Database Training Microsoft SQL Server (Day 2)

Authorized & published by Summitworks Technologies Inc

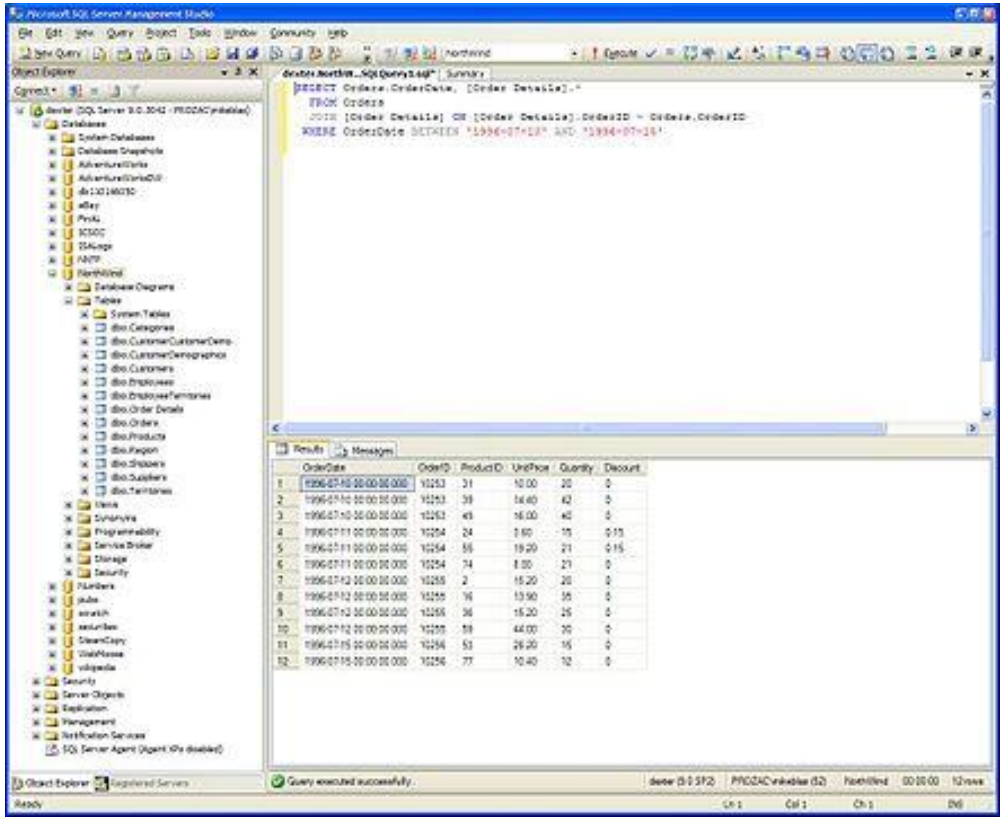
SummitWorks™
GLOBAL SOLUTION ARCHITECTS

Agenda Day 2

- Introduction to various SQL server tools and SQL Server Management Studio
- Keys and Database Constraints
- Introduction to T-SQL
- DDL Queries (Create, Alter, Drop, Truncate)
- DML Queries (Insert, Update, Delete, Select)
- DCL Data Control Language
- TCL Transaction Control Language
- Joins
- Sub-Queries
- Union/ Union All
- Views
- Built-in functions: null(), Numeric, String and Date functions
- Aggregate functions
- Conditional statements

SQL Server Management Studio

SQL Server Management Studio (SSMS) is a software application first launched with the Microsoft SQL Server 2005 that is used for configuring, managing, and administering all components within Microsoft SQL Server. The tool includes both script editors and graphical tools which work with objects and features of the server.



```
SELECT Orders.OrderDate, [Order Details].*
  FROM Orders
  JOIN [Order Details] ON [Order Details].OrderID = Orders.OrderID
 WHERE OrderDate BETWEEN '1996-07-10' AND '1996-07-15'
```

	OrderDate	OrderID	ProductID	UnitPrice	Quantity	Discount
1	1996-07-10 00:00:00.000	10253	31	10.00	20	0
2	1996-07-10 00:00:00.000	10253	39	14.40	42	0
3	1996-07-10 00:00:00.000	10253	49	16.00	40	0
4	1996-07-11 00:00:00.000	10254	24	3.60	15	0.15
5	1996-07-11 00:00:00.000	10254	55	19.20	21	0.15
6	1996-07-11 00:00:00.000	10254	74	8.00	21	0
7	1996-07-12 00:00:00.000	10255	2	15.20	20	0
8	1996-07-12 00:00:00.000	10255	16	13.90	35	0
9	1996-07-12 00:00:00.000	10255	36	15.20	25	0
10	1996-07-12 00:00:00.000	10255	59	44.00	30	0
11	1996-07-15 00:00:00.000	10256	53	26.20	15	0
12	1996-07-15 00:00:00.000	10256	77	10.40	12	0

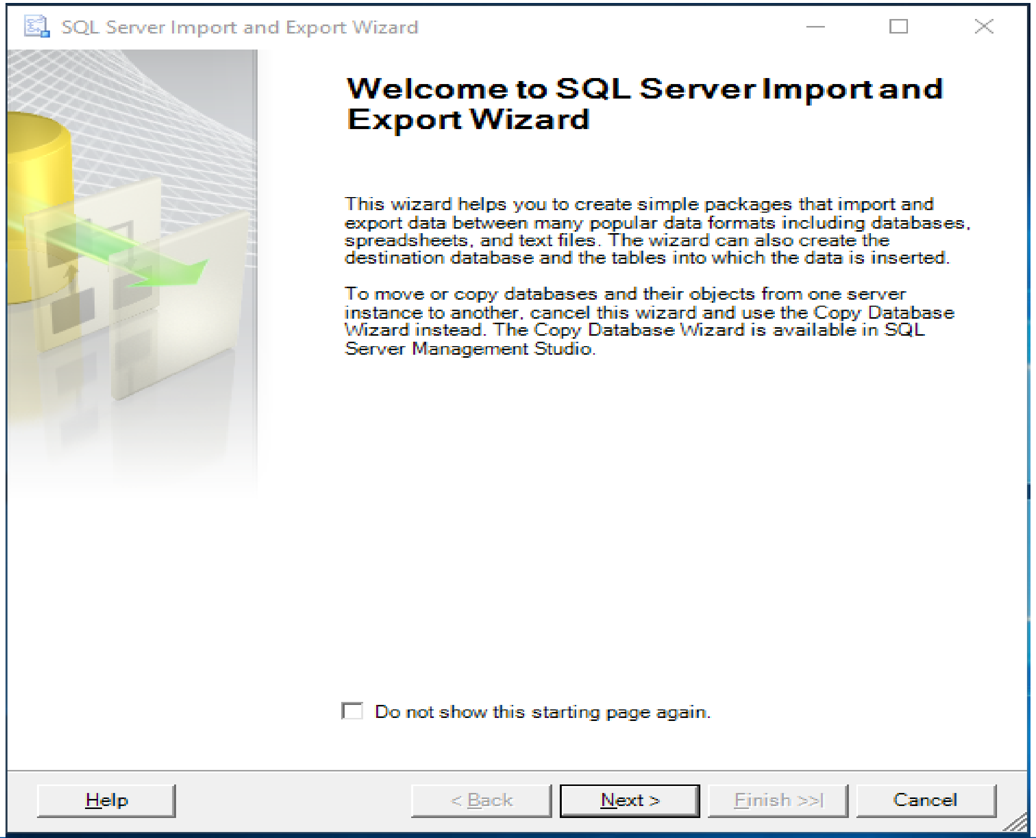
Query executed successfully.

SQL Server Import and Export Wizard

The SQL Server Import and Export Wizard offers the simplest method to create a Integration Services package that copies data from a source to a destination.

The SQL Server Import and Export Wizard can copy data to and from any data source for which a managed .NET Framework data provider or a native OLE DB provider is available. The list of available providers includes the following data sources:

- SQL Server
- Flat files
- Microsoft Office Access
- Microsoft Office Excel
- Microsoft Azure Blob Storage



Welcome to SQL Server Import and Export Wizard

This wizard helps you to create simple packages that import and export data between many popular data formats including databases, spreadsheets, and text files. The wizard can also create the destination database and the tables into which the data is inserted.

To move or copy databases and their objects from one server instance to another, cancel this wizard and use the Copy Database Wizard instead. The Copy Database Wizard is available in SQL Server Management Studio.

□ Do not show this starting page again.

SQL Server Profiler

SQL Server Profiler is a rich interface to create and manage traces and analyze and replay trace results. The events are saved in a trace file that can later be analyzed or used to replay a specific series of steps when trying to diagnose a problem. It can be used for activities such as:

- Stepping through problem queries to find the cause of the problem.
- Finding and diagnosing slow-running queries.
- Capturing the series of Transact-SQL statements that lead to a problem. The saved trace can then be used to replicate the problem on a test server where the problem can be diagnosed.
- Monitoring the performance of SQL Server to tune workloads
- Correlating performance counters to diagnose problems.

Introduction to T-SQL

- Transact – Structure Query Language (T-SQL) is Microsoft’s (& Sybase’s) proprietary extension to SQL.
- Its used for querying, altering and defining databases.
- Transact-SQL is central to using Microsoft SQL Server.
- All applications that communicate with an instance of SQL Server do so by sending Transact-SQL statements to the server, regardless of the user interface of the application.
- Although you can often avoid writing SQL, but using SQL GUI tools, there are still many situations where knowing basic T-SQL code allow to achieve tasks difficult or impossible with the GUI.

SQL statements Categories

SQL statements are divided into two major categories:

- **data definition language (DDL)**
- **data manipulation language (DML)**
- **DCL (Data Control Language)**

  DCL statements control the level of access that users have on database objects.

