insightsoftware

Simba SDK

SQL Generator User Guide

Version 10.3

August 2024

Copyright

Contact Us

Insightsoftware

https://insightsoftware.com/contact-us/.

Table of Contents

Contact Us

For more information or help using this product, please contact our Technical Support staff. We welcome your questions, comments, and feature requests.

Note:

To help us assist you, prior to contacting Technical Support prepare a detailed summary of the client and server environment including operating system version, patch level, and configuration. Contact Technical Support using the following link: https://insightsoftware.com/contact-us/.

Third-Party Trademarks

Third Party Licenses

The licenses for the third-party libraries that are included in this product are listed below.

ICU License - ICU 1.8.1 and later

OpenSSL

Original SSLeay License

Expat License

Stringencoders License

dtoa License

CityHash License

http:code.google.com/p/cityhash/

Overview

The ODBC specification requires ODBC connectors to support an “SQL 92 like” syntax in order to query the underlying data source. However, some data sources (for example, custom data sources) may have their own query language which means that an ODBC connector needs to translate “SQL 92” queries into the data source specific query syntax.

The Simba SDK elegantly solves this issue by providing components to parse and tokenize an SQL-92 query and then convert those tokens into a query string of another language.

This feature, collectively known as the “SQL Generator,” is comprised of two main components of the Simba SDK:

- **PSParser**: a lexical parser which takes in an “SQL-92 like” query and generates a tree of parse nodes each of which represents a token of the input query. The PSParser returns the root node of the parse tree.

- **SQLGenerator**: takes in the parse node tree (for example, the root node) returned from the PSParser and generates a query string from the tree nodes in the desired query language. By default, the Simba SDK supports SQL 92 data sources and therefore includes a SQLGenerator implementation which generates SQL 92 queries. When adding support for other queries languages, you will either extend or replace this component with your implementation.

Figure 1 summarizes these components and shows how an SQL-92 query is translated into a custom query string:

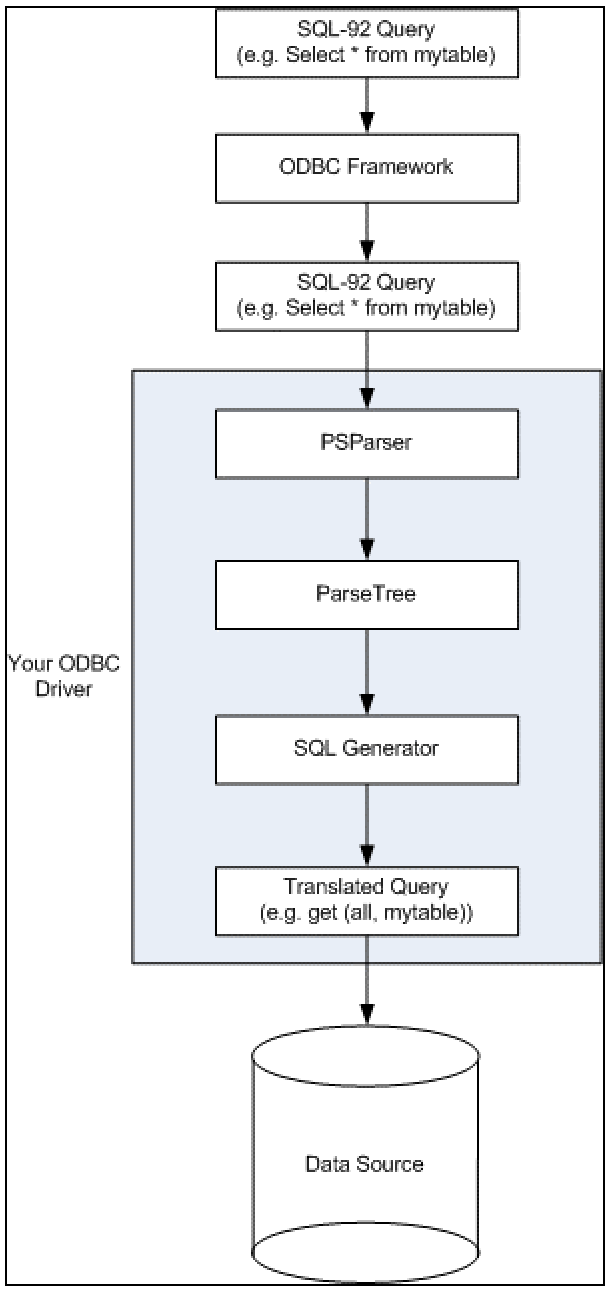

Figure 1: Summary of Components including the Input and Output Query Strings.

PSParser

The PSParser class is defined in PSParser.h located in `[INSTALL_ DIRECTORY]\SimbaEngineSDK\10.3\ DataAccessComponents\Include\SQLEngine\Parser` and contains only one public static method:

```
static PSRootParseNode* Parse(
                    Simba::DSI::IConnection* in_parentConnection,
                    const simba_wstring& in_sqlStatementText,
                    bool in_ignoreLimits = false);
```

Note:

In order to use the PSParser and related nodes in your project, you will need to link against the parser lib located in: `[INSTALL_DIRECTORY]\SimbaEngineSDK\10.3\DataAccessComponents\Lib\ [PLATFORM_CONFIGURATION]\[CONFIGURATION].`

For example:

```
C:\SimbaTechnologies\SimbaEngineSDK\10.3
\DataAccessComponents\Lib\Win32\Debug_MTDLL\Parser_Debug_MTDLL.lib.
```

The Parse() method serves as an entry point to parsing and takes in the following parameters:

- **in_parentConnection**: an IConnection object representing a DSN connection. This is a required component which is used by the PSParser to obtain the limits of the data source (e.g. the maximum length of a query, etc.), however, a simplified implementation can easily be provided as described in "Query Translator Example" on page 17.
- **in_sqlStatementText**: the query (for example, SQL 92 statement) to be parsed and converted into a parse tree.
- **in_ignoreLimits**: a flag indicating if the query limits defined in the connection properties should be checked or ignored.

Upon successfully parsing the query, PSParser will return a PSRootParseNode which is a wrapper class around the root node of the parse tree. This class contains accessors for the underlying root "PSParseNode", most notably the GetRootNode() method which returns the root PSParseNode of the tree.

The tree is comprised of various PSParseNode sub classed objects which represent and contain the information for each token of the query. These PSParseNodes are either "non-terminal" (has child nodes containing further information), or "terminal" (has no child nodes because there is no further information to store). For example, in the query `SELECT col1, col2 FROM t WHERE col1 < 1,` the literal node "1" would be terminal and the select list "(col1, col2)" would be non-terminal.

The Simba SDK provides the following PSParseNode subclasses that represent the components for SQL-92 queries:

- **PSFlagParseNode**: represents a quantifier (for example, ASC) specified in the query and has a PSDataType of PS_DT_FLAG.

- **PSNullParseNode**: an element that has not been specified in the query (for example, an optional parameter that was not specified in the input query). This node has a PSDataType of PS_DT_NULL.
- **PSIdentifierParseNode**: the node contains an identifier (for example, table name) and has a PSDataType of PS_DT_IDENTIFIER.
- **PSIntervalDataTypeParseNode**: the node contains the definition of a field where the type is an interval from the query (for example, interval year(3) to month). The node has a PSDataType of PS_DT_INTERVAL_DATATYPE.
- **PSIntervalLiteralParseNode**: the node contains a literal that was specified for an “interval” quantifier (for example, “1990” for a year interval) and has a PSDataType of PS_DT_ INTERVAL_LITERAL.
- **PSLiteralParseNode**: contains a literal (for example, string, character, number, etc.) that was specified in the query and has a PSDataType of PS_DT_LITERAL. The node also has a PSLiteralType indicating the type of literal contained in the node.
- **PSNonTerminalParseNode**: represents a non-terminal node in the parse tree which contains one or more child nodes. Non-terminal parse nodes form the majority of most parse trees because they represent the bulk of the query statements which need to store sub information in child nodes. PSNonTerminalParseNodes are assigned appropriate PSNonTerminalType enum by the parser.
- **PSParameterParseNode**: contains a function parameter and has a PSDataType of PS_DT_ PARAMETER.
- **PSParseNode**: the abstract base class from which all other parse nodes are derived from.
- **PSScalarOrAggrFnParseNode**: contains the definition of a function call to a scalar or aggregate function and has a PSDataType of PS_DT_SCALARORAGGRFN.

