

First Edition

BASIC

SQL QUERIES

By: Kamal Rawat & Akshay Patel



Basic SQL Queries

*This free book is provided by courtesy of C# Corner and Mindcracker Network and its authors. Feel free to share this book with your friends and co-workers. **Please do not reproduce, republish, edit or copy this book.***

Kamal Rawat & Akshay Patel

Author C# Corner

Sam Hobbs

Editor, C# Corner

Table of Content

1. Environment Setup

2. SQL Queries

- 2.1 Get all columns and all rows from table
- 2.2 Get desired columns and all rows from table
- 2.3 Get all columns from table where column name is equal to xxx
- 2.4 Get all columns from table where column name between xxx and xxx
- 2.5 Get all rows from table where column name is aaa,bbb,xxx
- 2.6 Get all rows from table where column name starts with 'x' letter
- 2.7 Get all rows from table where column name ends with 'x' letter
- 2.8 Get all rows from table where column name starts with 'x/z' letter
- 2.9 Get all rows from table where column name does not start with 'x/z' letter
- 2.10 Get single column with a combination of two columns and all rows from table
- 2.11 Get all rows from table where column name is not containing null values
- 2.12 Get all rows from table in descending order of column name
- 2.13 Get all rows from table in ascending order of column name
- 2.14 Get unique rows from table based on column name
- 2.15 Get top 5 rows from table name
- 2.16 Get maximum column value from table by column name
- 2.17 Get maximum column value from table by column name using COMPUTE
- 2.18 Get even row from table 'Product'
- 2.19 Get odd row from table 'Product'
- 2.20 Count number of records in a 'Product' table
- 2.21 List all the categories with products (Inner Join)
- 2.22 List all rows from category, with products (Left Join)
- 2.23 List all the rows from products (Right Join)
- 2.24 List all the rows from the category table and product table (Full Join)
- 2.25 Get average value of price from product table

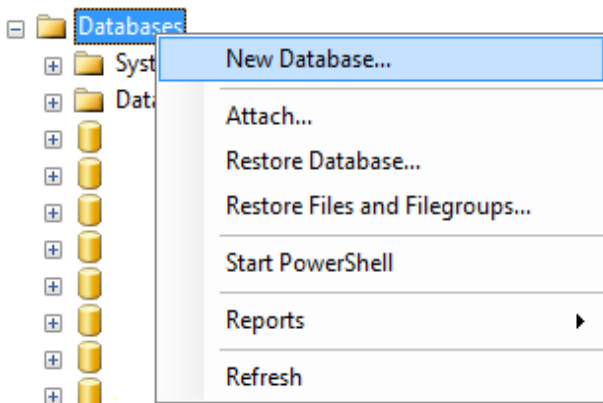
- 2.26 Get number of products from product table
- 2.27 Get Maximum price of product from product table
- 2.28 Get minimum price of product from product table
- 2.29 Get total number of qty of products from product table
- 2.30 Get all product name is in upper case from product table
- 2.31 Get all product name is in lower case from product table
- 2.32 List all columns of all tables in a database
- 2.33 List all tables which have column name like 'CategoryId'
- 2.34 List number of records in each table in a database
- 2.35 List all tables in a database
- 2.36 List all procedures from a database
- 2.37 List all tables in all databases
- 2.38 List all Stored Procedures in All Databases
- 2.39 List sizes of all tables in a database
- 2.40 List sizes of all database files
- 2.41 Generate xsd of a table without data
- 2.42 Generate xsd of a table with data



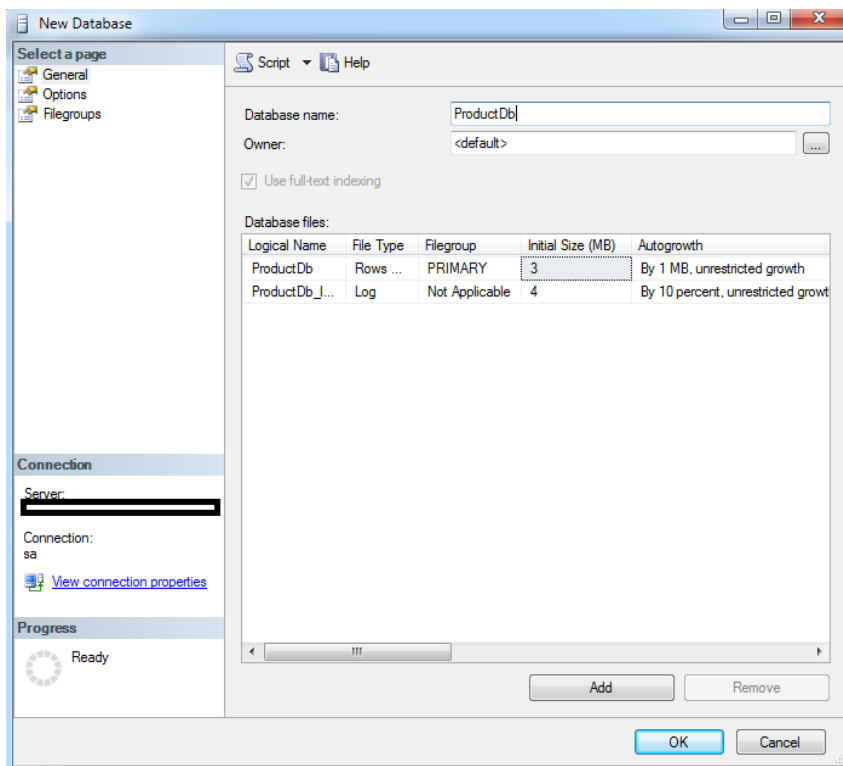
1. Environment Setup

Follow the following steps in your sql server before you execute all the queries available in this book.

Step 1: Create a database named 'ProductDb'

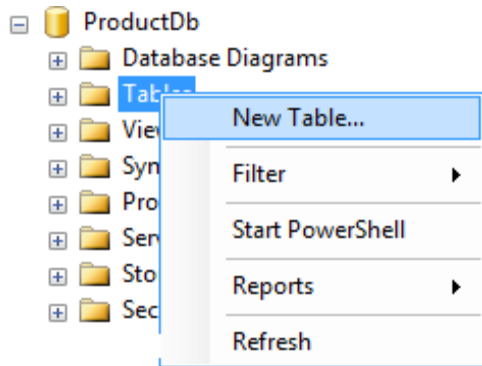


Right click on databases and select 'New Database...' option.