  GRANT – allows users to read/write on certain database objects
  REVOKE – keeps users from read/write permission on database objects

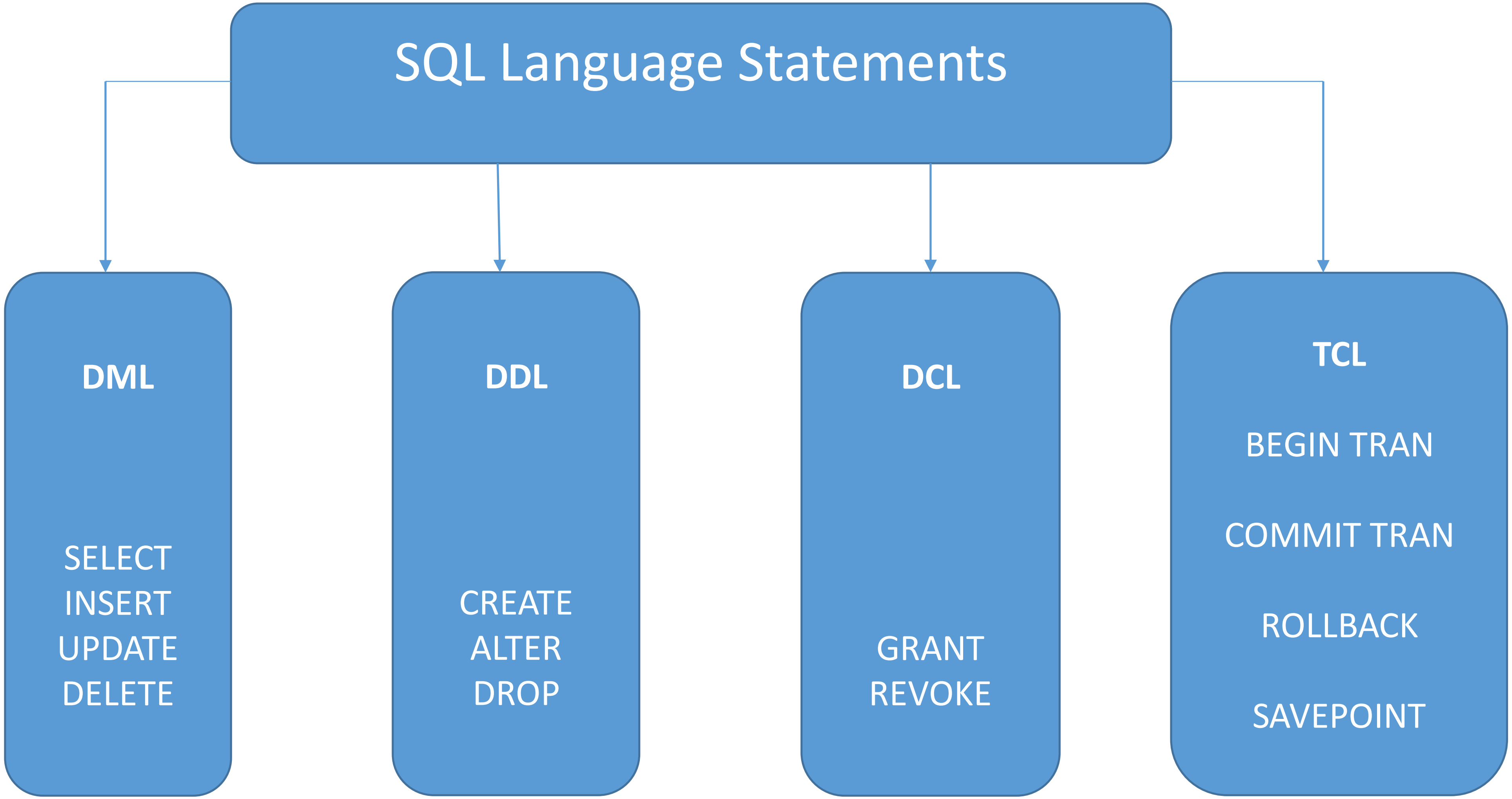
- **TCL (Transaction Control Language)**

  TCL statements allow you to control and manage transactions to maintain the integrity of data within SQL statements.

  BEGIN Transaction – opens a transaction
  COMMIT Transaction – commits a transaction
  ROLLBACK Transaction – ROLLBACK a transaction in case of any error

  SAVE Transaction – sets a Savepoint within a transaction

SQL statements Categories



SQL Language Statements

DML

SELECT
INSERT
UPDATE
DELETE

DDL

CREATE
ALTER
DROP

DCL

GRANT
REVOKE

TCL

BEGIN TRAN

COMMIT TRAN

ROLLBACK

SAVEPOINT

DDL Statement

Data Definition Language (DDL) is a vocabulary used to define data structures in SQL Server. Use these statements to create, alter, or drop data structures in an instance of SQL Server.

- Create Statements

  CREATE statements are used to define new entities.

```
CREATE DATABASE database_name
```

```
CREATE TABLE table_name
(
column_name1 data_type(size),
column_name2 data_type(size),
column_name3 data_type(size),
....
);
```

DDL Statement

- ALTER Statement
  - ALTER statements are used to modify the definition of existing entities

```
ALTER DATABASE { database_name  | CURRENT }
{
   MODIFY NAME = new_database_name
  | COLLATE collation_name
  | <file_and_filegroup_options>
  | <set_database_options>
}
```

```
ALTER TABLE table_name{
ALTER COLUMN column_name datatype
}
```

DDL Statement

- DROP Statement
  - Use DROP statement to remove existing entities.

```
DROP DATABASE [ IF EXISTS ] { database_name | database_snapshot_name } [ ,...n ] [;]

DROP TABLE [ IF EXISTS ] [ database_name . [ schema_name ] . | schema_name . ]
    table_name [ ,...n ]
     [ ; ]
```

DDL Statement

Truncate table

- Removes all rows from a table or specified partitions of a table, without logging the individual row deletions. The SQL **TRUNCATE TABLE** command is used to delete complete data from an existing table.

- You can also use DROP TABLE command to delete complete table but it would remove complete table structure from the database and you would need to re-create this table once again if you wish you store some data.

```
  TRUNCATE TABLE
[ { database_name .[ schema_name ] . | schema_name . } ]
table_name
[ WITH ( PARTITIONS ( { <partition_number_expression> | <range> }
[ , ...n ] ) ) ]
[ ; ]
```

Restrictions (Truncate)

You cannot use TRUNCATE TABLE on tables that:

- Are referenced by a FOREIGN KEY constraint. (You can’t truncate a table that has a foreign key that references itself.)
- Participate in an indexed view.
- Are published by using transactional replication or merge replication.
- TRUNCATE TABLE cannot activate a trigger because the operation does not log individual row deletions.

Data Manipulation Language (DML) Statements

- DML statements are used to work with the data IN tables.
- DML statements affect records in a table. These are basic operations we perform on data such as selecting a few records from a table, inserting new records, deleting unnecessary records, and updating/modifying existing records.
- When you are connected to most multi-user databases (whether in a client program or by a connection from a Web page script), you are in effect working with a private copy of your tables that can't be seen by anyone else until you are finished (or tell the system that you are finished).
- SELECT Statement is considered to be part of DML even though it just retrieves data rather than modifying it.

SELECT – select records from a table
INSERT – insert new records
UPDATE – update/Modify existing records
DELETE – delete existing records

INSERT (Transact-SQL)

- Adds one or more rows in the table

```
INSERT INTO table_name
VALUES (value1, value2, value3…)
```

```
INSERT INTO TraineeList VALUES (1, ‘John’ , ‘Smith’)
```

```
INSERT INTO table_name
(column2, column5,column6)
VALUES (value2, value5, value6)
INSERT INTO TraineeList
(FirstName,LastName)
VALUES (‘Sara’ , ‘Wilson’)
```

DELETE (Transact-SQL)

- Removes one or more rows from a table or view in SQL Server

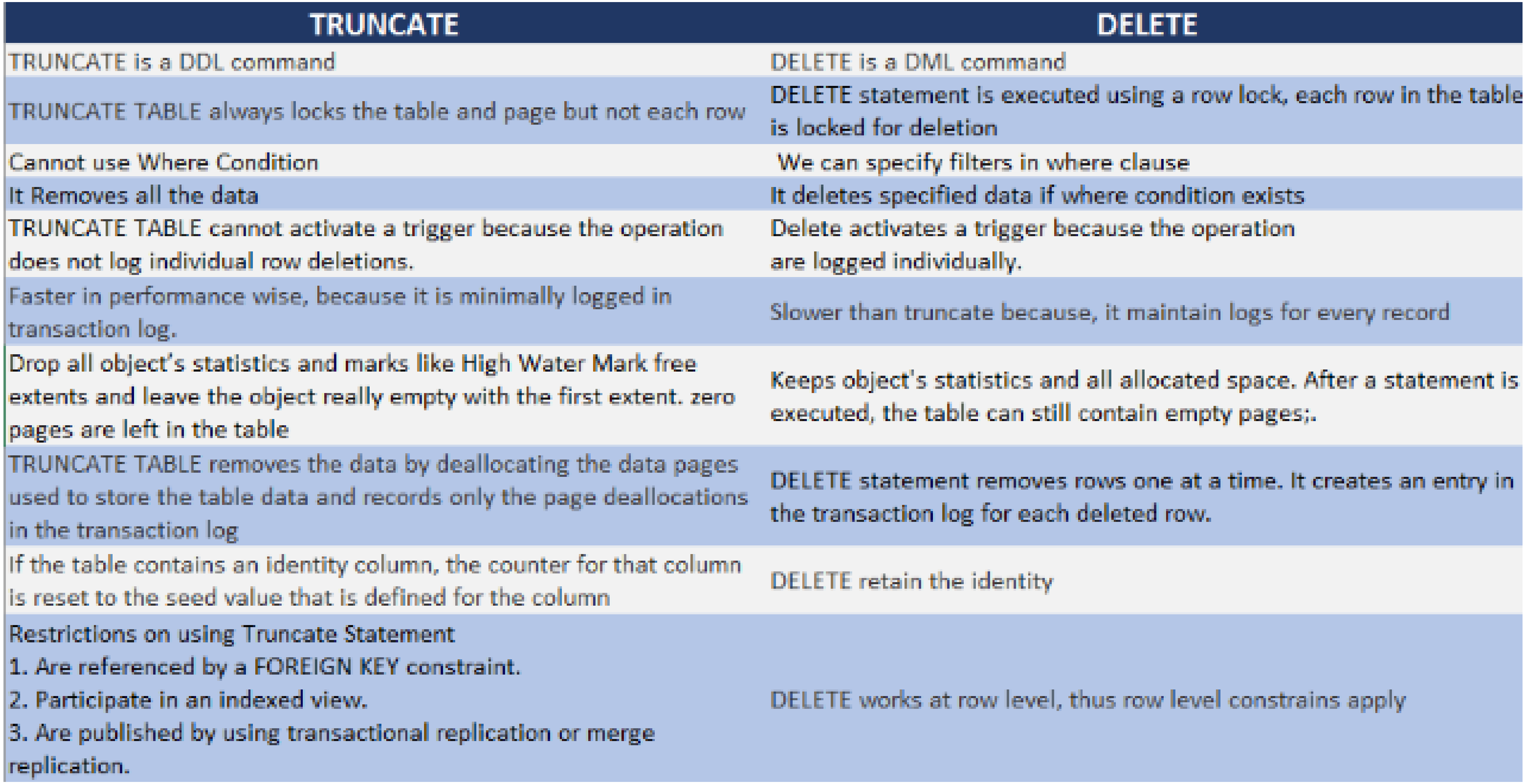
```
DELETE * FROM table_name

DELETE FROM table_name
  WHERE some_column = some_value
              (condition)
```

