** Hour (00..23)
9	**%h** Hour (01..12)
15	**%M** Month name (January..December)
16	**%m** Month, numeric (00..12)
17	**%p** AM or PM
18	**%r** Time, 12-hour (hh:mm:ss followed by AM or PM)
19	**%S** Seconds (00..59)
20	**%s** Seconds (00..59)
26	**%W** Weekday name (Sunday..Saturday)
27	**%w** Day of the week (0 = Sunday..6 = Saturday)
31	**%y** Year, numeric (two digits)

Numeric Functions

SQL SQRT – SQL Server and mySQL

SQL SQRT function is used to find out the square root of any number. You can Use SELECT statement to find out square root of any number as follows:

```
select SQRT(16);
```

SQL RAND – SQL Server and mySQL

SQL has a **RAND** function that can be invoked to produce random numbers between 0 and 1:

```
SELECT RAND( ), RAND( ), RAND( );
```

Convert Function – SQL Server

```
Syntax:
CONVERT( type [ (length) ], expression [ , style ]  )
```

Example:

- SELECT CONVERT(int, 14.85) *Result:* 14 *(result is truncated)*
- SELECT CONVERT(float, 14.85) *Result: 14.85 (result is not truncated)*
- SELECT CONVERT(float, '15.6') *Result: 15.6*
- SELECT CONVERT(nvarchar, 15.6) *Result: '15.6'*
- SELECT CONVERT(nvarchar(4), 15.6) *Result: '15.6'*
- SELECT CONVERT(datetime, '2014-05-02')

  Result: '2014-05-02 00:00:00.000'

Exercise – Convert:

- Convert the Price on Products to float
- Convert the Quantity on Orderdetails to float
- Convert the Price on Products to Real
- Convert the Quantity on Orderdetails to Real
- Convert the Price on Products to Decimal
- Convert the Quantity on Orderdetails to Decimal
- Convert the Price on Products to Numeric

String Functions

SQL Server		MySQL

UPPER(column)	SELECT upper(CustomerName)
	AS Customer FROM Customers;
	Return the Customer Name column in Upper Case
		Same

LOWER(column)	SELECT LOWER(column_name)
	FROM table_name;
	Return the Customer Name column in Lower Case
		Same

LEN(column)	select len(Notes) from employees Example: select len('foobar') ***returns 6***	**LENgth()**
SQL CONCAT	Select concat(firstname, lastname) from employees	**Same**
<u>REPLACE</u> ()	Replaces occurrences of a specified string REPLACE(str, from_str, to_str) e.g. SELECT REPLACE('foobar', 'bar', 'pub'); ***returns foopub***	**Same**
LEFT(str,len)	Returns the left-most characters from the string str, or NULL if any argument is NULL. E.G.SELECT Left('foobar', 4); ***returns: foob***	**Same**
RIGHT(str,len)	Returns the right-most characters from the string str, or NULL if any argument is NULL. E.G. SELECT RIGHT('foobar', 4); ***returns: obar***	**Same**

SUBSTRING	Returns a substring of a string (part of a string)
	SUBSTRING(string/column to inspect, from position, how many)
	SELECT SUBSTRING('Foobar', 1, 1); --***gives f***
	SELECT SUBSTRING('Foobar', 1, 2);-- ***gives fo***
	SELECT SUBSTRING('Foobar', 1, 3);-- ***gives foo***
	SELECT SUBSTRING('Foobar', 1, 4);--***gives foob***
		Same

LTRIM(str)	Returns the string str with leading space characters removed. SELECT LTRIM(' barbar'); **RETURNS: barbar**	**Same**

RTRIM(str)	Returns the string str with trailing space characters removed. SELECT RTRIM('barbar '); **RETURNS: barbar**	**Same**
CHARINDEX •	CHARINDEX(what to find, where to find it) E.G. • Select charindex('dumb','Dumb and Dumber') – gives 1 OR: • CHARINDEX(what to find, where to find it, from position) E.G. • Select charindex('dumb','Dumb and Dumber',4) – gives 10 • select description , CHARINDEX('pasta', description, 20) from categories • select description , CHARINDEX('cheese', description) from categories	**Position or Locate** Or Instr
Charindex with substring	select description , substring (description, -- *which field to look in* CHARINDEX('cheese', description) , --*from position* LEN('CHEESE') --*how many characters to return*) from categories select description , CHARINDEX('cheese', description) as SPosition , LEN('CHEESE') as MyLength , substring (description, -- which field to look in CHARINDEX('cheese', description) , --from position LEN('CHEESE') --how many characters to return) as Substring_StartAt_SPosition_LengthOf_MyLength from categories	

Exercise with string functions

- From the Employees table, show **EmployeeID** and **Firstname**, **Lastname** concatenated ,
- From the Employees table, Show **Firstname** in LowerCase , **Lastname** in Uppercase, and the **EmployeeID**
- Show **Firstname**, **Firstname** and the length of the **Notes** field on Employees
- Show **Categoryname** in uppercase , the **Description** field and the length of the **Description** field from Categories
- Show the first 3 characters of a **CategoryName** on categories
- On the Suppliers Table:
    - The first character in the **SupplierName**
    - The **Suppliername**,
    - Show the first 4 characters of a **PostalCode**
    - The length of the **Supppliername**
- Show the **SupplierName**, position of the first space in the **Contactname**, separate the **Firstname** and **Surname** of the **Contactname**, ordered by **Surname**

```
select CompanyName
        , contactname
        , charindex(' ',contactname, 1)
        , left ( contactname, charindex(' ',contactname, 1))
        , right ( contactname , len(contactname) - charindex(' ',contactname,
1)+1)
        , substring ( contactname, 1, charindex(' ',contactname, 1))
        , substring ( contactname, charindex(' ',contactname,
1),len(contactname) - charindex(' ',contactname, 1) +1)
from suppliers
```

- In the categories table, do you have the word “sweet” in a description of the **Description** field and at which position was it found?

```
select
        categoryname
        , description
        , iif ( charindex('sweet',description) >0
                  , 'Found at position ' +
convert(varchar,charindex('sweet',description))
                  , 'Not Found')    as Sweet
from categories
```

- In the categories table, do you have the word “meat” in the **Description** field and at which position was it found?
- In the categories table, do you have the word “meat” in the **CategoryName** field and at which position was it found?