ParseNodes are hierarchically organized with parent PSParseNodes "owning" child ParseNodes in a tree. Note that ParseNodes are unidirectional from parent to child, cannot share children, and always form a tree (as opposed to a graph).

For example, a PSNonTerminalParseNode could represent a “SELECT” statement in which the child nodes contain further information such as the columns to select, the conditions in the “where” clause, etc. Some or all of the child nodes in turn may also be PSNonTerminalParseNodes.

The other PSParseNode types are all “terminal”. For example, a node of type PSFlagParseNode could represent an “ASC” keyword found in the query indicating that the results of an “order by” operation be returned in ascending order. Since there is no further “sub” information to record for an “ASC” keyword, a PSFlagParseNode (terminal node) is used to represent the keyword.

Each non-terminal node in the tree is assigned a PSNonTerminalType enum by the parser, accessible via the node’s GetNonTerminalType() method. For more information about these types see "Appendix E - PSNonTerminalTypes" on page 26.

Other PSParseNode “get” methods that provide type information include:

- **GetIntervalDataType()**: used for PSIntervalDataTypeParseNodes to specify interval information. See "Appendix C - PSIntervalDataTypes" on page 24 for more information.
- **GetLiteralType()**: used for PSLiteralParseNodes to specify the types of literals found in the query. See "Appendix B - PSLiteralTypes" on page 22for more information.
- **GetFlagValue()**: used for PSFlagParseNodes to indicate options found in the query (for example, “ASC” for ascending). See "Appendix A - PSFlagValues" on page 20 for more information.
- **GetDataType()**: used for all nodes to indicate a node’s type. See "Appendix D - PSDataTypes" on page 25 for more information.

The following diagram illustrates a simple tree generated for the query `SELECT column1 FROM table1 WHERE column1 > 0`:

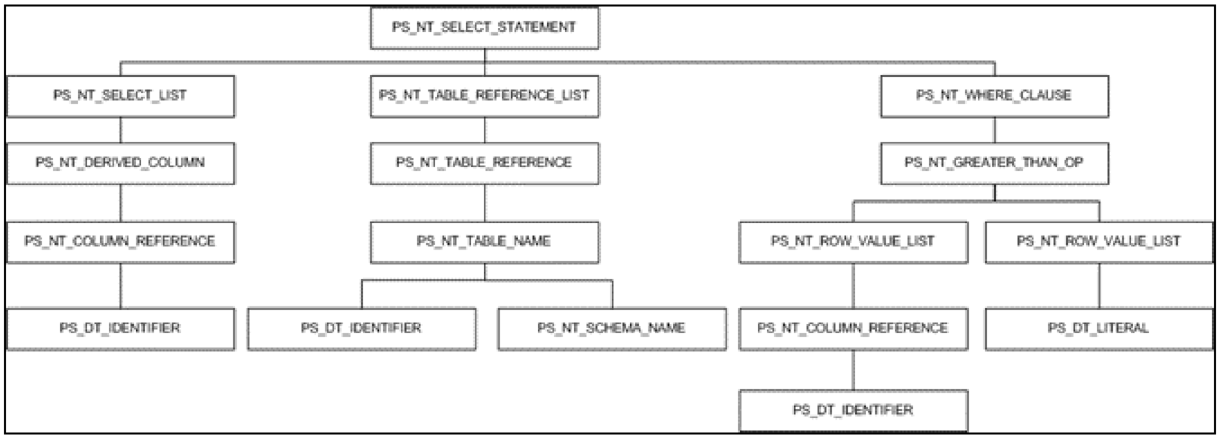

Figure 2 - Example Parse Tree for SELECT column1 from table1 where column1 > 0 with null nodes removed.

SQLGenerator

The Simba SDK includes a SQLGenerator implementation called “PSSql92Generator” which takes in a parse tree and generates an SQL 92 compliant query string.

This SQLGenerator is defined in `[INSTALL_DIRECTORY]\SimbaEngineSDK\10.3\DataAccessComponents\Include\SQLEngine\Parser\PSSql92Generator.h`.

Note:

in order to use PSSql92Generator, you will need to link against the core lib located in: `[INSTALL_DIRECTORY]\SimbaEngineSDK\10.3\DataAccessComponents\Lib\[PLATFORM_CONFIGURATION>\[BUILD]`.

For example:

```
C:\SimbaTechnologies\SimbaEngineSDK\10.3
\DataAccessComponents\Lib\Win32\Debug_MTDLL\Core_Debug_MTDLL.lib.
```

If the syntax supported by your underlying data source is similar to SQL 92, you may want to extend PSSql92Generator, as described below in "Query Translator Example" on the next page. If the syntax differs drastically from SQL 92, you should consider implementing the ISqlGenerator interface from scratch.

The notable methods of this interface are:

- **SetRootNode()**: takes in the PSRootParseNode generated by PSParser.
- **GenerateSqlStatement()**: generates a query string from the PSRootParseNode passed in using SetRootNode().

"Query Translator Example" on the next page describes the Query Translator example included with the Simba SDK.

Query Translator Example

The Simba SDK includes the “Query Translator” example which shows how to extend the PSSql92Generator to handle Transact SQL (T SQL). Since T SQL is similar to standard SQL 92 syntax, the PSSql92Generator was inherited from and extended into a new class called TransactSqlGenerator.

The example itself is a console application which links against the various Simba SDK libs including the core and parser libs.

The application takes in an SQL 92 query and a -log <log file> parameter. It then passes the query to PSParser for parsing, which returns a PSRootParseNode. This PSRootParseNode is then passed into TransactSqlGenerator’s GenerateSqlStatement() method which traverses the tree and returns the translated query:

```
int TranslateQuery(const std::vector<simba_wstring>& in_arguments)
{
        char* SimbaEngineDir = getenv("SIMBAENGINE_DIR");
        if (SimbaEngineDir)
        {
                simba_string errorPath(SimbaEngineDir);
                errorPath.append("\\ErrorMessages\\");
                SimbaSettingReader::SetErrorMessagesPath(errorPath);
        }
.
.
.

        const simba_wstring& sql = (in_arguments.size() == 1) ? in_arguments[0] : in_arguments[2];
.
.
.

        std::wcout << L"Parsing '" << sql.GetAsPlatformWString() << L"'" << std::endl;
        AutoPtr<Simba::SQLEngine::PSRootParseNode> root(Simba::SQLEngine::PSParser::Parse(
                &dummyConnection,
                sql));
.
.
.
```

```
        Simba::QueryTranslator::TransactSqlGenerator sqlGenerator;
sqlGenerator.SetRootNode(root.Get());
}
```

The output is then logged to the file specified in the -log command.

In order to satisfy the required IConnection parameter of PSParser::Parse(), the example defines a simple connection class called "DummyConnection" which implements DSIConnection:

```
// Fake connection just to satisfy PSParser interface.
class DummyConnection : public Simba::DSI::DSIConnection
{
public:
        DummyConnection() : Simba::DSI::DSIConnection(NULL, true) {}
private:
        virtual void Connect(const Simba::DSI::DSIConnSettingRequestMap& in_connectionSettings)
        {
                assert(false);
        }
        virtual Simba::DSI::IStatement* CreateStatement()
        {
                assert(false);
                return NULL;
        }
        virtual void Disconnect()
        {
                assert(false);
        }
        virtual bool PromptDialog(
                Simba::DSI::DSIConnSettingResponseMap& io_connResponseMap,
                Simba::DSI::DSIConnSettingRequestMap& io_connectionSettings,
                HWND in_parentWindow,
                Simba::DSI::PromptType in_promptType)
        {
                assert(false);
                return false;
        }
        virtual void UpdateConnectionSettings(
const Simba::DSI::DSIConnSettingRequestMap& in_connectionSettings,
                Simba::DSI::DSIConnSettingResponseMap& out_connectionSettings)
        {
                assert(false);
        }
} dummyConnection;
```

The bulk of the connection related functionality is defined in DSIConnection, while “stub” methods for operations that are not invoked by the generator (for example, connecting, displaying a prompt dialog etc.), are defined in the DummyConnection class.

Therefore, the DummyConnection class is a good “stub” class to use when working directly with a SQLGenerator.

