Cognos(R) 8 Business Intelligence

DATA MANAGER

COGNOS SQL

THE NEXT LEVEL OF PERFORMANCE™

Product Information

This document applies to Cognos(R) 8 Version 8.2 and may also apply to subsequent releases. To check for newer versions of this document, visit the Cognos Global Customer Services Web site (http://support.cognos.com).

Copyright

Table of Contents

Introduction

This document is intended for use with the Cognos SQL data manipulation language (DML) as defined in ANSI/ISO SQL standards.

Using This Document

The examples in this document use Bakus-Naur Form (BNF) to describe the syntax of the Data Manager language. The BNF syntax consists of the following:

- A set of terminal symbols that are the words, commands, or punctuation of the language and command-line interface.
- A set of non-terminal symbols that are essentially place holders. They appear in angled brackets < >, for example, <refdata_file>. A definition of each non-terminal symbol may appear elsewhere in the syntax definition. However, not all non-terminal symbols are defined. For a complete definition of the scripting language, including the description of all non-terminal symbols, see the *Function and Scripting Reference Guide*.
- A set of rules that you apply when interpreting the BNF definitions:
  - ::= means 'is defined to be'.
  - Square brackets [ ] indicate that the enclosed symbols are optional.
  - Braces { } indicate that the enclosed symbols may be repeated zero or more times.
  - The piping symbol | indicates that you should choose only one of the items that it separates.

The following example defines the <options> symbol to be an optional -C, followed by a <var_list> symbol. It then defines the <var_list> symbol to be zero, one, or more instances of -V followed by a <name>=<value> pair:

```
<options> ::=
[-C] <var_list>
<var_list> ::=
{-V<name>=<value>
```

Audience

You should have a basic knowledge of SQL.

Related Documentation

Our documentation includes user guides, getting started guides, new features guides, readmes, and other materials to meet the needs of our varied audience. The following documents contain related information and may be referred to in this document. To view these documents, either consult your administrator to determine the online location of Cognos documentation or search the Product Documentation library of the Cognos Global Customer Services Web site (http://support.cognos.com/kb-app/knowledgebase). If you require logon credentials, either consult your administrator or send an email to support.america@cognos.com.

Document	Description
Cognos 8 New Features	Describing the new features and changes in this release

Document	Description
Data Manager Getting Started	Introducing Data Manager Designer to new users
Data Manager User Guide	Using Data Manager to create data warehouses and data repositories
Function and Scripting Reference Guide	Describing the Data Manager functions and the scripting language that can be used within the engine or from Data Manager Designer
Data Manager Installation and Configuration Guide	Installing and configuring Data Manager

Finding Documentation

To find the most current Cognos documentation, including all localized documentation, access the Cognos Global Customer Services Web site (http://support.cognos.com). Click the **Documentation** link to access product documentation. Click the **Knowledge Base** link to access all documentation, technical papers, and multimedia materials.

Product-specific documentation is available in online help from the **Help** menu or button in Cognos products. You can also download documentation in PDF format from the Cognos Global Customer Services Web site, or click the following link to open a printable version of this document (PDF).

You can also read PDF versions of the product readme files and installation guides directly from Cognos product CDs.

Getting Help

For more information about using this product or for technical assistance, visit the Cognos Global Customer Services Web site (http://support.cognos.com). This site provides product information, services, user forums, and a knowledge base of documentation and multimedia materials. To create a case, contact a support person, or provide feedback, click the **Contact Support** link. For information about education and training, click the **Training** link.

Printing Copyright Material

You can print selected pages, a section, or the whole book. Cognos grants you a non-exclusive, non-transferable license to use, copy, and reproduce the copyright materials, in printed or electronic format, solely for the purpose of operating, maintaining, and providing internal training on Cognos software.

Chapter 1: Cognos SQL

Data Manager supports both native SQL and Cognos SQL. This book describes Cognos SQL, which is a subset of the SQL 99 data manipulation language (DML).

You should use this book if you set up your Data Manager connections to use Cognos SQL. Normally, you will use the native SQL of the connection, such as Oracle SQL for Oracle connections.

Using Cognos SQL allows a greater degree of portability between mixed database environments by allowing a common dialect to be used. For example, you may want to use Cognos SQL to build data sources that are more easily ported between database platforms where the source tables and columns are the same, such as Cognos 8 Analytic Applications that are intended to run at multiple sites on different database platforms.

If you use native SQL, you can use any SQL supported by the database you are working with, including non-standard SQL extensions and hints.

Notes:

- If you are using Data Manager Connector for SAP R/3, you must use Cognos SQL, as native SQL is not supported.
- You cannot use Cognos SQL for SQLTXT connections.