- Show the **CategoryID**, **CategoryName** and use an expression that will create a new column that will a Show 'meat' if it appears in the **Categoryname**, otherwise ' '.

```
select
  CategoryName
        ,   case
                          when CHARINDEX('meat', description)> 0
                          then SUBSTRING(Description, CHARINDEX('meat',
description), 4)
                          else ''
        end as 'Meat?'
        , Description
from categories
```

- Show 'cheese' of a category description as a substring
- Show the first letter of the supplier name and how many suppliernames start with that letter of the alphabet

Sub Queries

General
In , Any, Some, All
Expressiona
Exists, Not exists
CTE
Correlated:
In the Where. Select , From and Having Clauses

Sub-queries:

Sub-Query in the Where Clause

Sub-Query as an Expression in the Select Clause

Sub-Query in the Having Clause

Sub-Query in the From Clause

Exists / Not Exists

Correlated Sub-Query

CTE Tables

Sub Query in the Where Clause

What is a subquery : (a query nested inside another query)

Subqueries

SYNTAX:
SELECT "column_name1"
FROM "table_name1"
WHERE "column_name2" [Comparison Operator] (e.g. < > >=)

```
        (SELECT "column_name3"
        FROM "table_name2"
        WHERE "condition");
```

Examples

```
SELECT categoryID  , productname
FROM   products
WHERE  categoryId in      (   4,5,6,7,8  )
```

```
SELECT categoryID  , productname
FROM   productS
        wHERE  categoryId    IN
                    (  SELECT categoryID
                FROM categories
                where categoryID  > 3   )
```

The same fieldname

One column only

EXAMPLES

```
SELECT categoryID  , categoryname
FROM   categories
WHERE  categoryId in
                (   SELECT        categoryID
                    FROM          products
                    where         productID = 2   )
```

```
SELECT supplierID  , supplierName
FROM   suppliers
WHERE  supplierID  IN
            (         SELECT        supplierID
                      FROM          products          where   productID = 6    )
```

Subqueries with IN, ANY or SOME

operand IN (***subquery***)
operand comparison_operator ANY (***subquery***)
operand comparison_operator SOME (***subquery***)

The ANY keyword, which must follow a comparison operator:
means "return TRUE if :
the comparison is TRUE for ANY of the values in the column that the subquery returns."
For example:

```
SELECT categoryID, categoryname, description
FROM categories
 WHERE categoryID  IN (SELECT categoryID FROM products);

SELECT categoryID, categoryname, description
FROM categories
WHERE categoryID  =  ANY (SELECT categoryID FROM products);
```

IN and Any: When used with a subquery, the word IN is an alias for = ANY. .

NOT IN is not an alias for <> ANY, but for <> ALL.

SOME and ANY: The word SOME is an alias for ANY. Thus, these two statements are the same:

```
SELECT categoryID, categoryname, description
FROM categories
WHERE categoryID  <> ANY  (SELECT categoryID  FROM Products);

SELECT categoryID, categoryname, description
FROM categories
WHERE categoryID <> SOME (SELECT categoryID FROM products);
```

```
select employeeid
 from Employees
 where  EmployeeID = any  (
                              select 1
                              union select 2
                              union select 3
                              )
```

Try to do the queries without looking at the result:

1. Select the SupplierID and SupplierName for suppliers who supply products 2 to 6

```
SELECT supplierID  , supplierName
FROM   suppliers
WHERE  supplierID  IN
              (    SELECT    supplierID
                   FROM      products
                   where     productID between 2 and 6  )
```

2. Select the productid's and productnames for ordered's > 10400

```
Select ProductId, ProductName
From products
Where productID in
              (    Select     ProductID
                   From       Orderdetails
                   Where      OrderID > 10400   )
```

3. Use A SUBQUERY

```
Select ProductId, quantity
From OrderDetails
Where OrderID in
              (    where the oderdate is in the last 6 months 0f 1996   )
```

4. Use A SUBQUERY

```
Select    ProductId, ProductName , price
From Products
where price >
            ( select avg(price)
             from products            )
```

5. Use A SUBQUERY

```
Select ProductId, quantity
From Orderdetails
Where OrderID in
            (            for the employee with first name ANNE   )
```

Subqueries Used in Place of an Expression

Use Subquery in the select clause

```
-- show the product name , price ,
-- average price as average using an inline-subquery
-- price - the average price using an inline-subquery
```

Sub query Solution

```
SELECT
            CategoryID,
            ProductName,
            Price,
            (  SELECT  AVG(Price)  FROM Products  )  AS Average,
            Price - (SELECT AVG(Price)   FROM Products  ) as Difference
FROM Products
WHERE categoryID = 1
order by CategoryID;
```

CTE

CTE stands for Common Table expressions. It was introduced with SQL Server 2005. It is a temporary result set and typically it may be a result of complex sub-query. Unlike temporary table its life is limited to the current query. It is defined by using WITH statement. CTE improves readability and ease in maintenance of complex queries and sub-queries. Always begin CTE with semicolon.

A sub query without CTE is given below : (SQL Server and MySQL)

```
SELECT * FROM (
 SELECT Firstname, LastName, Notes From Employees
) Temp
WHERE LastName Like ‘m%’
ORDER BY Temp.FirstNAME
```

By using CTE above query can be re-written as follows : (Not MySQL)

```
With CTE1(Name, SurName, Notes)--Column names for CTE, which are optional
AS
(
SELECT Firstname, LastName, Notes From Employees
)
SELECT * FROM CTE1 --Using CTE
WHERE CTE1.SurName Like ‘m%’
ORDER BY CTE1.NAME
```

When to use CTE

1. This is used to store result of a complex sub query for further use.
2. This is also used to create a recursive query.