Appendix A - PSFlagValues

PSFlagParseNodes represent optional parameters/arguments found in the query and have a PSFlagValue indicating what type of parameter/argument the node represents. The PSFlagValue enum values are as follows:

- **PS_FLAG_ALL**: “all” logical operator.
- **PS_FLAG_ALL_PRIVILEGES**: not currently supported.
- **PS_FLAG_AND**: “and” logical operator.
- **PS_FLAG_ANY**: “any” logical operator,
- **PS_FLAG_AS**: “as” conversion.
- **PS_FLAG_ASC**: “asc” (ascending) option.
- **PS_FLAG_CASCADE**: not currently supported.
- **PS_FLAG_DEFAULT**: not currently supported.
- **PS_FLAG_DELETE**: not currently supported.
- **PS_FLAG_DESC**: the node contains the “desc” (descending) option.
- **PS_FLAG_DISTINCT**: “distinct” quantifier.
- **PS_FLAG_EXCLUDE**: the value for a PS_NT_INCLUDE_NULLS flag node (child of PS_NT_UNPIVOT_CLAUSE non-terminal node) to specify whether the unpivot clause contained 'EXCLUDE NULLS' or 'INCLUDE NULLS'
- **PS_FLAG_INCLUDE**: the value for a PS_NT_INCLUDE_NULLS flag node (child of PS_NT_UNPIVOT_CLAUSE non-terminal node) to specify whether the unpivot clause contained 'EXCLUDE NULLS' or 'INCLUDE NULLS'
- **PS_FLAG_INVALID**: default enum value.
- **PS_FLAG_IS**: “is” logical operator.
- **PS_FLAG_NOT**: “not” logical operator.
- **PS_FLAG_NOT_NULL**: ”not null” logical operator.
- **PS_FLAG_NULL**: “null” keyword.
- **PS_FLAG_OR**: “or” logical operator.
- **PS_FLAG_PRIMARY_KEY**: “primary key” constraint.
- **PS_FLAG_PROCEDURE_RETURN_VALUE**: a return value for a stored procedure.

- **PS_FLAG_PUBLIC**: not currently supported.
- **PS_FLAG_RESTRICT**: not currently supported.
- **PS_FLAG_SELECT**: “select” statement.
- **PS_FLAG_SOME**: not currently supported.
- **PS_FLAG_STAR**: “*” in a sub select query.
- **PS_FLAG_TOP_PERCENT**: “top percent” select clause.
- **PS_FLAG_UNIQUE**: “unique” constraint.
- **PS_FLAG_USAGE**: not currently supported.
- **PS_FLAG_USER**: “user” object.

Appendix B - PSLiteralTypes

The PSLiteralType enumeration specifies what type of literal is contained in a PSLiteralParseNode (for example, a string, character, etc.). The enum can be one of the following values:

- **PS_LITERAL_APPROXNUM**: an approximate number (for example, float)
- **PS_LITERAL_BINARY**: a binary value (for example, bit).
- **PS_LITERAL_CASTFORMAT**: a format string specified in the FORMAT clause of CAST.
- **PS_LITERAL_CHARSTR**: character string.
- **PS_LITERAL_DATATYPE**: a SQL_TSI_... data type used in a timestamp operation.
- **PS_LITERAL_DATE**: a date (for example, yyyy-mm-dd
- **PS_LITERAL_DECIMAL**: a decimal value.
- **PS_LITERAL_DEFAULT**: the default enumeration value (set to PS_LITERAL_INVALID).
- **PS_LITERAL_GUID**: a GUID value specified by the “guid” keyword.
- **PS_LITERAL_INTERVAL_DAY**: a string literal containing the number of days for a “day” interval.
- **PS_LITERAL_INTERVAL_DAY_HOUR**: a string literal containing the number of days to convert into hours for a “day to hour” interval conversion.
- **PS_LITERAL_INTERVAL_DAY_MINUTE**: a string literal containing the number of days to convert into minutes for a “day to minute” interval conversion.
- **PS_LITERAL_INTERVAL_DAY_SECOND**: a string literal containing the number of days to convert into seconds for a “day to second” interval conversion.
- **PS_LITERAL_INTERVAL_HOUR**: a string literal containing the number of hours for an “hour” interval.
- **PS_LITERAL_INTERVAL_HOUR_MINUTE**: a string literal containing the number of hours to convert into minutes for an “hour to minute” interval conversion.
- **PS_LITERAL_INTERVAL_HOUR_SECOND**: a string literal containing the number of hours to convert into seconds for an “hour to second” interval conversion.
- **PS_LITERAL_INTERVAL_MINUTE**: a string literal containing the number of minutes for a “minute” interval.
- **PS_LITERAL_INTERVAL_MINUTE_SECOND**: a string literal containing the number of minutes to convert into seconds for a “minute to second” interval conversion.
- **PS_LITERAL_INTERVAL_MONTH**: a string literal containing the number of months for a “month” interval.

- **PS_LITERAL_INTERVAL_SECOND**: a string literal containing the number of seconds for a "second" interval.
- **PS_LITERAL_INTERVAL_YEAR**: a string literal containing the number of years for a "year" interval.
- **PS_LITERAL_INTERVAL_YEAR_MONTH**: a string literal containing the number of years to convert into months for a "year to month" interval conversion.
- **PS_LITERAL_INVALID**: default enumeration value.
- **PS_LITERAL_MAX = PS_LITERAL_NULL**: the "null" literal.
- **PS_LITERAL_MIN = PS_LITERAL_APPROXNUM**: approximate numeric literals.
- **PS_LITERAL_NULL**: a literal string containing the word "null"
- **PS_LITERAL_TIME**: a time value.
- **PS_LITERAL_TIMESTAMP**: a timestamp value.
- **PS_LITERAL_USINT**: an unsigned integer value.

Appendix C - PSIntervalDataTypes

PSIntervalDataTypeParseNodes have a PSIntervalDataType enum field specifying the interval data type they represent. The following are the PSIntervalDataType enum values:

- **PS_DATATYPE_INTERVAL_DAY**: an interval specified in days.
- **PS_DATATYPE_INTERVAL_DAY_HOUR**: day to hour conversion.
- **PS_DATATYPE_INTERVAL_DAY_MINUTE**: day to minute conversion.
- **PS_DATATYPE_INTERVAL_DAY_SECOND**: day to second conversion
- **PS_DATATYPE_INTERVAL_HOUR**: an interval specified in hours.
- **PS_DATATYPE_INTERVAL_HOUR_MINUTE**: hour to minute conversion.
- **PS_DATATYPE_INTERVAL_HOUR_SECOND**: hour to second conversion.
- **PS_DATATYPE_INTERVAL_MINUTE**: an interval specified in minutes
- **PS_DATATYPE_INTERVAL_MINUTE_SECOND**: minute to second conversion.
- **PS_DATATYPE_INTERVAL_MONTH**: an interval specified in months.
- **PS_DATATYPE_INTERVAL_SECOND**: an interval specified in seconds.
- **PS_DATATYPE_INTERVAL_YEAR**: an interval specified in years.
- **PS_DATATYPE_INTERVAL_YEAR_MONTH**: year to month conversion.

Appendix D - PSDataTypes

The PSDataType enumerations indicate what type of data is stored in the node and therefore represent a node's type.

- **PS_DT_FLAG**: the node is a PSFlagParseNode representing a quantifier (for example, distinct) specified in the query.
- **PS_DT_IDENTIFIER**: the node is a PSIdentifierParseNode containing the identifier (for example, table name) for an object specified in the query.
- **PS_DT_INTERVAL_DATATYPE**: the node is a PSIntervalDataTypeParseNode containing a field definition where the field type is an interval (for example, interval year(3) to month).
- **PS_DT_INTERVAL_LITERAL**: the node is a PSIntervalLiteralParseNode containing a literal that was specified for an "interval" quantifier (for example, "1990" for a year interval).
- **PS_DT_INVALID**: the default enumeration value used for nodes which haven't been assigned yet.
- **PS_DT_LITERAL**: the node contains a literal value such as a string, character, etc.
- **PS_DT_NULL**: the node does not contain any data or represent any operations. This type of node is often a "default" when an optional query element (for example, quantifier) has not been specified.
- **PS_DT_PARAMETER**: the node is a PSParameterParseNode representing a parameter in a prepared statement or procedure.
- **PS_DT_PARENT**: The datatype of non-terminal nodes.
- **PS_DT_SCALARORAGGRFN**: the node represents a scalar or aggregate function call.