Insert database name as 'ProductDb' and press ok button.

Step 2: Create a table 'Category'



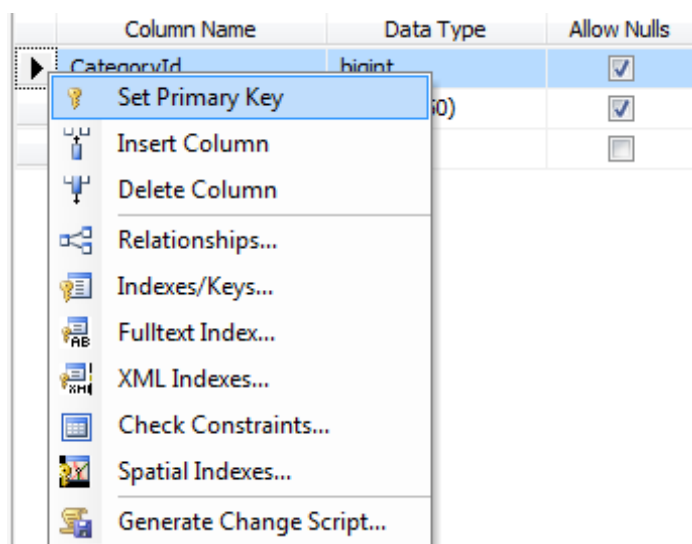
Right click on tables and select 'New Table...'.

Column Name	Data Type	Allow Nulls
CategoryId	bigint	<input checked="" type="checkbox"/>
CategoryName	varchar(50)	<input checked="" type="checkbox"/>

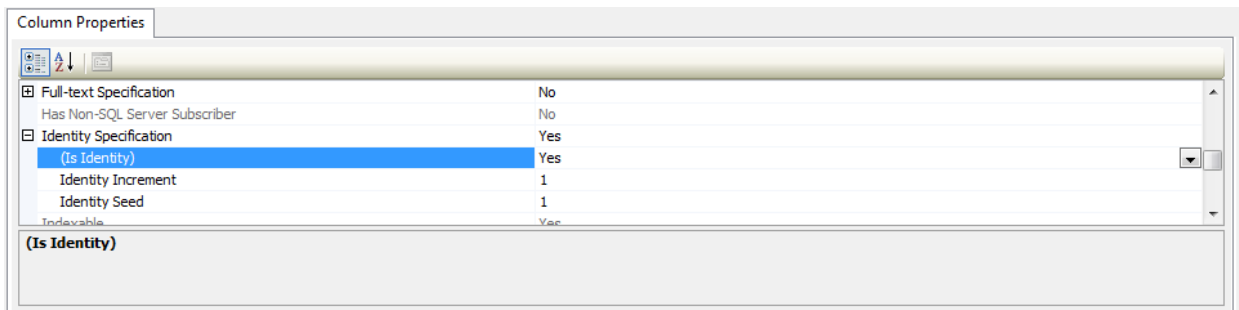
Add CategoryId of type bigint and CategoryName of type varchar with the size of 50.

Now set primary key on the column 'CategoryId'.

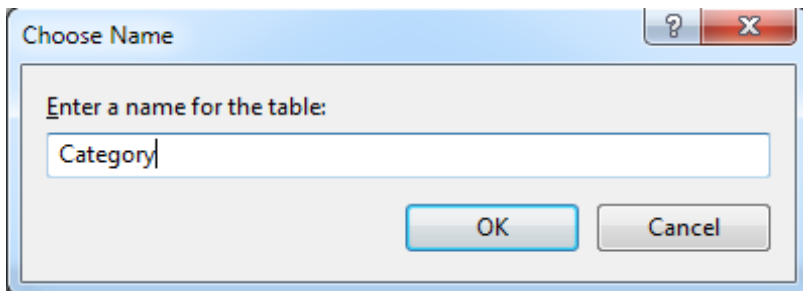
Right click on the field 'CategoryId' and select 'Set Primary Key'.



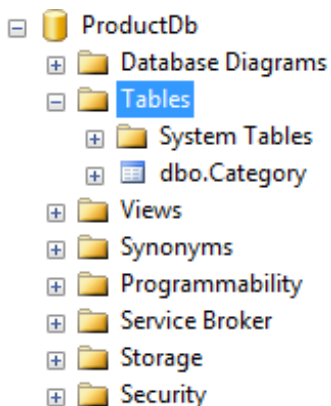
Now set identity specification in column properties of 'CategoryId' column.



Press **ctrl + s** to save the table. Give name 'Category'




Now you can see the table 'Category' under the 'Tables' Folder.



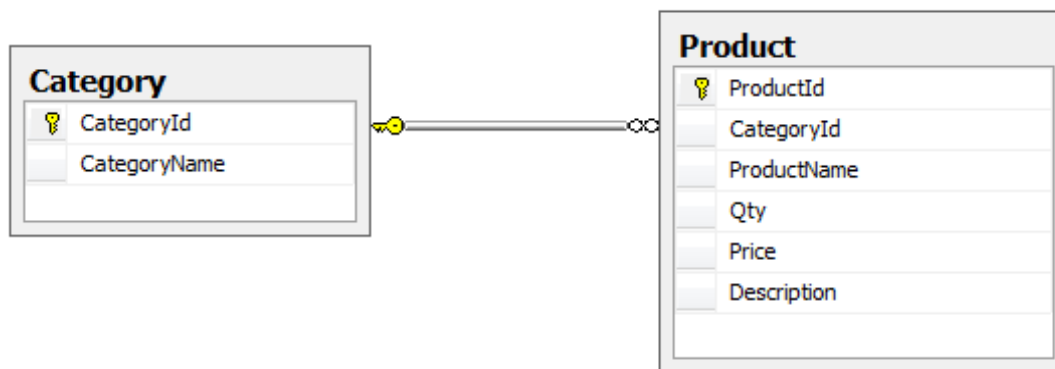
Now follow the above steps once again and create one more table 'Product'.

