



Teradata[®] RDBMS

SQL Reference - Volume 3

Data Types and Literals

V2R5.0

B035-1101-122A
December 2002

The product described in this book is a licensed product of NCR Corporation.

BYNET is an NCR trademark registered in the U.S. Patent and Trademark Office.

CICS, CICS/400, CICS/600, CICS/ESA, CICS/MVS, CICS/PLEX, CICS/VIEW, CICS/VSE, DB2, DFSMS/MVS, DFSMS/VM, IBM, NQS/MVS, OPERATING SYSTEM/2, OS/2, PS/2, MVS, QMS, RACF, SQL/400, VM/ESA, and VTAM are trademarks or registered trademarks of International Business Machines Corporation in the U. S. and other countries.

DEC, DECNET, MICROVAX, VAX and VMS are registered trademarks of Digital Equipment Corporation.

HEWLETT-PACKARD, HP, HP BRIO, HP BRIO PC, and HP-UX are registered trademarks of Hewlett-Packard Co.

KBMS is a trademark of Trinzic Corporation.

INTERTEST is a registered trademark of Computer Associates International, Inc.

MICROSOFT, MS-DOS, MSN, The Microsoft Network, MULTIPLAN, SQLWINDOWS, WIN32, WINDOWS, WINDOWS 2000, and WINDOWS NT are trademarks or registered trademarks of Microsoft Corporation.

SAS, SAS/C, SAS/CALC, SAS/CONNECT, and SAS/CPE are registered trademarks of SAS Institute Inc.

SOLARIS, SPARC, SUN and SUN OS are trademarks of Sun Microsystems, Inc.

TCP/IP protocol is a United States Department of Defense Standard ARPANET protocol.

TERADATA and DBC/1012 are registered trademarks of NCR International, Inc.

UNICODE is a trademark of Unicode, Inc.

UNIX is a registered trademark of The Open Group.

X and X/OPEN are registered trademarks of X/Open Company Limited.

YNET is a trademark of NCR Corporation.

THE INFORMATION CONTAINED IN THIS DOCUMENT IS PROVIDED ON AN "AS-IS" BASIS, WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OF IMPLIED WARRANTIES, SO THE ABOVE EXCLUSION MAY NOT APPLY TO YOU. IN NO EVENT WILL NCR CORPORATION (NCR) BE LIABLE FOR ANY INDIRECT, DIRECT, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES, INCLUDING LOST PROFITS OR LOST SAVINGS, EVEN IF EXPRESSLY ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

The information contained in this document may contain references or cross references to features, functions, products, or services that are not announced or available in your country. Such references do not imply that NCR intends to announce such features, functions, products, or services in your country. Please consult your local NCR representative for those features, functions, products, or services available in your country.

Information contained in this document may contain technical inaccuracies or typographical errors. Information may be changed or updated without notice. NCR may also make improvements or changes in the products or services described in this information at any time without notice.

To maintain the quality of our products and services, we would like your comments on the accuracy, clarity, organization, and value of this document. Please e-mail: teradata-books@lists.ncr.com

or write:

Information Engineering
NCR Corporation
100 North Sepulveda Boulevard
El Segundo, CA 90245-4361
U.S.A.

Any comments or materials (collectively referred to as "Feedback") sent to NCR will be deemed non-confidential. NCR will have no obligation of any kind with respect to Feedback and will be free to use, reproduce, disclose, exhibit, display, transform, create derivative works of and distribute the Feedback and derivative works thereof without limitation on a royalty-free basis. Further, NCR will be free to use any ideas, concepts, know-how or techniques contained in such Feedback for any purpose whatsoever, including developing, manufacturing, or marketing products or services incorporating Feedback.

**Copyright © 1996-2002, NCR Corporation
All Rights Reserved**

Preface

Supported Software Release

This book supports Teradata RDBMS V2R5.0.