Appendix E - PSNonTerminalTypes

The following summarizes the PSNonTerminalTypes enum values:

- **PS_NT_ACTION_LIST**: represents a list of actions which are contained in the PSFlagParseNode children, each with a PSNonTerminalType of PS_NT_ACTION_TYPE representing an action statement (for example, select, delete, or usage statement).
- **PS_NT_ACTION_TYPE**: represents an action type (select, delete, or usage) and has no child nodes.
- **PS_NT_ADD_COLUMN_DEFINITION**: represents an “add column” operation, where the column name to be added is defined in the PSNonTerminalParseNode child with a PSNonTerminalType of PS_NT_COLUMN_DEFINITION.
- **PS_NT_AGGR_EXPRESSION**: represents an aggregation in a PIVOT expression. Has two children, the first being the actual expression, and the second being an optional correlation specification.
- **PS_NT_AGGR_EXPRESSION_LIST**: represents the list of aggregation expressions in a PIVOT expression (PS_NT_PIVOT_CLAUSE), and has 1 or more PS_NT_AGGR_EXPRESSION children.
- **PS_NT_ALTER_TABLE_STATEMENT**: represents an “ALTER table” statement where the table and column names are specified in two PSNonTerminalParseNode children, with PSNonTerminalType types of PS_NT_TABLE_REFERENCE and PS_NT_ADD_COLUMN_DEFINITION respectively.
- **PS_NT_AND**: represents an “and” condition where both sides of the condition are represented in the two PSNonTerminalParseNodes children, each with the respective PSNonTerminalType.
- **PS_NT_AVG**: represents an “average” function where an optional set quantifier (for example, “distinct”) may be contained in a PSFlagParseNode child with the respective PSFlagValue (for example, PS_FLAG_DISTINCT), while the expression to be averaged is contained in a PSNonTerminalParseNode child with the respective PSNonTerminalType.
- **PS_NT_BETWEEN**: represents a “between” operator where the first child is the value to test, the second child is a PSParseFlagNode specifying if it’s a “not” condition, and the third child is a PSNonTerminalParseNode containing the end points of the range to test.
- **PS_NT_BINARY_MINUS_SIGN**: represents a minus operator where the two children represent the left and right values to use in the operation. The children can be PSNonTerminalParseNodes or terminal nodes such as PSLiteralParseNode.
- **PS_NT_BINARY_PLUS_SIGN**: represents a plus operator where the two children represent the left and right values to use in the operation. The children can be PSNonTerminalParseNodes or terminal nodes such as PSLiteralParseNode.
- **PS_NT_CHECK_CONSTRAINT_DEFINITION**: represents a check constraint being placed on a column where the boolean expression to constrain by is represented in the child PSNonTerminalParseNode

- **PS_NT_COALESCE**: represents a coalesce operation where the parameters are stored in the child PSNonTerminalParseNode with a PSNonTerminalType of PS_NT_COALESCE_LIST.

- **PS_NT_COALESCE_LIST**: contains the collection of value expressions for use in a coalesce operation, each stored in child PSNonTerminalParseNodes with a PSNonTerminalType of PS_NT_ROW_VALUE_LIST.

- **PS_NT_COLUMN_ALIAS_LIST**: represents an alias for a group in an UNPIVOT expression. Has one or more children representing a value expression.

- **PS_NT_COLUMN_CONSTRAINT_DEFINITION**: represents a column constraint of “not null” where the constraint is stored in the child PSNonTerminalParseNode with a PSNonTerminalType of PS_NT_NOT_NULL.

- **PS_NT_COLUMN_DEFINITION**: represents a column definition. The first child is a PSIdentifierParseNode which contains the column name. The second child is a PSNonTerminalParseNode of type PS_NT_DATA_TYPE containing the column data type. The third child is an optional PSNonTerminalParseNode of type PS_NT_COLUMN_ CONSTRAINT_DEFINITION which defines the column constraint

- **PS_NT_COLUMN_DEFINITION_LIST**: represents a collection of column definitions (for example, when creating a table) where each column is defined in a child PSNonTerminalParseNode of type PS_NT_COLUMN definition.

- **PS_NT_COLUMN_NAME**: represents a column name in a select list as a PSFlagParseNode with a PSFlagValue of PS_FLAG_STAR.

- **PS_NT_COLUMN_NAME_LIST**: represents a collection of column identifiers each contained in a PSIdentifierParseNode child.

- **PS_NT_COLUMN_REFERENCE**: represents a reference to an existing column (e.g in a select statement), with optional PSIdentifierParseNode nodes for the catalog, schema, and table, and a child PSIdentifierParseNode containing the column name.

- **PS_NT_COLUMN_REFERENCE_LIST**: represents a list of column references (used in PIVOT & UNPIVOT), has one or more PS_NT_COLUMN_REFERENCE children.

- **PS_NT_CONCAT_SIGN**: represents a concat operation (“||”) where both children are PSNonTerminalParseNodes each of type PS_NT_COLUMN_REFERENCE.

- **PS_NT_CONSTRAINT_NAME**: not currently supported.

- **PS_NT_COUNT**: represents a count aggregate function where an optional set quantifier (for example, “distinct”) may be contained in a PSFlagParseNode child with the respective PSFlagValue (for example, PS_FLAG_DISTINCT), while the expression to be counted is contained in a PSNonTerminalParseNode child with the respective PSNonTerminalType.

- **PS_NT_CREATE_INDEX_STATEMENT**: represents an index creation operation where an optional unique quantifier may be stored in the first (PSFlagParseNode) child with a PSFlagValue of PS_FLAG_UNIQUE. The second child contains an optional index type (for example, “clustered”). The third child is a PSIdentifierParseNode containing the index name.

The fourth child is a PSNonTerminalParseNode of type PS_NT_TABLE_NAME containing table information. The fifth child is a PSNonTerminalParseNode of type PS_NT_ORDER_COLUMN_LIST containing the collection of columns to index on.

- **PS_NT_CREATE_TABLE_AS_SELECT_STATEMENT**: represents a 'CREATE TABLE ... AS SELECT ...' statement. The first child will be a PS_NT_TABLE_NAME node containing the name of the table to create, the second child is an optional PS_NT_COLUMN_NAME_LIST containing the column names of the created table, and the third child represents the SELECT which is the initial value of the table.

- **PS_NT_CREATE_TABLE_STATEMENT**: represents a table creation operation where the first child is a PSNonTerminalParseNode of type PS_NT_TABLE_NAME containing table information, the second child is a PSNonTerminalParseNode of type PS_NT_COLUMN_DEFINITION_LIST containing the columns to create, and the third child is an optional PSNonTerminalParseNode of type PS_NT_TABLE_CONSTRAINT_DEFINITION_LIST which defines the constraints.

- **PS_NT_CREATE_VIEW_STATEMENT**: represents a view creation operation where the first child is a PSNonTerminalParseNode of type PS_NT_TABLE_NAME containing the table name to create the view on. The second child is an optional PSNonTerminalParseNode of type PS_NT_COLUMN_NAME_LIST containing the column names. The third child is a PSNonTerminalParseNode of type PS_NT_SELECT_STATEMENT containing the select statement tokens specified in the view creation statement.

- **PS_NT_CROSS_JOIN**: represents a cross join between two tables where references to the tables are specified in the child PSNonTerminalParseNodes each of the respective PSNonTerminalType (for example, PS_NT_TABLE_REFERENCE for a table join).

- **PS_NT_CUSTOM_AGGR**: represents a custom aggregate function. The first child is a PSScalarOrAggrFnParseNode containing the function name. The second child is a PSFlagParseNode containing the optional set quantifier (for example, distinct) with a respective PSFlagValue (for example, PS_FLAG_DISTINCT). The third child is a PSNonTerminalParseNode of type PS_NT_PARAMETER_LIST containing the parameters for the function such as column names.

- **PS_NT_DATA_TYPE**: specifies a data type (for example, when defining columns during the creation of a table). The first child is a PSIdentifierParseNode containing the data type name, and the second child is an optional PSNonTerminalParseNode of type PS_NT_DATA_TYPE_ATTRIBUTE_LIST containing attributes of the type.

- **PS_NT_DATA_TYPE_ATTRIBUTE_LIST**: specifies the attributes for a column definition which are contained in the child PSLiteralParseNodes.

- **PS_NT_DATE_LITERAL**: contains the literal for a date/time specified in the SQL statement.

- **PS_NT_DEFAULT_OPTION_FLAG**: Not currently supported by the parse

- **PS_NT_DELETE_STATEMENT_SEARCHED**: represents a “delete” statement where the first child is a PSNonTerminalParseNode of type PS_NT_TABLE_NAME containing the table name to delete, and the second (optional) child is a PSNonTerminalParseNode of type PS_

NT_WHERE_CLAUSE which contains the where clause information, if specified in the SQL query.