Sample:

```
DELETE FROM TraineeList
WHERE FirstName = ‘sara’
```

Differences between Truncate and Delete Statements



TRUNCATE	DELETE
TRUNCATE is a DDL command	DELETE is a DML command
TRUNCATE TABLE always locks the table and page but not each row	DELETE statement is executed using a row lock, each row in the table is locked for deletion
Cannot use Where Condition	We can specify filters in where clause
It Removes all the data	It deletes specified data if where condition exists
TRUNCATE TABLE cannot activate a trigger because the operation does not log individual row deletions.	Delete activates a trigger because the operation are logged individually.
Faster in performance wise, because it is minimally logged in transaction log.	Slower than truncate because, it maintain logs for every record
Drop all object's statistics and marks like High Water Mark free extents and leave the object really empty with the first extent. zero pages are left in the table	Keeps object's statistics and all allocated space. After a statement is executed, the table can still contain empty pages;.
TRUNCATE TABLE removes the data by deallocating the data pages used to store the table data and records only the page deallocations in the transaction log	DELETE statement removes rows one at a time. It creates an entry in the transaction log for each deleted row.
If the table contains an identity column, the counter for that column is reset to the seed value that is defined for the column	DELETE retain the identity
Restrictions on using Truncate Statement 1. Are referenced by a FOREIGN KEY constraint. 2. Participate in an indexed view. 3. Are published by using transactional replication or merge replication.	DELETE works at row level, thus row level constrains apply

UPDATE (TRANSACT-SQL)

- Changing existing data in a table or view
- UPDATE *table_name*

  SET

  column_name1 = new_value1,

  column_name2 = new value2

  WHERE *some_column = some_value*

  (condition)

Sample:

UPDATE *TraineeList*

SET

FirstName = ‘Philip’

WHERE *Trainee_ID = 1*

SELECT (Transact-SQL)

Retrieves rows from the database and enables the selection of one or many rows or columns from one or many tables in SQL Server.

SELECT *select_list* (* | column_name)

FROM *table_name*

WHERE *search_condition*

GROUP BY *group_by_expression*

HAVING *search_condition*

ORDER BY *order_expression* ASC | DESC

The UNION, EXCEPT and INTERSECT operators can be used between queries to combine or compare their results into one result set.

Select statements

- **WHERE Clause**

  Specify an actual value as condition to filter the SELECT statement

  SELECT * as QueryResults

  WHERE *ColumnName* = SomeValue AND|OR|NOT

  ColumnName = SomeValue

Logical Operators

AND|OR|NOT →passing more than one condition

Select statements

Operators (to pass some value)

=	→equal to [value]
!=	→!= not equal to [value]
<	→Less than [value]
>	→Greater than [value]
<=	→less than or equal to [value]
>=	→greater than or equal to [value]
LIKE	[wildcards]
BETWEEN	[VALUES AND VALUES]
IN	[VALUE1, VALUE2, …]

Select statements

- WILDCARDS
  - ‘any text‘ → text/string enclosed in single quotation marks; for numerical values, NO NEED to enclose in ‘ ‘ marks.
  - ^ → not
  - ‘%a’, ‘a%’, ‘%a%’ (enclosed in ‘ ‘ marks)

    a% → starts with a

    %a → ends with a

    %a% → with a in between

```
SELECT * as QueryResults
WHERE ColumnName  LIKE  ‘a%’

SELECT * as QueryResults
WHERE ColumnName  LIKE  ‘[abcde]%’

SELECT * as QueryResults
WHERE ColumnName   BETWEEN [1 AND 25]

SELECT * as QueryResults
WHERE ColumnName IN   [1,4,6,10]
```

Group By Clause

ORDER BY *column_name* ASC | DESC

→sort out the result of the SELECT query in alphabetical or ordinal order by default.

Select * from table_name Order by column_name

DESC → descending order

- GROUP BY *column_name*
  - *Used to group a selected set of rows into summary of rows by the values of one or more columns or expression*
  - *ALWAYS used in conjunction with one or more <u>AGGREGATE function</u>*
- HAVING *search_condition*
  - Pass condition after the AGGREGATE function

Joins

- SQL joins are used to combine rows from two or more tables, based on a common field between them.

  - Types of Joins:
    - Inner join
    - Outer join
      - Left outer join
      - Right outer join
      - Full outer join
    - Cross join
    - Self join

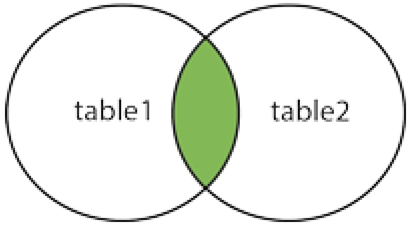
Types of Joins

- Inner join: The INNER JOIN keyword selects all rows from both tables as long as there is a match between the columns in both tables
- Outer join
  - Left outer join: LEFT JOIN keyword returns all rows from the left table (table1), with the matching rows in the right table (table2). The result is NULL in the right side when there is no match.
  - Right outer join: RIGHT JOIN keyword returns all rows from the right table (table2), with the matching rows in the left table (table1). The result is NULL in the left side when there is no match
  - Full outer join: The FULL OUTER JOIN keyword returns all rows from the left table (table1) and from the right table (table2).

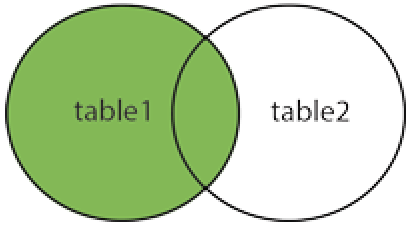
  The FULL OUTER JOIN keyword combines the result of both LEFT and RIGHT joins
- Cross join
- Self join

Types of Joins

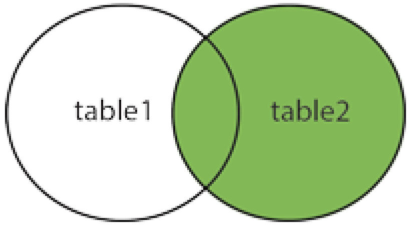
INNER JOIN



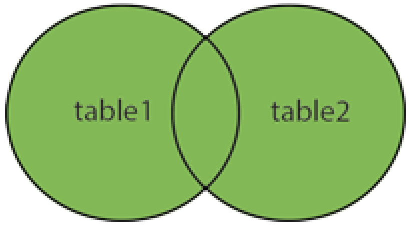
LEFT JOIN



RIGHT JOIN



FULL OUTER JOIN



Sample tables

Employee table

LastName	DepartmentID
Rafferty	31
Jones	33
Heisenberg	33
Robinson	34
Smith	34
Williams	NULL

Department table

DepartmentID	DepartmentName
31	Sales
33	IT
34	Clerical
35	Marketing

Inner Join

```
SELECT E.Lastname,E.DepartmentID, D.DepartmentName,D.DepartmentID
FROM employee E
INNER JOIN department D
ON E.DepartmentID = D.DepartmentID;
```

Employee.LastName	Employee.DepartmentID	Department.DepartmentName	Department.DepartmentID
Robinson	34	Clerical	34
Jones	33	IT	33
Smith	34	Clerical	34
Heisenberg	33	IT	33
Rafferty	31	Sales	31

Left Outer Join

SELECT * FROM employee E

LEFT OUTER JOIN D

ON E.DepartmentID = D.DepartmentID;

Employee.LastName	Employee.DepartmentID	Department.DepartmentName	Department.DepartmentID
Jones	33	IT	33
Rafferty	31	Sales	31
Robinson	34	Clerical	34
Smith	34	Clerical	34
Williams	NULL	NULL	NULL
Heisenberg	33	IT	33

Right Outer Join

SELECT * FROM employee **RIGHT OUTER JOIN** department **ON** employee.DepartmentID = department.DepartmentID;

Employee.LastName	**Employee.DepartmentID**	**Department.Department Name**	**Department.DepartmentID**
Smith	34	Clerical	34
Jones	33	IT	33
Robinson	34	Clerical	34
Heisenberg	33	IT	33
Rafferty	31	Sales	31
NULL	NULL	Marketing	35

Full Outer Join

SELECT * FROM employee **FULL OUTER JOIN** department **ON** employee.DepartmentID = department.DepartmentID;

Employee.LastName	Employee.DepartmentID	Department.DepartmentName	Department.DepartmentID
Smith	34	Clerical	34
Jones	33	IT	33
Robinson	34	Clerical	34
Williams	NULL	NULL	NULL
Heisenberg	33	IT	33
Rafferty	31	Sales	31
NULL	NULL	Marketing	35

CROSS JOIN

Returns the Cartesian product of rows from tables in the join. In other words, it will produce rows which combine each row from the first table with each row from the second table.

SELECT * FROM employee **CROSS JOIN** department;

Self Join

A self-join is joining a table to itself

```sql
SELECT F.EmployeeID, F.LastName, S.EmployeeID, S.LastName, F.Country
FROM Employee F
INNER JOIN Employee S
ON F.Country = S.Country
WHERE F.EmployeeID < S.EmployeeID
ORDER BY F.EmployeeID, S.EmployeeID;
```

Employee Table (Sample table)

EmployeeID	LastName	Country	DepartmentID
123	Rafferty	Australia	31
124	Jones	Australia	33
145	Heisenberg	Australia	33
201	Robinson	United States	34
305	Smith	Germany	34
306	Williams	Germany	NULL

Employee Table after Self-join by Country

EmployeeID	LastName	EmployeeID	LastName	Country
123	Rafferty	124	Jones	Australia
123	Rafferty	145	Heisenberg	Australia
124	Jones	145	Heisenberg	Australia
305	Smith	306	Williams	Germany

SUB-QUERIES

- A Subquery or Inner query or Nested query is a query within another SQL query and embedded within the WHERE clause.

- A subquery is used to return data that will be used in the main query as a condition to further restrict the data to be retrieved.

- Subqueries can be used with the SELECT, INSERT, UPDATE, and DELETE statements along with the operators like =, <, >, >=, <=, IN, BETWEEN etc.