CTE Example

```
with CTE1 ( FirstName, LastName, Age, notes )
 as
 (
                select FirstName, LastName, datediff( (year , getdate(), birthdate)  , notes
                from Employees
                where datediff( (year , getdate(), birthdate)  >  21
 )
 Select *
 from CTE1
 where Age  between 30 and 65
```

CTE Example

```
 with CTE1 ( ProductId, Productname,  Price)
 as
 (
                              select ProductId, Productname, UnitPrice
                              from  Products
                              where UnitPrice  >  50

 )
 select *
 from CTE1  order by Price
```

Exercise:

Do the following:
Select Firstname, Lastname, Birthdate using a CTE Table and list all the columns from the CTE table where the last name starts with k

Exercise:

Do the following:
Select Productname, Suppliername, Country using a CTE Table and list all the columns from the CTE table where the country starts with u and the price is between 10 and 500

Nested CTE

```
with newtable1 (Customerid, Name, Country)
as ( select c.customerid, customername, country
        from customers c
)
,  newtable2 (orderdid, customerid)
    as (select orderid, customerid from orders
)
select *
    from newtable2 n2
    join newtable1 n1 on n2.customerid =
n1.Customerid
    where n1.country like 'Mexico%'
```

EXERCISE

```
-- extract records:
 -- select firstname, productname, categoryname, quantity
 -- from orders table , orderdetails and product, employees
 -- for the orders of the employee STEVEN

 -- select fistname, categoryname, revenue for seafood
```

```
with CTE ( Firstname, Productname, Categoryname, Quantity, Price)
as (
        select firstname, productname, categoryname, quantity, price
        from employees e
                join orders o       on e.EmployeeID = o.employeeID
                join orderdetails od on o.OrderID    = od.OrderID
                join products p                  on od.ProductID = p.ProductID
                join categories c       on p.CategoryID = c.CategoryID
        where firstname like 'Steven%'
)
select firstname, categoryname, sum(CTE.price * CTE.quantity) as Revenue
from CTE
        where categoryname like 'Seafood%'
group by  firstname, categoryname
```

CTE Examples

Joining with CTE tables

```
 With CTE (CustomerId, CustomerName, Country , City )
 as
 (
             select  CustomerId, CompanyName, Country , City
             from Customers
             where county like ‘germany’
)
select CTE.CustomerId, CTE.CustomerName, CTE.Country, Orderdate, OrderID
from CTE
join orders on cte.CustomerId=orders.CustomerID
```

CTE Examples

Create a table with one value

```
with cte ( avgPrice )
as  (
          select avg(Price) from products
)
select * from cte
```

with crossJoin (Cartesian Product)

Join the CTE table with any table

```
with cte ( avgPrice )
as(
          select avg(Price) from products
)
select Productname, Price,  cte.avgPrice

from cte, products    - this is the cross join or Cartesian product
```

Subqueries with EXISTS

When a subquery is introduced with the keyword EXISTS, the subquery functions as an existence test. The WHERE clause of the outer query tests whether the rows that are returned by the subquery exist. The subquery does not actually produce any data; it returns a value of TRUE or FALSE.
A subquery introduced with EXISTS has the following syntax:

WHERE [NOT] EXISTS (subquery)

```
SELECT ProductName
FROM Products
WHERE   EXISTS
   (SELECT *
    FROM Categories
    WHERE categoryID > 20)
```

```
Select orders.OrderID, Orderdate
from  orders
join orderdetails  on orders.orderID = orderdetails.orderID
WHERE  EXISTS
             (SELECT * FROM [dbo].[Products]  WHERE price > 1800)
```

- The keyword EXISTS is not preceded by a column name, constant, or other expression.
- The select list of a subquery introduced by EXISTS almost always consists of an asterisk (*). There is no reason to list column names because you are just testing whether rows that meet the conditions specified in the subquery exist.

Subqueries with NOT EXISTS

NOT EXISTS works like EXISTS, except the WHERE clause in which it is used is satisfied if no rows are returned by the subquery.

```
SELECT ProductName
FROM Products
WHERE NOT  EXISTS
   (SELECT *
    FROM Categories
    WHERE categoryID > 20)


SELECT ProductName
FROM Products
WHERE   EXISTS
   (SELECT *
    FROM Categories
    WHERE categoryID > 20)
```

Correlated Subquery

Correlated Subquery

In a **SQL** database query, a **correlated subquery** (also known as a synchronized**subquery**) is a **subquery** (a query nested inside another query) that uses values from the outer query. ... Because the **subquery** is **correlated** with a column of the outer query, it is be re-executed for each row of the result.
SQL Correlated Subquery

Summary: in this tutorial, you will learn about the **SQL correlated subquery**, which is a subquery that depends on the outer query.

This tutorial requires a good knowledge of subquery. If you don’t know anything about the subquery, check it out the subquery tutorial before moving forward with this tutorial.

Introduction to SQL correlated subquery (WHERE clause)

A correlated subquery is a subquery that depends on the outer query.
It means that the WHERE clause of the correlated subquery uses the data of the outer query.

The main difference between a correlated subquery and a non-correlated subquery is that you cannot execute a correlated subquery alone like a non-correlated subquery. In addition, a correlated subquery executes once for each selected row from the outer query.

A correlated subquery is also known as repeating subquery or synchronized subquery.

SQL correlated subquery in WHERE clause example

You can also use the correlated subquery in a WHERE clause.

For example, the following example uses a correlated subquery in WHERE clause: to find customers that have total sales more than a certain amount:

```
select customerid, customername, city
from customers
where  2000 <
        (
                select sum ( price * quantity )
                from orders
                join orderdetails on orders.OrderID = orderdetails.OrderID
                join products on orderdetails.ProductID = products.ProductID
                where orders.CustomerID = customers.CustomerID
                group by CustomerID
        )
```

For each customer, the correlated subquery calculates the total sales. The WHERE clause checks if the total sales, which is returned by the correlated subquery, is greater than 100K.

SQL correlated subquery in the SELECT clause example

The following query selects top five customers by sales:

```
select customerid
        , customername
        , city
        , (
                select sum ( price * quantity )
                from orders
                join orderdetails on orders.OrderID = orderdetails.OrderID
                join products on orderdetails.ProductID = products.ProductID
                where orders.CustomerID = customers.CustomerID
                group by CustomerID
                having 2000 < sum ( price * quantity )
        ) as customerTotals
from customers
order by customerTotals desc
```

The correlated subquery calculates total sales for each selected customer from
the customers table. The selected customerid from the outer query is passed to the correlated subquery for getting the corresponding sales data.