- **PS_NT_DERIVED_COLUMN**: represents a derived column (i.e. a expression that represents multiple columns). The first child is a PSNonTerminalParseNode of type PS_NT_COLUMN_REFERENCE which references the column to derive. The second child is a PSIdentifierParseNode containing the identifier of the column name to derive.

- **PS_NT_DIVISION_SIGN**: represents a division operator where the two children represent the left and right values to use in the operation. The children can be PSNonTerminalParseNodes or terminal nodes such as PSLiteralParseNode.

- **PS_NT_DROP_BEHAVIOR**: contains the specified behavior to apply in a “drop view” operation (for example, cascade). This node is a PSFlagParseNode and has no children.

- **PS_NT_DROP_INDEX_STATEMENT**: represents a “drop index” operation where the first child is a PSIdentifierParseNode containing the index name to drop, and the second child is a PSNonTerminalParseNode of type PS_NT_TABLE_NAME containing the name of the table to perform the operation on.

- **PS_NT_DROP_TABLE_STATEMENT**: represents a “drop table” operation where the child is a PSNonTerminalParseNode of type PS_NT_TABLE_NAME and contains the name of the table to drop.

- **PS_NT_DROP_VIEW_STATEMENT**: represents a “drop view” operation. The first child is a PSNonTerminalParseNode of type PS_NT_TABLE_NAME which specifies the view to remove. The second child is a PSFlagParseNode which specifies the behavior to apply for the operation (for example, for cascade, the flag will be PS_FLAG_CASCADE).

- **PS_NT_ELSE_CLAUSE**: represents an “else” clause in a case statement. The child contains a value expression for the else branch of the condition.

- **PS_NT_EQUALS_OP**: represents an “equals” operation where the two child nodes are both PSNonTerminalParseNodes of type PS_NT_ROW_VALUE_LIST. Each child specifies a row value to compare on the respective side of the operator. This node is also used as a child of PS_NT_SET_CLAUSE_LIST, where it represents the assignment in an update (for example, ‘update T1 set C1 = 5’)

- **PS_NT_EXCEPT**: represents an “except” statement where the two child nodes are both PSNonTerminalParseNodes each containing an expression (for example, of type PS_NT_SELECT_STATEMENT).

- **PS_NT_EXCEPT_ALL**: represents an “except all” statement where the two child nodes are both PSNonTerminalParseNodes each containing an expression (for example, of type PS_NT_SELECT_STATEMENT for select statements).

- **PS_NT_EXISTS**: represents an “exists” condition where the child node is a PSNonTerminalParseNode containing the expression for the condition (for example, of type PS_NT_SELECT_STATEMENT for a sub select statement).

- **PS_NT_FULL_OUTER_JOIN**: represents a "full outer join" between two tables where references to the tables are specified in the first two child PSNonTerminalParseNodes. The third child is a PSNonTerminalParseNode containing the boolean expression to join on and is of the respective type (for example, PS_NT_EQUALS_OP for an "equals" comparision).

- **PS_NT_FUNC**: represents a function (for example, a custom function invoked in a select statement). The first child is a PSScalarOrAggrFnParseNode containing the function name, and the second child is a PSNonTerminalParseNode of type PS_NT_PARAMETER_LIST containing the parameters passed into the function.

- **PS_NT_GRANT_STATEMENT**: represents a grant statement where the first child is a PSNonTerminalParseNode of the type PS_NT_ACTION_LIST containing the privileges to grant or PS_NT_PRIVILEGES (if "all privileges" was specified), the second child is a PSNonTerminalParseNode of type PS_NT_TABLE_NAME containing the table to grant privileges on, and the third child is a PSNonTerminalParseNode of type PS_NT_GRANTEE_LIST containing the users or roles to grant permissions to.

- **PS_NT_GRANTEE**: represents a user or role to grant permission to in a grant statement. Contains one child node that can either be a PSIdentifierParseNode containing the name of the user or role to grant permissions to, or a PSFlagParseNode of type PS_FLAG_PUBLIC indicating that access is to be granted to "public".

- **PS_NT_GRANTEE_LIST**: contains the list of users or roles to grant permissions to in a grant statement. Each child is a PSNonTerminalParseNode of type PS_NT_GRANTEE containing a user or object to grant permission to.

- **PS_NT_GREATER_THAN_OP**: represents a "greater than" condition where both children are PSNonTerminalParseNodes of type PS_NT_ROW_VALUE_LIST containing the values to compare on.

- **PS_NT_GREATER_THAN_OR_EQUALS_OP**: represents a "greater than or equal to" condition where both children are PSNonTerminalParseNodes of type PS_NT_ROW_VALUE_LIST containing the values to compare on.

- **PS_NT_GROUP_BY**: represents a "group by" operation where the grouping columns are specified in the child PSNonTerminalParseNode of type PS_NT_ROW_VALUE_LIST.

- **PS_NT_GROUP_DEF**: represents one UNPIVOT group. Has a PS_NT_COLUMN_REFERENCE_LIST child defining the column or columns in the group, and an optional PS_NT_COLUMN_ALIAS_LIST child defining an alias for the group.

- **PS_NT_GROUP_DEF_LIST**: represents the groups defined in an UNPIVOT expression (PS_NT_UNPIVOT_CLAUSE). It has one or more PS_NT_GROUP_DEF children.

- **PS_NT_GROUPBY_EXPRESSION_LIST**: contains the columns for a "group by" operation where each column is specified in a child PSNonTerminalParseNode.

- **PS_NT_GUID_LITERAL**: contains a GUID literal that was specified in a query. The literal is contained within the child PSLiteralParseNode of type PS_LITERAL_GUID.

- **PS_NT_HAVING**: represents a “having” operation where the boolean expression is contained in the child PSNonTerminalParseNode of the respective boolean PSNonTerminalType (for example, PS_NT_GREATHER_THAN_OP).

- **PS_NT_IF**: represents the 'IF' scalar function. The first child is a boolean expression, the second is an expression which should be returned if the boolean expression evaluates to true, and the third is an expression which should be returned if the boolean expression evaluates to false.

- **PS_NT_IN**: represents an “in” constraint for a “where” condition. The first child is a PSNonTerminalParseNode of type PS_NT_ROW_VALUE_LIST which specifies the columns to compare rows by. The second child is a PSParseFlagNode specifying the conditional expression to compare on (for example, PS_NT_IS_OR_IS_NOT). The third child is a PSNonTerminalParseNode containing the collection items to search in which could be a value list or subquery.

- **PS_NT_INNER_JOIN**: represents an “inner join” between two tables where references to the tables are specified in the first two child PSNonTerminalParseNodes each of the respective PSNonTerminalType (for example, PS_NT_TABLE_REFERENCE for a table join). The third child is a PSNonTerminalParseNode containing the boolean expression to join on and is of the respective type (for example, PS_NT_EQUALS_OP for an “equals” comparision).

- **PS_NT_INSERT_ACTION**: represents an “insert” action on which permission is being granted as part of a “grant” operation. The child can optionally be a PSNonTerminalParseNode of type PS_NT_COLUMN_NAME_LIST specifying the columns to grant insert permissions for.

- **PS_NT_INSERT_LIST**: represents the items to insert, in an “insert into” operation. The first child is a PSNonTerminalParseNode of type PS_NT_COLUMN_NAME_LIST specifying the columns to insert into, and the second child is a PSNonTerminalParseNode containing the values to insert into the columns or a select statement.

- **PS_NT_INSERT_STATEMENT**: represents an “insert into” operation where the first child is a PSNonTerminalParseNode of type PS_NT_TABLE_NAME specifying the table to insert into and the second child is a PSNonTerminalParseNode of type PS_NT_INSERT_LIST containing the columns and values to insert.

- **PS_NT_INTERSECT**: represents an INTERSECT operation. Has two children, one for each relation to intersect.

- **PS_NT_INTERSECT_ALL**: represents an INTERSECT ALL operation. Has two children, one for each relation to intersect.

- **PS_NT_INVALID**: a default value used for indicating that the current node is not a non terminal.

- **PS_NT_IS_OR_IS_NOT**: represents an is/is not condition from the query (for example, for a between condition, like, in, and null predicate). The node is a PSFlagParseNode of type PS_NT_IS_OR_IS_NOT and has no children. The PSFlagValue will be set to PS_FLAG_IS or PS_FLAG_NOT depending on the condition specified in the query.

- **PS_NT_LEFT_OUTER_JOIN**: represents a “left outer join” between two tables where references to the tables are specified in the child PSNonTerminalParseNodes each of the respective PSNonTerminalType (for example, PS_NT_TABLE_REFERENCE for a table join). The third child is a PSNonTerminalParseNode containing the boolean expression to join on and is of the respective type (for example, PS_NT_EQUALS_OP for an “equals” comparison).