The proposed schema for product table follows is as follows:

	Column Name	Data Type	Allow Nulls
	ProductId	bigint	<input type="checkbox"/>
	CategoryId	bigint	<input checked="" type="checkbox"/>
	ProductName	varchar(50)	<input checked="" type="checkbox"/>
	Qty	int	<input checked="" type="checkbox"/>
	Price	decimal(18, 2)	<input checked="" type="checkbox"/>
	Description	text	<input checked="" type="checkbox"/>

After creating product table set foreign key relation between 'CategoryId' of Category table and 'CategoryId' of Product table.

Now our database diagram looks like this:



Now insert some data in both the tables by executing following query:

First insert into 'Category' table:

```

INSERT INTO Category (CategoryId) VALUES ('Stationery')
INSERT INTO Category (CategoryId) VALUES ('Computer')
  
```

Now your 'Category' table contains following data:

	CategoryId	CategoryName
1	1	Stationery
2	2	Computer

Now insert few records into 'Product' table:

```
INSERT INTO Product (CategoryId,ProductName,Qty,Price,Description) VALUES (1,'Pen',10,15,'Good pen for exam')
INSERT INTO Product (CategoryId,ProductName,Qty,Price,Description) VALUES (1,'Pencil',100,5,'Good pencil for drawing')
INSERT INTO Product (CategoryId,ProductName,Qty,Price,Description) VALUES (1,'Notebook',50,25,'Notebook for primary students')
INSERT INTO Product (CategoryId,ProductName,Qty,Price,Description) VALUES (1,'Fullscape',200,15,'Fullscape for higher secondary students')
INSERT INTO Product (CategoryId,ProductName,Qty,Price,Description) VALUES (1,'Eraser',1000,1.50,'Eraser for KG Students')
INSERT INTO Product (CategoryId,ProductName,Qty,Price,Description) VALUES (2,'Mouse',20,200,'USB 2.0 Mouse')
INSERT INTO Product (CategoryId,ProductName,Qty,Price,Description) VALUES (2,'Keyboard',20,250,'USB 3.0 Keyboard')
INSERT INTO Product (CategoryId,ProductName,Qty,Price,Description) VALUES (2,'USB Stick',1000,450,'4 GB Pendrives')
```

Now 'Product' table contains following data:

	ProductId	CategoryId	ProductName	Qty	Price	Description
▶	1	1	Pen	10	15.00	Good pen for exam
	2	1	Pencil	100	5.00	Good pencil for ...
	3	1	Notebook	50	25.00	Notebook for pri...
	4	1	Fullscape	200	15.00	Fullscape for hig...
	5	1	Eraser	1000	1.50	Eraser for KG St...
	6	2	Mouse	20	200.00	USB 2.0 Mouse
	7	2	Keyboard	20	250.00	USB 3.0 Keyboard
	8	2	USB Stick	1000	450.00	4 GB Pendrives

2. Sql Queries

2.1: Get all columns and all rows from 'Product' table

Query:

```
SELECT * FROM Product
```

Result:



ProductId	CategoryId	ProductName	Qty	Price	Description
1	1	Pen	10	15	Good pen for exam
2	1	Pencil	100	5	Good pencil for drawing
3	1	Notebook	50	25	Notebook for primary students
4	1	Fullscape	200	15	Fullscape for higer secondary students
5	1	Eraser	1000	1.5	Eraser for KG Students
6	2	Mouse	20	200	USB 2.0 Mouse
7	2	Keyboard	20	250	USB 3.0 Keyboard
8	2	USB Stick	1000	450	4 GB Pendrives

Messages:

(8 row(s) affected)

Description:

It feteches all the records available in the product table. '*' is used in conjunction with query to fetch all the columns value.

Note: '*' can be replaced with column names if only sppecific columns need to be shown.

2.2: Get Column with their respective values from the table.

Query:

SELECT ProductName, Qty, Price **FROM** Product

Result:

ProductName	Qty	Price
Pen	10	15
Pencil	100	5
Notebook	50	25
Fullscape	200	15
Eraser	1000	1.5
Mouse	20	200
Keyboard	20	250
USB Stick	1000	450

Messages:

(8 row(s) affected)

Description:

It fetches columns value from product table and columns are separated by comma operator. Here you are not using *, as you want to fetch only specific columns.

2.3: Get all columns from 'Product' table with where clause.

'ProductId' is equal to 2

Query:

```
SELECT * FROM Product WHERE ProductId=2
```

Result:

ProductId	CategoryId	ProductName	Qty	Price	Description
2	1	Pencil	100	5	Good pencil for drawing

Messages:

(1 row(s) affected)

Description:

It fetches record from product table where ProductId = 2.

In the above query, where clause is used to specify condition. E.g

```
SELECT * FROM Product WHERE ProductId=2
```

Hence, Only those records will be displayed whose productid = 2.

2.4: Get all columns from 'Product' table using Between Operator.

Query:

```
SELECT * FROM Product WHERE ProductId Between 4 and 8
```

Result:

ProductId	CategoryId	ProductName	Qty	Price	Description
4	1	Fullscape	200	15	Fullscape for higer secondary students
5	1	Eraser	1000	1.5	Eraser for KG Students
6	2	Mouse	20	200	USB 2.0 Mouse
7	2	Keyboard	20	250	USB 3.0 Keyboard
8	2	USB Stick	1000	450	4 GB Pendrives

Messages:

(5 row(s) affected)

Description:

It fetches records from the product table where productid ranges between 4 and 8.

For this purpose **BETWEEN** operator is used in conjunction with **WHERE** clause.

Note: **BETWEEN** operator can't be used without **WHERE** clause.

2.5: Get all rows from 'Product' table where ProductName is Mouse,Keyboard,USB Stick

Query:

```
SELECT * FROM Product WHERE ProductName in ('Mouse','Keyboard','USB Stick')
```

Result:

ProductId	CategoryId	ProductName	Qty	Price	Description
6	2	Mouse	20	200	USB 2.0 Mouse
7	2	Keyboard	20	250	USB 3.0 Keyboard
8	2	USB Stick	1000	450	4 GB Pendrives

Messages:

(3 row(s) affected)

Description:

It fetches the records from the product table where productname could be any of the names specified in '**in clause**' For e.g.

```
SELECT * FROM Product WHERE ProductName in ('Mouse','Keyboard','USB Stick')
```

It fetches those rows from the table where ProductName could be either **Mouse** or **Keyboard** or **USB Stick**.

