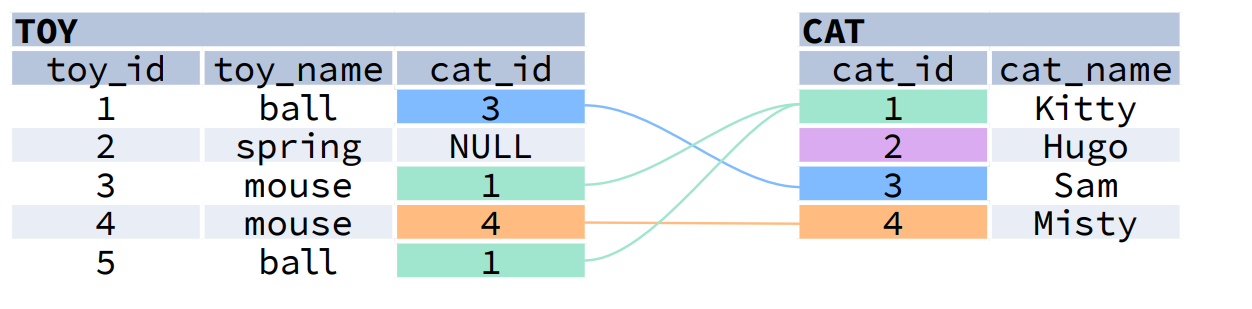
**Joins in SQL server.**

Joins combines data from two or more tables.



JOIN typically combines rows with equal values for the specified columns.

Usually, one table contains a primary key, which is a column or columns that uniquely identify rows in the table (the cat\_id column in the cat table).

The other table has a column or columns that refer to the primary key columns in the first table (the cat\_id column in the toy table). Such columns are foreign keys.

The JOIN condition is the equality between the primary key columns in one table and columns referring to them in the other table.

**JOIN CONDITIONS**

The JOIN condition doesn't have to be an equality – it can be any condition you want. JOIN doesn't interpret the JOIN condition, it only checks if the rows satisfy the given condition.

To refer to a column in the JOIN query, you have to use the full column name: first the table name, then a dot (.) and the column name:

ON cat. Cat\_id = toy. Cat\_id

You can omit the table name and use just the column name if the name of the column is unique within all columns in the joined tables.

**JOIN/INNERJOIN**

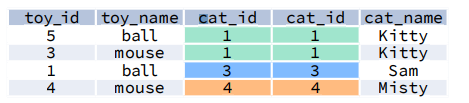
JOIN returns all rows that match the ON condition. JOIN is also called INNER JOIN

SELECT \*

FROM toy

JOIN cat

ON toy.cat\_id = cat.cat\_id;



There is also another, older syntax, but it isn't recommended. List joined tables in the FROM clause, and place the conditions in the WHERE clause.

SELECT \*

FROM toy, cat

WHERE toy.cat\_id = cat.cat\_id;

**LEFT JOIN/ LEFT OUTER JOIN**

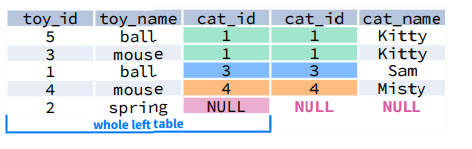
LEFT JOIN returns all rows from the left table with matching rows from the right table. Rows without a match are filled with NULLs. LEFT JOIN is also called LEFT OUTER JOIN.

SELECT \*

FROM toy

LEFT JOIN cat

ON toy.cat\_id = cat.cat\_id;



**RIGHT JOIN / RIGHT OUTER JOIN**

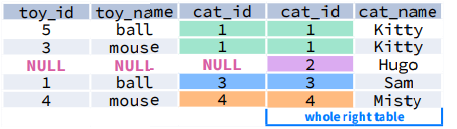
RIGHT JOIN returns all rows from the right table with matching rows from the left table. Rows without a match are filled with NULLs. RIGHT JOIN is also called RIGHT OUTER JOIN.

SELECT \*

FROM toy

RIGHT JOIN cat

ON toy.cat\_id = cat.cat\_id;



**FULL JOIN/ FULL OUTER JOIN**

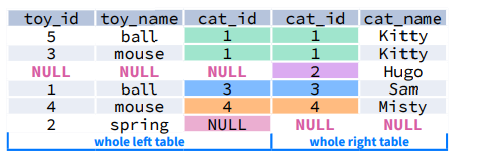
FULL JOIN returns all rows from the left table and all rows from the right table. It fills the non-matching rows with NULLs. FULL JOIN is also called FULL OUTER JOIN.

SELECT \*

FROM toy

FULL JOIN cat

ON toy.cat\_id = cat.cat\_id;



**CROSS JOIN**

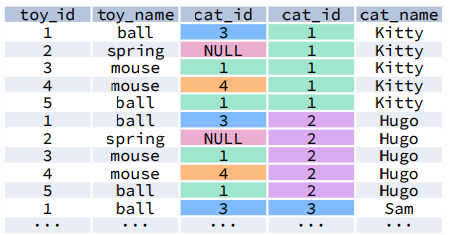
CROSS JOIN returns all possible combinations of rows from the left and right tables.