Chapter 2: Syntax

SELECT Statements

This is the syntax for the top level of a SELECT statement.

Syntax

```
<select_statement> ::=
   SELECT [DISTINCT]
      <select_list>
   FROM
<table_reference>
   [<where_clause>]
   [<group_by_clause>
   [<having_clause>]]
   [<order_by_clause>]
```

Symbol	Description
SELECT	This keyword introduces each SELECT statement
DISTINCT	This keyword eliminates duplicate data rows from the result set
<select_list>	The set of returned data columns
FROM	This keyword introduces the set of tables from which data is acquired
<table_reference>	The set of tables from which data is acquired
<where_clause>	A set of conditions that filter the data
<group_by_clause>	Specifications for grouping the returned data
<having_clause>	A set of conditions that filter the groups of data rows
<order_by_clause>	Specifications for sorting the returned data

Select List

Syntax

```
<select_list> ::=
   <source_columns>|*
<source_columns> ::=
   <source_column>{, <source_column>}
<source_column> ::=
   <aggregate_expression>|<value_expression>
<aggregate_expression> ::=
   <table_aggregate>|<column_aggregate>
<table_aggregate> ::=
   COUNT(*)
<column_aggregate> ::=
   <aggregate_function> ([DISTINCT|ALL] <value_expression>)
```

```
<aggregate_function> ::=
   AVG|SUM|MAX|MIN|COUNT
<value_expression> ::=
   <literal>|<column_spec>|<function_result>|<table_spec>
<column_spec> ::=
   [<table_name>.]<column_name> [AS <column_alias>]
<table_spec> ::=
   <table_name>.*
```

Symbol	Description
<select_list>	The set of returned data columns. You can use the asterisk (*) to indicate all available columns from the source tables, or you can name the columns.
<source_columns>	A named set of data columns.
<source_column>	A named column or all columns from a named source table.
<aggregate_expression>	An expression that may consolidate many data source rows, such as the average, minimum, or maximum values in the set of rows. If the expression includes the DISTINCT keyword, duplicate values are ignored. If the expression includes the ALL keyword (the default), all non-null values are considered, whether or not they contain duplicates.
<value_expression>	A literal value, a data source column or table, or the result of evaluating an expression or function.
<table_aggregate>	The COUNT aggregation function applied to an entire table. COUNT(*) counts the number of data source rows, including all duplicates and those that contain null values.
<column_aggregate>	An aggregation function applied to a data source column.
<aggregate_function>	A function used to consolidate each group of data rows • AVG returns the average value of the group • SUM returns the sum of the members of the group • MAX returns the maximum value of the group • MIN returns the minimum value of the group • COUNT returns the number of rows in the group These functions ignore null values.
<literal>	A constant value.
<column_spec>	The name of a data source column, optionally qualified by the name of the table in which it resides, together with an optional alias.
<table_name>	The name of the data source table that contains the named column. This is required if the column name is ambiguous, such as when it appears in more than one source table.
<function_result>	The result of evaluating a function.
<column_name>	The name of a data source column.
<column_alias>	An optional alias for the column.

Reference Tables

Cognos SQL can specify data directly from the base tables, from the join of base tables, and from derived tables.

Syntax

```
<table_reference> ::=
   <table>{, <table>}
<table> ::=
<base_table>|<joined_table>|<derived_table>
```

Symbol	Description
<table>	A logical data table which may be a base table, the result of joining two tables, or a derived table
<base_table>	A base table or a view of the source data
<joined_table>	The result of joining two or more tables
<derived_table>	The result table from a SELECT statement

Base Tables

Syntax

```
<base_table> ::=
   <table_name> [[AS] <table_alias>]
```

Symbol	Description
<table_name>	The name of a data source table or view
<table_alias>	An alias, or correlation name, that you assign to the table

Joined Tables

Syntax

```
<joined_table> ::=
   <table> <join_type> JOIN <table> ON <conditions>
<join_type> ::=
   INNER|<outer_join>
<outer_join>
   <outer_join_type> [OUTER]
<outer_join_type>
   LEFT|RIGHT|FULL
```

Symbol	Description
<joined_table>	A table formed by joining two or more tables which can be any combination of base tables, joined tables, or derived tables.
<table>	A data table.
<join_type>	A specification of how to join the tables.
JOIN	This keyword specifies a join.
INNER	The returned data must be present in both the joined tables.

Symbol	Description
OUTER	The join returns data from one or both of the tables even if it does not match data from the other table. Null is used for the missing values.
LEFT	The join includes all the data from the first table, whether or not it matches data from the second. The join includes data from the second table only if it matches data from the first table.
RIGHT	The join includes all the data from the second table, whether or not it matches data from the first. The join includes data from the first table only if it matches data from the second table.
FULL	The join includes all the data from each table, whether or not it matches data from the other table.
<conditions>	Test conditions that must be true for each row of the joined table.

