# Comments:

* - - used to comment at line level( for one line only)
* /\* … \*/ 🡪 used to comment the block of code.
* Comments are ignored by sql parser.

# Identifiers:

* Used in the naming convention of sql statements while creating the table name, view name, synonym name, column name, index name, function name, procedure name, user name, role name and so on.
* Two types of identifiers

1. Undelimited identifier.
2. Delimited identifier.

* Undelimited identifier:

1. Undelimited table and column names must start with a letter and contain only letters, digits, or underscores "\_".

* Delimited Identifier:

1. Delimited identifiers are enclosed in the delimiter, double quotes. The identifier can then contain any character including special characters. "AB$%CD" is a valid identifier name for example.

Ex: . "AB$%CD" 🡪 valid identifier.

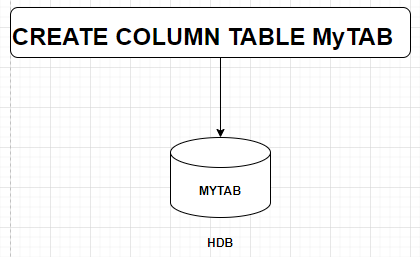
* Limitations:

1. "\_SYS\_" is reserved exclusively for database engine and is therefore not allowed at the beginning of schema object names.
2. The role name and username must be specified as undelimited identifiers.
3. The maximum length for identifiers is 127 characters.

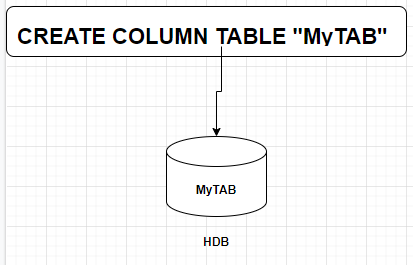
# Identifiers and case sensitivity:

* Identifiers without double-quotes in SQL syntax are converted to upper case when processed by the server.
* Ex: CREATE COLUMN TABLE MyTAB

Here ‘MyTAB’ 🡪 with out double quotes, but internally create with capital letter- that is ‘MYTABLE’.



* Ex: CREATE COLUMN TABLE "MyTab" creates a table called MyTab.



* Specifying identifiers without double-quotes is allowed but can cause ambiguity(unclear) later when querying or performing operations on objects where casing in the identifier name is significant. A recommendation is to standardize to using double-quotes around all identifiers in SQL statements where ambiguity may be a concern.

Quotation marks:

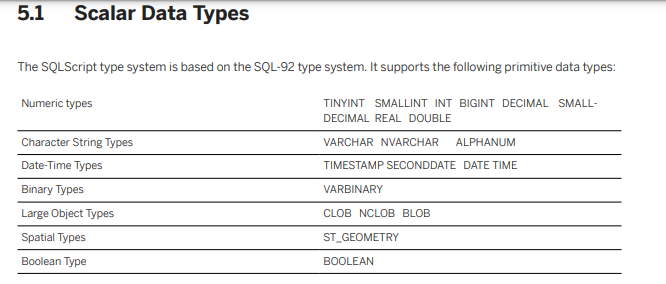
* Single quotation marks(‘) are used to delimit string literals.

A single quotation mark itself can be represented using two single quotation marks.

* Double quotation marks(“) are used to delimit identifiers. A double quotation mark itself can be represented using two double quotation marks.

# Data types:

Scalar data types:



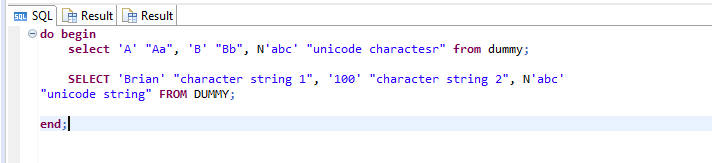
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Numeric | Character | Date-Time | Binary | Large Obj | Special | Boolean | Multi-Valued |
| TINYINT | VARCHAR | TIMESTAMP | VARBINARY | CLOB | ST\_GEOMETRY | BOOLEAN | ARRAY |
| SMALLINT | NVARCHAR | SECONDDATE |  | NCLOB | ST\_POINT |  |  |
| INT | ALPHANUM | DATE |  | BLOB |  |  |  |
| BIGINT | SHORTTEXT | TIME |  |  |  |  |  |
| DECIMAL |  |  |  |  |  |  |  |
| SMALLDECIMAL |  |  |  |  |  |  |  |
| REAL |  |  |  |  |  |  |  |
| DOUBLE |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |

# Typed Constants:

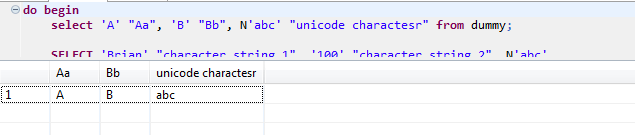
* A constant is a symbol that represents a specific fixed data value.
* Character string constant:

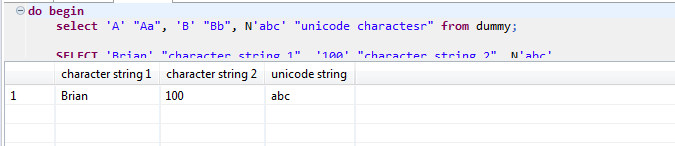
1. A character string constant is enclosed in single quotation marks, for example: 'Brian' or '100'.
2. Unicode strings have a similar format to character strings but are preceded by an N identifier (N stands for National Language in the SQL-92 standard).