- **PS_NT_LESS_THAN_OP**: represents a “less than” operation where the two child nodes are both PSNonTerminalParseNodes of type PS_NT_ROW_VALUE_LIST. Each child specifies a row value to compare on the respective side of the operator.

- **PS_NT_LESS_THAN_OR_EQUALS_OP**: represents a “less than or equals” operation where the two child nodes are both PSNonTerminalParseNodes of type PS_NT_ROW_VALUE_LIST. Each child specifies a row value to compare on the respective side of the operator.

- **PS_NT_LIKE**: represents a “like” condition where the first child is a PSNonTerminalParseNode containing the expression to evaluate (for example, column name), the second child is a PSFlagParseNode indicating the condition (for example, has a PSFlagValue of PS_FLAG_IS), the third child is a PSNonTerminalParseNode containing the value to compare against, and the fourth node is an optional PSLiteralParseNode containing an optional escape character (i.e. wildcard), if specified in the query.

- **PS_NT_LIMIT**: used internally by the parser only.

- **PS_NT_LIMIT_SKIP**: used internally by the parser only.

- **PS_NT_MAX**: represents a “min” function where an optional set quantifier (for example, “distinct”) may be contained in a PSFlagParseNode child with the respective PSFlagValue (for example, PS_FLAG_DISTINCT), while the expression to perform the operation on is contained in a PSNonTerminalParseNode child with the respective PSNonTerminalType.

- **PS_NT_MEASURE_COLUMN_LIST**: represents the list of measure columns in an UNPIVOT (PS_NT_UNPIVOT_CLAUSE). Will have one or more PSIdentifierParseNode children.

- **PS_NT_MERGE_OPERATION_SPECIFICATION**: represents the operation an MERGE SQL statement should do. Has two children, the first being a PS_NT_MERGE_WHEN_MATCHED_CLAUSE, and the second being a PS_NT_MERGE_WHEN_NOT_MATCHED_CLAUSE, one of which may not be present (replaced by a PSNullParseNode).

- **PS_NT_MERGE_STATEMENT**: represents a MERGE SQL statement. The first child will be a PS_NT_TABLE_NAME node containing the target table for the merge, the second is an optional PSIdentifierParseNode with an alias for the target table, the third is a PS_NT_TABLE_REFERENCE containing the source table for the merge, the fourth is the boolean expression from the ON clause, and the fifth is a PS_NT_MERGE_OPERATION_SPECIFICATION detailing what exactly the merge should do.

- **PS_NT_MERGE_WHEN_MATCHED_CLAUSE**: represents what a MERGE SQL statement should do if a matching row is found. Has a single PS_NT_SET_CLAUSE_LIST child which describes what to set when a row matches the condition.

- **PS_NT_MERGE_WHEN_NOT_MATCHED_CLAUSE**: represents what a MERGE SQL statement should do if no matching row is found for a row in the target table. Has two children, the first being an optional PS_NT_COLUMN_NAME_LIST which specifies the columns being inserted into, and the second being a PS_NT_ROW_VALUE_LIST containing the values to insert.

- **PS_NT_MIN**: represents a “max” function where an optional set quantifier (for example, “distinct”) may be contained in a PSFlagParseNode child with the respective PSFlagValue (for example, PS_FLAG_DISTINCT), while the expression to perform the operation on is contained in a PSNonTerminalParseNode child with the respective PSNonTerminalType.

- **PS_NT_MULTIPLICATION_SIGN**: represents a multiplication operator where the two children represent the left and right values to use in the operation. The children can be PSNonTerminalParseNodes or terminal nodes such as a PSLiteralParseNode.

- **PS_NT_NATIVE_QUERY**: represents a 'native' query using the {NATIVE ...} escape syntax. The first child is PS_LITERAL_CHARSTR litral node containing the text of the native query, the second is an optional PS_NT_PASSING_CLAUSE_DEFINITION node representing the PASSING clause from the native escape, and the third is a PS_NT_COLUMN_DEFINITION_LIST node describing the columns returned from the native query (for example, GREATER_THAN_OP, etc.).

- **PS_NT_NATIVE_VALUE** represents a 'native' expression using the {NATIVE ... RETURNING ...} escape syntax. The first child is PS_LITERAL_CHARSTR litral node containing the text of the native expression, the second is an optional PS_NT_PASSING_CLAUSE_DEFINITION node representing the PASSING clause from the native escape, and the third is a PS_NT_RETURNING_CLAUSE node describing the type returned by the native expression.

- **PS_NT_NOT**: represents a “not” condition where the condition is contained in the child PSNonTerminalParseNode, with a boolean PSNonTerminalType (for example, PS_NT_LESS_THAN_OP, PS_NT_GREATER_THAN_OP etc.).

- **PS_NT_NOT_EQUALS_OP**: represents a “not equals” operation where the two child nodes are both PSNonTerminalParseNodes of type PS_NT_ROW_VALUE_LIST. Each child specifies a row value to compare on the respective side of the operator.

- **PS_NT_NOT_NULL**: represents a “not null” constraint specified in the creation of a table. This is a PSFlagParseNode of type PS_FLAG_NOT_NULL and it has no children.

- **PS_NT_NOVALUE**: default value for PSNonTerminalType nodes.

- **PS_NT_NULL**: represents a test for “null” (i.e. an “is null” statement within a where clause) where the first child is a PSNonTerminalParseNode of type PS_NT_ROW_VALUE_LIST containing the columns to test, and the second child is a PSFlagParseNode of type PS_NT_IS_OR_IS_NOT with a PSFlagValue of PS_FLAG_IS or PS_FLAG_NOT.

- **PS_NT_NULLIF**: represents a “nullif” expression where the two children are PSNonTerminalParseNodes of type PS_NT_ROW_VALUE_LIST specifying the expressions to test for equality.

- **PS_NT_OR**: represents an “or” condition where both sides of the condition are contained in the (two) PSNonTerminalParseNodes children, each of a boolean PSNonTerminalType (for example, PS_NT_LESS_THAN_OP, PS_NT_GREATER_THAN_OP etc.).

- **PS_NT_ORDER_BY**: specifies an “order by” condition for an operation. The child is a PSNonTerminalParseNode of type PS_NT_SORT_SPECIFICATION_LIST which contains the collection of columns to order by.

- **PS_NT_ORDER_COLUMN**: contains a column to sort on in a “create index” statement. The first child is a PSIdentiferParseNode containing the column to sort on. The second (optional) child is a PSFlagParseNode representing the sort order, with the respective PSFlagValue, if specified (for example, PS_FLAG_ASC for ascending order).

- **PS_NT_ORDER_COLUMN_LIST**: contains the list of columns specified in a “create index” statement. Each child is a PSNonTerminalParseNode of type PS_NT_ORDER_COLUMN specifying a column to sort on.

- **PS_NT_ORDERING_SPECIFICATION_OPT**: not currently supported.

- **PS_NT_OUTER_JOIN_VT**: represents an outer join escape sequence where the child is a PSNonTerminalParseNode with the respective join PSNonTerminalType.

- **PS_NT_PARAMETER**: not currently supported.

- **PS_NT_PARAMETER_LIST**: contains the list of parameters that were passed into a scalar function or stored procedure within the query. Each parameter is in a PSNonTerminalParseNode specifying the value or name of a column to perform the function on.

- **PS_NT_PASSING_CLAUSE_DEFINITION**: represents a value being passed to the native query in a native query escape. Has 3 children, the first being an expression defining the value that's being passed, the second being an optional PS_NT_DATA_TYPE or PSIntervalDataTypeParseNode describing the type to pass the value as, and the third being a PS_NT_COLUMN_DEFINITION_LIST node describing the columns returned.

- **PS_NT_PASSING_CLAUSE_LIST**: represents the PASSING clause from a native query escape, and has one or more PS_NT_PASSING_CLAUSE_DEFINITION children.

- **PS_NT_PIVOT_CLAUSE**: represents a PIVOT operation. The first child is a PS_NT_AGGR_EXPRESSION_LIST representing the aggregations to do in the pivot, the second is a PS_NT_COLUMN_REFERENCE_LIST representing the column(s) the pivot values are coming from, the third is a PS_NT_PIVOT_COLUMN_LIST representing the pivot columns/values, the fourth is an optional PSIdentifierParseNode containing an alias for the result of the pivot, and the fifth is an optional PS_NT_COLUMN_NAME_LIST containing renamed columns for the result of the pivot.