Examples

```
SELECT T1.CurrentYear, T2.ProductCode, T2.Quantity
FROM (SalesFact T2 LEFT OUTER JOIN TimeDimension T1 ON
T2.TimeCode = T1.TimeCode)
```

Derived Tables

Syntax

```
<derived_table> ::=
   (<select_statement>) [AS] <table_alias>
```

Symbol	Description
<derived_table>	The result table from an intermediate SELECT statement
<select_statement>	The intermediate SELECT statement
<table_alias>	The correlation name (alias) for the derived table

Examples

```
SELECT T1.ProdNo AS c1,
       T1.ProdName AS c2,
       T1.DateSold AS c3,
       T1.UnitsSold AS c4
FROM (SELECT P.ProductNumber AS ProdNo,
       P.ProductName AS ProdName,
       S.OrderDate AS DateSold,
       S.Quantity AS UnitsSold
FROM ProductDimension P, SalesFact S
WHERE P.ProductNumber = S.ProductNumber) AS T1
```

Test Conditions

Syntax

```
<conditions> ::=
   <predicate> {<boolean_operator><predicate>}
<predicate> ::=
   [NOT][(]<comparison>|<like_predicate>|<between_predicate>|<in_predicate>|
<null_predicate>|<sub_query_predicate>[)]
<boolean_operator> ::=
   AND|OR
```

Symbol	Description
<conditions>	Test conditions that must be TRUE to include the row or group in the result data.
<predicate>	An expression that evaluates to either TRUE or FALSE.
<boolean_operator>	Either AND or OR. AND returns TRUE if, and only if, both conditions are TRUE. OR returns TRUE if either, or both, conditions are TRUE.
NOT	This keyword negates the predicate.
Parentheses ( )	Use parentheses to force an order of evaluation. (optional)
<comparison>	The comparison of two values to each other.
<like_predicate>	The comparison of a string value with a given pattern.
<between_predicate>	The test of whether a value lies in a specified, ascending range.
<in_predicate>	The test of a value for membership of a specified set.
<null_predicate>	The test of whether a value is NULL.
<sub_query_predicate>	The comparison of a value to the rows of an intermediate table.

Logical Comparison

Syntax

```
<comparison> ::=
   <value> <comparison_operator> <value>
<comparison_operator> ::=
   <|<=|=|>=|>|<>
<value> ::=
   [<table_name>.]<column_name>
   <literal>
   <expression>|
   <column_aggregate>|
   <column_alias>
```

Symbol	Description
<comparison>	A comparison of two values.
<value>	A literal value, the value of a data column, or the result of an expression.

Symbol	Description
<comparison_operator>	A logical operator that is used for comparing two values. Valid operators are • < specifies less than • <= specifies less than or equal to • = specifies equal to • >= specifies greater than or equal to • > specifies greater than • <> specifies not equal to
<table_name>	The name of a base data table or view.
<column_name>	The name of a data column.
<literal>	A constant value.
<expression>	An expression that results in a single numeric or string value.
<column_aggregate>	An aggregate function applied to a data source column. In logical comparisons, column aggregates are valid only in HAVING clauses.
<column_alias>	An alias for the column or column group.

Examples

```
SELECT ProductNo, ProductName, UnitsSold
FROM SalesFact
WHERE UnitsSold > 100
```

Pattern Matching

Syntax

```
<like_predicate> ::=
   <value> [NOT] LIKE <pattern>
<pattern> ::=
   '{<char>|<metachar>}' [ESCAPE'<char>']
<metachar> ::=
   %|_
```

Symbol	Description
<like_predicate>	The comparison of a string value to a wildcard template.
<value>	A literal string value, a data column, or the result of converting a value to a character data type.
LIKE	This keyword specifies a wildcard comparison. The value being tested must match the wildcard template for the condition to be TRUE. The optional NOT keyword inverts the result.
<pattern>	A string value that can include the wildcard characters.
<char>	A single character.
<metachar>	A wildcard character, either a percentage symbol (%) or an underscore (_). A percentage symbol matches any number of characters, whereas an underscore matches a single character. For example, 'catalog' LIKE '%a_og' is TRUE.

Symbol	Description
ESCAPE	This keyword introduces the optional escape character, which is required when the pattern contains a percentage symbol (%) or an underscore (_) that is not a wildcard.

Examples

```
SELECT * FROM Product
WHERE ProductCode LIKE 'SV%'

SELECT * FROM Product
WHERE ProductCode LIKE 'SV\_DB%'ESCAPE '\'
```

Value is in a Range

Syntax

```
<between_predicate> ::=
   <value> BETWEEN <literal> AND <literal>
```

Symbol	Description
<between_predicate>	A condition that tests whether a value lies in a specified range.
<value>	The value being tested.
<literal>	A constant value. The first literal value specifies the lower bound, and the second specifies the upper bound.