Hence, **IN** clause is used to specify more than one choices/options in **conjunction** with WHERE clause.

2.6: Get all rows from 'Product' table where the name starts with 'K' letter.

Query:

```
SELECT * FROM Product WHERE ProductName Like 'K%'
```

Result:

ProductId	CategoryId	ProductName	Qty	Price	Description
7	2	Keyboard	20	250	USB 3.0 Keyboard

Messages:

(1 row(s) affected)

Description:

It fetches records from the product table where ProductName starts with K.

For this purpose, LIKE operator is used to define the pattern.

Few LIKE Patterns:

LIKE 'K%' : ProductName starts with K.

LIKE '%K' : ProductName ends with K.

NO NEED OF THIS QUERY

2.7: Get all rows from 'Product' table where the name ends with 'K' letter.

Query:

```
SELECT * FROM Product WHERE ProductName Like '%K'
```

Result:

ProductId	CategoryId	ProductName	Qty	Price	Description
3	1	Notebook	50	25	Notebook for primary students
8	2	USB Stick	1000	450	4 GB Pendrives

Messages:

(2 row(s) affected)

Description:

2.8: Get all rows from 'Product' table where the name starts with 'E/F' letter.

Query:

```
SELECT * FROM Product WHERE ProductName Like '[EF]%'
```

Result:

ProductId	CategoryId	ProductName	Qty	Price	Description
4	1	Fullscape	200	15	Fullscape for higer secondary students
5	1	Eraser	1000	1.5	Eraser for KG Students

Messages:

(2 row(s) affected)

Description:

It fetches records from the product table where ProductName start with either 'E' or 'F'.

2.9: Get all rows from 'Product' table where the name not starts with 'E/F' letter.

Query:

```
SELECT * FROM Product WHERE ProductName Like '[^EF]'
```

Result:

ProductId	CategoryId	ProductName	Qty	Price	Description
1	1	Pen	10	15	Good pen for exam
2	1	Pencil	100	5	Good pencil for drawing
3	1	Notebook	50	25	Notebook for primary students
6	2	Mouse	20	200	USB 2.0 Mouse
7	2	Keyboard	20	250	USB 3.0 Keyboard
8	2	USB Stick	1000	450	4 GB Pendrives

Messages:

(6 row(s) affected)

Description:

It fetches records from the product table where ProductName doesn't start with either 'E' or 'F'.

2.10: Get single column with a combination of Product and Qty added together with a space from Product' table

Query:

```
SELECT (ProductName + SPACE(1) + CONVERT(VARCHAR,Qty)) AS ProductWithQty FROM Product
```

Result:

ProductWithQty
Pen 10
Pencil 100
Notebook 50
Fullscape 200
Eraser 1000
Mouse 20
Keyboard 20
USB Stick 1000

Messages:

(8 row(s) affected)

Description:

It fetches ProductName with Quantity have space between them.

In this query, Additional data i.e SPACE(1) is added to add space between ProductName and ProductQty. So additional data can be added within the select query using '+' operator.

2.11: Get all rows from 'Product' table where ProductName column is not containing null values

Query:

```
SELECT * FROM Product WHERE Description IS NOT NULL
```

Result:

ProductId	CategoryId	ProductName	Qty	Price	Description
1	1	Pen	10	15	Good pen for exam
2	1	Pencil	100	5	Good pencil for drawing
3	1	Notebook	50	25	Notebook for primary students
4	1	Fullscape	200	15	Fullscape for higer secondary students
5	1	Eraser	1000	1.5	Eraser for KG Students
6	2	Mouse	20	200	USB 2.0 Mouse
7	2	Keyboard	20	250	USB 3.0 Keyboard
8	2	USB Stick	1000	450	4 GB Pendrives

Messages:

(7 row(s) affected)

Description:

It fetches records from the Product database where Description is not null.

In some cases data could remain NULL, so in order to find out only those records which are not null 'IS NOT NULL' clause is used in conjunction with WHERE clause/condition.

2.12: Get all rows from 'Product' table in descending order of ProductId

Query:

```
SELECT * FROM Product ORDER BY ProductId DESC
```

Result:

ProductId	CategoryId	ProductName	Qty	Price	Description
8	2	USB Stick	1000	450	4 GB Pendrives
7	2	Keyboard	20	250	USB 3.0 Keyboard
6	2	Mouse	20	200	USB 2.0 Mouse
5	1	Eraser	1000	1.5	Eraser for KG Students
4	1	Fullscape	200	15	Fullscape for higer secondary students
3	1	Notebook	50	25	Notebook for primary students
2	1	Pencil	100	5	Good pencil for drawing
1	1	Pen	10	15	Good pen for exam

Messages:

(8 row(s) affected)

Description:

It fetches records from Product table in Descending order.

Note: Ascending order is by default.

2.13: Get all rows from 'Product' table in ascending order of ProductId**Query:**

```
SELECT * FROM Product ORDER BY ProductId
```

Result:

ProductId	CategoryId	ProductName	Qty	Price	Description
1	1	Pen	10	15	Good pen for exam
2	1	Pencil	100	5	Good pencil for drawing
3	1	Notebook	50	25	Notebook for primary students
4	1	Fullscape	200	15	Fullscape for higer secondary students
5	1	Eraser	1000	1.5	Eraser for KG Students
6	2	Mouse	20	200	USB 2.0 Mouse
7	2	Keyboard	20	250	USB 3.0 Keyboard
8	2	USB Stick	1000	450	4 GB Pendrives

Messages:

(8 row(s) affected)

Description:

It fetches records from Product table in ascending order.

2.14: Get Unique row from 'Product' table based on CategoryId

Query:

```
SELECT DISTINCT CategoryId FROM Product
```

Result:

CategoryId
1
2

Messages:

(2 row(s) affected)

Description:

It fetches unique records from Product table based on CategoryId.

