LTI
Let's Solve

Celebrating 20 Years

SAP HANA internal training – Session 4

March 2018

A Larsen & Toubro Group Company

HANA SQL – Basics and Advanced

Basics

- SQL stands for Structured Query Language.
- It is a standardized language for communicating with a relational database.
- SQL is used to retrieve, store or manipulate information in the database.
- SQL statements perform the following tasks:
  - Schema definition and manipulation
  - Data manipulation
  - System management
  - Session management
  - Transaction management

HANA SQL – Basics and Advanced

Basics – Data Types

- Classification of Data Types:

Classification	Data Type
Datetime types	DATE, TIME, SECONDDATE, TIMESTAMP
Numeric types	TINYINT, SMALLINT, INTEGER, BIGINT, SMALLDECIMAL, DECIMAL, REAL, DOUBLE
Boolean type	BOOLEAN
Character string types	VARCHAR, NVARCHAR, ALPHANUM, SHORTTEXT
Binary types	VARBINARY
Large Object types	BLOB, CLOB, NCLOB, TEXT
Multi-valued types	ARRAY
Spatial types	ST_GEOMETRY, ST_POINT

HANA SQL – Basics and Advanced

Basics - Operators

- **Reserved Words:** Reserved words are words which have a special meaning to the SQL parser in the SAP HANA database and cannot be used as when defining an identifier.
- You can obtain the list of Keywords by querying the RESERVED_KEYWORDS system view (SELECT * FROM RESERVED_KEYWORDS;).
- **Operators:** Operators can be used for calculation, value comparison or to assign values.
  - Unary and Binary Operators
    - Ex.: **Unary:** +,-, NOT. **Binary:** +,-,*,/,AND,OR, >, <, >=, <=, =, !=
  - Arithmetic Operators
    - Ex. – (negation), +, -, *, /

    NOTE: For All operators except division (/), Result is NULL if any of the expression is NULL. In case of division, If either expression is NULL, or if the second expression is 0, an error is returned.
  - String Operators
    - Ex. || (Concatenation)
  - Comparison Operators
    - Ex. =, >, <, >=, <= , !=/<>
  - Logical Operators
    - AND, OR, NOT

HANA SQL – Basics and Advanced

Basics - Operators

- **Operator Precedence:**

Precedence	Operator	Operation
Highest	()	parentheses
	+,-	unary positive and negative operation
	*, /	multiplication, division
	+, -	addition, subtraction
	||	concatenation
	=, !=, , <=, >=, IS NULL, LIKE, BETWEEN	comparison
	NOT	logical negation
	AND	Conjunction
Lowest	OR	disjunction

HANA SQL – Basics and Advanced

Basics - Expressions

- **Expressions:** An expression is a clause that can be evaluated to return values.
  - **Case Expressions**

    Syntax – <case_expression> ::= <simple_case_expression> | <search_case_expression>

    <simple_case_expression> ::=
    CASE <expression>
    WHEN <expression> THEN <expression>
    [{ WHEN <expression> THEN <expression>}…]
    [ELSE <expression>]
    END

    If the expression following the CASE statement is equal to the expression following the WHEN statement, the expression following the THEN statement is returned. Otherwise, the expression following the ELSE statement is returned if it exists.
  - **Function Expressions**

    Syntax - <function_expression> ::= <function_name> (<expression> [{, <expression>}...])

    SQL built-in functions can be used as expressions.
  - **Aggregate Expressions**

    Syntax – **<aggregate_expression> ::= COUNT(*) | COUNT (DISTINCT <expression_list>) |<agg_name> ([ALL | DISTINCT] <expression>) | STRING_AGG (<expression> [,<delimiter>] [<aggregate_order_by_clause>]) <agg_name> ::= CORR | CORR_SPEARMAN | COUNT | MIN | MEDIAN | MAX | SUM | AVG | STDDEV | VAR | STDDEV_POP | VAR_POP | STDDEV_SAMP | VAR_SAMP**

    <delimiter> ::= <string_constant><aggregate_order_by_clause> ::= ORDER BY <expression> [ASC | DESC][NULLS FIRST | NULLS LAST]

    An aggregate expression uses an aggregate function to calculate a single value from the values of multiple rows in one or more columns
  - **Subqueries in Expressions**