Examples

```
SELECT T1.ProductNumber, T2.ProductName, 'Volume Band 2'
AS Qty
FROM SalesFact T1 INNER JOIN ProductDimension T2
ON (T2.ProductNumber = T1.ProductNumber)
WHERE T1.UnitsSold BETWEEN 50 AND 100
```

Value is a Member of a Set

Syntax

```
<in_predicate> ::=
   <test_value> IN (<value_list>|<select_statement>)
<value_list> ::=
   <value>{, <value>}
```

Symbol	Description
<in_predicate>	A condition that tests whether a value is a member of a specified set.
<test_value>	The value being tested.
<value_list>	A set of discrete values.
<value>	A discrete member of the test set.
<select_statement>	A SELECT statement that must return only one column. An error occurs if the statement returns more.

Examples

```
SELECT T1.ProductNumber AS c1, T1.ProductName AS c2
```

```
FROM ProductDimension
WHERE ProductNumber IN
(SELECT ProductNumber FROM SalesFact
WHERE SalesDate BETWEEN '2006-01-01' AND '2006-03-31')

SELECT T1.ProductNumber AS c2, T1.ProductName AS c2
FROM ProductDimension
WHERE ProductNumber IN (123, 456, 789)
```

Value is NULL

Syntax

```
<null_predicate> ::=
   <value> IS [NOT] NULL
```

Symbol	Description
<null_predicate>	A condition that tests whether a value is null
<value>	The value being tested

Examples

```
SELECT * FROM Product WHERE ProductName IS NOT NULL
```

Logical Comparison Extended to a Sub-Query

Syntax

```
<sub_query_predicate> ::=
   <value> <comparison_operator> <quantifier> <select_statement>|
   EXISTS <select_statement>
<quantifier> ::=
   SOME|ANY|ALL
```

Symbol	Description
<sub_query_predicate>	A condition that tests a value against an intermediate table.
<value>	The value being tested.
<comparison_operator>	A logical comparison operator from <, <=, =, >=, >, and <>.
<quantifier>	A keyword that specifies the form of the comparison. SOME and ANY both return TRUE if the comparison is TRUE for one or more rows, and FALSE otherwise. ALL returns TRUE if the comparison is TRUE for all rows or the intermediate table is empty.
<select_statement>	The SQL specification of the intermediate table used in the condition. When used with SOME, ANY, or ALL, the intermediate table must contain one column only.
EXISTS	A keyword that returns TRUE if the associated SELECT statement returns at least one data row.

Examples

```
SELECT T1.ProductNumber, T1.ProductName FROM ProductDimension T1
WHERE EXISTS
(SELECT T2.ProductNumber FROM SalesFact T2
WHERE T1.ProductNumber = T2.ProductNumber)
```

Operator Precedence

In the conditions of WHERE and HAVING clauses, and in join conditions, the AND operator has higher precedence than the OR operator. The NOT operator has higher precedence than both the AND and the OR operator.

Consider the expression, FALSE AND FALSE OR TRUE. Here, AND has the higher precedence and is processed first to give FALSE OR TRUE, which evaluates to TRUE.

Consider the expression, FALSE AND FALSE OR NOT TRUE. Here, NOT has the higher precedence and is processed first to give FALSE AND FALSE OR FALSE. Next, AND is processed to give FALSE OR FALSE, which evalutates to FALSE.

Where Clauses

Syntax

```
<where_clause> ::=
   WHERE <conditions>
```

Symbol	Description
<where_clause>	A set of conditions that filter the data
<conditions>	Test conditions that must be true

Examples

```
SELECT T1.ProductName AS c1, SUM(T2.SaleTotal) AS c2
FROM ProductDimension T1, SalesFact T2
WHERE (T1.ProductCode = T2.ProductCode)
AND ((T1.ProductName IS NOT NULL) AND (T1.ProductName
LIKE 'ABC%'))
GROUP BY c1
ORDER BY c1 ASC
```

Group By Clauses

Syntax

```
<group_by_clause> ::=
   GROUP BY <group_column>{, <group_column>}
<group_column> ::=
   <column_name>|<column_alias>
```

Symbol	Description
<group_by_clause>	Specifications for grouping the returned data
<group_column>	A column by which to group the data
<column_name>	The name of the column
<column_alias>	The alias, or correlation name, of the column

Examples

```
SELECT T1.ProductName AS c1, SUM(T2.SaleTotal) AS c2
FROM ProductDimension T1, SalesFact T2
WHERE (T1.ProductCode = T2.ProductCode)
GROUP BY c1
ORDER BY c1 ASC
```

Having Clauses

Each HAVING clause is applied to the data groups that result from a GROUP BY clause.