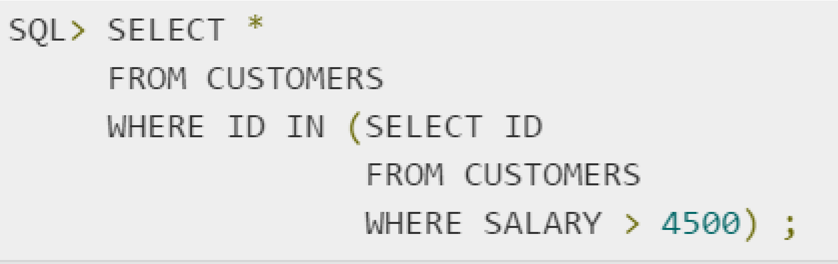
SUB-QUERIES

There are a few rules that subqueries must follow:

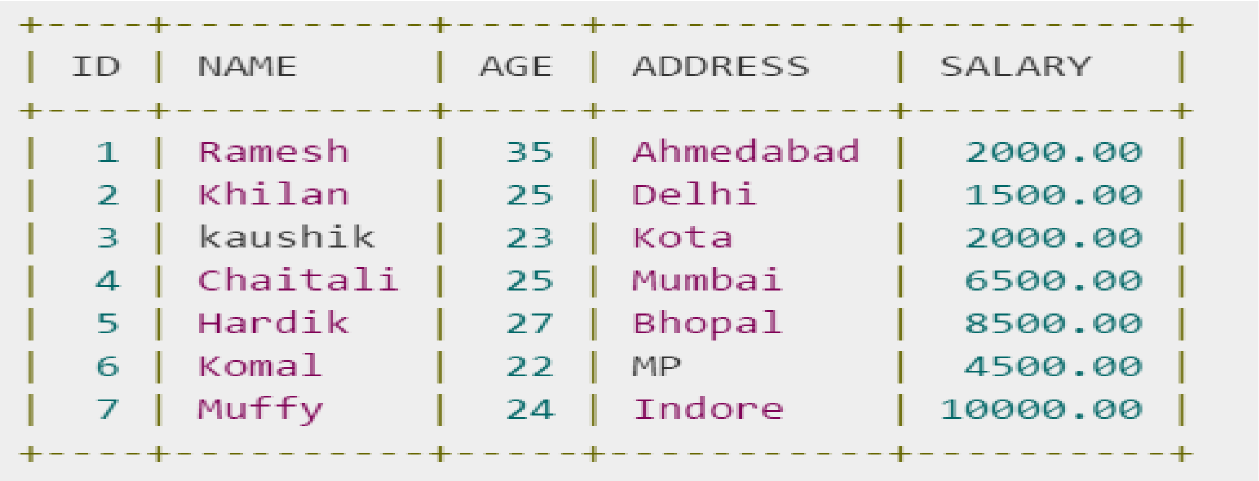
- Subqueries must be enclosed within parentheses.
- A subquery can have only one column in the SELECT clause, unless multiple columns are in the main query for the subquery to compare its selected columns.
- An ORDER BY cannot be used in a subquery, although the main query can use an ORDER BY. The GROUP BY can be used to perform the same function as the ORDER BY in a subquery.
- Subqueries that return more than one row can only be used with multiple value operators, such as the IN operator.
- A subquery cannot be immediately enclosed in a set function.
- The BETWEEN operator cannot be used with a subquery; however, the BETWEEN operator can be used within the subquery.

Subquery

SELECT column_name [, column_name]
FROM table1 [, table2]
WHERE column_name OPERATOR (SELECT column_name
[, column_name]
FROM table1 [, table2] [WHERE])

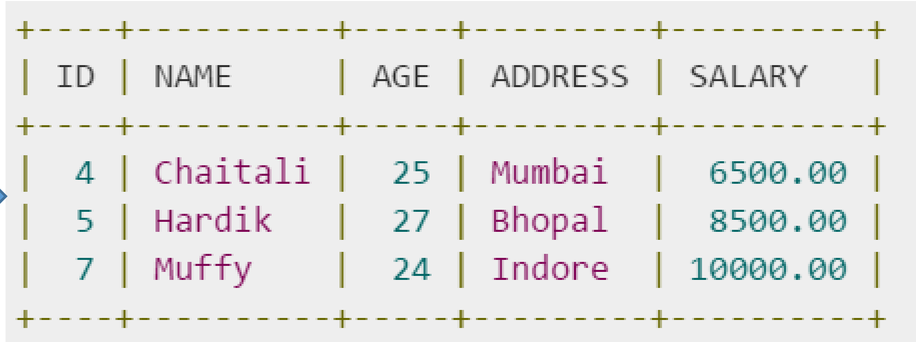


```
SQL> SELECT *
     FROM CUSTOMERS
     WHERE ID IN (SELECT ID
                  FROM CUSTOMERS
                  WHERE SALARY > 4500) ;
```



```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  35 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

Result



```
+----+----------+-----+---------+----------+
| ID | NAME     | AGE | ADDRESS | SALARY   |
+----+----------+-----+---------+----------+
|  4 | Chaitali |  25 | Mumbai  |  6500.00 |
|  5 | Hardik   |  27 | Bhopal  |  8500.00 |
|  7 | Muffy    |  24 | Indore  | 10000.00 |
+----+----------+-----+---------+----------+
```

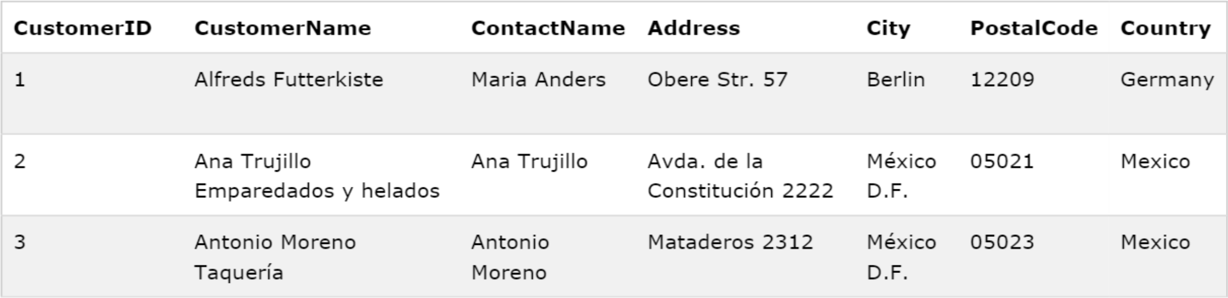
UNION/ UNION ALL

The UNION operator is used to combine the result-set of two or more SELECT statements.
Each SELECT statement within the UNION must have the same number of columns.
The columns must also have similar data types. Also, the columns in each SELECT statement must be in the same order.
The UNION operator selects only distinct values by default. To allow duplicate values, **use the ALL keyword with UNION.**

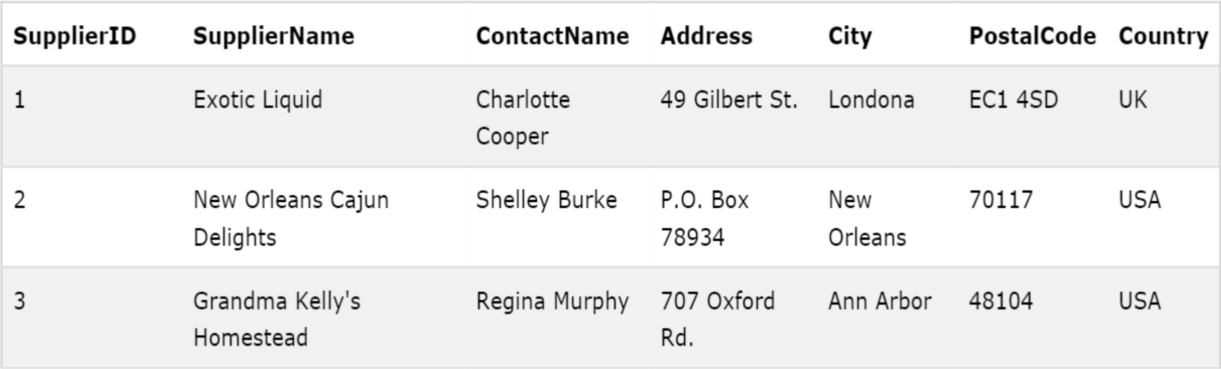
Select Columnname from table 1

Union

Select Columnname from table 2



CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico



SupplierID	SupplierName	ContactName	Address	City	PostalCode	Country
1	Exotic Liquid	Charlotte Cooper	49 Gilbert St.	Londona	EC1 4SD	UK
2	New Orleans Cajun Delights	Shelley Burke	P.O. Box 78934	New Orleans	70117	USA
3	Grandma Kelly's Homestead	Regina Murphy	707 Oxford Rd.	Ann Arbor	48104	USA

Result

Results will get all the data from both the tables

VIEWS

- A view is a virtual table based on the result-set of an SQL statement.
- A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database.
- You can add SQL functions, WHERE, and JOIN statements to a view and present the data as if the data were coming from one single table.

```
CREATE VIEW view_name

AS
SELECT column_name(s)
FROM table_name
WHERE condition
```

```
SELECT * FROM view_name
```

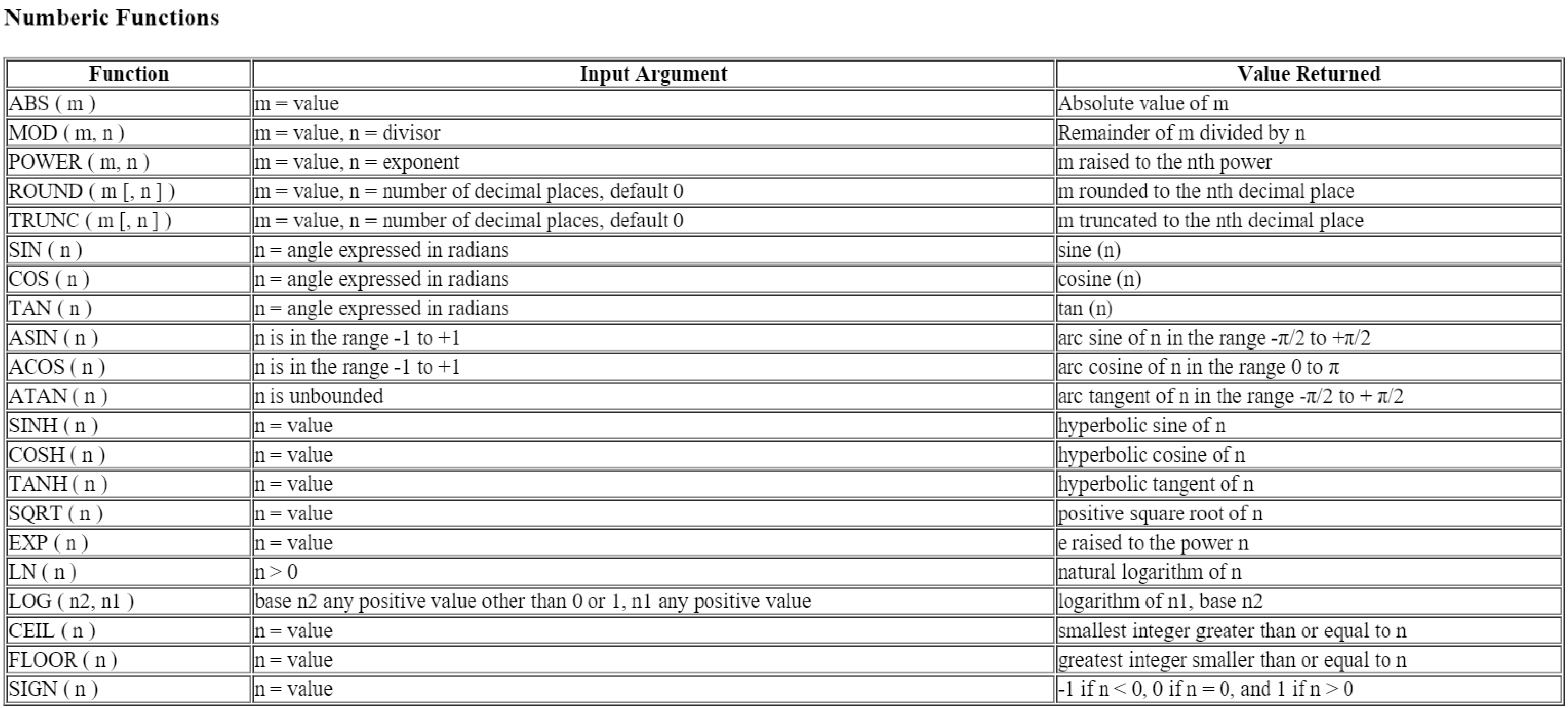
```
REPLACE VIEW view_name AS
SELECT column_name(s)
FROM table_name
WHERE condition
```