- **PS_NT_PIVOT_COLUMN**: represents a pivot column in a PIVOT. Has two children, the first being a PS_NT_VALUE_LIST representing the value(s) for that pivot column (each child can be either a general expression, or a PSIdentifierParseNode, in which case it should be treated as a string value), and the second being an optional alias (as a PSIdentifierParseNode).

- **PS_NT_PIVOT_COLUMN_LIST**: represents the list of pivot columns in a PIVOT (PS_NT_PIVOT_CLAUSE). It has one or more PS_NT_PIVOT_COLUMN children.

- **PS_NT_PRIVILEGES**: represents “all privileges” within a “grant” statement. Not currently supported by the SQLGenerator.

- **PS_NT_PROCEDURE**: represents a stored procedure that is invoked in a query. The first child is a PSNonTerminalParseNode of type PS_NT_PROCEDURE_NAME containing the name of the procedure being invoked. The second child is an optional PSNonTerminalParseNode of type PS_NT_PARAMETER_LIST containing the parameters passed into the procedure, if specified in the query.

- **PS_NT_PROCEDURE_CALL**: represents the invocation of a stored procedure. The first child is an optional PSParameterParseNode containing the return value if specified. The second child is a PSNonTerminalParseNode of type PS_NT_PROCEDURE containing the procedure to be invoked.

- **PS_NT_PROCEDURE_NAME**: contains information related to the name of a procedure that is being invoked in a query. The first child is a PSNonTerminalParseNode of type PS_NT_SCHEMA_NAME containing the schema in which the stored procedure is contained. The second child is a PSIdentifier node containing the procedure name.

- **PS_NT_QUANTIFIED_COMPARISON_PREDICATE**: contains a comparison predicate specified in a subquery (for example, “any”, “all”, or “some”). The first child is a PSNonTerminalParseNode containing the operator used on the left hand side of the comparison and is of the respective type (for example, PS_NT_GREATER_THAN_OP). The second child is a PSParseFlagNode specifying the quantifier type and has the respective PSFlagValue corresponding to the specified quantifier (for example, PS_FLAG_ALL).

- **PS_NT_QUANTIFIER**: represents an “any” quantifier used in a where clause. This is a PSParseFlagNode with a PSFlagValue of PS_NT_QUANTIFIER.

- **PS_NT_REFERENCES_ACTION**: represents a “references” action on which permission is being granted as part of a “grant” operation. The child can optionally be a PSNonTerminalParseNode of type PS_NT_COLUMN_NAME_LIST specifying the columns to grant references permissions for.

- **PS_NT_REFERENCES_SPECIFICATION**: contains a “references” statement as part of a foreign key definition. The first child is a PSNonTerminalParseNode of type PS_NT_TABLE_REFERENCE containing the foreign table being referenced. The second child is an optional PSNonTerminalParseNode of type PS_NT_COLUMN_NAME_LIST containing the columns in the table to set the foreign key on.

- **PS_NT_REFERENTIAL_CONSTRAINT_DEFINITION**: not currently supported.

- **PS_NT_REFERENTIAL_CONSTRAINT_DEFINITION**: not currently supported.

- **PS_NT_RETURN_ESCAPE**: represents a DML statement using the {RETURN <columns> FROM <DML QUERY>} escape syntax. The first child is a PS_NT_COLUMN_NAME_LIST representing the columns to be returned by the statement for updated/deleted/inserted rows, and the second is the parse tree for the DML statement.

- **PS_NT_RETURNING_CLAUSE**: used as a child of PS_NT_NATIVE_VALUE to indicate the data type being returned by the native expression. It has one child, either PS_NT_DATA_TYPE or PSIntervalDataTypeParseNode.

- **PS_NT_RIGHT_OUTER_JOIN**: represents a "right outer join" between two tables where references to the tables are specified in the first two child PSNonTerminalParseNodes. The third child is a PSNonTerminalParseNode containing the boolean expression to join on and is of the respective type (for example, PS_NT_EQUALS_OP for an "equals" comparision).

- **PS_NT_ROW_VALUE_LIST**: a general purpose node used for containing row values (for example, values to compare in an expression). The children are PSNonTerminalParseNodes with value expressions. In some contexts these can even be default/null flag nodes.

- **PS_NT_SCALAR_OR_AGGR_FN**: indicates that a scalar or aggregate function was found in the query. The child node is a PSNonTerminalParseNode of type PS_NT_FUNC or PS_NT_CUSTOM_AGGR which contains information about the respective type of function.

- **PS_NT_SCHEMA_NAME**: represents the name of a schema which forms part of a table name reference (for example,catalogname.schemaname.tablename) where the first child is an optional PSNonTerminalParseNode of type PSIdentifierParseNode containing the schema name if specified, and the second child is an optional PSNonTerminalParseNode of type PSIdentifierParseNode containing the table name if specified.

- **PS_NT_SEARCHED_CASE**: represents a searched case operation where the first child is a PSNonTerminalParseNode of type PS_NT_SEARCHED_WHEN_CLAUSE_LIST containing the when clause elements, and the second child is an optional PSNonTerminalParseNode of type PS_NT_ELSE_CLAUSE containing the else condition of the clause, if specified.

- **PS_NT_SEARCHED_WHEN_CLAUSE**: represents a when clause in a search case operation. The first child is a PSNonTerminalParseNode containing the boolean operation and is of the respective non terminal type (for example, PS_NT_LESS_THAN_OR_EQUALS_OP). The second child contains the value for the boolean expression and is of the respective type (for example, a PSLiteralParseNode containing a literal).

- **PS_NT_SEARCHED_WHEN_CLAUSE_LIST**: contains a list of clauses for a searched case operation where each child is a PSNonTerminalParseNode of type PS_NT_SEARCHED_WHEN_CLAUSE.

- **PS_NT_SELECT_LIMIT**: represents a limit condition in a select statement. The first child is a PSLiteralParseNode containing the value to limit by. The second child is an optional PSNonTerminalParseNode of type PS_NT_TOP_PERCENT_OPT containing the optional top percent constraint, if specified.

- **PS_NT_SELECT_LIMIT_SKIP**: same as PS_NT_SELECT_LIMIT, but the second child is a PSLiteralParseNode that represents the number of rows to skip at the beginning of the resultset.

- **PS_NT_SELECT_LIST**: contains the list of columns specified in a select statement. Each child is a PSNonTerminalParseNode of type PS_NT_COLUMN_REFERENCE OR PS_NT_DERIVED_COLUMN depending on whether the column is from the table or derived.

- **PS_NT_SELECT_STATEMENT**: represents a select operation. The first child is an optional PSNonTerminalParseNode containing the set quantifier, if specified. The second child is an optional PSNonTerminalParseNode of type PS_NT_SELECT_LIMIT containing the limit, if specified. The third child is a PSNonTerminalParseNode of type PS_NT_SELECT_LIST containing the columns to select. The fourth child is a PSNonTerminalParseNode of type PS_NT_TABLE_REFERENCE_LIST containing the tables specified in the select statement. The fifth child is an optional PSNonTerminalParseNode of type PS_NT_WHERE_CLAUSE containing the "where" condition if specified. The sixth child is a PSNonTerminalParseNode of type PS_NT_GROUP_BY containing the "group by" information if specified. The seventh child is an optional PSNonTerminalParseNode of type PS_NT_HAVING containing the having condition information.

- **PS_NT_SET_CATALOG**: represents a "set catalog" operation. The child is a PSLiteralParseNode containing the name of the catalog.

- **PS_NT_SET_CLAUSE_LIST**: contains the operations for a set clause. Each child is a PSNonTerminalParseNode of type PS_NT_EQUALS_OP specifying an assignment operation for the set clause.

- **PS_NT_SET_SCHEMA**: represents a "set schema" operation. The child is a PSLiteralParseNode containing the schema name.

- **PS_NT_SET_QUANTIFIER_OPT**: represents a set quantifier for an operation. The node type is a PSFlagParseNode with the respective PSFlagValue.

- **PS_NT_SIMPLE_CASE**: represents a simple case statement. The first child is a PSNonTerminalParseNode containing the value expression to which the statement applies. The second child is a PSNonTerminalParseNode of type PS_NT_WHEN_CLAUSE_LIST representing the statement's when clause. The third child is an optional PSNonTerminalParseNode of type PS_NT_ELSE_CLAUSE containing the else condition of the case statement, if specified.

