SAVVY SQL CHART

Author

Harshal Chaudhari

A Compilation of WorkShop book by Industry Expert.

ACKNOWLEDGMENTS

It is with our deepest gratitude and warmest affection that we indebted to

Dr. Subhash Chaudhary.
Mr. C. G. Patil.
Dr. Shirish Jadhav.
Mr. Omprakash Mishra (Symantec)
Prof. Pankaj Bhangale.
(H.O.D.(Civil) at SSGB Engineering college)
Dr. Chetan Chaudhari.
(Director of Sinhgad Management Institute)

Who has been a constant source of knowledge and inspiration.

INTRODUCTION

Welcome to Savvy SQL Chart \\WorkShopBook We think your time is too valuable to spend struggling with new concepts.

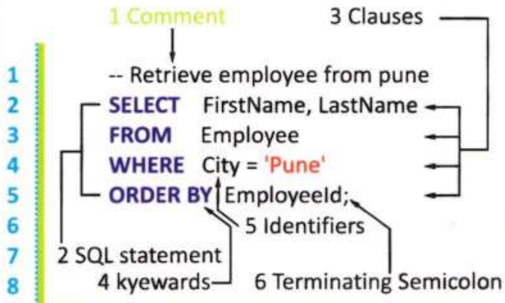
Using the latest research in cognitive science and learning theory to craft a multi-sensory learning experience, \\WorkShopBook use rich format designed for the way your Brain works, not a text heavy approach that put you to sleep.

Practical and precise, Savvy SQL Chart \\WorkShopBook shows you just what you need to know to stay competitive in a shifting marketplace.

Learn SQL Savvy Way!

TABLE OF CONTENTS

SQL Basic Syntax



```
-- Retrieve employee from pune
SELECT   FirstName, LastName
FROM     Employee
WHERE    City = 'Pune'
ORDER BY EmployeeId;
```

Note : SQL is not cause sensitive language

Each column in a table has a single Data Type

Category	Type
String	char, varchar, text, nchar, nvarchar, ntext
Numeric	bigint, int, smallint, tinyint decimal, numeric, float, real
Boolean	bit
DateTime	datetime, smalldatetime
UniqueIdentifiers	uniquedentifier, identity
BLOB	binary, varbinary, image

Example of Literals

STRING	'LastName', 'Is', '30', 'Savvy'
NUMERIC	1, 2, 3, ...-24, -105
BOOLEAN	True, False, NULL
DATATIME	'10/06/1986';'19800130'
UNIQUE IDENTIFIERS	{MIKKAd24-89bc-self7 jj7e-7bb3db3809bc}

Apply It

Retrieving columns with **SELECT** and **FROM**
The **SELECT** clause lists the columns to display.
The **FROM** clause

Syntax

```
SELECT columns
FROM table;
```

BASIC EXAMPLE

```
SELECT city
FROM Employee;
```

Extra

To retrieve all columns from a table

```
SELECT *
FROM Employee;
```

Apply It

Creating column aliases with **AS**
As clause to create a column alias
A column alias is an alternative name.

Syntax

```
SELECT column [AS] AliasName
FROM table;
```

BASIC EXAMPLE

syntactic variations of the AS clause

```
SELECT    FirstName AS 'Student Name',
          LastName AS "SurName",
          EmailID AS [Email]
FROM Employee;
```

Apply It

Eliminating Duplicate rows with **DISTINCT**
A result that list each duplicate only once.

Syntax

```
SELECT DISTINCT column
FROM table;
```

BASIC EXAMPLE

```
SELECT DISTINCT  city
FROM Employee;
```

Apply It

Sorting rows with **ORDER BY**
ORDER BY clause to sort rows by a specified column or columns in ascending (lowest to highest) or descending (highest to lowest)

Syntax

```
SELECT column
FROM table
ORDER BY sortColumn [ASC|DESC];
```

BASIC EXAMPLE

```
SELECT  FirstName, LastName
FROM Employee
ORDER BY LastName ASC;
```

TIPS The **ORDER BY** clause always the last clause in a select statement

Example

```
SELECT  FirstName, City, Zip
FROM  Employee
ORDER BY
   CASE WHEN City = 'Pune' THEN Zip
   ELSE City END
DESC;
```

Apply It

Filtering rows with WHERE
WHERE clause to filter unwanted rows from the result.