Changes to This Book

This book includes the following changes to support the current release:

Date	Description
December 2002	• COMPRESS default value control phrase now supports multivalued, comma-separated list of values • Addition of new formatting characters to FORMAT phrase to support cultural formatting feature • Replaced information on DefaultDateFormat with information on the default date format in a custom specification for data formatting (SDF) file • TIME and TIMESTAMP data types now support FORMAT phrase, system default format settings, and implicit conversion from CHARACTER types during assignment and comparison
June 2001	• Material on data literals from <i>Teradata RDBMS SQL Reference, Volume 1</i> moved into its own chapter in this volume • This volume renamed • General reorganization and rewriting of material to enhance usability • Performed DR and RFC enhancements
September 2000	• Usability enhancements
June 2000	• General reorganization and rewriting of material to enhance usability • Addition and extensive rewriting and reorganization of embedded SQL material formerly found in <i>Teradata RDBMS Application Programming with C, COBOL and PL/I</i>

About This Book

Purpose

Teradata RDBMS SQL Reference, Volume 3 describes how to use data types and literals within the Teradata Structured Query Language (SQL).

This preface describes the organization of *Teradata RDBMS SQL Reference, Volume 3* and identifies information you should know before using it. Use this book in conjunction with the other volumes of *Teradata RDBMS SQL Reference*.

Audience

System administrators, database administrators, and security administrators are the principal audience for this manual. NCR field engineers and other technical personnel responsible for designing and maintaining the Teradata RDBMS may also find this manual useful.

How This Book Is Organized

This book has eight chapters and one appendix.

- [Chapter 1: “SQL Data Definition”](#) describes the concept of data definition in SQL, Teradata extensions to ANSI-defined data typing, data type phrases, and column and table constraints.
- [Chapter 2: “Data Literals”](#) describes numeric, datetime, interval, character, graphic, and hexadecimal literals.
- [Chapter 3: “Decimal/Numeric Data Types”](#) describes the decimal/numeric data types supported by Teradata, including several extensions to the ANSI definitions. Particular attention is given to usage considerations for the system default DECIMAL(18) maximum value.
- [Chapter 4: “DateTime and Interval Data Types”](#) describes the DateTime and Interval data types and the Calendar view.
- [Chapter 5: “Character Data Types”](#) describes character data types. In addition, the chapter provides some discussion of the various character sets supported by Teradata.
- [Chapter 6: “Byte Data Types”](#) describes the BYTE and VARBYTE data types, both of which are Teradata extensions of the ANSI standard.
- [Chapter 7: “Default Value Control Phrases”](#) describes the default column definition phrases COMPRESS and NOT NULL.
- [Chapter 8: “Output Format Phrases”](#) describes the Teradata output formatting phrases FORMAT, AS, NAMED, and TITLE, all of which are extensions of the ANSI standard.
- [Appendix A: “Notation Conventions”](#) explains how to read syntax diagrams, character shorthand notation, and predicate calculus notation.

Prerequisites

You should be familiar with basic relational database management technology and SQL. This manual is not an SQL primer.

If you are not familiar with the Teradata RDBMS, you may find it useful to read *Introduction to the Teradata RDBMS* before reading this document.

For additional information about developing applications using embedded SQL, see *Teradata Preprocessor2 (PP2) Programming Guide*.

List of Acronyms

The following acronyms, listed in alphabetical order, apply to this release:

AMP	Access Module Processor vproc
ANSI	American National Standards Institute
BTEQ	Basic TeraData Query facility
BYNET	Banyan Network - High speed interconnect
CJK	Chinese, Japanese, and Korean
CLIV2	Call Level Interface Version 2
cs0, cs1, cs2, cs3	Four code sets (codeset 0, 1, 2, and 3) used in EUC encoding.
E2I	External-to-Internal
EUC	Extended UNIX Code
FK	Foreign Key
HI	Hash Index
I2E	Internal-to-External
JI	Join Index
JIS	Japanese Industrial Standards
LT/ST	Large Table/Small Table (join)
NUPI	Non-Unique Primary Index
NUSI	Non-Unique Secondary Index
OLAP	On-Line Analytical Processing
OLTP	On-Line Transaction Processing
PDE	Parallel Database Extensions
PE	Parsing Engine vproc
PK	Primary Key
RDBMS	Relational Database Management System
SDF	Specification for Data Formatting
SQL	Structured Query Language
UCS-2	Universal Coded Character Set containing 2 bytes
UPI	Unique Primary Index