Subqueries in the FROM clause

Example

```
select AveragePrice
from
            (
            select  avg(price) as AveragePrice
            from products
            where price is not null
            ) as TempTable
```

With cross join

```
 select AveragePrice, Price, Productname
 from
           (
                        select  avg(price) as AveragePrice
                        from products
           ) as TempTable,  products
                                     where price is not null
```

same as CTE example

```
  with cte ( avgPrice )
  as  (
            select avg(Price) from products
            where price is not null
  )
  select Productname , avgPrice,  Price
  from cte, products
```

Example

```
     select *
     from   (  select  orders.OrderID
                    , datepart(year,orders.orderdate)              as oYear
                    , datepart(month, orderdate)        as oMonth
                   from Orders
                   join orderdetails on orders.OrderID = Orderdetails.OrderID
            )
      as myTable
                   where     oYear > 1996
                   and             oMonth > 6
     order by oYear
```

Same as the cte

```
with cte  ( orderid, oYear, oMonth )
as ( select  orders.OrderID
    , datepart(year,orders.orderdate)
    , datepart(month, orderdate)
    from Orders
    join orderdetails on orders.OrderID = Orderdetails.OrderID
)
 select *
 from cte  where   oYear = 1996  and  oMonth >6 order by oYear
```

SQL correlated subquery in HAVING clause example

```
select t1.CategoryID, CategoryName, avg (t1.price)

from products t1
join categories on t1.categoryID = categories.categoryid

group by t1.CategoryID, CategoryName

having max(t1.price) >  (

                                          select 2 * avg(t2.price)
                                          from products t2
                                          where t1.CategoryID = t2.CategoryID

                                          )
```

```
select cOuter.CustomerID,  Customername, sum(p.price * od.quantity)
from orders o
            join orderdetails od on o.OrderID = od.OrderID
            join products p on od.ProductID = p.ProductID
            join Customers cOuter  on o.CustomerID = cOuter.CustomerId
group by cOuter.CustomerID, Customername
having 2000 <  (
      select sum(p.price * od.quantity)
      from orders o
                  join orderdetails od on o.OrderID = od.OrderID
                  join products p on od.ProductID = p.ProductID
      where cOuter.customerid = o.CustomerID
      group by o.CustomerID
```

In the above query:

- The subquery calculate the average unit price in each category and multiply it with 2.
- The outer query selects category of the product whose unit price is greater than the double average unit price returned by the correlated subquery.

CTE example

```
with cte1 ( customername , orderdate)
as
(
        select customername , max (orderdate)
        from customers
        join orders on Customers.CustomerID = orders.CustomerID
        where datepart (year, orderdate) = 1997
        group by customername

)
select cte1.customername, format(cte1.orderdate, 'dd MMM yy') as orderdate
        , datediff ( day, cte1.orderdate , '1997-12-31' ) as 'Bonus earned'
from cte1
order by customername
```

- Use intersect as subquery in the where

```
SELECT COUNTRY , CustomerName, OrderID, OrderDate
FROM Customers
JOIN ORDERS ON CustomerS.CustomerID= ORDers.CustomerID
where country in (
                    select country from customers
                    intersect
                    select country from Suppliers
)
```

- Find customers who buy from suppliers in the same country as themselves

```
SELECT COUNTRY , CustomerName, OrderID, OrderDate
FROM Customers c
JOIN ORDERS o ON c.CustomerID= o.CustomerID
where exists (
                    select country
                    from Suppliers
                    where c.Country = Country
)
```

Views

Create a view

```
Syntax:
CREATE VIEW ViewName AS
SELECT *
FROM table
WHERE condition
```

- Views cannot ORDER By and cannot * for columns

To alter a view:

```
Syntax:

Alter VIEW ViewName AS
SELECT *
FROM table
WHERE condition
```

To DELETE a View

```
Syntax:

DROP VIEW view_name
-- sales per customer per year, per product, per employee
```

Exercise:

1. Create a view that will have columns:

   Show the Orderdate, OrderId , CustomerName , FirstName, ShipperName , OrderDetailsID, Productname , Quantity , Price, CategoryName, **SupplierName** ordered by Orderdate, OrderID
   *Tables: Orders , Customers , Employees, Shippers , OrderDetails , Products, Categories, **Suppliers***

2. Alter the view to Include the

   Notes, DateOfBirth from employees, and
   the supplier country , supplier contactname and supplier city
   and the customer contact, customer country and customer city

3. Queries using the view

   1)Which employees sell products from suppliers in the UK to customers in the UK
   2)Which 3 employees sell the most beverages
   3)List the country , category , employee and customer name

Export

Export the views above to Excel

Stored Procedures

Stored Procedures

1. Create Procedure
2. Execute the procedure
3. Alter procedure
4. Drop Procedure

Stored Procedures (SQL Server)

Create Procedure without parameter

```
Create  PROCEDURE EmpName
AS
BEGIN
            SELECT Firstname, Lastname , employeeID
            FROM employees
            WHERE employeeid=1
END
```

Create Procedure with an input parameter

```
Create  PROCEDURE EmpNameNext
(
   @eid INT                  --Input parameter ,  id  of the employee
)
AS
BEGIN
            SELECT Firstname, Lastname , employeeID
            FROM employees
            WHERE employeeid=@eid
END
```

Execute the procedure

Either :   `Execute EmpName 1`   - to display the employee name where the employeeid is 1

Or   **`Exec EmpName 3`**   -- to display the employee name where the employeeid is 2

`exec wildcard_name 'n%';`   --wildcard on nvarchars

Alter Procedure

```
Alter   PROCEDURE EmpName
(
   @eid INT                          --Input parameter ,  id  of the employee
)
AS
BEGIN
            SELECT employeeID , Concat(Firstname, Lastname )
            FROM employees
            WHERE employeeid=@eid
            Order by employeeID
END
```

For the stored procedures:

Question 1

Question 1a
Create a stored procedure:
Input parameters: EmployeeID
To do:
Display all EmployeeID, Firstname, Categoryname, Productname, OrderID, Orderdate, Quantity where the EmployeeID matches the input parameter
order by EmployeeID, Categoryname , Productname (orders processed by an employee)

Execute the procedure with value: 1
Execute the procedure with value: 2
Execute the procedure with value: 4

Question 1b - Alter the stored procedure:
Input parameters: Firstname
To do:
Display all EmployeeID, Firstname, Categoryname, Productname, OrderID, Orderdate, Quantity where the Firstname matches the input parameter
order by EmployeeID, Categoryname , Productname (orders processed by an employee)

Execute the procedure with value: 'michael'
Execute the procedure with value: 'janet'
Execute the procedure with value: 'robert'

Question 1c - Alter the stored procedure:
Input parameters: Firstname
To do:
Display all EmployeeID, Firstname, Categoryname , Productname, OrderID, Orderdate, Quantity, where the Firstname is LIKE the input parameter using a wildcard pattern
order by EmployeeID, Categoryname , Productname (orders processed by an employee)

Execute the procedure with value: 'm%'
Execute the procedure with value: 'j%'
Execute the procedure with value: 'r%'

Question 1d - Alter the stored procedure:
Input parameters: Firstname and Lastname
To do:
Display all EmployeeID, Firstname, Lastname, Categoryname , Productname, OrderID, Orderdate, Quantity where the Firstname and Lastname match the input parameters
order by EmployeeID, Categoryname , Productname (orders processed by an employee)

Execute the procedure with value: 'nancy, 'Davolio',
Execute the procedure with value: 'mike', 'fuller
Execute the procedure with value: 'robert','King'

Question 1e
Create a stored procedure:
Input parameters: Firstname and Lastname and Categoryname
To do:
Display all EmployeeID, Firstname, Lastname, Categoryname , Productname, OrderID, Orderdate, Quantity where the Firstname and Lastname and Categoryname match the input parameters order by EmployeeID, Categoryname , Productname (how does this employee support this category)

Execute the procedure with value: 'mike', 'Davolio', 'Beverages'
Execute the procedure with value: 'mike', 'Davolio', 'Seafood'
Execute the procedure with value: 'mike','Davolio', 'Condiments'
Execute the procedure with value: 'robert,'King', 'Condiments'

Solution

```
create procedure proc_emplCatProd (
      @var1 nvarchar(255), @var2 nvarchar(255), @var3 nvarchar(255)
)as
begin
      select
      e.EmployeeID, Firstname, Lastname
      , Categoryname, Productname
      , o.OrderID, Orderdate, Quantity
      from orderdetails od
      join products p            on od.ProductID = p.ProductID
      join categories c   on p.supplierid = c.categoryid
      join orders o       on od.orderid   = o.orderid
      join employees e    on o.employeeid = e.employeeid
      where Firstname=@var1 and Lastname= @var2 and Categoryname=@var3
      order by e.EmployeeID, Categoryname , Productname
end
Exec proc_emplCatProd 'nancy', 'Davolio', 'Beverages'
Exec proc_emplCatProd 'nancy', 'Davolio', 'Seafood'
Exec proc_emplCatProd 'nancy','Davolio', 'Condiments'
Exec proc_emplCatProd 'robert','King', 'Condiments'
```

Question 1f
Create a stored procedure:
Input parameters: Firstname and Categoryname
To do:
Display all EmployeeID, Firstname, Lastname, Categoryname , Productname, OrderID, Orderdate, Quantity
where the   Firstname is LIKE the firstinput parameter Lastname is LIKE the second input parameter and Categoryname LIKE the last input parameter
order by EmployeeID, Categoryname , Productname (how does this employee support this category)

Execute the procedure with value: 'm%', 'Beverages%'
Execute the procedure with value: 'r%', 'Sea%'
Execute the procedure with value: 's%', 'C%'

More Questions

Question 2
Create a stored procedure:
Input parameter: Country
To do: Display CustomerNAME, Address, City, County from **CUSTOMERS** where the country matching the input parameter

Execute the procedure with value: 'France'
Execute the procedure with value: 'Germany'
Execute the procedure with value: 'Finland'

Question 3
Create a stored procedure:
Input parameter: Country
To do: Display all Suppliername, Address, Country, City from **SUPPLIERS** where the country matching the input parameter

Execute the procedure with value: 'France'
Execute the procedure with value: 'Germany'
Execute the procedure with value: 'Finland'

Question 4
Create a stored procedure:
Input parameters: Categoryname
To do: Display the top 5 best sellers for a category
List the Productname and Categoryname

Execute the procedure with value: 'beverages'
Execute the procedure with value: 'seafood
Execute the procedure with value: 'Condiments'

Question 5
Create a stored procedure:
Input parameters: Supplier Country
To do: Display for the country in the suppliers table like the input parameter.
Display the Country , Suppliername, ProductName, the total quantity ordered , and the total amount (quantity * price)

Execute the procedure with value: 'mexico%'
Execute the procedure with value: 'italy%'
Execute the procedure with value: 'uk%' or 'united k%'

Question 6
Create a stored procedure:
Input parameters: Product ID
To do: Display the orderID, order date, customer name, quantity , price ordered by customer name, order by orderID where the Product ID matches the input parameter

Question 7
Create a stored procedure:
Input parameters: CustomerName
To do: Display the CustomerName, ProductName , CustomerCountry and price by CustomerCountry, ProductName for the CustomerName matching the input parameter

Question 8
Create a stored procedure:
Input parameters: ShipperName
To do: Display the ShipperName, Orderdate , CustomerCountry by ShipperName, CustomerCountry and Orderdate for the ShipperName matching the input parameter

Question 13
Create a stored procedure:
Input parameters: IncludeExtendedInformation int
On Fridays you do a full report and for this you give the procedure an integer value of 1. All other days you want an abbreviated report and then you give the procedure an integer value of 0.
To do: Display on Fridays: ProductID, ProductName , ShipperName, unit and price , otherwise simply productname, unit and price