Ex:



Output:





SELECT 'Brian' "character string 1", '100' "character string 2", N'abc' "unicode string" FROM DUMMY;

Or

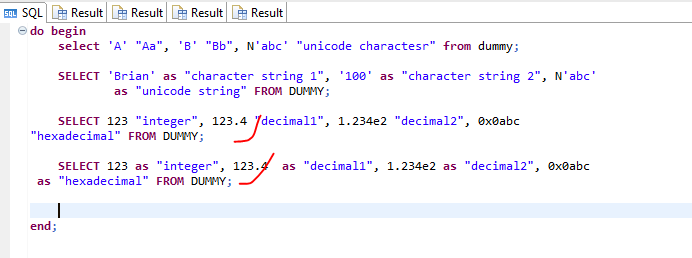
SELECT 'Brian' as "character string 1", '100' as "character string 2", N'abc' as "unicode string" FROM DUMMY;

Here: "character string 1"/"character string 2"/"unicode string" are headings/aliases in the output.

* Number Constant:

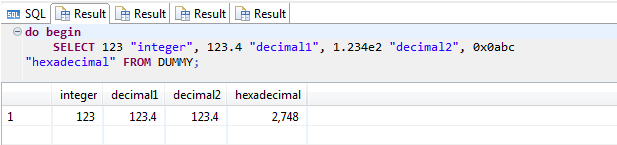
1. A number constant is represented by a string of numbers that are not enclosed in quotation marks.
2. Numbers may contain a decimal point or a scientific notation. For example, 123, 123.4, or 1.234e2.
3. A hexadecimal number constant is a string of hexadecimal numbers and has the prefix 0x. For example, 0x0abc.

Ex:



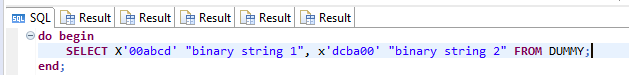
SELECT 123 "integer", 123.4 "decimal1", 1.234e2 "decimal2", 0x0abc "hexadecimal" FROM DUMMY;

Output



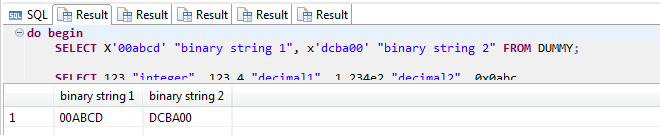
* Binary string constant

1. A binary string has the prefix X and is a string of hexadecimal numbers that are enclosed in quotation marks. For example, X'00abcd' or x'dcba00'.



SELECT X'00abcd' "binary string 1", x'dcba00' "binary string 2" FROM DUMMY;

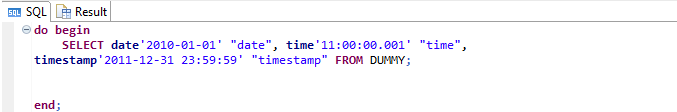
Output:



* Date/Time/Timestamp constant:

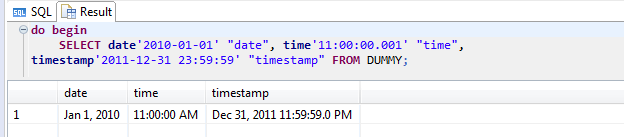
1. Date, Time, and Timestamp each have the following prefixes:
2. Date constant has the prefix with ‘date’: date'2010-01-01'
3. Time constant has the prefix with ‘time’: time'11:00:00.001'
4. Timestamp has the prefix with ‘timestamp’: timestamp'2011-12-31 23:59:59'

Ex:



SELECT date'2010-01-01' "date", time'11:00:00.001' "time", timestamp'2011-12-31 23:59:59' "timestamp" FROM DUMMY;

Output:



* Binary Data Types(VARBINARY)

1. Binary types are used to store bytes of binary data.
2. A value of type binary can be converted to a value of type (N)VARCHAR if its size is smaller than or equal to 8192. It can therefore be used like a value of type (N)VARCHAR except for full text search operations and numeric operations.

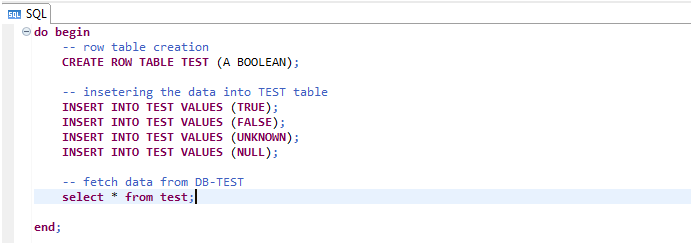
VARBINARY:

1. The VARBINARY() data type is used to store binary data of a specified maximum length in bytes, where indicates the maximum length and is an integer between 1 and 5000.
2. If the length is not specified, then the default is 1.