Syntax

```
<having_clause> ::=
<conditions>
```

Symbol	Description
<having_clause>	A set of conditions that filter the groups of data rows
<conditions>	Test conditions that must be true to include the data group in the result set

Examples

```
SELECT T1.ProductNumber AS c1, T1.ProductName AS c2, SUM(T2.SaleTotal)
AS c3
FROM ProductDimension T1, SalesFact T2
WHERE (T1.ProductCode = T2.ProductCode)
AND ((T1.ProductName IS NOT NULL) AND (T1.ProductName LIKE 'ABC%'))
GROUP BY c1, c2
HAVING SUM(T2.SaleTotal) >= 44594.14
```

Order By Clauses

Each ORDER BY clause specifies how the returned data is sorted. By default, the order is undefined.

Syntax

```
<order_by_clause> ::=
   ORDER BY <column_spec>{ ,<column_spec>}]
<column_spec> ::=
   <column> [ASC|DESC]
<column> ::=
   <column_name>|<column_alias>
```

Symbol	Description
<order_by_clause>	Specifications for sorting the returned data.
<column_spec>	The name of a column of returned data used to sort the data, together with the direction of sort. ASC (the default) sorts the data in ascending order, and DESC in descending order.
<column>	A column used to sort the data.
<column_name>	The name of a column.
<column_alias>	The alias, or correlation name, of a column.

Examples

```
SELECT T1.ProductNumber AS c1, T1.ProductName AS c2, SUM(T2.SaleTotal)
AS c3
FROM ProductDimension T1, SalesFact T2
WHERE (T1.ProductCode = T2.ProductCode)
AND ((T1.ProductName IS NOT NULL) AND (T1.ProductName
LIKE 'ABC%'))
GROUP BY c1, c2
HAVING SUM(T2.SaleTotal) >= 44594.14
```

```
ORDER BY c1 ASC, c3 DESC, c2 ASC
```

Other SQL Statements

You can modify data using the INSERT, UPDATE, and DELETE statements that execute within a transaction. When a transaction commits, the changes are visible for new transactions that access the database.

UPDATE Statements

Syntax

```
update <target_table>
   set <set_clause_list>
   [ where <search_condition> ]
```

Symbol	Description
<target_table>	The name of the target table
<set_clause_list>	The list of columns to be updated
<search_condition>	The condition for which to search

Examples

```
Update SalesFact set UnitsSold=UnitsSold - 100, Cost = Cost * 0.10 where
ProductNo = 123456
```

```
Update SalesFact set Cost = Cost + 50
```

INSERT Statements

Syntax

```
insert into <target_table>
   values (value-1, value-2,.... ,value-n)
```

Symbol	Description
<target_table>	The name of the target table

Examples

```
Insert into ProductDimension (ProductNumber, ProductName) values ( 123478,
'Paper Towels')
```

DELETE Statements

Syntax

```
delete from <target_table>
   [ where <search_condition> ]
```

Symbol	Description
<target_table>	The name of the target table
<search_condition>	The condition for which to search

Examples

```
Delete from ProductDimension where ProductNumer = 1234678
```

```
Delete from ProductDimension
```

Chapter 3: Functions

You can use the following functions in Cognos SQL statements:

- numeric
- string
- type conversion
- conditional expressions
- date and time

Numeric Functions

Numeric functions operate on non-numeric data to return numeric results. Cognos SQL provides the following numeric functions:

- POSITION
- EXTRACT
- CHAR_LENGTH

POSITION

Returns an integer to represent the position, starting from one, of one string in another. This function returns NULL if either operand is NULL. It returns zero if the first string does not occur in the second.

Syntax

```
POSITION(<string1> IN <string2>)
```

Symbol	Description
<string1>	A string value
<string2>	A string value

Examples

```
SELECT T1.ProductName AS c1, (POSITION('ABC' IN T1.ProductName))AS c2
FROM ProductDimension T1
```

EXTRACT

Returns an integer to represent the value of the required component of a date or time value, or of an interval expression. If the date value or interval expression is NULL, EXTRACT returns NULL. The following table shows how EXTRACT represents each component for date values.

Component	Representation
Years	Four digits, such as 2006
Months	Month number, where January=1

Component	Representation
Days	Day of the month, such as 25 to represent the day component of February 25, 2006
Hours	24-hour format in the range (0..23)
Minutes	A number in the range (0..59)
Seconds	A number in the range (0..59)

The following table shows the value ranges for time interval expressions.