```
DROP VIEW view_name
```

Built-In Functions

- Numeric Functions
- String Functions
- String / Number Conversion Functions
- Formats for TO_CHAR Function
- Group Functions
- Date and Time Functions
- Date Conversion Functions
- Date Formats

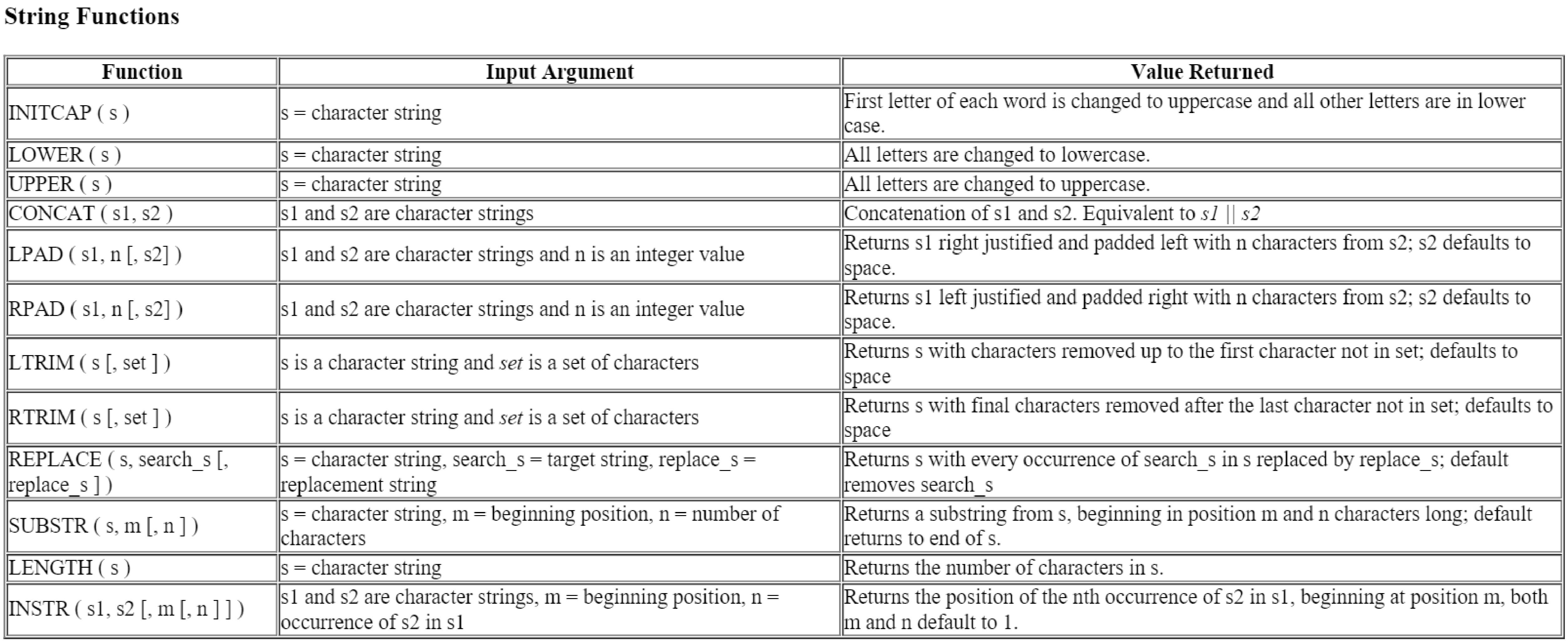
Number Functions



Numberic Functions

Function	Input Argument	Value Returned
ABS (m)	m = value	Absolute value of m
MOD (m, n)	m = value, n = divisor	Remainder of m divided by n
POWER (m, n)	m = value, n = exponent	m raised to the nth power
ROUND (m [, n])	m = value, n = number of decimal places, default 0	m rounded to the nth decimal place
TRUNC (m [, n])	m = value, n = number of decimal places, default 0	m truncated to the nth decimal place
SIN (n)	n = angle expressed in radians	sine (n)
COS (n)	n = angle expressed in radians	cosine (n)
TAN (n)	n = angle expressed in radians	tan (n)
ASIN (n)	n is in the range -1 to +1	arc sine of n in the range -π/2 to +π/2
ACOS (n)	n is in the range -1 to +1	arc cosine of n in the range 0 to π
ATAN (n)	n is unbounded	arc tangent of n in the range -π/2 to + π/2
SINH (n)	n = value	hyperbolic sine of n
COSH (n)	n = value	hyperbolic cosine of n
TANH (n)	n = value	hyperbolic tangent of n
SQRT (n)	n = value	positive square root of n
EXP (n)	n = value	e raised to the power n
LN (n)	n > 0	natural logarithm of n
LOG (n2, n1)	base n2 any positive value other than 0 or 1, n1 any positive value	logarithm of n1, base n2
CEIL (n)	n = value	smallest integer greater than or equal to n
FLOOR (n)	n = value	greatest integer smaller than or equal to n
SIGN (n)	n = value	-1 if n < 0, 0 if n = 0, and 1 if n > 0

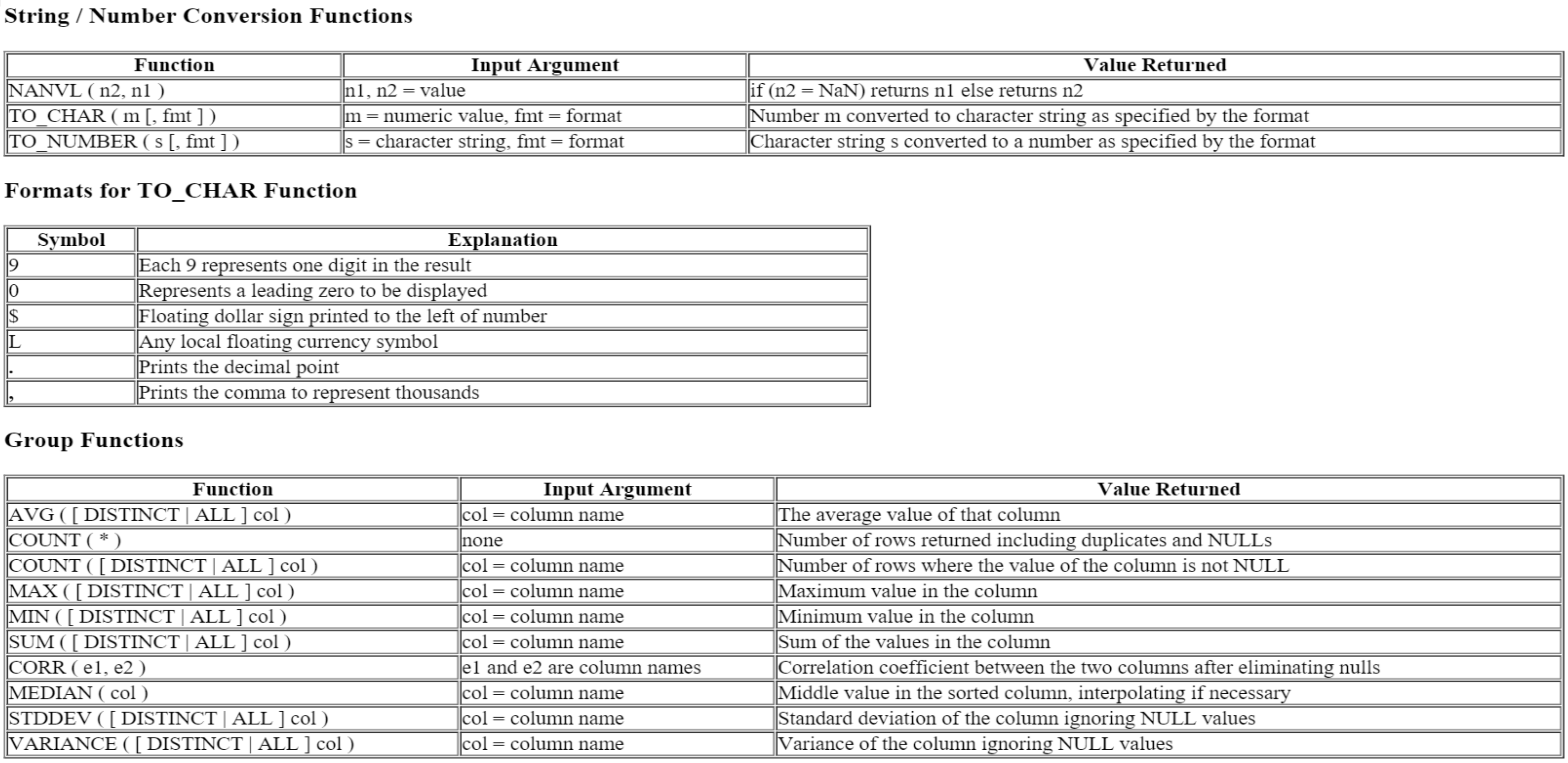
String Functions



String Functions

Function	Input Argument	Value Returned
INITCAP (s)	s = character string	First letter of each word is changed to uppercase and all other letters are in lower case.
LOWER (s)	s = character string	All letters are changed to lowercase.
UPPER (s)	s = character string	All letters are changed to uppercase.
CONCAT (s1, s2)	s1 and s2 are character strings	Concatenation of s1 and s2. Equivalent to *s1* || *s2*
LPAD (s1, n [, s2])	s1 and s2 are character strings and n is an integer value	Returns s1 right justified and padded left with n characters from s2; s2 defaults to space.
RPAD (s1, n [, s2])	s1 and s2 are character strings and n is an integer value	Returns s1 left justified and padded right with n characters from s2; s2 defaults to space.
LTRIM (s [, set])	s is a character string and *set* is a set of characters	Returns s with characters removed up to the first character not in set; defaults to space
RTRIM (s [, set])	s is a character string and *set* is a set of characters	Returns s with final characters removed after the last character not in set; defaults to space
REPLACE (s, search_s [, replace_s])	s = character string, search_s = target string, replace_s = replacement string	Returns s with every occurrence of search_s in s replaced by replace_s; default removes search_s
SUBSTR (s, m [, n])	s = character string, m = beginning position, n = number of characters	Returns a substring from s, beginning in position m and n characters long; default returns to end of s.
LENGTH (s)	s = character string	Returns the number of characters in s.
INSTR (s1, s2 [, m [, n]])	s1 and s2 are character strings, m = beginning position, n = occurrence of s2 in s1	Returns the position of the nth occurrence of s2 in s1, beginning at position m, both m and n default to 1.

String and Number Conversion



String / Number Conversion Functions

Function	Input Argument	Value Returned
NANVL (n2, n1)	n1, n2 = value	if (n2 = NaN) returns n1 else returns n2
TO_CHAR (m [, fmt])	m = numeric value, fmt = format	Number m converted to character string as specified by the format