Example -testing conditions

```
alter procedure newP1 (
@IncludeExtendedInformation int
)
as
begin
IF @IncludeExtendedInformation = 1
          begin
           SELECT productID,productname,suppliername,categoryname,unit,price
           FROM   Products join
            ....
          end
ELSE
          begin
           SELECT productname,unit,price
           FROM   products
          end
end
```

To execute:

```
exec newP1 1
```

Stored Procedure with OUTPUT parameter

Create

```
alter procedure OutPut (

@eid int,
@firstname nvarchar(255) out,
@lastname nvarchar (255) out
)
as
begin
            select @firstname =
                                    firstname
                                    from Employees where EmployeeID=@eid
            select @lastname =
                                    lastname
                                    from employees where EmployeeID=@eid

end
```

To execute

```
Declare @Fname as nvarchar(50)
Declare @Lname as nvarchar(50)    -- Declaring the variable to collect the Studentemail
Execute OutPut  1 , @Fname output , @Lname output
select 'using the output ' ,@Fname , @Lname     -- "Select" Statement is used to show the output
from Procedure
```

User defined Functions

- Creating custom functions

User defined functions:

- Create Function
- Execute the function
- Alter Function
- Drop function

Create a function

```
CREATE FUNCTION max_productID (
   @p_id int
)
RETURNS float
AS
BEGIN

        DECLARE @qty float
        SELECT @qty = 0  -- initialize the variable at 0:
        SELECT @qty = MAX(quantity)
              FROM orderdetails
              WHERE productID = @p_id
        RETURN ISNULL(@qty, 0)
        -- or you could do : SELECT @qty = isnull( MAX(quantity), 0 ) FROM orderdetails
        -- WHERE productID = @p_id ; return @qty

END
```

Execute the function in a query with parameter in ()

```
SELECT max_productID (1)
```

Exercise:

Find the revenue for with the productid as the parameter

Test your 2 functions

```
SELECT productname
   , max_p roductID (1) as max
from products
where productid = 1
```

Stored procedures: MYSQL:

https://www.mysqltutorial.org/mysql-stored-procedure-tutorial.aspx

No parameter

```
use training
DELIMITER //
create PROCEDURE GetAllP()
BEGIN
   SELECT categoryname  FROM categories;
END //
DELIMITER ;
call GetAllP()
```

Input Parameter

```
use training
DELIMITER //
CREATE PROCEDURE GetOfficeByCountry(
   IN countryName VARCHAR(255)
)
BEGIN
   SELECT *     FROM suppliers    WHERE country = countryName;
END //

DELIMITER ;
CALL GetOfficeByCountry('USA');
DELIMITER ;
```

Output Parameter

```
DELIMITER //
create PROCEDURE GetCount2 (
    IN  InID INT,
    OUT total INT
)
BEGIN
    SELECT orderID
                INTO total FROM orders WHERE shipperid = InID
    limit 1;
END //
DELIMITER ;
CALL GetCount2(1,@total);
SELECT @total;  -- some versions of Mysqlyou need not this line
DELIMITER ;
DROP PROCEDURE dbo.uspMyProc;
```

Stored procedures (MySQL)

```
DELIMITER //
CREATE  PROCEDURE `process1`()
BEGIN
SELECT FirstName, LastName, EmployeeID
FROM Employees;
END //
call Process1()

DELIMITER //
CREATE PROCEDURE `Process2`(
IN EID INT)
BEGIN
SELECT FirstName, LastName, EmployeeID
FROM Employees
WHERE EmployeeID=EID;
END //
call Process2(1)

DELIMITER //
CREATE PROCEDURE `Process3`(
IN EID INT,
out MaxW real)
BEGIN
SELECT MAX(Quantity)
FROM orderdetails o
join products p on o.ProductID = p.ProductID
WHERE CategoryID=EID;
END //
SET @M = 0;
call Process3(1, @M)
select @M
Alter: right click and follow the wizard
```

MYSQL : Functions

```
Create function

DELIMITER //
CREATE FUNCTION CalcIncome3 ( val INT )
RETURNS INT
BEGIN
  DECLARE income INT;
  SET income = 0;
  select floor( avg(price) ) into income
  from products
  where categoryid = val
  group by categoryid;
  RETURN income;
END; //

DELIMITER ;
-- You could then reference your new function as follows:
SELECT CalcIncome3 (2);
```

```
DROP FUNCTION CalcIncome;
```

MySQL
table valued function not possible. MySQL does not support table-valued (aka set-returning) functions.

Optional :

Looping

```
DECLARE @counter int = 0;
BEGIN
	WHILE (@counter < 10) BEGIN
		SELECT  ( DATEADD( day, RAND() * 365 * 50, GETDATE()));

		SET @counter = @counter + 1;
	END;
END;
```

EXAMPLE

The Stored Procedure has a date parameter.
proc_student should run in a loop while the date is between '2010-01-06' and '2010-01-25'.
I don't want to add anything inside the stored procedure ***proc_student***,
would prefer if just running it from a query outside the stored procedure ***proc_student***.

```
..........................................................
CREATE TABLE Student (
Id int,
Name varchar(20),
Class int,
Date datetime
)
.............................................................
insert into Student values(1, 'Komal',10,'2010-01-05');
insert into Student values(2, 'Ajay',10,'2010-01-07');
insert into Student values(3, 'Santosh',10,'2010-01-10');
insert into Student values(4, 'Rakesh',10,'2010-01-28');
insert into Student values(5, 'Bhau',10,'2010-01-17');
.............................................................

CREATE PROCEDURE proc_student ( @Date datetime  )
AS
Begin
        Select * from Student where Date = @Date
End
.............................................................
exec proc_student @Date = '2010-01-05'
.............................................................
```

Below is the external query to execute Stored Procedure in a Loop:

```
DECLARE @Date datetime
DECLARE @MaxDate datetime
SET @Date = '2010-01-05'
SET @MaxDate = '2010-01-25'

WHILE @Date < @MaxDate
BEGIN
      SET @Date = @Date + 1
      IF         @Date = '2010-01-06' OR
            @Date = '2010-01-07' OR
            @Date = '2010-01-11' OR
            @Date = '2010-01-20' OR
            @Date = '2010-01-25'
      EXEC proc_student @Date
END
```