SELECT \*

FROM toy

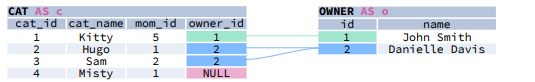
CROSS JOIN cat;

Other syntax

SELECT \* FROM toy, cat;  
  


**SELF JOIN**

You can join a table to itself, for example, to show a parent-child relationship.

This is useful when you have a table with hierarchical or parent-child data relationships, and you want to retrieve information about related records within the same table. To perform a self-join, you use table aliases to distinguish between the two instances of the same table. Here's a basic example of how to perform a self-join:  
  


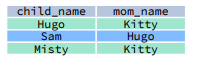
A column alias renames a column in the result. A table alias renames a table within the query. If you define a table alias, you must use it instead of the table name everywhere in the query. The AS keyword is optional in defining aliases.  
  
SELECT

child.cat\_name AS child\_name,

mom.cat\_name AS mom\_name

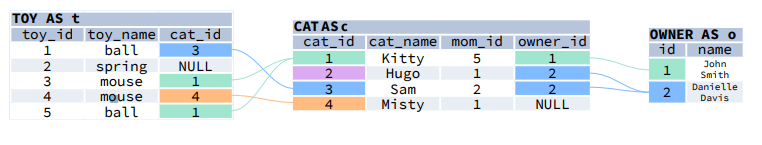
FROM cat AS child   
JOIN cat AS mom

ON child.mom\_id = mom.cat\_id

Here is the output of the above self-join query  
  


**MULTIPLE JOINS**

You can join more than two tables together. First, two tables are joined, then the third table is joined to the result of the previous joining.

There are two tables and there is no direct relation between them but the two-table having relation with another table in that case we can use multiple join   
  


Let us do some multiple join and their query result tables for the above tables toy, cat and owner

**JOIN & JOIN**

SELECT

t.toy\_name,

c.cat\_name,

o.name AS owner\_name

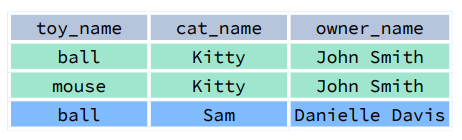
FROM toy t

JOIN cat c

ON t.cat\_id = c.cat\_id

JOIN owner o

ON c.owner\_id = o.id;



**JOIN AND LEFT JOIN**

SELECT

t.toy\_name,

c.cat\_name,

o.name AS owner\_name

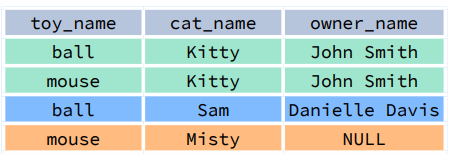
FROM toy t

JOIN cat c

ON t.cat\_id = c.cat\_id

LEFT JOIN owner o

ON c.owner\_id = o.id;



**LEFT JOIN AND LEFT JOIN**

SELECT

t.toy\_name,

c.cat\_name,

o.name AS owner\_name

FROM toy t

LEFT JOIN cat c

ON t.cat\_id = c.cat\_id

LEFT JOIN owner o

ON c.owner\_id = o.id;



**Functions in SQL server**

In SQL Server, a function is a stored program that can accept parameters and return a value. Functions are pre-built actions that perform calculations, manipulate data, and return results. They can simplify complex queries and automate repetitive tasks.

There are two types of functions in sql server, there are:

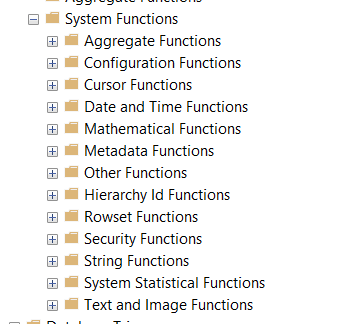
**Built in functions**

**User defined functions**

Here are some of the most used system defined functions in sql server

* String functions
* Date and time functions
* Aggregate functions
* Mathematical functions
* Ranking functions

When we open SSMS object explorer



User defined functions in SQL server

There are three types of user defined functions

* Scalar functions--------------returns a single value. The return value of any type except   
  text, ntext, image, cursor, and timestamp.
* Inline table-valued functions---------------returns a table value.
* Multi-statement table valued functions

create function fn\_employees

(

@date date

)

returns table

as

return (select employeeid,

firstname+' '+lastname as empname,

hiredate as date\_of\_joining,

datediff(year,birthdate, @date) as age

from Employees )

select \* from fn\_employees('2023-12-04');

**Top n clause**

It is used to retrieve the first n rows in a table below is the syntax in sql server

select \* from orders

select top 10 \* from orders

select top 10 percent \* from orders

select top 10 with ties \* from orders

order by ordered

SELECT TOP (70) PERCENT

ID, Name, EmailID, Gender, CITY, Department

FROM Employee

WHERE Gender = 'Male'

ORDER BY ID;

**Set operators in SQL server:**

Set operators are special type of operator used to combine the result set of two queries.

