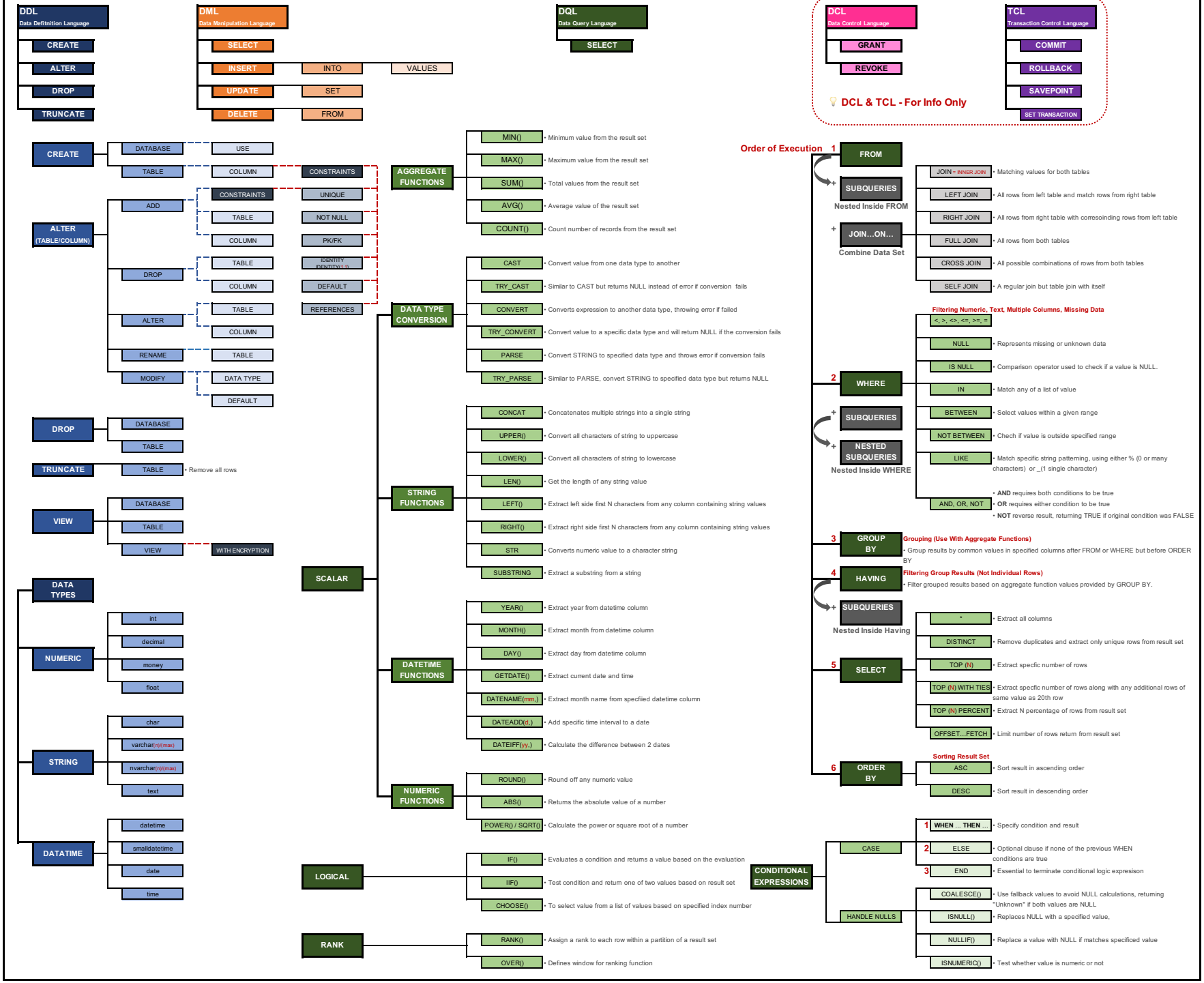
Querying Data withMicrosoft Transact-SQL