TO_NUMBER (s [, fmt])	s = character string, fmt = format	Character string s converted to a number as specified by the format

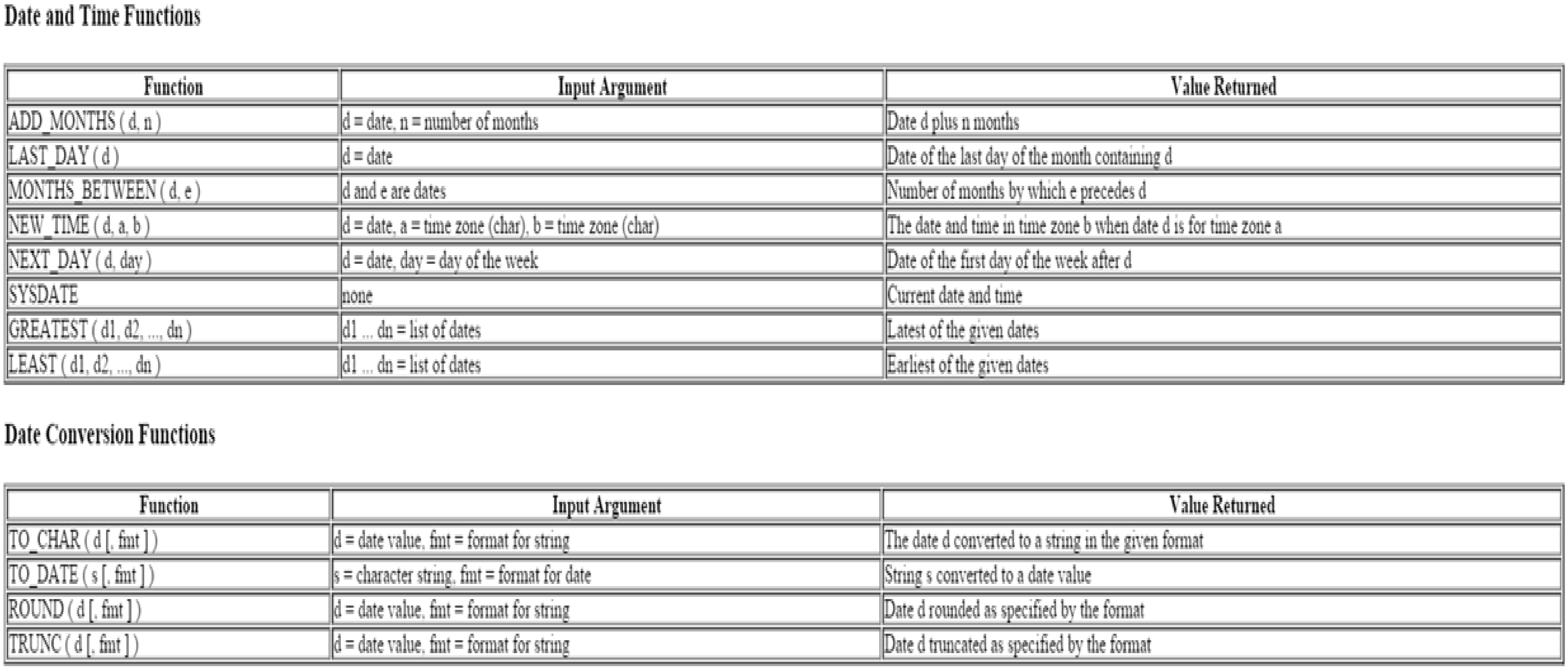
Formats for TO_CHAR Function

Symbol	Explanation
9	Each 9 represents one digit in the result
0	Represents a leading zero to be displayed
$	Floating dollar sign printed to the left of number
L	Any local floating currency symbol
.	Prints the decimal point
,	Prints the comma to represent thousands

Group Functions

Function	Input Argument	Value Returned
AVG ([DISTINCT | ALL] col)	col = column name	The average value of that column
COUNT (*)	none	Number of rows returned including duplicates and NULLs
COUNT ([DISTINCT | ALL] col)	col = column name	Number of rows where the value of the column is not NULL
MAX ([DISTINCT | ALL] col)	col = column name	Maximum value in the column
MIN ([DISTINCT | ALL] col)	col = column name	Minimum value in the column
SUM ([DISTINCT | ALL] col)	col = column name	Sum of the values in the column
CORR (e1, e2)	e1 and e2 are column names	Correlation coefficient between the two columns after eliminating nulls
MEDIAN (col)	col = column name	Middle value in the sorted column, interpolating if necessary
STDDEV ([DISTINCT | ALL] col)	col = column name	Standard deviation of the column ignoring NULL values
VARIANCE ([DISTINCT | ALL] col)	col = column name	Variance of the column ignoring NULL values

Date and Time Function



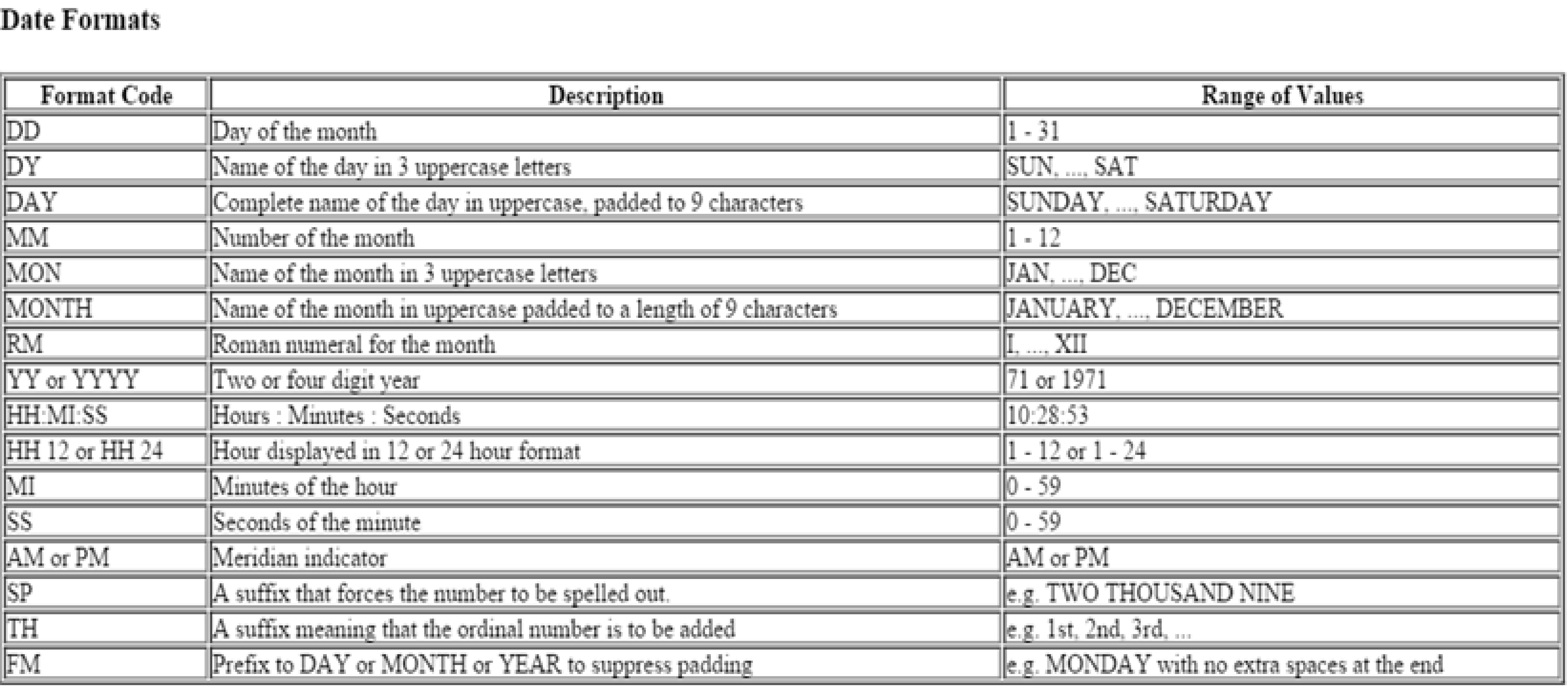
Date and Time Functions

Function	Input Argument	Value Returned
ADD_MONTHS (d, n)	d = date, n = number of months	Date d plus n months
LAST_DAY (d)	d = date	Date of the last day of the month containing d
MONTHS_BETWEEN (d, e)	d and e are dates	Number of months by which e precedes d
NEW_TIME (d, a, b)	d = date, a = time zone (char), b = time zone (char)	The date and time in time zone b when date d is for time zone a
NEXT_DAY (d, day)	d = date, day = day of the week	Date of the first day of the week after d
SYSDATE	none	Current date and time
GREATEST (d1, d2, ..., dn)	d1 ... dn = list of dates	Latest of the given dates
LEAST (d1, d2, ..., dn)	d1 ... dn = list of dates	Earliest of the given dates

Date Conversion Functions

Function	Input Argument	Value Returned
TO_CHAR (d [, fmt])	d = date value, fmt = format for string	The date d converted to a string in the given format
TO_DATE (s [, fmt])	s = character string, fmt = format for date	String s converted to a date value
ROUND (d [, fmt])	d = date value, fmt = format for string	Date d rounded as specified by the format
TRUNC (d [, fmt])	d = date value, fmt = format for string	Date d truncated as specified by the format

Date Functions



Date Formats

Format Code	Description	Range of Values
DD	Day of the month	1 - 31
DY	Name of the day in 3 uppercase letters	SUN, ..., SAT
DAY	Complete name of the day in uppercase, padded to 9 characters	SUNDAY, ..., SATURDAY
MM	Number of the month	1 - 12
MON	Name of the month in 3 uppercase letters	JAN, ..., DEC
MONTH	Name of the month in uppercase padded to a length of 9 characters	JANUARY, ..., DECEMBER
RM	Roman numeral for the month	I, ..., XII
YY or YYYY	Two or four digit year	71 or 1971
HH:MI:SS	Hours : Minutes : Seconds	10:28:53
HH 12 or HH 24	Hour displayed in 12 or 24 hour format	1 - 12 or 1 - 24
MI	Minutes of the hour	0 - 59
SS	Seconds of the minute	0 - 59
AM or PM	Meridian indicator	AM or PM
SP	A suffix that forces the number to be spelled out.	e.g. TWO THOUSAND NINE
TH	A suffix meaning that the ordinal number is to be added	e.g. 1st, 2nd, 3rd, ...
FM	Prefix to DAY or MONTH or YEAR to suppress padding	e.g. MONDAY with no extra spaces at the end

Sample build-in functions

SAMPLE BUILT-IN FUNCTION:

```
select round (83.28749, 2) as QueryResults;

select sqrt (3.67) as QueryResults;

select power (2.512, 5) as QueryResults;

select to_char ( sysdate, 'MON DD, YYYY' ) as QueryResults;

select to_char ( sysdate, 'HH12:MI:SS AM' ) as QueryResults;

select to_char ( new_time ( sysdate, 'CDT', 'GMT'), 'HH24:MI' ) as QueryResults;

select greatest ( to_date ( 'JAN 19, 2000', 'MON DD, YYYY' ), to_date ( 'SEP 27, 1999', 'MON DD, YYYY' ), to_date ( '13-Mar-2009', 'DD-Mon-YYYY' ) ) as
QueryResults;

 select next_day ( sysdate, 'FRIDAY' ) as QueryResults;

select last_day ( add_months ( sysdate, 1 ) ) as QueryResults;

select concat ('Alan', 'Turing') as "NAME" as QueryResults;

select 'Alan' || 'Turing' as "NAME" as QueryResults;

select initcap ("now is the time for all good men to come to the aid of the party") as "SLOGAN" as QueryResults;

select substr ('Alan Turing', 1, 4) as "FIRST" as QueryResults;
```

AGGREGATE FUNCTIONS

- SQL aggregate functions return a single value, calculated from values in a column.

- Useful aggregate functions:
    - AVG() - Returns the average value
    - COUNT() - Returns the number of rows
    - FIRST() - Returns the first value
    - LAST() - Returns the last value
    - MAX() - Returns the largest value
    - MIN() - Returns the smallest value
    - SUM() - Returns the sum

AGGREGATE FUNCTIONS

- GROUP BY column_name
  - Used to group a selected set of rows into summary of rows by the values of one or more columns or expression
  - ALWAYS used in conjunction with one or more AGGREGATE function
  - Columns that does not have an aggregate function must be contain in a GROUP BY clause
- HAVING search_condition
  - Pass condition after the AGGREGATE function takes place

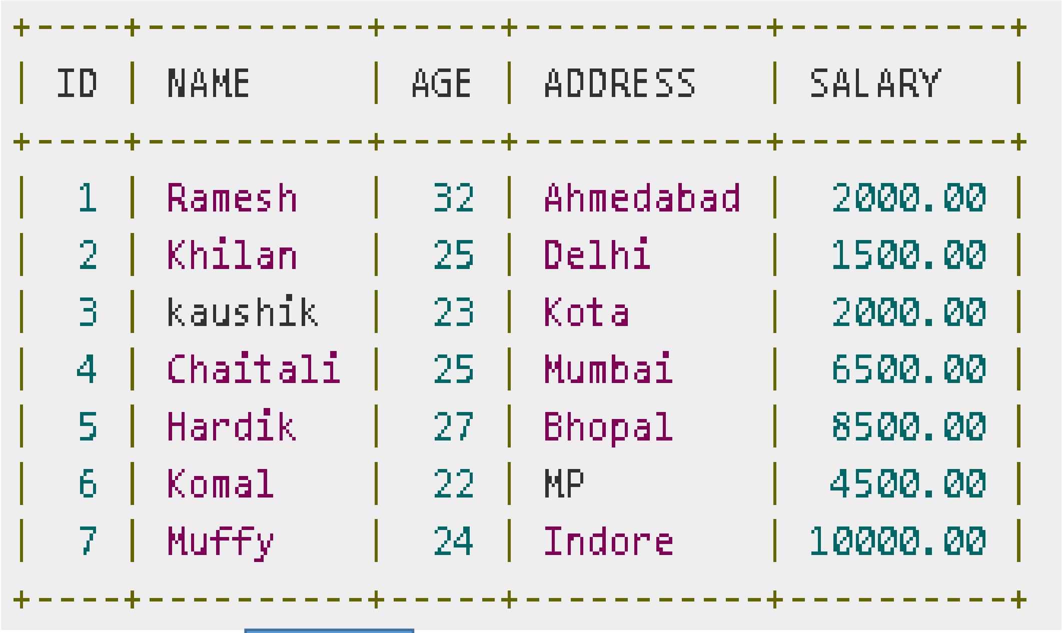
AGGREGATE FUNCTIONS

Sample:

```
SELECT COUNT(name) as total_names
FROM TraineeList
```

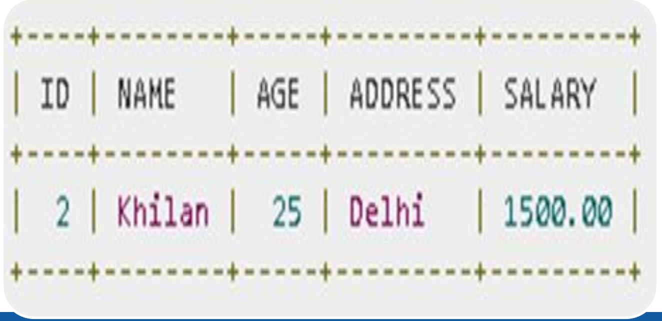
```
SELECT Column1, Column2, AVG(Column3)
FROM TableName
GROUP BY Column1, Column2
```

```
SELECT ID, NAME, AGE, ADDRESS, SALARY
FROM CUSTOMERS
GROUP BY age
HAVING COUNT(ID) = 2;
```



```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

result



```
+----+--------+-----+---------+---------+
| ID | NAME   | AGE | ADDRESS | SALARY  |
+----+--------+-----+---------+---------+
|  2 | Khilan |  25 | Delhi   | 1500.00 |
+----+--------+-----+---------+---------+
```

SCALAR Functions

- SQL scalar functions return a single value, based on the input value.
- Useful scalar functions:
  - UCASE() - Converts a field to upper case
  - LCASE() - Converts a field to lower case
  - MID() - Extract characters from a text field
  - LEN() - Returns the length of a text field
  - ROUND() - Rounds a numeric field to the number of decimals specified
  - NOW() - Returns the current system date and time
  - FORMAT() - Formats how a field is to be displayed

Conditional Statements in SQL

- If-Else Statement

  - Imposes conditions on the execution of a Transact-SQL statement. The Transact-SQL statement that follows an IF keyword and its condition is executed if the condition is satisfied: the Boolean expression returns TRUE. The optional ELSE keyword introduces another Transact-SQL statement that is executed when the IF condition is not satisfied: the Boolean expression returns FALSE.

  - An IF...ELSE construct can be used in batches, in stored procedures, and in ad hoc queries. When this construct is used in a stored procedure, it is frequently used to test for the existence of some parameter.

  - IF tests can be nested after another IF or following an ELSE. The limit to the number of nested levels depends on available memory.

```
IF Boolean_expression
{ sql_statement | statement_block }
[ ELSE
{ sql_statement | statement_block } ]
```

Boolean_expression

- Is an expression that returns TRUE or FALSE. If the Boolean expression contains a SELECT statement, the SELECT statement must be enclosed in parentheses.

{ sql_statement | statement_block }

- Is any Transact-SQL statement or statement grouping as defined by using a statement block. Unless a statement block is used, the IF or ELSE condition can affect the performance of only one Transact-SQL statement.
- To define a statement block, use the control-of-flow keywords BEGIN and END.

```
DECLARE @val int;
SET @val = 10

IF @val <25
PRINT 'hey its greater than 15'
ELSE
PRINT 'no its not greater than 15'
GO
```

Conditional Statements in SQL

- CASE Statements
  - SQL CASE is a very unique conditional statement providing if/then/else logic for any ordinary SQL command, such as SELECT or UPDATE. It then provides when-then-else functionality (WHEN this condition is met THEN do_this).

-
  ```
  SELECT product,
            'Status' =
            CASE
            WHEN quantity > 0
            THEN 'in stock‘
            ELSE 'out of stock‘
            END
  FROM dbo.inventory;
  ```

-
  ```
  SELECT ColumnName,
  
   'Status' = CASE
  
      WHEN SomeID > 5 and SomeID <= 10 THEN 'second batch'
  
      WHEN SomeID > 10 THEN 'last batch'
  
       ELSE 'first batch'
  
       END
  
   FROM TableName;
  ```

Null Functions

- Null Functions- used to handle NULL values in database. The objective of the general NULL handling functions is to replace the NULL values with an alternate value.
    - ISNULL() function is used to specify how we want to treat NULL values.
    - SELECT ProductName,UnitPrice*(UnitsInStock+ISNULL(UnitsOnOrder,0))
      FROM Products
    - * if "UnitsOnOrder" is NULL it will not harm the calculation, because ISNULL() returns a zero if the value is NULL
- NULLIF()-The NULLIF function compares two arguments expr1 and expr2. If expr1 and expr2 are equal, it returns NULL; else, it returns expr1. Unlike the other null handling function, first argument can't be NULL.
- NULLIF (expr1, expr2);

NVL

- NVL()

substitutes an alternate value for a NULL value. both the parameters are mandatory. Note that NVL function works with all types of data types. And also that the data type of original string and the replacement must be in compatible state. If arg1 is a character value, then converts replacement string to the data type compatible with arg1 before comparing them and returns VARCHAR2 in the character set of expr1. If arg1 is numeric, then determines the argument with highest numeric precedence, implicitly converts the other argument to that data type, and returns that data type.

```
NVL( Arg1, replace_with )
```

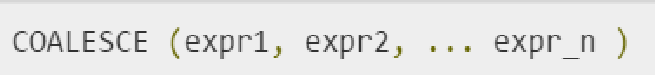
```
SELECT first_name, NVL(JOB_ID, 'n/a')
FROM employees;
```

Coalesce

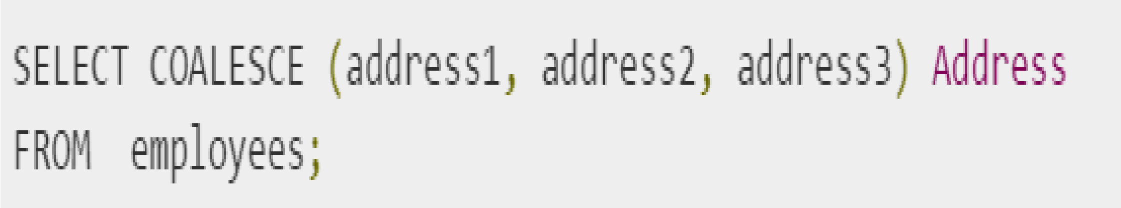
- COALESCE()

  COALESCE function, a more generic form of NVL, returns the first non-null expression in the argument list. It takes minimum two mandatory parameters but maximum arguments has no limit.