The following are covered under set operators.

**UNION**

**UNION ALL**

**INTERSECT**

**EXCEPT or MINUS**

**UNION AND UNION ALL**

SELECT ID, Name, Gender, Department FROM EmployeeIndia

UNION ALL

Select ID, Name, Gender, Department FROM EmployeeUK

SELECT ID, Name, Gender, Department FROM EmployeeIndia

UNION

Select ID, Name, Gender, Department FROM EmployeeUK

**Group by and having clauses**

The group by clause is used to group rows in a table based on one or more then one column in a table

It is used for grouping data

Select

name,

city

from customers

Group by city

Aggregate functons

SELECT category,

SUM(sales\_amount) AS total\_sales

FROM sales\_data

GROUP BY category;

Group by and having

SELECT

category,

AVG(price) AS average\_price

FROM products

GROUP BY category

HAVING AVG(price) > 50;

**Ranking functions**

**Row number**

The ROW\_NUMBER() is a built-in function in SQL Server that assigns a sequential integer number to each row within a partition of a result set. The row number always starts with 1 for the first row in each partition and then increases by 1 for the next row onwards in each partition. The syntax to use the ROW\_NUMBER function is given below.

Rownumber() over (partition by <value> order by <value>)

select

firstname ,

gender,

emailaddress,

totalchildren

ROW\_NUMBER() over(partition by totalchildren order by firstname)

from DimCustomer

**Rank function in SQL server:**

In SQL Server, the RANK() function is a window function used to assign a unique rank to each distinct row within a partition of a result set. It calculates the rank of rows within the partition and leaves gaps in the ranking when there are ties (i.e., rows with the same values).

select

firstname ,

gender,

emailaddress,

totalchildren,

RANK() over(partition by totalchildren order by firstname)

from DimCustomer

**Dense rank in sql server**

In SQL Server, DENSE\_RANK() is a window function used to assign a unique rank value to each distinct row within a partition of a result set. It calculates the rank of rows within the partition, but unlike RANK(), it does not leave gaps when there are ties (i.e., rows with the same values).

select

firstname ,

gender,

emailaddress,

totalchildren,

DENSE\_RANK() over(partition by totalchildren order by firstname)

from DimCustomer

**CTE (common table expression):**

WITH

cteReports (EmpID, FirstName, LastName, MgrID, EmpLevel)

AS

(

SELECT EmployeeID, FirstName, LastName, ManagerID, 1

FROM Employees

WHERE ManagerID IS NULL

UNION ALL

SELECT e.EmployeeID, e.FirstName, e.LastName, e.ManagerID,

r.EmpLevel + 1

FROM Employees e

INNER JOIN cteReports r

ON e.ManagerID = r.EmpID

)

Select \* from cteReports.

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Practice some scenarios on the NORTHWIND DATABSE**

To install the northwind database in your local run the following script in SSMS.

Steps to install.

1. Open SSMS and login to the corresponding server
2. Open the git hub link [https://github.com/kirankumargunna/north-wind-data-base-](https://github.com/kirankumargunna/north-wind-data-base-%20) Copy the script in INSTNWD.SQL.
3. Create a database northwind sample database and run the script in query editor in the database.
4. Then the northwind database created

***Sql assignment questions beginner level***

**1. How many customers do we have in our database?**

**2. How many of our customer names begin with the letter "b"?**

**3. How many of our customer names contain the letter "s" ?**

**4. How many customers do we have in each city?**

**5. What are the top three cities where we have our most customers?**

**6. Who has been our top customer - please list CustomerID, CompanyName, ContactName for the customer that we have sold the most to?**

**7. Who has been our top customer - please list CustomerID, CompanyName in the year 1997?**

**8. Name the top 3 countries that we ship our orders to?**

**9. Which shipper do we use the most to ship our orders out through?**

**10. List the following employee information (EmployeeID, LastName, FirstName)**

**11. What are the last names of all employees who were born in the month of November?**

**12. List each employee (lastname, firstname, territory) and sort the list by territory and then by employee last name. Remember employees may work for more than one territory.**

**13. In terms of sales value, what has been our best selling product of all time?**

**14. In terms of sales value, and only include products that have at least been sold once, which has been our worst selling product of all time?**

**15. In terms of sales value, which month has been traditionally best for sales?**

**16. What is the name of our best sales person?**

**17. Product report (productID, ProductName, Supplier Name, Product Category). Order the list by product category.**

**Scenario1**:

You are working for a marketing team and need to segment customers based on their historical buying behaviour. Create SQL queries to categorize customers into different segments, such as high-value, frequent buyers, or lapsed customers, using data from the Northwind database.

**Scenario2:**

As a procurement manager, you want to evaluate the performance of your suppliers. Write a query to calculate metrics such as on-time delivery based on historical order data in the Northwind database.