Data Definition Language (DDL) Commands

Command	Description	Syntax	Example
CREATE	Creates a new data base and objects, such as a table, index, view, or stored procedure.	CREATE TABLE table_name (column1 datatype1, column2 datatype2);	CREATE TABLE employees (employee_id INT PRIMARYKEY, first_name VARCHAR(50), last_name VARCHAR(50), age INT);
ALTER	Adds, deletes, or modifies columns in an existing table.	ALTER TABLE table_name ADD column_named atatype;	ALTER TABLE customers ADD email VARCHAR(100);
DROP	Drop an existing table in a database	DROP TABLE table_name;	DROP TABLE customers;
TRUNCATE	Delete the data inside a table, but not the table itself.	TRUNCATE TABLE table_name;	TRUNCATE TABLE customers;

Data Manipulation Language (DML) Commands

Command	Description	Syntax	Example
SELECT	Retrieves data from a database	SELECT column1, column2 FROM table_name;	SELECT first_name, last_name FROM customers;
INSERT	Adds new records to a table.	INSERT INTO table_name (column1, column2) VALUES (value1, value2);	INSERT INTO customers (first_name, last_name) VALUES ('Mary', 'Doe');
UPDATE	Modify existing records in a table.	UPDATE table_name SET column1 = value1, column2 = value2 WHERE condition;	UPDATE employees SET employee_name = 'John Doe', department = 'Marketing';
DELETE	Removes records from a table.	DELETE FROM table_name WHERE condition;	DELETE FROM employees WHERE employee_name = 'John Doe';

Data Query Language (DQL) Commands

Querying Data

Command	Description	Syntax	Example
SELECT	Retrieve data from a database.	SELECT column1, column2 FROM table_name;	SELECT first_name, last_name FROM customers;
WHERE	Filter rows based on a specified condition.	SELECT * FROM table_name WHERE condition;	SELECT * FROM customers WHERE age > 30;
ORDER BY	Sort the result set in ascending or descending order based on a specified column.	SELECT * FROM table_name ORDER BY column_name ASC|DESC;	SELECT * FROM products ORDER BY price DESC;
GROUP BY	Groups rows based on the values in a specified column. It is often used with aggregate functions like COUNT, SUM, AVG, etc.	SELECT column_name, COUNT(*) FROM table_name GROUP BY column_name;	SELECT category, COUNT(*) --counts rows in each category FROM products GROUP BY category;
HAVING	Filters grouped results based on a specified condition.	SELECT column_name, COUNT(*) FROM table_name GROUP BY column_name HAVING condition;	SELECT category, COUNT(*) --counts rows in each category FROM products GROUP BY category HAVING COUNT(*) > 5;

Join Data

Command	Description	Syntax	Example
JOIN **(INNER JOIN)**	Returns rows with matching values in both tables.	SELECT * FROM table1 JOIN table2 ON table1.column = table2.column;	SELECT * FROM employees JOIN departments ON employees.department_id = departments.id;
LEFT JOIN **(LEFT OUTER JOIN)**	Returns all rows from the left table (first table) and the matching rows from the right table (second table).	SELECT * FROM table1 LEFT JOIN table2 ON table1.column = table2.column;	SELECT * FROM employees LEFT JOIN departments ON employees.department_id = departments.id;
RIGHT JOIN **(RIGHT OUTER JOIN)**	Returns all rows from the right table (second table) and the matching rows from the left table (first table).	SELECT * FROM table1 RIGHT JOIN table2 ON table1.column = table2.column;	SELECT * FROM employees RIGHT JOIN departments ON employees.department_id = departments.id;
FULL JOIN **(FULL OUTER JOIN)**	Returns all rows when there is a match in either the left table or the right table.	SELECT * FROM table1 FULL JOIN table2 ON table1.column = table2.column;	SELECT * FROM employees FULL JOIN departments ON employees.department_id = departments.id;
CROSS JOIN	Combines every row from the first table with every row from the second table, creating a Cartesian product.	SELECT * FROM table1 CROSS JOIN table2;	SELECT * FROM employees CROSS JOIN departments;
SELF JOIN	Joins a table with itself.	SELECT * FROM table1 t1, table1 t2 WHERE t1.column = t2.column;	SELECT * FROM employees t1, employees t2 WHERE t1.employee_id = t2.employee_id;