Component	Representation
Days	Unbounded
Hours	-23 to +23
Minutes	-59 to 59
Seconds	-59.9 to 59.9

Syntax

```
EXTRACT(<component> FROM <value>)
<component> ::=
   YEAR|MONTH|DAY|HOUR|MINUTE|SECOND
```

Symbol	Description
<component>	The required date or time unit
<value>	The date, time, or time interval from which to extract the component

Examples

```
SELECT
EXTRACT(YEAR FROM T1.SalesDate)) AS c1,
(EXTRACT(MONTH FROM T1.SalesDate)) AS c2,
(EXTRACT(DAY FROM T1.SalesDate)) AS c3
FROM Sales T1

SELECT
T1.CurrentYear AS c1,
EXTRACT(HOUR FROM CURRENT_TIMESTAMP)
FROM TimeDimension T1
```

CHAR_LENGTH

Returns the number of characters in its argument. CHAR_LENGTH returns NULL if the argument is NULL.

Syntax

```
CHAR[ACTER]_LENGTH(<string>)
```

Symbol	Description
<string>	A string value

Examples

```
SELECT
T1.ProductNumber AS c1,
CHAR_LENGTH(T1.ProductDescription)) AS c2
FROM Product

SELECT
T1.ProductNumber AS c1,
CHARACTER_LENGTH(T1.ProductDescription)) AS c2
FROM Product
```

String Functions

String functions operate on and return string values. Cognos SQL provides the following string functions:

- SUBSTRING
- UPPER
- LOWER
- TRIM

SUBSTRING

Returns the portion of a string of specified length starting from a specified position. By default, SUBSTRING returns from the starting position to the end of the source string. SUBSTRING returns NULL if the starting position is NULL, and returns a zero-length string if either the start position is greater than the number of characters in the source string or the source string is a zero-length string. If the specified length is greater than the length of the source string minus the starting position, SUBSTRING ignores the length specification.

Syntax

```
SUBSTRING(<source_string> FROM <start> [FOR <length>])
```

Symbol	Description
<source_string>	The source string from which to extract a substring
<start>	The starting position, where the left character of the source string being in position 1
<length>	The specified length of the substring

Examples

```
SELECT
T1.ProductName AS c1,
(SUBSTRING(T1.ProductName FROM 4 FOR 2)) AS c2
FROM ProductDimension T1

SUBSTRING('Catalog' FROM 5) returns 'log'
SUBSTRING('Catalog' FROM 8) returns an empty (zero-length) string
```

UPPER

Converts a string to upper case.

Syntax

```
UPPER(<string>)
```

Symbol	Description
<string>	A string value

Examples

```
SELECT
T1.ProductNumber AS c1,
(UPPER(T1.ProductName)) AS c2
FROM ProductDimension T1

UPPER('catalog') returns 'CATALOG'
```

LOWER

Converts a string to lower case.

Syntax

```
LOWER(<string>)
```

Symbol	Description
<string>	A string value

Examples

```
SELECT
T1.ProductNumber AS c1,
(LOWER(T1.ProductName)) AS c2
FROM ProductDimension T1

LOWER('CATALOG')returns 'catalog'
```

TRIM

Removes all spaces from the beginning of a string, the end of a string, or both. By default, TRIM removes spaces from both ends of the string.

Syntax

```
TRIM([<direction> FROM ]<string>)
<direction> ::=
   LEADING|TRAILING|BOTH
```

Symbol	Description
<string>	A string value
LEADING	Option to remove spaces from the leading (left) end of <string>
TRAILING	Option to remove spaces from the trailing (right) end of <string>
BOTH	Option to remove spaces from both ends of <string>

Examples

`TRIM('   catalog   ')` returns 'catalog'

`TRIM(LEADING FROM '   catalog   ')` returns 'catalog   '

Type Conversion Functions

Type conversion functions convert data from one data type to another. Cognos SQL provides the following type conversion functions:

- CAST

CAST

Converts a value to a different data type.

Syntax

```
CAST(<value> AS <datatype>)
<datatype> ::
   CHARACTER[(<length>)]
   CHAR[(<length>)]
   VARCHAR(<length>)
   NUMERIC[(<precision>[,<scale>])]
   DECIMAL[(<precision>[,<scale>])]
   INTEGER|
   SMALLINT|
   REAL|
   FLOAT[(<precision>)] |
   DATE|
   TIME|
   TIMESTAMP|
   INTERVAL
```

Symbol	Description
<value>	The value to convert to a different data type.
<datatype>	The data type to which to convert the value. The following SQL-99 data types are valid: • CHARACTER • VARCHAR • CHAR • NUMERIC • DECIMAL • INTEGER • SMALLINT • REAL • FLOAT • DATE • TIME • TIMESTAMP • INTERVAL
<length>	The maximum number of characters in the returned string.
<precision>	The number of digits to use.
<scale>	The number of digits that follow the decimal point.