    Syntax – **<scalar_subquery_expression> ::= (<subquery>)**

    A subquery is a SELECT statement enclosed in parentheses

HANA SQL – Basics and Advanced

Basics - Predicates

- **Predicates:** A predicate is specified by combining one or more expressions, or logical operators, and returns one of the following logical/truth values: TRUE, FALSE, or UNKNOWN.
  - Comparison Predicates: Two values are compared using comparison predicates, and the comparison returns true, false, or unknown.
  - BETWEEN Predicates: Compares a value with a list of values within the specified range and returns true or false.
  - EXISTS Predicate: Tests for the presence of a value in a set and returns either true or false.
  - IN Predicate: Searches for a value in a set of values and returns true or false.
  - LIKE Predicate: Performs a comparison to see if a character string matches, or does not match, a specified pattern.
  - NULL Predicate: Performs a comparison of the value of an expression with NULL.
  - CONTAINS Predicate: Matches a search string with the results of a subquery

HANA SQL – Basics and Advanced

Basics – SQL Functions

- **Aggregate Functions**: Aggregate functions are analytic functions that calculate an aggregate value based on a group of rows.
- **Date Type Conversion Functions**: Data type conversion functions convert data from one data type to another data type.
- **Datetime Functions**: Date and time functions perform operations on date and time data types or return date or time information.
- **Fulltext Functions**: Fulltext functions perform operations on data that has a fulltext index.
- **Hierarchy Functions**: Hierarchy functions allow you to work with hierarchical data such as tables with rows arranged in a tree or directed graph.
- **Miscellaneous Functions**: SAP HANA supports many functions that return system values and perform various operations on values, expressions, and return values of other functions.
- **Numeric Functions**: Numeric functions perform mathematical operations on numerical data types or return numeric information.
- **Series Data Functions**: Series data functions provide special functionality for series data and series tables.
- **String Functions**: String functions perform extraction and manipulation on strings, or return information about strings.
- **Security Functions**: Security functions provide special functionality for security purposes.
- **Window Functions:** Window functions allow you to perform analytic operations over a set of input rows.

HANA SQL – Basics and Advanced

Basics – SQL Functions

- **Some Important SQL Functions and syntaxes:**
- Create Table:
  - Row Store Type:
    - Create Table “<schema_name>”.”<table_name>” (column_name data type, primary key ());
  - Column Store Type:
    - Create Column Table “<schema_name>”.”<table_name>” (column_name data type, primary key ());
- Create Procedure:
  - CREATE PROCEDURE “<schema_name>"." <procedure_name“ (IN <parameter_name>, OUT <parameter_name>, INOUT <parameter_name>) LANGUAGE SQLSCRIPT AS <variable declaration> BEGIN <body> END;
- Create Trigger:
  - CREATE TRIGGER “<schema_name>".“<trigger_name>" AFTER/BEFOREINSTEAD OF INSERT/DELETE/UPDATE ON "SAP_STUDENT"."TRIG_FROM“ REFERENCING NEW/OLD <list> FOR EACH ROW/STATEMENT BEGIN <body> END

HANA SQL – Basics and Advanced

Basics – SQL Functions

- **Some Important SQL Functions and syntaxes:**
- Alter Table:
  - Add a new column
    - ALTER TABLE “<schema_name>”.”<table_name>” ADD (column_name data type);
  - Delete an Existing Column
    - ALTER TABLE “<schema_name>”.”<table_name>” DROP (column_name);
  - Modify Name of a column
    - RENAME COLUMN <schema_name>.<table_name>.<old_column_name> TO <new_column_name>;
  - Modify Data type of a column
    - ALTER TABLE “<schema_name>”.”<table_name>” ALTER (column_name data type);
  - Add primary key
    - ALTER TABLE “<schema_name>”.”<table_name>” ADD PRIMARY KEY (column_name(s));
  - Delete primary key
    - ALTER TABLE “<schema_name>”.”<table_name>” DROP PRIMARY KEY;
  - Modify Primary Key : First delete the existing key and then add new key as:
    - ALTER TABLE “<schema_name>”.”<table_name>” DROP PRIMARY KEY;
    - ALTER TABLE “<schema_name>”.”<table_name>” ADD PRIMARY KEY (column_name(s));