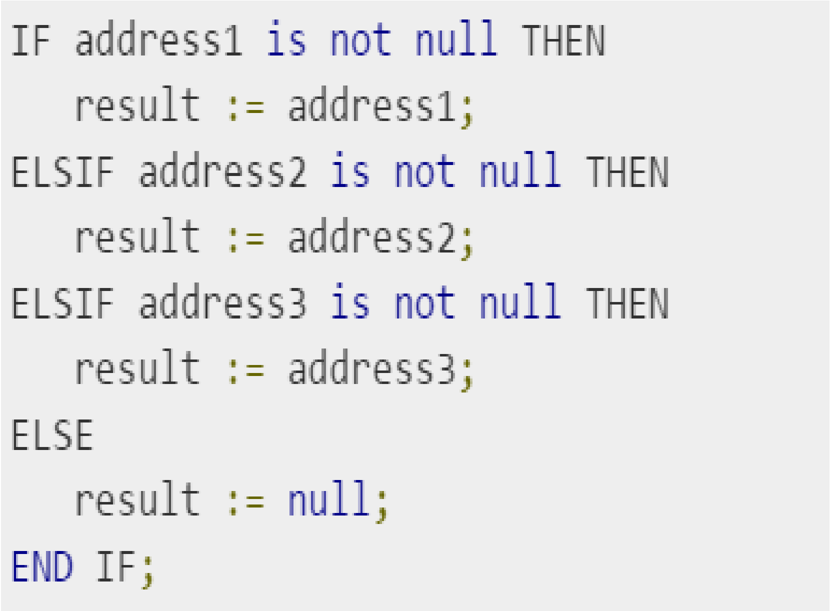
SYNTAX:



```
COALESCE (expr1, expr2, ... expr_n )
```



```
SELECT COALESCE (address1, address2, address3) Address
FROM  employees;
```



```
IF address1 is not null THEN
   result := address1;
ELSIF address2 is not null THEN
   result := address2;
ELSIF address3 is not null THEN
   result := address3;
ELSE
   result := null;
END IF;
```

Stored Procedure

Stored Procedure in SQL Server can be defined as the set of logical group of SQL statements which are grouped to perform a specific task. There are many benefits of using a stored procedure. The main benefit of using a stored procedure is that it increases the performance of the database. The other benefits of using the Stored Procedure are given below.

Benefits of using stored procedure:

- One of the main benefits of using the Stored procedure is that it reduces the amount of information sent to the database server.
- Compilation step is required only once when the stored procedure is created.
- It helps in re usability of the SQL code
- Stored procedure is helpful in enhancing the security

Stored Procedure

Maintainability

- Because scripts are in one location, updates and tracking of dependencies based on schema changes becomes easier

Testing

- Can be tested independent of the application

Isolation of Business Rules

- no confusion of having business rules spread over potentially disparate code files

Speed / Optimization

- Stored procedures are cached on the server
- Execution plans for the process are easily reviewable

Utilization of Set-based Processing

- The power of SQL is its ability to quickly and efficiently perform set-based processing on large amounts of data; the coding equivalent is usually iterative looping, which is generally much slower

Security

- Limit direct access to tables via defined roles in the database
- Provide an "interface" to the underlying data structure so that all implementation

Stored Procedure

```
Create  PROCEDURE
GetstudentnameInOutputVariable
(
   @studentid INT,
              -- Input parameter
   @studentname VARCHAR(200)  OUT
              -- Out parameter
)
AS
BEGIN
   SELECT @studentname= Firstname+'
'+Lastname
   FROM tbl_Students
   WHERE studentid=@studentid
END
```

```
EXEC GetstudentnameInOutputVariable 1
```

User Defined Functions

SQL Server user-defined functions are routines that accept parameters, perform an action, such as a complex calculation, and return the result of that action as a value. The return value can either be a single scalar value or a result set.

Benefits:

- They allow modular programming : You can create the function once, store it in the database, and call it any number of times in your program. User-defined functions can be modified independently of the program source code.
- They allow faster execution : Similar to stored procedures, Transact-SQL user-defined functions reduce the compilation cost of Transact-SQL code by caching the plans and reusing them for repeated executions. This means the user-defined function does not need to be reparsed and optimized with each use resulting in much faster execution times.
- They can reduce network traffic : An operation that filters data based on some complex constraint that cannot be expressed in a single scalar expression can be expressed as a function. The function can then invoked in the WHERE clause to reduce the number or rows sent to the client.

User Defined Functions(Types)

- Scalar Function

  User-defined scalar functions return a single data value of the type defined in the RETURNS clause.
  For an inline scalar function, there is no function body; the scalar value is the result of a single statement.
  For a multistatement scalar function, the function body, defined in a BEGIN...END block, contains a series of Transact-SQL statements that return the single value. The return type can be any data type excepttext, ntext, image, cursor, and timestamp.

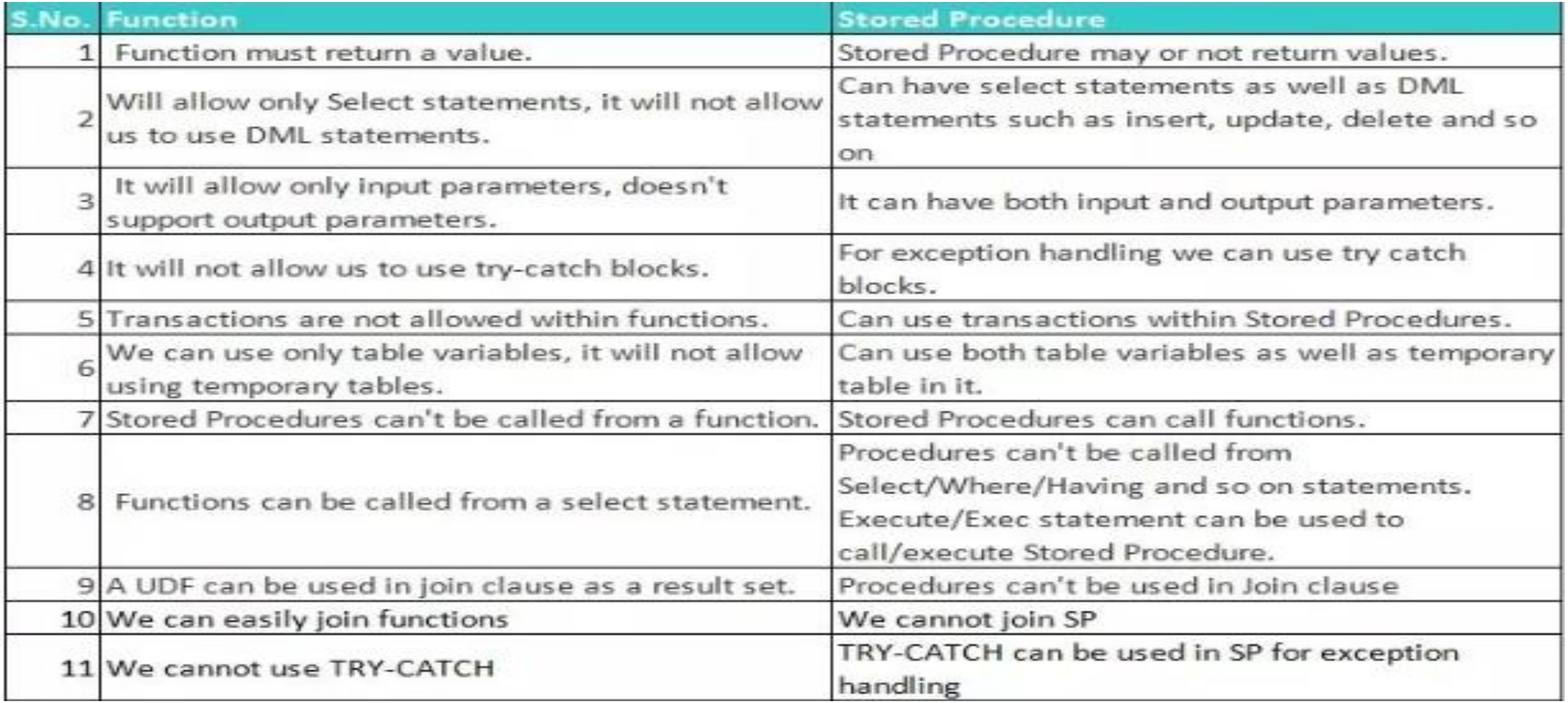
- Table-Valued Functions

  User-defined table-valued functions return a table data type. For an inline table-valued function, there is no function body; the table is the result set of a single SELECT statement.

- System Functions

  SQL Server provides many system functions that you can use to perform a variety of operations. They cannot be modified.

Difference between Stored Procedure & Function



S.No.	Function	Stored Procedure
1	Function must return a value.	Stored Procedure may or not return values.
2	Will allow only Select statements, it will not allow us to use DML statements.	Can have select statements as well as DML statements such as insert, update, delete and so on
3	It will allow only input parameters, doesn't support output parameters.	It can have both input and output parameters.
4	It will not allow us to use try-catch blocks.	For exception handling we can use try catch blocks.
5	Transactions are not allowed within functions.	Can use transactions within Stored Procedures.
6	We can use only table variables, it will not allow using temporary tables.	Can use both table variables as well as temporary table in it.
7	Stored Procedures can't be called from a function.	Stored Procedures can call functions.
8	Functions can be called from a select statement.	Procedures can't be called from Select/Where/Having and so on statements. Execute/Exec statement can be used to call/execute Stored Procedure.
9	A UDF can be used in join clause as a result set.	Procedures can't be used in Join clause
10	We can easily join functions	We cannot join SP
11	We cannot use TRY-CATCH	TRY-CATCH can be used in SP for exception handling