Subqueries

Command	Description	Syntax	Example
IN	Determine whether a value matches any value in a subquery result. It is often used in the WHERE clause.	SELECT column(s) FROM table WHERE value IN (subquery);	SELECT CustomerID, SalesOrderID FROM Sales.SalesOrderHeader WHERE CustomerID IN (SELECT CustomerID FROM Sales.Customer WHERE CountryRegion = 'Canada');
ANY	Compare a value to any value returned by a subquery. It can be used with comparison operators like =, >, <, etc.	SELECT column(s) FROM table WHERE value < ANY (subquery);	SELECT SalesOrderID, ProductID, OrderQty FROM Sales.SalesOrderDetail WHERE SalesOrderID = (SELECT MAX(SalesOrderID) FROM Sales.SalesOrderHeader);
ALL	Compare a value to all values returned by a subquery. It can be used with comparison operators like =, >, <, etc.	SELECT column(s) FROM table WHERE value > ALL (subquery);	SELECT * FROM orders WHERE order_amount > ALL (SELECT total_amount FROM previous_orders);

Aggregate Functions

Command	Description	Syntax	Example
COUNT()	Counts the number of rows or non-null values in a specified column.	SELECT COUNT(column_name) FROM table_name;	SELECT COUNT(age) FROM employees;
SUM()	Calculate the sum of all values in a specified column.	SELECT SUM(column_name) FROM table_name;	SELECT SUM(revenue) FROM sales;
AVG()	Calculate the average (mean) of all values in a specified column.	SELECT AVG(column_name) FROM table_name;	SELECT AVG(price) FROM products;
MIN()	Returns the minimum (lowest) value in a specified column.	SELECT MIN(column_name) FROM table_name;	SELECT MIN(price) FROM products;
MAX()	Returns the maximum (highest) value in a specified column.	SELECT MAX(column_name) FROM table_name;	SELECT MAX(SalesOrderID) FROM Sales.SalesOrderHeader

String Functions

Command	Description	Syntax	Example
CONCAT()	Concatenates two or more strings into a single string.	SELECT CONCAT(string1, string) AS concatenated_string FROM table_name;	SELECT CONCAT(first_name, ' ', last_name) AS full_name FROM employees;
UPPER()	Converts all characters in a string to uppercase.	SELECT UPPER(string) AS uppercase_string FROM table_name;	SELECT UPPER(first_name) AS uppercase_first_name FROM employees;
LOWER()	Converts all characters in a string to lowercase.	SELECT LOWER(string) AS lowercase_string FROM table_name;	SELECT LOWER(last_name) AS lowercase_last_name FROM employees;
LEN()	Return the length of a string.	SELECT LEN(string_expression) FROM table_name;	SELECT LEN('James, Bond') AS StringLength; FROM employees;
LEFT()	Returns a specified number of characters from the left of a string.	SELECT LEFT(string, num_characters) AS left_string FROM table_name;	SELECT LEFT(product_name, 5) AS left_product_name FROM products;
RIGHT()	Returns a specified number of characters from the right of a string.	SELECT RIGHT(string, num_characters) AS right_string FROM table_name;	SELECT RIGHT(order_number, 4) AS right_order_number FROM orders;
STR()	Convert a numeric value to a string.	SELECT STR(numeric_expression, length, decimal) FROM table_name;	SELECT STR(123.45, 7, 2) AS ConvertedNumber; FROM orders;
SUBSTRING()	Extracts a substring from a string.	SELECT SUBSTRING(string FROM start_position [FOR length]) AS substring FROM table_name;	SELECT SUBSTRING(product_name FROM 1 FOR 5) AS substring FROM products