USI	Unique Secondary Index
vproc	Virtual Process

Technical Information on the Web

The NCR home page (<http://www.ncr.com>) provides links to numerous sources of information about Teradata. Among the links provided are sites that deal with the following subjects:

- Contacting technical support
- Enrolling in customer education courses
- Ordering and downloading product documentation
- Accessing case studies of customer experiences with Teradata
- Accessing third party industry analyses of Teradata data warehousing products
- Accessing white papers
- Viewing or subscribing to various online periodicals

Contents

Preface

About This Book	ii
List of Acronyms	iv
Technical Information on the Web.....	vi

Chapter 1:

SQL Data Definition

Data Definition in SQL	1-2
Data Type Attributes: ANSI and Teradata Mode.....	1-4
Data Type Phrases	1-6
Column and Table Constraints	1-11
Constraints	1-12
Uniqueness Constraints.....	1-13
CHECK Constraints	1-14
Referential Constraints	1-15

Chapter 2:

Data Literals

Numeric Literals.....	2-2
Integer Literals	2-4
Decimal Literals	2-5
Floating Point Literals.....	2-7
Date and Time Literals	2-9
DateTime Literals	2-10
Date Literals	2-11
Time Literals.....	2-13
Timestamp Literals.....	2-17
Interval Literals.....	2-22
Interval Year-Month Literals	2-24
INTERVAL YEAR Literals	2-25
INTERVAL YEAR TO MONTH Literals	2-26
INTERVAL MONTH Literals.....	2-27
Interval Day-Time Literals	2-28
INTERVAL DAY Literals	2-29

INTERVAL DAY TO HOUR Literals	2-30
INTERVAL DAY TO MINUTE Literals	2-31
INTERVAL DAY TO SECOND Literals.....	2-32
INTERVAL HOUR Literals.....	2-33
INTERVAL HOUR TO MINUTE Literals.....	2-34
INTERVAL HOUR TO SECOND Literals	2-35
INTERVAL MINUTE Literals.....	2-36
INTERVAL MINUTE TO SECOND Literals	2-37
INTERVAL SECOND Literals	2-38
Character Data Literals	2-39
Graphic Literals	2-42
Hexadecimal Literals	2-44

Chapter 3:

Decimal/Numeric Data Types

ANSI/Teradata Correspondences for Naming Numeric Data Types.....	3-2
Rounding	3-3
DECIMAL/NUMERIC Data Types	3-5
Usage Considerations for DECIMAL(18)	3-9
DECIMAL(18) and the MaxDecimal DBSControl Flag.....	3-10
Behavior of SQL Operations Independent of MaxDecimal	3-12
Behavior for Different Values of MaxDecimal.....	3-13
CAST and the DECIMAL Data Type.....	3-15
DECIMAL Data Type Compatibility Issues	3-16
BYTEINT Data Type	3-17
SMALLINT Data Type.....	3-19
INTEGER Data Type.....	3-21
FLOAT/REAL/DOUBLE PRECISION Data Types.....	3-23

Chapter 4:

DateTime and Interval Data Types

DateTime Fields	4-2
Time Zones	4-4
DATE	4-8
DATE Data Type Extensions with CALENDAR System View	4-16
DATE Formats	4-18
TIME	4-22
TIMESTAMP	4-24
TIME WITH TIME ZONE	4-27
TIMESTAMP WITH TIME ZONE	4-29
TIME and TIMESTAMP Formats.....	4-31