2.15: Get top 5 Rows from 'Product' table

Query:

```
SELECT Top 5 * FROM Product
```

Result:

ProductId	CategoryId	ProductName	Qty	Price	Description
1	1	Pen	10	15	Good pen for exam
2	1	Pencil	100	5	Good pencil for drawing
3	1	Notebook	50	25	Notebook for primary students
4	1	Fullscape	200	15	Fullscape for higer secondary students
5	1	Eraser	1000	1.5	Eraser for KG Students

Messages:

(5 row(s) affected)

Description:

It fetches top 5 rows from the Product table.

Hence, **TOP n** statement is used to fetch top n rows from the table.

2.16: Get maximum value Of ProductId from 'Product' table**Query:**

```
SELECT MAX(ProductId) FROM Product
```

Result:

(No column name)
8

Messages:

(1 row(s) affected)

Description:

It fetches the row from Product table having maximum value.

In this case ProductId is passed in **MAX()** function, so returns the row having maximum value of ProductId.

2.17: Get maximum value Of Qty from 'Product' table**Query:**

```
SELECT * FROM Product COMPUTE MAX(Qty)
```

Result:

ProductId	CategoryId	ProductName	Qty	Price	Description
1	1	Pen	10	15	Good pen for exam
2	1	Pencil	100	5	Good pencil for drawing
3	1	Notebook	50	25	Notebook for primary students
4	1	Fullscape	200	15	Fullscape for higer secondary students
5	1	Eraser	1000	1.5	Eraser for KG Students
6	2	Mouse	20	200	USB 2.0 Mouse
7	2	Keyboard	20	250	USB 3.0 Keyboard
8	2	USB Stick	1000	450	4 GB Pendrives

max
1000

Messages:

(9 row(s) affected)

Description:

2.18: Get even row from table 'Product'

Query:

```
SELECT Temp.* FROM
    (SELECT ROW_NUMBER() OVER (ORDER BY ProductId) AS row_num,
      Product.* FROM Product) Temp
WHERE ((Temp.row_num) % 2) = 0
```

Result:

row_num	ProductId	CategoryId	ProductName	Qty	Price	Description
2	2	1	Pencil	100	5	Good pencil for drawing
4	4	1	Fullscape	200	15	Fullscape for higer secondary students
6	6	2	Mouse	20	200	USB 2.0 Mouse
8	8	2	USB Stick	1000	450	4 GB Pendrives

Messages:

(4 row(s) affected)

Description:

It fetches out even rows from the Product table.

2.19: Get odd row from table 'Product'

Query:

```
SELECT Temp.* FROM
    (SELECT ROW_NUMBER() OVER (ORDER BY ProductId) AS row_num,
      Product.* FROM Product) Temp
WHERE ((Temp.row_num) % 2) = 1
```

Result:

row_num	ProductId	CategoryId	ProductName	Qty	Price	Description
1	1	1	Pen	10	15	Good pen for exam
3	3	1	Notebook	50	25	Notebook for primary students
5	5	1	Eraser	1000	1.5	Eraser for KG Students
7	7	2	Keyboard	20	250	USB 3.0 Keyboard

Messages:

(4 row(s) affected)

Description:

It fetches out odd rows from Product table.

2.20: Count number of records in a 'Product' table

Query:

```
SELECT COUNT(*) AS RecordCount FROM Product
```

Result:

RecordCount
8

Messages:

(1 row(s) affected)

Description:

It fetches total number of records available in Product table.

2.21: List all the categories with products (Inner Join)

Query:

```
SELECT * FROM Category  
INNER JOIN Product  
ON Category.CategoryId = Product.CategoryId
```

Result:

CategoryId	CategoryName	ProductId	CategoryId	ProductName	Qty	Price	Description
1	Stationery	1	1	Pen	10	15	Good pen for exam
1	Stationery	2	1	Pencil	100	5	Good pencil for drawing
1	Stationery	3	1	Notebook	50	25	Notebook for primary students
1	Stationery	4	1	Fullscape	200	15	Fullscape for higer secondary students
1	Stationery	5	1	Eraser	1000	1.5	Eraser for KG Students
2	Computer	6	2	Mouse	20	200	USB 2.0 Mouse
2	Computer	7	2	Keyboard	20	250	USB 3.0 Keyboard
2	Computer	8	2	USB Stick	1000	450	4 GB Pendrives

Messages:

(8 row(s) affected)

Description:

The INNER JOIN keyword returns rows when there is at least one match in both tables. If there are rows in "Product" that do not have matches in "Category", those rows will NOT be listed.

Inner join creates a new result table by combining column values of two tables (A and B) based upon the join-predicate. The query compares each row of A with each row of B to find all pairs of rows which satisfy the join-clause.

The "**explicit join notation**" uses the **JOIN** keyword to specify the table to join, and the **ON** keyword to specify the predicates for the join, as in the following example:


```
SELECT * FROM Category
INNER JOIN Product
ON Category.CategoryId = Product.CategoryId
```

The following example is equivalent to the previous one, but this time using **implicit join** notation:

```
SELECT * FROM Category, Product
WHERE Category.CategoryId = Product.CategoryId
```

2.22: List all rows from category, even if there are no matches in the product table (Left Join)

Query:

```
SELECT * FROM Category
LEFT JOIN Product
ON Category.CategoryId = Product.CategoryId
```

Result:

CategoryId	CategoryName	ProductId	CategoryId	ProductName	Qty	Price	Description
1	Stationery	1	1	Pen	10	15	Good pen for exam
1	Stationery	2	1	Pencil	100	5	Good pencil for drawing
1	Stationery	3	1	Notebook	50	25	Notebook for primary students
1	Stationery	4	1	Fullscape	200	15	Fullscape for higer secondary students
1	Stationery	5	1	Eraser	1000	1.5	Eraser for KG Students
2	Computer	6	2	Mouse	20	200	USB 2.0 Mouse
2	Computer	7	2	Keyboard	20	250	USB 3.0 Keyboard
2	Computer	8	2	USB Stick	1000	450	4 GB Pendrives
3	Mobile	NULL	NULL	NULL	NULL	NULL	NULL
4	Tablet	NULL	NULL	NULL	NULL	NULL	NULL

Messages:

(10 row(s) affected)

Description:

The LEFT JOIN keyword returns all the rows from the left table (Category), even if there are no matches in the right table (Product).

This means that if the **ON** clause matches 0 (zero) records in **Table B** (for a given record in **Table A**), the join will still return a row in the result (for that record)—but with NULL in each column from **Table B**.