Syntax

```
SELECT columns
FROM table
WHERE column operator value;
```

BASIC EXAMPLE

```
SELECT  FirstName, LastName
FROM Employee
WHERE Mobile = '9403019549';
```

TIPS

Place the **WHERE** clause before the **ORDER** BY clause, in a **SELECT** statement in which both appear.

Types of Conditions

Condition	SQL OPERATORS
Comparison	=, < >, <, < =, >, > =, ! =
Pattern matching	LIKE
Range filtering	BETWEEN
List filtering	IN
NULL testing	IS NULL

Apply It

Combining and Negating Conditions with **AND**, **OR** and **NOT**
The AND operator connects two conditions and returns true only if both conditions are true.

Syntax

```
SELECT columns
FROM table
WHERE Condition1 AND Condition2;
```

BASIC EXAMPLE

```
SELECT FirstName, Salary
FROM Employee
WHERE Salary > 15000 AND Salary < 60000;
```

AND table

Condition1	Condition2	Result
True	True	True
True	False	False
False	True	False
False	False	False

Apply It

The **OR** operator connects two conditions and returns true if either condition is true or if both conditions are true.

Syntax

```
SELECT columns
FROM table
WHERE Condition1 OR Condition2
```

BASIC EXAMPLE

```
SELECT FirstName, City
FROM Employee
WHERE City = 'Pune'  OR  City = 'Mumbai';
```

OR table

Condition1	Condition2	Result
True	True	True
True	False	True
False	True	True
False	False	False

Apply It

The **NOT** operator negates (reverses) a Single Condition.

Syntax

```
SELECT columns
FROM table
WHERE NOT (Condition)
```

BASIC EXAMPLE

```
SELECT FirstName, City
FROM Employee
WHERE NOT (City = 'Bangalore');
```

NOT table

Condition1	NOT Result
True	False
False	True

Apply It

Matching pattern with **LIKE**
LIKE to retrieve rows based on partial information. **LIKE** is useful if you don't know an exact value like works with only string.

Syntax

```
SELECT columns
FROM table
WHERE column [NOT] LIKE 'Pattern';
```

BASIC EXAMPLE

```
SELECT FirstName
FROM Employee
WHERE FirstName LIKE '_ a%';
```

Wildcard Operators

Operator	Matches
%	a percent sign matches any string of Zero or more character
_	An underscore matches any one character

Example of % and _ Patterns

PATTERN	**MATCHES**
'H%'	Matches a strings of length >,1 that begins with A
'%L'	Matches a strings of length >,1 that ends with L
'%in%'	Matches a strings of length >,2 that contains in anywhere
'_in_'	Matches any four character string that has in as its second & third characters
'[L-S]et'	Matches Let, Net, Set but not vet

Apply It

Range filtering with **BETWEEN** to determine whether a given value falls with a specified range.

Syntax

```
SELECT column
FROM table
WHERE [NOT] column BETWEEN LowValue
                         AND HighValue;
```

BASIC EXAMPLE

```
SELECT  FirstName, Salary
FROM Employee
WHERE Salary BETWEEN 15000
                 AND 60000;
```

Apply It

List filtering with **IN** to determine whether a given value matches any value in a specified list.

Syntax

```
SELECT columns
FROM table
WHERE  column [NOT] IN
          (Value1, Value2);
```

BASIC EXAMPLE

```
SELECT  FirstName, [State]
FROM Employee
WHERE [State] IN ('MH', 'GH');
```

Apply It

Deal with **NULL**
Null represent missing or unknown values.

Syntax

```
SELECT column
FROM table
WHERE  column IS [NOT] NULL;
```

BASIC EXAMPLE

```
SELECT  FirstName, City
FROM Employee
WHERE City  IS NULL;
```

Apply It

Performing arithmetic operations

Syntax

```
SELECT value1 + value2;
SELECT value1 - value2;
SELECT value1 * value2;
SELECT value1 / value2;
```

BASIC EXAMPLE

```
                                                  Result
SELECT  (9 + 1);                                      10
SELECT  (7 - 1);                                       6
SELECT  (2 * 3);                                       6
SELECT  (12 / 2);                                      6
SELECT  CONVERT (DECIMAL (3,1),5) /                  2.5
        CONVERT (DECIMAL (3,1),3);
```

TIPS

Its good programming style to add parentheses (even when they're unnecessary) to make code portable & easier to read.