INTERVAL YEAR	4-37
INTERVAL YEAR TO MONTH	4-38
INTERVAL MONTH	4-39
INTERVAL DAY	4-40
INTERVAL DAY TO HOUR	4-41
INTERVAL DAY TO MINUTE	4-42
INTERVAL DAY TO SECOND	4-44
INTERVAL HOUR	4-46
INTERVAL HOUR TO MINUTE	4-47
INTERVAL HOUR TO SECOND	4-48
INTERVAL MINUTE	4-50
INTERVAL MINUTE TO SECOND	4-51
INTERVAL SECOND	4-53

Chapter 5:

Character Data Types

Character Data	5-2
CHARACTER Type	5-3
VARCHAR Type	5-9
LONG VARCHAR Type	5-14
Defining Case Sensitivity for Table Columns	5-16
CASESPECIFIC Phrase	5-17
UPPERCASE Phrase	5-21
Client Character Set.....	5-23
Teradata Character Strings and Client Physical Bytes.....	5-24
CHARACTER SET Clause	5-28
LATIN Server Character Set	5-29
UNICODE Server Character Set.....	5-30
GRAPHIC Server Character Set	5-31
KANJISJIS Server Character Set.....	5-32
KANJI1 Server Character Set.....	5-33

Chapter 6:

Byte Data Types

BYTE and VARBYTE Data	6-2
BYTE	6-3
VARBYTE	6-5

Chapter 7:	
Default Value Control Phrases	
Using Default Value Control Phrases.....	7-2
COMPRESS	7-3
NOT NULL	7-7
DEFAULT	7-8
WITH DEFAULT	7-13

Chapter 8:	
Output Format Phrases	
Data Display Default Formats	8-2
Output Format Phrases.....	8-5
FORMAT	8-6
FORMAT Phrase and CHARACTER Formats	8-10
FORMAT Phrase and NUMERIC Formats	8-12
FORMAT Phrase and DateTime Formats	8-29
FORMAT Phrase, DateTime Formats, and Japanese Character Sets	8-37
Naming Columns and Expressions	8-40
AS.....	8-43
NAMED	8-44
TITLE	8-46

Appendix A:	
Notation Conventions	
Syntax Diagram Conventions.....	A-2
Character Shorthand Notation Used In This Manual	A-6
Predicate Calculus Notation Used in This Manual	A-8

Index.....	Index-1
-------------------	----------------

Chapter 1:

SQL Data Definition

This chapter describes some of the general principles of SQL data definition.

The specific topics described in the chapter are:

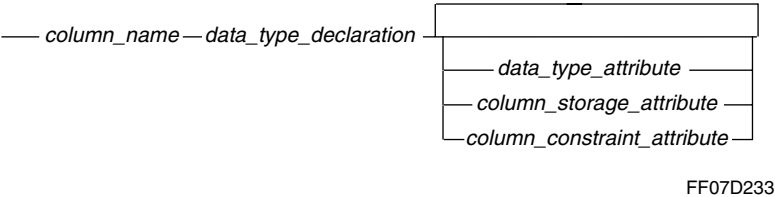
- Data definition in SQL, including:
 - Data type attributes: ANSI and Teradata mode
 - Data type phrases
- Column and table constraints, including:
 - Uniqueness constraints
 - CHECK constraints
 - Referential constraints

Data Definition in SQL

Introduction

Data type attributes control the internal representation of stored data and determine how data is presented for a column or an expression result.

Syntax



where:

Syntax element ...	Is further described in ...
<i>data_type_declaration</i>	the summary table under “Data Type Phrases” on page 1-6.
<i>data_type_attribute</i>	the summary tables under: • “Core Data Type Attributes” on page 1-8. • “FLOAT/REAL/DOUBLE PRECISION Data Types” on page 3-23.
<i>column_storage_attribute</i>	“ALTER TABLE” and “CREATE TABLE” in the chapter “SQL Data Definition Language Statement Syntax” in <i>Teradata RDBMS SQL Reference, Volume 4.</i>
<i>column_constraint_attribute</i>	“ALTER TABLE” and “CREATE TABLE” in the chapter “SQL Data Definition Language Statement Syntax” in <i>Teradata RDBMS SQL Reference, Volume 4.</i> Also see “Column and Table Constraints” on page 1-11.