DateTime Functions

Command	Description	Syntax	Example
YEAR()	Extract the year.	YEAR(date)	SELECT YEAR('2022-10-15') AS ExtractedYear;
MONTH()	Extract the month.	MONTH(date)	SELECT MONTH('2022-10-15') AS ExtractedMonth;
DAY()	Extract the day component from a date.	DAY(date)	SELECT DAY('2022-10-15') AS ExtractedDay;
GETDATE()	Returns the current date and time.	SELECT GETDATE() AS current_datetime;	
DATENAME(mm,)	Return a specified part of a date as a character string.	DATENAME(datepart, date)	SELECT DATENAME(mm, '2022-10-15') AS MonthName;
DATEADD(d,) **DATESUB(d,)**	Adds or subtracts a specified number of days, months, or years to/from a date.	SELECT DATE_ADD(date_expression, INTERVAL value unit) AS new_date;	= DATE_ADD Example SELECT DATE_ADD('2024-04-11', INTERVAL 1 DAY) AS = DATE_SUB Example SELECT DATE_SUB('2024-04-11', INTERVAL 1 DAY) AS
DATEIFF(yy,)	Calculates the difference in days between two dates.	SELECT DATEDIFF(date1, date2) AS difference_in_days;	SELECT DATEDIFF('2024-04-11', '2024-04-10') AS difference_in_days;

Logical Functions

Command	Description	Syntax	Example
IF()	Evaluates a condition and returns a value based on the evaluation.	SELECT IF(condition, true_value, false_value) AS alias FROM table_name;	SELECT name, age, IF(age > 50, 'Senior', 'Junior') AS employee_category FROM employees;
IIF()	A shorthand way of writing a CASE statement with a simple expression.	IIF(condition, true_value, false_value)	SELECT AddressType, UseAddressFor FROM Sales.CustomerAddress;
CHOOSE()	Select value from a list of values based on specified index	SELECT column1, column2, CHOOSE(index, val1, val2, val3, ...) FROM table_name;	SELECT SalesOrderID, Status, CHOOSE(Status, 'Ordered', 'Shipped', 'Delivered') AS OrderStatus FROM Sales.SalesOrderHeader;

Rank Functions

Command	Description	Syntax	Example
RANK()	Assign a rank to each row within a partition of a result set.	RANK() OVER (PARTITION BY column_name ORDER BY column_name)	SELECT TOP 100 ProductID, Name, ListPrice, RANK() OVER(ORDER BY ListPrice DESC) AS RankByPrice FROM Production.Product AS p ORDER BY RankByPrice;
OVER()	Defines a window or a set of rows within a query result set to which the ranking function is applied.	RANK() OVER (PARTITION BY column_name ORDER BY column_name)	SELECT c.Name AS Category, p.Name AS Product, ListPrice, RANK() OVER(PARTITION BY c.Name ORDER BY ListPrice DESC) AS RankByPrice FROM Production.Product AS p JOIN Production.ProductCategory AS c ON p.ProductCategoryID = c.ProductcategoryID ORDER BY Category, RankByPrice;

Conditional Expression

Command	Description	Syntax	Example
CASE	Allows you to perform conditional logic within a query.	SELECT column1, column2, CASE WHEN conditional1 THEN result1 WHEN conditional2 THEN result2 ELSE default result END AS alias FROM table_name;	SELECT order_id, total_amount, CASE WHEN total_amount > 1000 THEN 'High Value Order' WHEN total_amount > 500 THEN 'Medium Value Order' ELSE 'Low Value Order' END AS order_status FROM orders;
COALESCE()	Returns the first non-null value from a list of values.	SELECT COALESCE(value1, value2) AS alias FROM table_name;	SELECT COALESCE(first_name, middle_name) AS alias FROM employees;
IS NULL()	Returns null if two specified expressions are equal.	SELECT NULLIF(expression1, expression2) AS alias FROM table_name;	SELECT NULLIF(total_amount, discounted_amount) AS diff_amount FROM orders;
NULLIF()	Returns null if two specified expressions are equal.	SELECT NULLIF(expression1, expression2) AS alias FROM table_name;	SELECT NULLIF(total_amount, discounted_amount) AS diff_amount FROM orders;
ISNUMERIC()	Determine whether an expression can be evaluated as a numeric value.	ISNUMERIC(expression)	SELECT ISNUMERIC('123') AS NumericCheck;