* Boolean Data Type:

1. The BOOLEAN data type stores boolean values.
2. Boolean values are TRUE(1), FALSE(0) and UNKNOWN
3. UNLNOWN is similar to NULL.

Ex: create the row table and inserted with Boolean values and see the output using select statement in DB level.



CREATE ROW TABLE TEST (A BOOLEAN);

INSERT INTO TEST VALUES (TRUE);

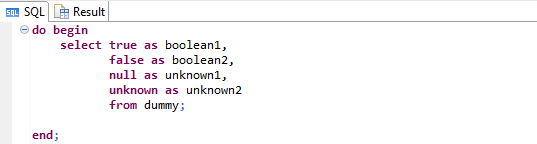
INSERT INTO TEST VALUES (FALSE);

INSERT INTO TEST VALUES (UNKNOWN);

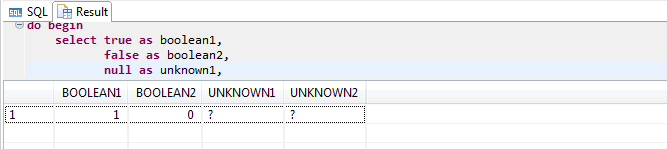
INSERT INTO TEST VALUES (NULL);

SELECT A "boolean" FROM TEST WHERE A = TRUE;

Ex:

: 

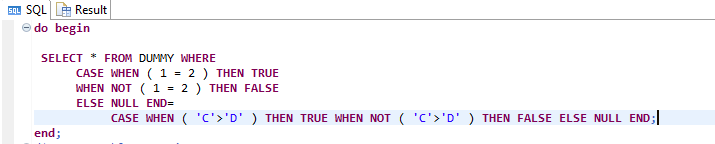
Output: UNLNOWN = NULL(Both are same)



For example, the following statement does not work:

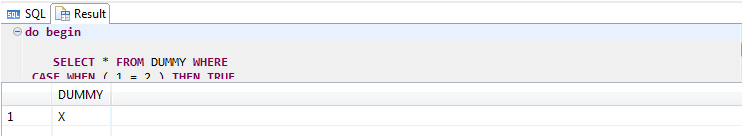
SELECT \* FROM DUMMY WHERE ( 'A'>'B' ) = ( 'C'>'D' );

The following statement is the correct way to achieve the results desired from the statement above:



SELECT \* FROM DUMMY WHERE CASE WHEN ( 'A'>'B' ) THEN TRUE WHEN NOT ( 'A'>'B' ) THEN FALSE ELSE NULL END= CASE WHEN ( 'C'>'D' ) THEN TRUE WHEN NOT ( 'C'>'D' ) THEN FALSE ELSE NULL END;

Output:



* Character String Data types:

1. Character string data types are used to store values that contain character strings.
2. VARCHAR data types contain 7-bit ASCII character strings.
3. NVARCHAR are used for storing Unicode character strings.
4. Character string data types in SAP HANA use 7-bit ASCII.
5. Extended ASCII characters are converted into corresponding Unicode characters.
6. If the data includes anything other than 7-bit ASCII characters, use Unicode character string types, such as NVARCHAR and NCLOB.
7. The SAP HANA database does not officially support the CHAR and NCHAR datatypes, Use VARCHAR and NVARCHAR instead.
8. Character string data types are variable length.
9. Character data types are 4 types
10. VARCHAR
11. NVARCHAR
12. ALPHANUM
13. SHORTTEXT

* VARCHAR:

DDL:

1. Syntax: VARCHAR(n), here n🡪 is the length of string.
2. Is the variable length character string.
3. N= provides the length of the string. i.e 1 to 5000
4. Maximum length 5000 and minimum length 1.
5. If length not defined in ddl statements🡺 then the default length is 1.

DML:

1. if the VARCHAR(n) data type used in a DML query. For Example CAST( A as VARCHAR(n)), here n- indicates maximum length of the string in a characters.
2. If the length not specified, the default length is 5000.

|  |  |  |  |
| --- | --- | --- | --- |
| Data type | MIN length | Max length | Default |
| VARCHAR | 1 | 5000 | 1 |
| NVARCHAR | 1 | 5000 | 1 |
| VARCHAR(Unicode) | 1 | 5000 | 5000 |
| NVARCHAR(Unicode) | 1 | 5000 | 5000 |
| ALPHANUM | 1 | 127 | If it is numeric only, adding leading zeros |
| SHORTTEXT |  |  | Used in column table only |

* Short text:

1. Can be used in column tables only, not in row table.
2. it can be support text search feature and string search features.

* Data Type Conversion

1. Both implicit and explicit data type conversions are allowed in the SAP HANA database.