HANA SQL – Basics and Advanced

Basics – SQL Functions

- **Some Important SQL Functions and syntaxes:**
- Select:
  - Select All
    - SELECT * FROM “<schema_name>”.”<table_name>”;
  - Select some desired Columns
    - SELECT COL1 COL2 <…> COLn FROM “<schema_name>”.”<table_name>”;
  - Select count of all records
    - SELECT COUNT(*) FROM “<schema_name>”.”<table_name>”;
  - Select count of all records for a specific column
    - SELECT COUNT(column_name) FROM “<schema_name>”.”<table_name>”;
  - Select Unique records of a column
    - SELECT DISTINCT(column_name) FROM “<schema_name>”.”<table_name>”;
  - Select count of unique records of a column
    - SELECT COUNT(DISTINCT(column_name)) FROM “<schema_name>”.”<table_name>”;
  - Select based on “WHERE” clause
    - SELECT * FROM “<schema_name>”.”<table_name>” WHERE <column_name> <operator> <value>;
  - Order By
    - SELECT * FROM “<schema_name>”.”<table_name>” ORDER BY <column_name> (By default Ascending, else use ASC/DSC)

HANA SQL – Basics and Advanced

Basics – SQL Functions

- **Some Important SQL Functions and syntaxes:**
- Select:
  - Top N
    - SELECT TOP N FROM “<schema_name>”.”<table_name>”;
  - Limit the number of records retrieved from a SQL Query
    - SELECT * FROM “<schema_name>”.”<table_name>” LIMIT N;

HANA SQL – Stored Procedures

- A Stored procedure allows you to group the SQL statement into a single block.
- Procedures are re-useable processing block with a specific sequence of data transformation.
- Stored Procedures can return data in the form of output parameters (integer or character) or a cursor variable.
- Stored Procedures are also used for performance optimization as it contains series of SQL statements and results from one set of statement determines next set of statements to be executed.

- Syntax:

```
CREATE PROCEDURE <proc_name> [(<parameter_clause>)]
        [LANGUAGE <lang>] [SQL SECURITY <mode>]
        [DEFAULT SCHEMA <default_schema_name>]
        [READS SQL DATA [WITH RESULT VIEW <view_name>]]
         AS {
                BEGIN
                        [SEQUENTIAL EXECUTION] <procedure_body>
                END | HEADER ONLY
        }
```

HANA SQL – Stored Procedures

- Syntax Elements:

ELEMENTS	DESCRIPTION
<proc_name>	Procedure Name
<parameter_clause>	The parameter is defined here. IN, OUT, INOUT parameter is there. Each parameter is marked using the keywords IN/OUT/INOUT • IN – Used for Pass Value To procedure as INPUT. It is Read Only parameter. • OUT – Used for Return Value from Procedure as OUTPUT. • INOUT – Used for Pass and Return Value To Procedure by same parameter.
LANGUAGE <Lang>	Defines the programming language used in the procedure. Default: SQLSCRIPT
SQL SECURITY <mode>	Specifies the security mode of the procedure. Default: DEFINER • DEFINER - Specifies that the execution of the procedure is performed with the privileges of the definer of the procedure. • INVOKER - Specifies that the execution of the procedure is performed with the privileges of the invoker of the procedure.
<default_schema_name>	It defines the schema for unqualified objects in the procedure body. If nothing is define, then the current schema of the session is used for the procedure.
READS SQL DATA	It marks the procedure as being read-only, it means the procedure does not modify the database data or its structure and that the procedure does not contain DDL or DML statements. This procedure only calls other read-only procedures.
WITH RESULT VIEW <view_name>	It defines the result view to be used as the output of a read-only procedure. If a result view is specified for a procedure, then it can be called by an SQL statement in the same process as a table or view.
SEQUENTIAL EXECUTION	This statement will force sequential execution of the procedure logic. No parallelism takes place.
<procedure body>	It defines the main body of the procedure based on the programming language selected.
HEADER ONLY	If Header Only is used, then only procedure properties are created with OID.

HANA SQL – Stored Procedures

- **Definer Mode and Invoker Mode:**

- -- definer-mode procedure declared by USER_A

```
CREATE PROCEDURE USER_A.PROC1 LANGUAGE SQLSCRIPT SQL SECURITY DEFINER AS
BEGIN
        SELECT CURRENT_USER "current user" FROM DUMMY;
END;
```

- -- USER_B executing USER_A.PROC1

```
CALL USER_A.PROC1;
```

- -- current user

```
USER_A
```

- -- invoker-mode procedure declared by USER_A

```
CREATE PROCEDURE USER_A.PROC2 LANGUAGE SQLSCRIPT SQL SECURITY INVOKER AS
BEGIN
        SELECT CURRENT_USER "current user" FROM DUMMY;
END;
```

- -- USER_B is executing USER_A.PROC

```
CALL USER_A.PROC2;
```

- -- current user

```
USER_B
```

LTI

Let’s Solve