Apply It

Concating string with +

Syntax

```
SELECT string1 + string2;
```

BASIC EXAMPLE

```
SELECT  FirstName + ' ' + LastName
FROM Employee;
```

TIPS

You can use + in **SELECT**, **WHERE** and **ORDER BY** clauses or anywhere an expression is allowed.

Apply It

Extracting a substring with **SUBSTRING ( )** to extract part of a string.

Syntax

1 **SUBSTRING** (expression, start, length)

Eg	Result
SELECT UPPER ('sql')	SQL
SELECT LOWER ('SQL')	sql
SELECT '<' + LTRIM (' sql ') + '>'	<SQL >
SELECT '<' + LTRIM (' sql ') + '>'	< SQL>
SELECT LEN ('SQL')	3
SELECT SUBSTRING ('sqlQuery', 1,4)	sqlQ
SELECT REVERSE ('SQL')	LQS
SELECT REPLACE ('SQL', 'SQL', 'savvySQL')	savvy SQL
SELECT LEFT ('savvySQL',5)	savvy
SELECT RIGHT ('savvySQL', 3)	SQL

Apply It

Getting the current Date and Time

	Function
1	GETDATE ( )
2	DATEPART (datePart, date)
3	DATEADD (datePart, number, date)
4	DATEDIFF (datePart, startDate, EndDate)
5	CONVERT (dateType (length), expression, style)

	Description
1	Return the current date and Time
2	Return the single part of a date/Time
3	Add or subtract a specified time interval from date
4	Return the time between two dates
5	Display date/Time data in different formats

Value (Century yyyy)	Input/Output	Standard
101	mm/dd/yyyy	USA
104	dd.mm.yyyy	German
110	mm-dd-yyyy	USA
111	yyyy/mm/dd	Japan

DatePart	abbreviation
year	yy, yyyy
month	mm, m
day	aa, a
week	wk, ww
hour	hh
minute	mi, n
second	ss, s

Apply It

Converting Data types with **CAST**
To convert one data type to another.

Syntax

```
SELECT CAST (expression AS dataType)
```

BASIC EXAMPLE

```
SELECT CAST (mobile AS CHAR(10))
FROM Employee;
```

Apply It

Evaluating conditional value with CASE is used to evaluate several condition and return a single value for the first true condition.

Syntax

```
CASE ComparisonValue
   WHEN   value1   THEN   result1
   WHEN   value2   THEN   result2
   WHEN   value    THEN   result
   [ELSE DefaultResult]
END;
```

BASIC EXAMPLE

```
SELECT FirstName
      CASE   Gender
        WHEN   0   THEN   'Female'
        WHEN   1   THEN   'Male'
        ELSE   'Unknown'
      END
FROM  Employee;
```

Apply It

Checking for NULLS with COALESCE ( ) is display a specific value instead of a null in a result.

Syntax

```
(expression1[, expression2, expressionN])
```

BASIC EXAMPLE

```
SELECT
FirstName, COALESCE ([state], 'N/A') AS [state]
FROM Employee;
```

Apply It

Comparing expressions with ISNULL( ) is used to convert a user define missing, unknown, or inapplicable value to null.

Syntax

```
ISNULL(expression1, expression2)
```

BASIC EXAMPLE

```
SELECT  FirstName
   ISNULL(contract, 1) AS "contract"
FROM Employee;
```

TIPS

To return a null if two expression are equivalent.
NULLIF( ) function you can use for avoiding division by zero problem.

BASIC EXAMPLE

```
SELECT  ( 51 / NULLIF(0, 0) ) AS Value;
```

Apply It Summarizing and Grouping data.

Aggregate Functions

Function	Return
MIN (column)	Minimum value in column
MAX (column)	Maximum value in column
SUM (column)	Sum of the values in column
AVG (column)	Average of the values in column
COUNT (column)	The number of non null values in column
COUNT (*)	The number of rows in a table or set.