Example

```
DECLARE @counter int = 0;
DECLARE @End int = 0;
select  @end = count(*) from employees;
select @counter, @end

BEGIN
  WHILE (@counter < @end) BEGIN
            select employeeid, firstname, birthdate from employees where employeeid=@counter
    update employees
    set birthdate = ( DATEADD( day, RAND() * 365 * 50, GETDATE())) ;
    SET @counter = @counter + 1;
             select employeeid, firstname, birthdate from employees where employeeid=@counter
            END;
END;
```

Case vs if

```
DECLARE @MyVal INT
DECLARE @TestVal INT
DECLARE @OUTPUTValues VARCHAR(200)
SET @MyVal = 1
SELECT @OUTPUTValues =
(
CASE  @MyVal
      WHEN 1 THEN 'test1'
      WHEN 2 THEN 'test2'
      WHEN 3 THEN 'test3'
      ELSE 'New'
END
)
PRINT @OUTPUTValues



--the same as
DECLARE @MyVal INT
DECLARE @testVal INT
DECLARE @OUTPUTValues VARCHAR(200)

SET @OUTPUTValues='hfashfsakjfs'
PRINT @OUTPUTValues

set @MyVal=5
IF @MyVal= 1
            SET @OUTPUTValues= 'test 1'
ELSE IF @MyVal= 2
            SET @OUTPUTValues= 'test 2'
ELSE IF @MyVal= 3
            SET @OUTPUTValues= 'test 3'
ELSE
            SET @OUTPUTValues= @MyVal

PRINT @OUTPUTValues
```

What is a table valued function?

Table-Valued Functions have been around since SQL Server version 2005. Basically a Table-Valued Function is a function that returns a table, thus it can be used as a table in a query.

User-defined **functions** that return a **table** data type can be powerful alternatives to views. These **functions** are referred to as **table-valued functions**. ... The **table**returned by a user-defined **function** can be referenced in the FROM clause of a Transact-SQL statement, but stored procedures that return result sets cannot.

What is a scalar function in SQL?
User-defined **scalar functions** return a single data value of the type defined in the RETURNS clause. ... The **function** takes one input value, a ProductID, and returns a single data value, the aggregated quantity of the specified product in inventory. Transact-**SQL**.

- Create function (returns @variablename table)
- Alter function
- `DROP FUNCTION DatesBetween;`

MySQL
table valued function not possible. MySQL does not support table-valued (aka set-returning) functions.

Creating a Table-Valued Function

First, let's create a small table to store some data:

```
CREATE TABLE TrackingItem (
  Id       int  NOT NULL IDENTITY(1,1),
  Issued   date NOT NULL,
  Category int  NOT NULL
);
CREATE INDEX X_TrackingItem_Issued ON TrackingItem (Issued);
```

And then add few rows for test data:

```
INSERT INTO TrackingItem (Issued, Category) VALUES ( DATEADD( day, 0, GETDATE()), 1);
INSERT INTO TrackingItem (Issued, Category) VALUES ( DATEADD( day, 1, GETDATE()), 2);
INSERT INTO TrackingItem (Issued, Category) VALUES ( DATEADD( day, 4, GETDATE()), 1);
INSERT INTO TrackingItem (Issued, Category) VALUES ( DATEADD( day, 4, GETDATE()), 2);
```

Now, if we would need a result set which would:

- Include all the columns from TrackingTable
- Include an extra Modified (date) column
- Not have even numbers in Category
- The Modified-column indicates when the changes into the data have been made
- Return only TrackingItem-rows having the Id greater than or equal to the parameter passed

The Table-Valued Function could look like this:

```
CREATE FUNCTION TrackingItemsModified(@minId int)
RETURNS @trackingItems TABLE (
  Id       int      NOT NULL,
  Issued   date     NOT NULL,
  Category int      NOT NULL,
  Modified datetime NULL
)
AS
BEGIN
  INSERT INTO @trackingItems (Id, Issued, Category)
  SELECT ti.Id, ti.Issued, ti.Category
  FROM   TrackingItem ti
  WHERE  ti.Id >= @minId;

  UPDATE @trackingItems
  SET Category = Category + 1,
      Modified = GETDATE()
  WHERE Category%2 = 0;

  RETURN;
END;
```

The function defines a new table called @trackingItems. This is a temporary table stored in the tempdb. The contents of this table will be return value for the function when the function exits.

First, the function inserts all the desired rows from the TrackingItem-table to the temporary table. After that, the contents of the temporary table are modified based on the specifications and then returned to the caller.

Using the Function

The next step is to use the function. If we want to select all the rows having Id equal to 2 or more, the query would look like:

```
SELECT * FROM TrackingItemsModified(2);
```

And the results:

```
Id  Issued     Category  Modified
--  ----------  --------  -----------------------
2   2011-03-11  3         2011-03-10 23:46:53.523
3   2011-03-14  1         NULL
4   2011-03-14  3         2011-03-10 23:46:53.523
```

As the result is a table, it can be used like one. For example, if we want to query all the original tracking items that don't exist in this subset, the query could be:

```
SELECT *
FROM  TrackingItem ti
WHERE ti.Id NOT IN (SELECT tim.Id
            FROM   TrackingItemsModified(2) tim)
```

An the results would be:

```
Id  Issued     Category
--  ----------  --------
1   2011-03-10  1
```

LOOPING in table valued function: Generating Data

So, **Table-Valued Functions** can be used to return modified data from one or more tables in the database. But since they are programmable functions, they can also generate data.
One quite common problem is to query all dates from a specified period and then have some results from a table which doesn't have entries for all the dates. In our test data, there is a row for today and tomorrow but the next few dates are missing. So, if we want to get the amount of tracking items for each day for the next seven days, it wouldn't be so simple. One typical solution is to create a table that contains all the necessary dates and then use that table in the query. Table-Valued Function can be used as an alternative. If we pass the date range to a function, we can create the necessary data on-the-fly with a simple loop.

```
CREATE FUNCTION DatesBetween(@startDate date, @endDate date)
RETURNS @dates TABLE (
  DateValue date NOT NULL
)
AS
BEGIN
  WHILE (@startDate <= @endDate) BEGIN
    INSERT INTO @dates VALUES (@startDate);
    SET @startDate = DATEADD(day, 1, @startDate);
  END;

  RETURN;
END;
```