2.23: List all the rows from product, even if there are no matches in the category table (Right Join)

Query:

```
SELECT * FROM Category
RIGHT JOIN Product
ON Category.CategoryId = Product.CategoryId
```

Result:

CategoryId	CategoryName	ProductId	CategoryId	ProductName	Qty	Price	Description
1	Stationery	1	1	Pen	10	15	Good pen for exam
1	Stationery	2	1	Pencil	100	5	Good pencil for drawing
1	Stationery	3	1	Notebook	50	25	Notebook for primary students
1	Stationery	4	1	Fullscape	200	15	Fullscape for higer secondary students
1	Stationery	5	1	Eraser	1000	1.5	Eraser for KG Students
2	Computer	6	2	Mouse	20	200	USB 2.0 Mouse
2	Computer	7	2	Keyboard	20	250	USB 3.0 Keyboard
2	Computer	8	2	USB Stick	1000	450	4 GB Pendrives
NULL	NULL	9	0	Table	10	1500	Computer Table
NULL	NULL	10	0	Chair	10	500	Chair

Messages:

(10 row(s) affected)

Description:

The RIGHT JOIN keyword returns all the rows from the right table (Product), even if there are no matches in the left table (Category).

2.24: List all the rows from the category table and all the rows from the product table (Full Join)

Query:

```
SELECT * FROM Category
FULL JOIN Product
ON Category.CategoryId = Product.CategoryId
```

Result:

CategoryId	CategoryName	ProductId	CategoryId	ProductName	Qty	Price	Description
1	Stationery	1	1	Pen	10	15	Good pen for exam
1	Stationery	2	1	Pencil	100	5	Good pencil for drawing
1	Stationery	3	1	Notebook	50	25	Notebook for primary students
1	Stationery	4	1	Fullscape	200	15	Fullscape for higher secondary students
1	Stationery	5	1	Eraser	1000	1.5	Eraser for KG Students
2	Computer	6	2	Mouse	20	200	USB 2.0 Mouse
2	Computer	7	2	Keyboard	20	250	USB 3.0 Keyboard
2	Computer	8	2	USB Stick	1000	450	4 GB Pendrives
3	Mobile	NULL	NULL	NULL	NULL	NULL	NULL
4	Tablet	NULL	NULL	NULL	NULL	NULL	NULL
NULL	NULL	9	0	Table	10	1500	Computer Table
NULL	NULL	10	0	Chair	10	500	Chair

Messages:

(12 row(s) affected)

Description:

The FULL JOIN keyword returns all the rows from the left table (Category), and all the rows from the right table (Product). If there are rows in "Category" that do not have matches in "Product", or if there are rows in "Product" that do not have matches in "Category", those rows will be listed as well.

2.25: Get average value of price from product table**Query:**

```
SELECT AVG (Price) AS PriceAverage FROM Product
```

Result:

PriceAverage
296.15

Messages:

(1 row(s) affected)

Description:**2.26: Get number of products from product table****Query:**

```
SELECT COUNT (ProductName) AS NoOfProducts FROM Product
```

Result:

NoOfProducts
10

Messages:

(1 row(s) affected)

Description:**2.27: Get Maximum price of product from product table****Query:**

```
SELECT MAX(Price) AS MaximumPrice FROM Product
```

Result:

MaximumPrice
1500

Messages:

(1 row(s) affected)

Description:

The MAX() function returns the maximum value of the selected column.

2.28: Get minimum price of product from product table**Query:**

```
SELECT MIN(Price) AS MinimumPrice FROM Product
```

Result:

MinimumPrice
1.5

Messages:

(1 row(s) affected)

Description:

The MIN() function returns the smallest value of the selected column.

2.29: Get total number of qty of products from product table

Query:

```
SELECT SUM(Qty) AS TotalProducts FROM Product
```

Result:

TotalProducts
2420

Messages:

(1 row(s) affected)

Description:

The SUM() function returns the total sum of a numeric column.

2.30: Get all product name is in upper case from product table

Query:

```
SELECT UPPER(ProductName) AS PRODUCTS FROM Product
```

Result:

PRODUCTS
PEN
PENCIL
NOTEBOOK
FULLSCAPE
ERASER
MOUSE
KEYBOARD
USB STICK
TABLE
CHAIR

Messages:

(10 row(s) affected)

Description:

Returns a character expression after converting character data to Uppercase.

2.31: Get all product name is in lower case from product table**Query:**

```
SELECT LOWER(ProductName) AS products FROM Product
```

Result:

Products
pen
pencil
notebook
fullscape
eraser
mouse
keyboard
usb stick
table
chair

Messages:

(10 row(s) affected)

Description:

Returns a character expression after converting character data to lowercase.

2.32: List all columns of all tables in a database

Query:

```
SELECT sys.objects.name AS TableName,sys.columns.name AS ColumnName
FROM sys.columns
INNER JOIN sys.objects
    ON sys.columns.object_id=sys.objects.object_id
    and type_desc = 'USER_TABLE'
ORDER BY sys.objects.name,sys.columns.column_id
```

Result:

TableName	ColumnName
Category	CategoryId
Category	CategoryName
Product	ProductId
Product	CategoryId
Product	ProductName
Product	Qty
Product	Price
Product	Description

Messages:

(8 row(s) affected)

Description:

It pulls out all the columns of all the tables of the active/current working database. In order to achieve this **sys.columns** table is used, as it is a system table (prefixed with sys).

* You can check rest of the fields/columns available in sys.columns table using the following query

```
Select * from sys.columns
```

These queries could be useful for DBA's or programmers to create scripts etc.

2.33: List all tables which have column name like 'CategoryId'

Query:

```
SELECT
    sys.tables.name AS table_name,
    SCHEMA_NAME(schema_id) AS schema_name,
    sys.columns.name AS column_name
FROM sys.tables
INNER JOIN sys.columns
    ON sys.tables.OBJECT_ID = sys.columns.OBJECT_ID
WHERE sys.columns.name LIKE '%CategoryId%'
ORDER BY schema_name, table_name
```

Result:

table_name	schema_name	column_name
Category	Dbo	CategoryId
Product	Dbo	CategoryId

Messages:

(2 row(s) affected)

Description:

It pulls out records from **sys.tables** having column 'CategoryId' available in any of the tables for the current working database.