Apply It

Grouping rows with **GROUP BY**

Syntax

```
SELECT
   columns
FROM table
[WHERE  SearchCondition]
GROUP BY
[HAVING  SearchCondition]
[ORDER BY  SortColumns];
```

BASIC EXAMPLE

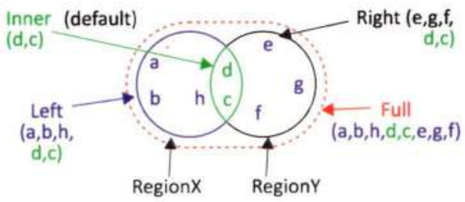
```
SELECT FirstName, City
        COUNT(*) AS "count(*)"
FROM Employee
GROUP BY  City;
```

Extra

The **GROUP BY** clause comes after the **WHERE** clause and before the **ORDER BY** clause

Apply It

A join is a table operation that uses related columns to combine rows from two input tables into one result table [you can chain join to retrieve rows from an unlimited number of tables]

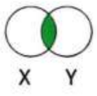


Syntax

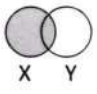
```
SELECT
FROM LeftTable;
[INNER|LEFT|FULL] join
RightTable ON condition;
```

BASIC EXAMPLE

```
SELECT  X. Point, Y. Point
FROM RegionX   X
INNER JOIN RegionY   Y
ON   X.Point = Y.Point;
```

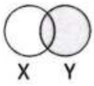


```
SELECT  X.Point
FROM RegionX   X
LEFT JOIN RegionY   Y
ON   X.Point = Y.Point;
```

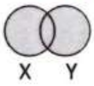


BASIC EXAMPLE

```
SELECT  Y.Point
FROM RegionX   X
RIGHT JOIN RegionY   Y
ON  X.Point = Y.Point;
```



```
SELECT  COALESCE (X.Point, Y.Point, 'N/A')
FROM RegionX   X
FULL JOIN RegionY   Y
ON  X.Point = Y.Point;
```



Extra

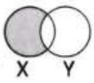
EXCEPT and **INTERSECT**
EXCEPT returns any distinct values from the left query that are not also found on the right query
INTERSECT returns any distinct values that are returned by both the query on the left & right sides of the INTERSECT Operand.

Syntax

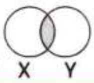
```
{ SQL Query1}
{EXCEPT | INTERSECT}
```

BASIC EXAMPLE

```
SELECT   Point   FROM   RegionX
EXCEPT
SELECT   Point   FROM   RegionY;
```



```
SELECT   Point   FROM   RegionX
INTERSECT
SELECT   Point   FROM   RegionY;
```



Apply It

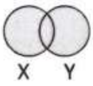
Combining Rows with **UNION** expression removes duplicate rows from the result; a **UNION ALL** expression doesn't remove duplicates.

Syntax

```
SELECT statement
UNION [ALL]
SELECT statement
```

BASIC EXAMPLE

```
SELECT  Point  FROM  RegionX
UNION
SELECT  Point  FROM  RegionY
```



```
SELECT  city   FROM  Employee
UNION ALL
SELECT  city   FROM  NewEmployee
```

Apply It

INSERTING, UPDATING And DELETING Rows
Inserting rows with **INSERT**

Syntax

To insert a row by using column positions

```
INSERT INTO table
VALUES (value1, value2, valueN);
```

To insert a row by using column names

```
INSERT INTO
(Column1, Column2, ColumnN)
VALUES (value1, value2, valueN);
```

To insert a rows from one table into another table

```
INSERT INTO
[(Column1, Column2, columnN)
SELECT Statement;
```

BASIC EXAMPLE

```
INSERT INTO Employee
VALUES
('Harry', 'Aderson',1,'ha@gmail.com',
 'Pune','MH', '6421763549', '411027',
 6000,  'Dev', 1);
```

```
INSERT INTO
(FirstName, LastName, EmailID, City, State,
Mobile, Zip, Salary)
VALUES ('Rick', 'Gate', 'rg@hotmail.com',
'Pune', 'MH', '9421763549', '411027', 9100);
```

BASIC EXAMPLE

```
INSERT INTO  Employee
SELECT
        FirstName, LastName, Gender
        ,EmailId, City, [State], Mobile,Zip
        ,Department, IsContract
FROM NewEmployee
WHERE City = 'Pune';
```

Apply It

Updating rows with UPDATE statement changes the values in a tables existing rows.

Syntax

```
UPDATE table
SET   Column = Value1, Column2 = Value2
[WHERE  SearchCondition]
```

BASIC EXAMPLE

```
UPDATE Employee
SET Salary = (Salary + 1000);
```

Apply It

Deleting rows with **DELETE** statement removes rows from a table.