And the query for the TrackingItem amounts would be:

```
SELECT d.DateValue,
      (SELECT COUNT(*)
       FROM   TrackingItem ti
       WHERE  d.DateValue = ti.Issued) AS Items
FROM DatesBetween(DATEADD(day, 1, GETDATE()), DATEADD(day, 7, GETDATE())) d
ORDER BY d.DateValue;
```

So the results would be something like:

```
DateValue   Items
----------  ------
2011-03-12  1
2011-03-13  0
2011-03-14  0
2011-03-15  2
2011-03-16  0
2011-03-17  0
2011-03-18  0
```

Example

```sql
CREATE FUNCTION DatesBetween(@startDate date, @endDate date)
RETURNS @dates TABLE (
  DateValue date NOT NULL PRIMARY KEY CLUSTERED
)
AS
BEGIN
  WHILE (@startDate <= @endDate) BEGIN
    INSERT INTO @dates VALUES (@startDate);
    SET @startDate = DATEADD(day, 1, @startDate);
  END;

  RETURN;
END;
```

Cursor

```
--  simple example of the SQL Server Cursor.

DECLARE @AccountID INT

DECLARE @getAccountID CURSOR
SET @getAccountID = CURSOR FOR
            SELECT CustomerID
            FROM Customers

OPEN @getAccountID
            FETCH NEXT FROM @getAccountID INTO @AccountID

WHILE @@FETCH_STATUS = 0
                        BEGIN
                                                PRINT @AccountID
                                                FETCH NEXT FROM @getAccountID INTO @AccountID
                        END


CLOSE @getAccountID
DEALLOCATE @getAccountID
```

http://stevestedman.com/2015/03/simple-introduction-to-tsql-cursors/

```
- - Declare all variable necessary for the block of code
DECLARE @ProductID as INT;
DECLARE @ProductName as NVARCHAR(50);

DECLARE @BusinessCursor as CURSOR;
-- set , does bnot run it.i.e like adeclare statement
SET @BusinessCursor = CURSOR FOR
SELECT Productid, ProductName
 FROM Products;

-- open and FECTH runs the query
-- @@Fetch_status is your like end-of-file indicator, checks if the previous fetch was successful
--, the fetch is really looping through every record
OPEN @BusinessCursor;
FETCH NEXT FROM @BusinessCursor INTO @ProductID, @ProductName;  -- reads the first record

WHILE @@FETCH_STATUS = 0 –if fetch was successful, enter the loop, otherwise skips to END
BEGIN
 PRINT cast(@ProductID as VARCHAR (50)) + ' ' + @ProductName;
 FETCH NEXT FROM @BusinessCursor INTO @ProductID, @ProductName;
END

CLOSE @BusinessCursor;
DEALLOCATE @BusinessCursor;
```

Exercise

Create a cursor for the above bur add a int varable @count that will count and print the lines of the result.

Solution

```
DECLARE @ProductID as INT;
DECLARE @ProductName as NVARCHAR(50);
Declare @Count as Int;
Set @count = 1;

DECLARE @BusinessCursor as CURSOR;
-- set , does bnot run it.i.e like adeclare statement
SET @BusinessCursor = CURSOR FOR
SELECT Productid, ProductName
 FROM Products
 where productname like 'c%';

-- open and FECTH runs the query
-- @@Fetch_status is your end-of-file indicator, the fetch is really looping through every record
OPEN @BusinessCursor;
FETCH NEXT FROM @BusinessCursor INTO @ProductID, @ProductName;  -- reads the first record

WHILE @@FETCH_STATUS = 0 --check if there is more, in that case enter the loop, otherwise skips
to END
BEGIN
 PRINT 'Line Number: ' + cast(@count as varchar(2))+' '+ cast(@ProductID as VARCHAR (50)) + ' '+
@ProductName; -- cast to print varchar
 FETCH NEXT FROM @BusinessCursor INTO @ProductID, @ProductName;
END

CLOSE @BusinessCursor;
DEALLOCATE @BusinessCursor;
```

https://www.youtube.com/watch?v=I0VqFAinzHM

do a backup for all databases

```
--declare variables
DECLARE @name as NVARCHAR(100);
DECLARE @path as NVARCHAR(256);
DECLARE @FileName as NVARCHAR(256);
DECLARE @FileDate as NVARCHAR(20);

set @path = 'C:\Users\pis\Desktop\Southend\'
set @FileDate = convert(nvarchar(20), getdate(), 112);

--declare cursors
DECLARE @db_cursor as CURSOR;
-- set cursor
set @db_cursor = cursor FOR
SELECT name
 FROM master.dbo.sysdatabases
 where name not in ('master','model','msdb','tempdb');

-- open cursor
OPEN @db_cursor;
-- fecth the first record of the cursor
FETCH NEXT FROM @db_cursor INTO @name;  -- reads the first record
                        set @FileName = @path+@name+'_'+@FileDate+'.BAK'
                        print @filename
-- loop for all
WHILE @@FETCH_STATUS = 0 --check that the last fetch was successful, in that case enter the
loop, otherwise skips to END
BEGIN
 set @FileName = @path+@name+'_'+@FileDate+'.BAK';
 backup database @name to disk = @filename with INIT, compression;
            print @filename;
 FETCH NEXT FROM @db_cursor INTO @name;
END

-- close and de-allocate to clean-up the memory
CLOSE @db_cursor;
DEALLOCATE @db_cursor;
```

(comment set @variabl = @@rowcount)

```sql
SELECT --TOP 1
  PERCENTILE_CONT(0.5)
  WITHIN GROUP (
           ORDER BY DATEDIFF(SECOND, [dateAdded],
[dateModified])
   ) OVER (PARTITION BY MONTH(DATEMODIFIED) ) AS
[MedianWaitTime]
FROM [pcpx].[dbo].[linkerDownloadQueue]
  WHERE [processStateID] = 2
  AND dateAdded >= convert(date, getdate())
```