You must define data type attributes when you define a column. You can omit the data type definitions when modifying data.

Using Data Definition

Data definition can occur in any order in a statement. Data definitions are also referred to as data description phrases.

When used to define the attributes of a data value, also referred to as data type attributes, the data definition immediately follows a column name.

In the following CREATE TABLE statement, the following columns are assigned attributes using SQL data definitions:

- EmpNo
- Name
- DeptNo
- JobTitle
- Salary
- YrsExp
- DOB
- Sex
- Race
- MStat
- EdLev
- HCap

```
CREATE TABLE Employee
(EmpNo PRIMARY KEY SMALLINT FORMAT '9(5)'
  CHECK(EmpNo BETWEEN 1000 AND 32001),
Name VARCHAR(12) CHARACTER SET LATIN NOT CASESPECIFIC NOT NULL,
DeptNo SMALLINT FORMAT '999'
  CHECK (DeptNo BETWEEN 100 AND 900),
JobTitle VARCHAR(12) CHARACTER SET LATIN NOT CASESPECIFIC,
Salary DECIMAL(8,2) FORMAT 'ZZZ,ZZ9.99'
  CHECK (Salary BETWEEN 1.00 AND 999000.00),
YrsExp BYTEINT FORMAT 'Z9'
  CHECK (YrsExp BETWEEN -99 AND 99),
DOB DATE FORMAT 'MMbDDbYYYY' NOT NULL,
Sex CHAR CHARACTER SET LATIN UPPERCASE NOT NULL,
Race CHAR CHARACTER SET LATIN UPPERCASE,
MStat CHAR CHARACTER SET LATIN UPPERCASE,
EdLev BYTEINT FORMAT 'Z9'
  CHECK (EdLev BETWEEN 0 AND 2) NOT NULL,
HCap BYTEINT FORMAT 'Z9'
  CHECK (HCap BETWEEN -99 AND 99)
INDEX (Name) ;
```

Data Type Attributes: ANSI and Teradata Mode

Introduction

When used to modify the attributes of returned values, the data description phrase immediately follows the column being modified. This usage is non-ANSI.

For example, in the following SELECT statement, a TITLE phrase overrides the default heading (the column name) for EmpNo, and a FORMAT phrase modifies the display format defined for Salary data in the CREATE TABLE statement.

```
SELECT EmpNo (TITLE 'Emp#'), Salary (FORMAT '$$$,$$9.99')
FROM Employee;
```

The expression “data type attributes” is non-ANSI, as are attributes such as TITLE and FORMAT. You should use the CAST function for applications that need to conform to ANSI as well as to modify attributes of a value. In this case, the modification is restricted to converting to an ANSI defined type. The CAST function is described later in detail.

Example

To select Name as a fixed length field and Salary as an integer value:

```
SELECT CAST (Name AS CHAR(12)), CAST (Salary AS INTEGER)
FROM Employee ;
```

ANSI Versus Teradata Mode

A move to be fully ANSI-compliant with regard to case sensitivity, would have caused large regression problems. To facilitate the transition to ANSI SQL compatibility, two modes of operation dealing with character comparison rules, referred to as Teradata and ANSI modes, are provided.

Installations may choose either Teradata or ANSI mode as a system default, and sessions can individually be set to override the system default. For further information on ANSI and Teradata processing modes, see Chapter 4: “Transaction Processing” in *Teradata RDBMS SQL Reference, Volume 2*.

Because some instances of comparison attributes occur with data definitions, and others with text in queries (or text from dictionary tables scanned as queries are executed), use of the *two modes* can be rather involved and you should exercise caution when doing so.

Use the most current Teradata SQL syntax for all newly written applications because it is the most ANSI-compliant. Teradata SQL still supports old applications, written in prior non-ANSI-compliant versions of Teradata SQL.

Data Type Phrases

Definition

A data type phrase:

- Specifies how to present data to the user
- Determines how to store data on Teradata RDBMS

A data type phrase must be specified for each column when a table is created. No default data type is provided. A data type phrase can also be used to specify data conversions in expressions.