Like **sys.columns**, **sys.tables** is also a system table and provide meaningful information about the tables.

2.34: List number of records in each table in a database

Query:

```
DECLARE @DSql VARCHAR(MAX)

SELECT @DSql =
    COALESCE(@DSql + CHAR(13) + ' UNION ALL ' + CHAR(13), '') +
    'SELECT ' + QUOTENAME(TABLE_NAME, '"') + ' as [Table Name], COUNT(*) AS [Records Count]
    FROM ' + QUOTENAME(Table_schema) + '.' + QUOTENAME(TABLE_NAME)
FROM INFORMATION_SCHEMA.TABLES
ORDER BY TABLE_NAME

EXEC( @DSql)
```

Result:

Table Name	Records Count
Category	2
Product	8

Messages:

(2 row(s) affected)

Description:

It pulls out total number of records available for each table in the current working database.

2.35: List all tables in a database**Query:**

```
SELECT Table_Schema AS [Table Schema], Table_Name AS [Table Name]
FROM INFORMATION_SCHEMA.Tables
```

Result:

Table Schema	Table Name
Dbo	Category
Dbo	Product

Messages:

(2 row(s) affected)

Description:

It lists out all the tables created under the active working database.

2.36: List all procedures from a database**Query:**

```
SELECT name,create_date, modify_date FROM sys.procedures
```

Result:

Name	create_date	modify_date
CategorySelectData	2012-12-09 15:48:02.513	2012-12-09 15:48:02.513
ProductSelectData	2012-12-09 15:48:17.960	2012-12-09 15:48:17.960

Messages:

(2 row(s) affected)

Description:

It lists out procedures created under active working database.

Again helpful for DBA's (Database Administrator) and programmers to keep track of number of procedures made along with meaningful details like **createddate, last modified date**.

2.37: List all tables in all databases**Query:**

```

CREATE TABLE #Temp
(
    DBName sysname,
    TableSchema sysname,
    TableName sysname
)
DECLARE @DSql NVARCHAR(MAX)

SELECT @DSql = COALESCE(@DSql, '') + '
    INSERT INTO #temp
    SELECT ' + QUOTENAME(name, '''') + ' as [DB Name], [Table_Schema] as [Table Schema],
    [Table_Name] as [Table Name]
    FROM ' + QUOTENAME(Name) + '.INFORMATION_SCHEMA.Tables;'

FROM sys.databases
ORDER BY name

EXECUTE(@DSql)

SELECT * FROM #temp

DROP TABLE #temp
    
```

Result:

DBName	TableSchema	TableName
Master	dbo	spt_fallback_db
Master	dbo	spt_fallback_dev
Master	dbo	spt_fallback_usg
Master	dbo	spt_monitor
Master	dbo	spt_values
Master	dbo	MSreplication_options
Msdb	dbo	sysutility_ucp_volume_powershell_path
Msdb	dbo	syspolicy_policy_execution_history
Msdb	dbo	syspolicy_policies_internal
Msdb	dbo	sysutility_ucp_instance_policies
Msdb	dbo	Sysproxies

.....

Messages:

(6 row(s) affected)

(0 row(s) affected)

.....

Description:

It lists out all the tables of all the databases.

Hence, **sys.databases** is used in the query to list out the details.

2.38: List all Stored Procedures in All Databases

Query:

```
CREATE TABLE #Temp
(
    DBName SYSNAME,
    SPName SYSNAME,
    create_date DATETIME,
    modify_date DATETIME
)

DECLARE @DSql NVARCHAR(MAX)
SET @DSql = ''
SELECT @DSql = @DSql + ' INSERT INTO #Temp
                        SELECT ' + QUOTENAME(name, '') + ', name, create_date, modify_date
                        FROM ' + QUOTENAME(name) + '.sys.procedures' from sys.databases

EXECUTE (@DSql)

SELECT * FROM #Temp ORDER BY DBName, SPName

DROP TABLE #temp
```

Result:

DBName	SPName	create_date	modify_date
master	sp_MScleanupmergepublisher	2012-08-23 00:10:07.397	2012-08-23 00:10:07.440
master	sp_MSrepl_startup	2012-08-23 00:10:07.327	2012-08-23 00:10:07.367
msdb	sp_add_alert	2012-08-23 00:09:37.657	2012-08-23 00:10:01.290
msdb	sp_add_alert_internal	2012-08-23 00:09:37.470	2012-08-23 00:10:01.283
msdb	sp_add_category	2012-08-23 00:09:33.243	2012-08-23 00:10:00.810
msdb	sp_add_dtscategory	2012-08-23 00:09:39.220	2012-08-23 00:10:02.000
msdb	sp_add_dtspackage	2012-08-23 00:09:39.163	2012-08-23 00:10:01.960
msdb	sp_add_job	2012-08-23 00:09:35.800	2012-08-23 00:10:01.090
master	sp_MScleanupmergepublisher	2012-08-23 00:10:07.397	2012-08-23 00:10:07.440

.....

Messages:

(2 row(s) affected)

(0 row(s) affected)

.....

Description:

It lists out all the procedures for all the databases.

2.39: List sizes of all tables in a database

Query:

```
CREATE TABLE #Temp
(
    TableName SYSNAME,
    ROWS BIGINT,
    reserved VARCHAR(100),
    data VARCHAR(100),
    index_size VARCHAR(100),
    unused VARCHAR(100)
)
```

```

)

DECLARE @DSql VARCHAR(MAX)
SELECT @DSql = COALESCE(@DSql, '') + '
INSERT INTO #Temp execute sp_spaceused ' + QUOTENAME(Table_Name, '''') FROM
INFORMATION_SCHEMA.TABLES

EXECUTE (@DSql)

SELECT * FROM #Temp ORDER BY TableName

```

Result:

TableName	ROWS	reserved	Data	index_size	unused
Category	2	16 KB	8 KB	8 KB	0 KB
Product	8	32 KB	24 KB	8 KB	0 KB

Messages:

(2 row(s) affected)

Description:

It displays sizes of all tables under the current working database.

Hence, **INFORMATION_SCHEMA.TABLES** is used in the query to extract the information.