- **PS_NT_SIMPLE_WHEN_CLAUSE**: contains a condition of a when clause. The first child contains the expression to compare with, and the second child contains the result if the comparison succeeds.

- **PS_NT_SIMPLE_WHEN_CLAUSE_LIST**: contains a collection of when clauses where each child is a PSNonTerminalParseNode of type PS_NT_SIMPLE_WHEN_CLAUSE and contains a when clause.

- **PS_NT_SORT_SPECIFICATION**: contains a piece of sorting information for an "order by" clause. The first child contains the expression to sort by (for example, column name). The second (optional) child, is a PSFlagParseNode which contains the sorting option, if specified, (for example, "asc") and has the respective PSFlagValue type.

- **PS_NT_SORT_SPECIFICATION_LIST**: represents the sorting information for an "order by" clause. Each piece of sorting information is stored in a child PSNonTerminalParseNode of type PS_NT_SORT_SPECIFICATION.

- **PS_NT_SORTED_SELECT_STATEMENT**: represents a sub query (select) with an "order by" clause. The first child is a PSNonTerminalParseNode of type PS_NT_SELECT_

STATEMENT representing the select clause and the second child is a PSNonTerminalParseNode of type PS_NT_ORDER_BY representing the order by clause.

- **PS_NT_STDDEV**: represents a "standard deviation" operation where the first child is an optional PSFlagParseNode specifying a set quantifier (for example, distinct) with the respective PSFlagValue (for example, PSFlagDistinct), and the second child is a PSNonTerminalParseNode containing the value expression to perform the operation on.

- **PS_NT_STDDEV_POP**: represents a "standard deviation pop" operation where the first child is an optional PSFlagParseNode specifying a set quantifier (for example, distinct) with the respective PSFlagValue (for example, PSFlagDistinct), and the second child is a PSNonTerminalParseNode containing the value expression to perform the operation on.

- **PS_NT_SUM**: represents a "sum" operation where the first child is an optional PSFlagParseNode specifying a set quantifier (for example, distinct) with the respective PSFlagValue (for example, PSFlagDistinct), and the second child is a PSNonTerminalParseNode containing the value expression to perform the operation on.

- **PS_NT_TABLE_CONSTRAINT_DEFINITION**: contains the definition of a table constraint for a table creation operation (for example, defining a primary key). The first child is a PSFlagParseNode of type PS_NT_UNIQUE_SPECIFICATION and the second child is a PSNonTerminalParseNode of type PS_NT_COLUMN_NAME_LIST containing the column names involved in the constraint.

- **PS_NT_TABLE_CONSTRAINT_DEFINITION_LIST**: contains the collection of table constraints specified in a table creation statement. Each child is a PSNonTerminalParseNode of type PS_NT_TABLE_CONSTRAINT_DEFINITION containing a constraint definition.

- **PS_NT_TABLE_NAME**: represents a table specified in the query. The first child is a PSNonTerminalParseNode of type PS_NT_SCHEMA_NAME which contains the table schema name and the second child is a PSIdentifierParseNode containing the name of the table.

- **PS_NT_TABLE_REFERENCE**: represents a reference to an existing table specified in the query. The first child is a PSNonTerminalParseNode of type PS_NT_TABLE_NAME which contains the table information. The second child is an optional PSIdentifierParseNode containing the table alias, if specified. The third child is an optional PSNonTerminalParseNode of type PS_NT_COLUMN_NAME_LIST containing any column identifiers specified.

- **PS_NT_TABLE_REFERENCE_LIST**: contains a collection of table references in a "from" clause from a "select" statement, where each child is a PSIdentifierParseNode of type PS_NT_TABLE_REFERENCE.

- **PS_NT_TABLE_VALUE_LIST**: contains a collection of row value lists to be inserted into a table. Each child is a PSNonTerminalParseNode of type PS_NT_ROW_VALUE_LIST containing the column values to insert.

- **PS_NT_TIME_LITERAL**: represents a "time" literal (for example, select TIME '10:50:00') where the child is a PSLiteralParseNode of type PS_LITERAL_TIME containing the specified time.

- **PS_NT_TIMESTAMP_LITERAL**: represents a “timestamp” literal (for example, select TIMESTAMP ’10:50:00’) where the child is a PSLiteralParseNode of type PS_LITERAL_TIME containing the specified time.

- **PS_NT_TOP_LEVEL_SELECT_STATEMENT**: represents a select statement in a select operation where “order by” is specified. The first child node is a PSNonTerminalParseNode of type PS_NT_SELECT_STATEMENT which contains the select statement. The second child is a PSNonTerminalParseNode of type PS_NT_ORDER_BY containing the order by clause.

- **PS_NT_TOP_PERCENT_OPT**: represents a “top percent” operation. This node is a PSFlagParseNode with a PSFlagValue of PS_FLAG_TOP_PERCENT.

- **PS_NT_UNARY_MINUS_SIGN**: represents a unary minus sign on a number (for example, -6). The value being modified is contained in the child PSLiteralParseNode with a respective PSLiteralType for the value (for example, PS_LITERAL_USINT).

- **PS_NT_UNARY_PLUS_SIGN**: represents a unary plus sign on a number (for example, +6). The value being modified is contained in the child PSLiteralParseNode with a respective PSLiteralType for the value (for example, PS_LITERAL_USINT).

- **PS_NT_UNION**: represents a union of rows selected from two tables. Both children are PSLiteralParseNodes each of type PS_NT_SELECT_STATEMENTS containing the select statements to perform the union with.

- **PS_NT_UNION_ALL**: represents a “union all” operation on columns selected from two tables. Both children are PSLiteralParseNodes each of type PS_NT_SELECT_STATEMENTS containing the select statements to perform the union with.

- **PS_NT_UNIQUE**: represents a “unique” constraint on an operation. This node is of type PSFlagParseNode with a PSFlagValue of PS_FLAG_UNIQUE and has no child nodes.

- **PS_NT_UNIQUE_SPECIFICATION**: represents a “unique” constraint on a column specified when creating a table. This is a PSFlagParseNode of type PS_NT_UNIQUE_SPECIFICATION with a PSFlagValue of PS_FLAG_UNIQUE.

- **PS_NT_UPDATE_ACTION**: represents an “update” action on which permission is being granted as part of a “grant” operation. The child can optionally be a PSNonTerminalParseNode of type PS_NT_COLUMN_NAME_LIST specifying the columns to grant update permissions for.

- **PS_NT_UPDATE_SOURCE**: represents the “default” keyword in an “update table” statement.

- **PS_NT_UPDATE_STATEMENT_SEARCHED**: represents an “update” statement where the first child is a PSNonTerminalParseNode of type PS_NT_TABLE_NAME containing the table name to update, and the second child is a PSNonTerminalParseNode of type PS_NT_SET_CLAUSE_LIST which contains the set clause information. The third child is an optional PSNonTerminalParseNode of type PS_NT_WHERE_CLAUSE containing the where condition, if specified.

- **PS_NT_UPSERT_STATEMENT_SEARCHED**: represents an UPSERT statement. The children are the same as for PS_NT_UPDATE_STATEMENT_SEARCHED.

- **PS_NT_VALUE**: a PSFlagParseNode of type PSFlagDefault representing the keyword "default" in a value list.

- **PS_NT_VALUE_LIST**: contains the list of values specified in an "in" predicate statement, to check for a row value in. Each child is a value node to compare against (for example, a PSLiteralParseNode of type PS_LITERAL_CHARSTR for a string literal).

- **PS_NT_VAR**: represents a "variance" operation where the first child is an optional PSFlagParseNode specifying a set quantifier (for example, distinct) with the respective PSFlagValue (for example, PSFlagDistinct), and the second child is a PSNonTerminalParseNode containing the expression to perform the operation on.

- **PS_NT_VAR_POP**: represents a "variance pop" operation where the first child is an optional PSFlagParseNode specifying a set quantifier (for example, distinct) with the respective PSFlagValue (for example, PSFlagDistinct), and the second child is a PSNonTerminalParseNode containing the expression to perform the operation on.

- **PS_NT_VS_PRED_LIST**: represents a "pred" keyword where both children are PSLiteralParseNodes of the respective PSLiteralType for the values provided to pred (for example, PS_LITERAL_USINT).

- **PS_NT_WHERE_CLAUSE**: represents a where clause in a select statement in which the boolean expression is contained in the child PSNonTerminalParseNode with an appropriate PSNonTerminalType (for example, PS_NT_LESS_THAN_OP for a "<" boolean expression).