Notes:

- When you convert a value of type TIMESTAMP to type DATE, the time portion of the timestamp value is ignored.
- When you convert a value of type TIMESTAMP to type TIME, the date portion of the timestamp is ignored.
- When you convert a value of type DATE to type TIMESTAMP, the time components of the timestamp are set to zero.
- When you convert a value of type TIME to type TIMESTAMP, the date component is set to the current system date.
- When you convert a value of type INTERVAL to type CHAR, CHARACTER, or VARCHAR, the resulting string is in the format DDD HH:MM:SS.S, where DDD is the number of days, HH the number of hours, MM the number of minutes, and SS.S the number of seconds and parts of a second.
- When you convert a value of type CHAR, CHARACTER, OR VARCHAR to type DATE, TIME, TIMESTAMP, or INTERVAL, leading and trailing spaces are removed and the string is converted to a temporal value only if it matches a supported format. Otherwise CAST returns NULL. For more information, see "Temporal Values" (p. 32).
- You cannot use the CAST function to convert between some data types. The following table indicates which conversions are valid.

Target data type (<datatype>)	Source data type (<value>)
CHARACTER	CHARACTER
CHAR	CHAR
VARCHAR	VARCHAR
	NUMERIC
	DECIMAL
	INTEGER
	SMALLINT
	REAL
	FLOAT
	DATE
	TIME
	TIMESTAMP
	INTERVAL
NUMERIC	CHARACTER
DECIMAL	CHAR
INTEGER	VARCHAR
SMALLINT	NUMERIC
REAL	DECIMAL
FLOAT	INTEGER
	SMALLINT
	REAL
	FLOAT

Target data type (<datatype>)	Source data type (<value>)
DATE	CHARACTER CHAR VARCHAR DATE TIMESTAMP
TIME	CHARACTER CHAR VARCHAR TIME TIMESTAMP
TIMESTAMP	CHARACTER CHAR VARCHAR DATE TIME TIMESTAMP
INTERVAL	CHARACTER CHAR VARCHAR INTERVAL

Examples

```
SELECT
T1.SalesTarget AS c1,
CAST(T2.CurrentYear AS INTEGER) AS c2,
CAST(T1.SalesTarget AS VARCHAR(10)) AS c3
FROM StaffDimension T1, TimeDimension T2, SalesFact T3
WHERE (T1.StaffCode = T3.StaffCode) AND (T2.TimeCode =
T3.TimeCode)
```

Conditional Expressions

Cognos SQL provides the following conditional expressions:

- CASE
- COALESCE
- NULLIF

CASE

Provides a choice of multiple conditions. It consists of a case list and an optional default case.

Each case has an expression to match, or a test condition to evaluate, together with the value that CASE returns if the expression is matched or the test condition evaluates to TRUE. All members of the case list must be expressions or they must all be test cases.

If none of the specified case list match, and no default is given, CASE returns NULL.

Syntax

```
CASE {<predicate_case>} [<default_case>] END|
CASE <expression> {<expression_case>}  [<default_case>] END
<expression_case> ::=
   WHEN <expression> THEN <value>
<predicate_case> ::=
   WHEN <predicate> THEN <value>
<default_case> ::=
   ELSE <value>
```

Symbol	Description
<default_case>	The value to return if none of the specified cases return TRUE.
<expression>	A literal value, data column, or expression that evaluates to a single, non-Boolean value.
<predicate>	An expression that evaluates to either TRUE or FALSE.
<value>	The value to return.

Examples

```
SELECT T1.ProductNumber AS c1, T2.ProductName AS c2,
   (CASE)
WHEN (T2.Quantity BETWEEN 1 AND 10) THEN ('Low volume')
WHEN (T2.Quantity BETWEEN 11 AND 100) THEN ('Medium volume')
WHEN (T2.Quantity IS NULL) THEN ('NULL value')
ELSE ('Ideal volume')
   END) AS c3
FROM ProductDimension T1, SalesFact T2
WHERE (T1.ProductCode = T2.ProductCode)

SELECT CASE T1.Country
WHEN 'France' THEN 'Europe'
WHEN 'Japan' THEN 'Asia'
WHEN 'United States' THEN 'North America'
ELSE NULL
END AS Continent
FROM SiteDimension T1
```

COALESCE

Returns the first non-NULL value from a series of expressions. It returns NULL if the series contains only NULL.

Syntax

```
COALESCE(<expression>{, <expression>})
```

Symbol	Description
<expression>	A literal value, column name, or expression that evaluates to a discrete value.