Data Type Declarations

Use the data type phrases in the following table to define the data type as numeric, character, datetime, interval, or byte.

Data Type	ANSI SQL	Non-ANSI SQL or Teradata Extension to ANSI SQL	Not Entry-Level ANSI SQL
Numeric			
INTEGER	X		
SMALLINT	X		
BYTEINT		X	X
DATE ¹		X	X
REAL	X		
DOUBLE PRECISION	X		
FLOAT [n]	X		
DECIMAL [(n[,m])]	X		
NUMERIC [(n[,m])]	X		
DateTime			
DATE ^a	X		
TIME [(n)]	X		
TIMESTAMP [(n)]	X		
TIME [(n)] WITH TIME ZONE	X		
TIMESTAMP [(n)] WITH TIME ZONE	X		

Data Type	ANSI SQL	Non-ANSI SQL or Teradata Extension to ANSI SQL	Not Entry-Level ANSI SQL
Interval			
INTERVAL	X		
INTERVAL YEAR [(n)]	X		
INTERVAL YEAR [(n)] TO MONTH	X		
INTERVAL MONTH	X		
INTERVAL DAY [(n)]	X		
INTERVAL DAY [(n)] TO HOUR	X		
INTERVAL DAY [(n)] TO MINUTE	X		
INTERVAL DAY [(n)] TO SECOND	X		
INTERVAL HOUR [(n)]	X		
INTERVAL HOUR [(n)] TO MINUTE	X		
INTERVAL HOUR [(n)] TO SECOND	X		
INTERVAL MINUTE [(n)]	X		
INTERVAL MINUTE [(n)] TO SECOND [(m)]	X		
INTERVAL SECOND [(n,[m])]	X		
Character			
CHAR[(n)]	X		
VARCHAR(n)	X		X
CHAR VARYING(n)	X		X
LONG VARCHAR		X	X
Byte^b			
BYTE[(n)]		X	X
VARBYTE(n)		X	X
Graphic^c			
GRAPHIC[(n)]		X	X
VARGRAPHIC(n)		X	X
LONG VARGRAPHIC		X	X

- a. DATE is supported both in its Teradata form and in the preferred ANSI DateTime form.
For any new development, define DATE using ANSI DATE type rather than REAL type.
- b. BYTE data types are a Teradata extension to the ANSI standard.
- c. The GRAPHIC types are equivalent to the type CHARACTER CHARACTER SET GRAPHIC, which is the form to use for all new development.

For further information on the different level of ANSI compliance in this release, see Appendix C: “Differences Between Teradata and ANSI SQL” in *Teradata RDBMS SQL Reference, Volume 1*.

Client System Data Types

In addition to SQL data types, Teradata supports numeric, character, and byte data types as used on various supported client systems.

Core Data Type Attributes

The following table summarizes the main Teradata data type attributes.

Data Type Attribute	ANSI	Teradata Extension to ANSI
NOT NULL	X	
UPPERCASE		X
[NOT] CASESPECIFIC		X
FORMAT <i>quote_string</i>		X
TITLE <i>quote_string</i>		X
NAMED <i>name</i>		X
DEFAULT <i>number</i>	X	
DEFAULT USER	X	
DEFAULT DATE		X
DEFAULT TIME		X
DEFAULT NULL	X	
WITH DEFAULT		X
CHARACTER SET	X	

Additional Data Type Attributes

The following table summarizes additional Teradata data type attributes.