Syntax

```
DELETE FROM table
[WHERE SearchConditon];
```

BASIC EXAMPLE

```
DELETE FROM NewEmployee;
```

Apply It

Creating a New Table with **CREATE TABLE**

Syntax

```
CREATE TABLE table
(
  Column1   DataType1,
  Column2   DataType2,
  ColumnN   DataTypeN
);
```

BASIC EXAMPLE

```
CRATE TABLE INTERN
( InternID      INT  NOT NULL
  FirstName     VARCHAR (15),
  LastName      VARCHAR (15),
  Degree        VARCHAR (15),
  Phone         VARCHAR (10),
  City          VARCHAR (15),
  State         VARCHAR (2),
  Zip           VARCHAR (10)
);
```

Extra

Create a temporary local table.

```
CREATE Table  #table (column dataType);
```

Create a temporary Global table

```
CREATE Table  ##table (column dataType);
```

To create a new table from an existing table.

Syntax

```
SELECT [TOP (number)]
   columns
INTO New Table
FROM Existing Table
[WHERE SearchCondition];
```

Eg. Create a new table with data

```
SELECT *
INTO EmployeeClon1
FROM Employee;
```

Create a new empty table

```
SELECT   TOP(0)  *
INTO EmployeeClon2
FROM Employee;
```

Apply It

Creating an Index with **CREATE INDEX**

Syntax

```
CREATE [UNIQUE] INDEX indexName
ON table (indexColumn);
```

BASIC EXAMPLE

```
CREATE INDEX IndexEmpSal
ON Employee (Salary);
```

Apply It

Creating a view with
CREATE VIEW

Syntax

```
CREATE VIEW view [(viewColumns)]
AS SELECT Statement;
```

BASIC EXAMPLE

```
CREATE VIEW EmployeeView (EmpName)
AS
SELECT
   FirstName + ' ' + LastName
FROM  Employee;
```

Extra

Retrieving data through a view

```
SELECT *
FROM EmployeeView;
```

Apply It

Creating stored **Procedure**

Syntax

```
CREATE {PROC I PROCEDURE} ProcName
[ Parameter AS dataType [= DefaultValue] ]
AS
[BEGIN]
        sql statements;
[END]
GO
```

BASIC EXAMPLE

```
CREATE PROC GetEmployee
(@EmployeeID   AS INT )
 AS
BEGIN
       SELECT *  FROM Employee
       WHERE EmployeeID=@EmployeeID;
END
GO

```

Execute stored procedure using **EXECUTE** clause

```
EXECUTE GetEmployee  2
```

Apply It

Create user defined **function**

Syntax

```
CREATE FUNCTION FunctionName
({@parameterName [AS] DataType})
RETURNS  returnDataType
BEGIN
     RETURN ScolarExpression
END
```

Use used define function

```
SELECT  dbo.SumofSalary ('PUNE')
```

BASIC EXAMPLE

```
CREATE FUNCTION dbo.SumOfSalary
(@City AS VARCHAR (10))
RETURNS DECIMAL (8,2)
AS
  BEGIN
    DECLARE @SumSalary AS DECIMAL (8,2);
    SELECT @SumSalary = sum (Salary)
    FROM dbo.Employee
    WHERE  City = @City;

    RETURN  @SumSalary;
END
GO
```

IMP SQLS

✓ Get a list of database

```
SELECT Name
FROM Sys.Objects;
```

✓ Get a list of table in a database

```
SELECT * FROM  Sys.Objects
WHERE type = 'U';
```

✓ Get a list of Stored Proc in database

```
SELECT Name, type
FROM dbo.Sys.Objects
WHERE TYPE  = 'P';
```

✓ Get a list of columns in a table

```
SELECT C.Name FROM Sys.Columns C
WHERE
C.Object_Id = Object_Id('Employee') ;

-- Alternate way
EXEC SP_Columns Employee

```

Helpful Database engine stored proc.

ProName	description
SP_help	Reports information about database object
SP_helprotect	Reports that has information about user permission
SP_depends	Displays object dependencies
SP_helptext	displays user define rute Eg. proc, function
SP_helpServer	Report information about server
SP_helpdb	Reports information about database
SP_helpfile	Return physical FileName
SP_ExecuteSql	Execute a Transact SQL Statement

Helpful Shortcut for SQL Management Studio

Purpose	Shortcuts
Run SQL	F5
Cancel the Executing Query	Alt + Break
Help	Alt + F1
Comment	Ctrl + K + C
Uncomment	Ctrl + K + U