Examples

```
SELECT
T1.ProductNo AS c1,
(COALESCE(T1.ProductName, T1.Description)) AS c2
FROM Product T1
```

NULLIF

Compares two values and returns NULL if they are equal. Otherwise, NULLIF returns the first value.

Examples

```
SELECT
T1.ProductNo AS c1,
(COALESCE(NULLIF(T1.ProductName, 'OldName'), 'NewName')) AS
c2
FROM Product T1
```

Date and Time Functions

Cognos SQL provides the following date and time functions:

- CURRENT_DATE, which returns the current system date
- CURRENT_TIME, which returns the current system time
- CURRENT_TIMESTAMP, which returns the current system date and time as a timestamp
- sysdate(), which returns the system date from the server on which the database is located (for use with SAP only)
- systime(), which returns the system time from the server on which the database is located (for use with SAP only)

Examples

```
SELECT
T1.ProductNumber AS c1,
CURRENT_DATE AS c2,
CURRENT_TIME AS c3,
CURRENT_TIMESTAMP AS c4
FROM ProductDimension

SELECT
   sysdate(),systime(), <column_name>
FROM <table_name>
```

Chapter 4: Literal Values

You can use these types of literal values in Cognos SQL statements:

- numeric values
- string values
- temporal values

Numeric Values

Numeric literal values can be integers or decimal numbers. Integers match to integer data types and are automatically converted to other data types. Decimal numbers match only precise numbers and floating-point data types.

Numeric literal values can take the following forms:

- Exact numeric literal.

  This consists of an exact numeric value and a sign, such as, -12 and 56789.
- Fixed precision numeric literal, or approximate numeric.

  This consists of a fixed number of digits, an indication of scale, and a sign, such as 12.345, -.091, and 12e-02.

You can use arithmetic operators in numeric expressions. Operators have the following order of precedence:

Order	Operator	Description
1	( )	Parentheses
2	/	Division
3	*	Multiplication
4	+	Addition
5	-	Subtraction

Examples

```
123 * 2
(123.45 + 4) / (23 - 2)
```

String Values

String literal values consist of a series of characters, the maximum length of which is determined by the underlying DBMS. Delimit string literal values using single quotation marks.

You can use the concatenation operator () to join two string values.

Because the single quotation mark is used to delimit string values, you must use the escape character, a single quotation mark, with each single quotation mark that appears in a literal value.

Examples

```
'Example ' || 'string'
'The string''s length'
```

Temporal Values

Temporal values are dates, times, and time intervals. In temporal literal values, specify the data type of the value and use the following formats.

Value	Format	Example	Notes
TIME	'HH:MI:SS'	'12:23:55.123'	Time of the day on the 24-hour clock. Seconds can be expressed to three decimal places.
DATE	'YYYY-MM-DD'	'2006-11-30'	Date as year-month-day.
TIMESTAMP	'YYYY-MM-DD HH:MI:SS'	'2006-11-30 12:23:55.123'	Combination of date and time.
INTERVAL	'DAYS HH:MI:SS'	'234 23:34:12.55'	A time interval. The first figure gives the number of days. The remainder give the hours, minutes, and seconds.

You can add and subtract temporal values to return other temporal values. The following table indicates the result of performing these operations on valid combinations of operands.

Operation	Result
date + interval	date
interval + date	date
date - interval	date
date - date	interval
time + interval	time
interval + time	time
time - interval	time
time - time	interval
timestamp + interval	timestamp
interval + timestamp	timestamp
timestamp - interval	timestamp
timestamp - timestamp	interval

In addition, you can multiply and divide intervals by numeric values to return intervals.

Examples

- These examples add intervals to date and time values:

```
DATE '2001-11-30' + INTERVAL '1
00:00:00.0' = DATE '2001-12-01'
TIME '12:05:04.5' + INTERVAL '0 02:02:00.0'
= TIME '14:07:04.5'
```

- This example returns all orders that took 30 or more days to close. The expression (T1.CLOSED_DATE - T1.ORDER_DATE) returns an interval, which is compared with an interval of 30 days in the WHERE clause:

```
SELECT T1.ORDER_NO AS c1
FROM ORDER T1
WHERE (T1.CLOSED_DATE - T1.ORDER_DATE >= INTERVAL '30 00:00:00.0')
```

- This example calculates shipping dates by adding five days to the date on which each order is placed:

```
SELECT
T1.ORDER_NO, T1.ORDER_DATE,
T1.ORDER_DATE + INTERVAL '5 00:00:00.0' AS SHIPPING_DATE
FROM ORDER T1
```

- This example select all orders on a specific date:

```
SELECT
T1.ORDER_NO, TI.ORDER_DATE
FROM ORDER T1
WHERE (T1.ORDER_DATE = DATE '2005-12-01’)
```

Index