2.40: List sizes of all database files

Query:

```

CREATE TABLE #Temp
(
    DBName SYSNAME,
    FILENAME VARCHAR(MAX),
    PhysicalName VARCHAR(MAX),
    Size DECIMAL(12,2)
)

```

```

DECLARE @DSql NVARCHAR(MAX)
SET @DSql = ''
SELECT @DSql = @DSql + 'USE' + QUOTENAME(name) + '
                INSERT INTO #Temp
                SELECT ' + QUOTENAME(name, '''') + ', Name,
                Physical_Name, size/1024.0 from sys.database_files'

FROM sys.databases
EXECUTE (@DSql)

SELECT * FROM #Temp ORDER BY DBName, FileName

DROP TABLE #Temp

```

Result:

DBName	FILENAME	PhysicalName	Size
master	master	C:\Program Files\Microsoft SQL Server\MSSQL10_50.SQL2008\MSSQL\DATA\master.mdf	0.5
master	mastlog	C:\Program Files\Microsoft SQL Server\MSSQL10_50.SQL2008\MSSQL\DATA\mastlog.ldf	0.13
model	modeldev	C:\Program Files\Microsoft SQL Server\MSSQL10_50.SQL2008\MSSQL\DATA\model.mdf	0.16
model	modellog	C:\Program Files\Microsoft SQL Server\MSSQL10_50.SQL2008\MSSQL\DATA\modellog.ldf	0.06
msdb	MSDBData	C:\Program Files\Microsoft SQL Server\MSSQL10_50.SQL2008\MSSQL\DATA\MSDBData.mdf	1.84
msdb	MSDBLog	C:\Program Files\Microsoft SQL Server\MSSQL10_50.SQL2008\MSSQL\DATA\MSDBLog.ldf	0.77
ProductDb	ProductDb	E:\Webinar\SqlBook\ProductDb.mdf	0.25
ProductDb	ProductDb_log	E:\Webinar\SqlBook\ProductDb_log.ldf	0.13
ReportServer	ReportServer	C:\Program Files\Microsoft SQL Server\MSSQL10_50.SQL2008\MSSQL\DATA\ReportServer.mdf	0.53
ReportServer	ReportServer_log	C:\Program Files\Microsoft SQL Server\MSSQL10_50.SQL2008\MSSQL\DATA\ReportServer_log.LDF	0.09

ReportServerTempDB	ReportServerTempDB	C:\Program Files\Microsoft SQL Server\MSSQL10_50.SQL2008\MSSQL\DATA\ReportServerTempDB.mdf	0.28
ReportServerTempDB	ReportServerTempDB_log	C:\Program Files\Microsoft SQL Server\MSSQL10_50.SQL2008\MSSQL\DATA\ReportServerTempDB_log.LDF	0.09
tempdb	tempdev	C:\Program Files\Microsoft SQL Server\MSSQL10_50.SQL2008\MSSQL\DATA\tempdb.mdf	1
tempdb	templog	C:\Program Files\Microsoft SQL Server\MSSQL10_50.SQL2008\MSSQL\DATA\templog.ldf	0.09

Messages:

(14 row(s) affected)

Description:

It lists out the sizes of all database files.

These files have extension (.ldf) and can be found at following location

C:\Program Files\....

2.41: Generate xsd of a table without data

Query:

```
DECLARE @myXsd XML
```

```
SET @myXsd = (SELECT TOP 0 * FROM Category FOR XML AUTO, ELEMENTS,
XMLSCHEMA('myTargetNamespace'))
```

```
SET @myXsd.modify('delete /xs:schema[1]/@targetNamespace')
```

```
SELECT @myXsd
```

Result:

```
<xsd:schema xmlns:schema="myTargetNamespace" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:sqltypes="http://schemas.microsoft.com/sqlserver/2004/sqltypes"
elementFormDefault="qualified">
```

```

<xsd:import namespace="http://schemas.microsoft.com/sqlserver/2004/sqltypes"
schemaLocation="http://schemas.microsoft.com/sqlserver/2004/sqltypes/sqltypes.xsd"
/>
<xsd:element name="Category">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="CategoryId" type="sqltypes:bigint" />
      <xsd:element name="CategoryName" minOccurs="0">
        <xsd:simpleType>
          <xsd:restriction base="sqltypes:varchar" sqltypes:localeId="1033"
sqltypes:sqlCompareOptions="IgnoreCase IgnoreKanaType IgnoreWidth"
sqltypes:sqlSortId="52">
            <xsd:maxLength value="50" />
          </xsd:restriction>
        </xsd:simpleType>
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
</xsd:schema>

```

Messages:

(1 row(s) affected)

Description:

2.42: Generate xsd of a table with data

Query:

```
DECLARE @XsdSchema XML
```

```
SET @XsdSchema = (SELECT * FROM Category FOR XML AUTO, ELEMENTS,
XMLSCHEMA('TestXsdSchema'))
```

```
SELECT @XsdSchema
```

Result:

```

<xsd:schema xmlns:schema="TestXsdSchema" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:sqltypes="http://schemas.microsoft.com/sqlserver/2004/sqltypes"
targetNamespace="TestXsdSchema" elementFormDefault="qualified">

```

```

    <xsd:import namespace="http://schemas.microsoft.com/sqlserver/2004/sqltypes"
      schemaLocation="http://schemas.microsoft.com/sqlserver/2004/sqltypes/sqltypes.xsd"
    />
  <xsd:element name="Category">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="CategoryId" type="sqltypes:bigint" />
        <xsd:element name="CategoryName" minOccurs="0">
          <xsd:simpleType>
            <xsd:restriction base="sqltypes:varchar" sqltypes:localeId="1033"
              sqltypes:sqlCompareOptions="IgnoreCase IgnoreKanaType IgnoreWidth"
              sqltypes:sqlSortId="52">
              <xsd:maxLength value="50" />
            </xsd:restriction>
          </xsd:simpleType>
        </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
<Category xmlns="TestXsdSchema">
  <CategoryId>1</CategoryId>
  <CategoryName>Stationery</CategoryName>
</Category>
<Category xmlns="TestXsdSchema">
  <CategoryId>2</CategoryId>
  <CategoryName>Computer</CategoryName>
</Category>

```

Messages:

(1 row(s) affected)