Attribute Family	Data Type Attribute	ANSI	Teradata Extension to ANSI
Column Storage	Column storage attributes are a Teradata extension to ANSI.		
	COMPRESS		X
	COMPRESS NULL		X
	COMPRESS <i>quote_string</i>		X
	COMPRESS <i>constant_value</i>		X
Column Constraints	Constraint names are intermediate ANSI SQL.		
	CONSTRAINT	X	
	CONSTRAINT UNIQUE	X	
	CONSTRAINT PRIMARY KEY	X	
	CONSTRAINT CHECK (<i>boolean_condition</i>)	X	
	CONSTRAINT REFERENCES <i>table_name</i> <i>column_name</i>	X	
Table Constraints	Table constraints are intermediate ANSI SQL.		
	FOREIGN KEY (<i>column_name_list</i>)	X	
	PRIMARY KEY	X	
	UNIQUE	X	
	CHECK (<i>boolean_expression</i>)	X	
	REFERENCES <i>table_name</i> [(<i>column_name_list</i>)]	X	

Attribute Family	Data Type Attribute	ANSI	Teradata Extension to ANSI
Character Data Literal Character Set Definition	Character set specifications for CHARACTER literals are full ANSI SQL compliant. Note that <code>_Kanji1</code> is <i>not</i> a valid character literal character set attribute. While the ANSI standard defines the concept of defining character sets for CHARACTER data literals, the supported character sets are vendor- and implementation-specific. For more detailed information about character data literal character set definition attributes, see “Character Data Literals” in Chapter 3: “SQL Lexicon” of <i>Teradata RDBMS SQL Reference, Volume 1</i>.		
	<code>_Graphic</code>	X	
	<code>_KanjiSJIS</code>	X	
	<code>_Latin</code>	X	
	<code>_Unicode</code>	X	

Column and Table Constraints

Introduction

You can specify constraints on column values when the columns are defined during table creation and modification.

These constraints can be defined on single columns as a part of the column definition, or on multiple columns. The full syntax for such constraints is in “ALTER TABLE” and “CREATE TABLE” in the chapter “SQL Data Definition Language Statement Syntax” in *Teradata RDBMS SQL Reference, Volume 4*.

Column Constraints

Column constraints include:

- PRIMARY KEY
- UNIQUE
- CHECK (*boolean_expression*)
- REFERENCES tablename [(*column_name_list*)]

Table Level Constraints

Table level constraints include:

- FOREIGN KEY (*column_name_list*)
- PRIMARY KEY
- UNIQUE
- CHECK (*boolean_expression*)
- REFERENCES tablename [(*column_name_list*)]

Constraints

Introduction

Constraints can be any of the following types:

- Uniqueness (see [“Uniqueness Constraints” on page 1-13](#)).
- CHECK (see [“CHECK Constraints” on page 1-14](#)).
- Referential integrity (see [“Referential Constraints” on page 1-15](#)).

Uniqueness Constraints

Definition

UNIQUE means the table cannot include two or more rows in which the values for the column or set of columns are identical.

PRIMARY KEY Constraint

A column or set of columns defined as PRIMARY KEY must also be NOT NULL.

Teradata RDBMS instantiates a PRIMARY KEY as a UNIQUE [PRIMARY] INDEX.

Only one PRIMARY KEY can be defined for a table.

For more details, see:

- “CREATE TABLE” in the chapter “SQL Data Definition Language Statement Syntax” of *Teradata RDBMS SQL Reference, Volume 4*.
- “Primary and Foreign Keys” in the chapter “SQL Data Definition, Control, and Manipulation” of *Teradata RDBMS SQL Reference, Volume 1*.
- “Constraint Specifications” in the chapter “Designing for Database Integrity” in *Teradata RDBMS Database Design*.

UNIQUE Constraint

A UNIQUE column definition requires that data cannot be NULL. Queries are flagged as non-entry-level ANSI when the SQL flagger is enabled if any listed column is not also defined as NOT NULL.

Teradata RDBMS instantiates UNIQUE as a unique primary or secondary index.

Also see “Constraint Specifications” in the chapter “Designing for Database Integrity” in *Teradata RDBMS Database Design*.

CHECK Constraints

Purpose

CHECK constraints compare values of a field or fields in the same row with constants or other fields.

Notes

CHECK constraints are applied to rows generated as candidates for INSERT and UPDATE operations.

You cannot specify CHECK constraints for Identity columns.

If the condition is met or the proposed INSERT or UPDATE contains NULLs, the operation is permitted.

If the condition is not met, an error is reported as a constraint violation. For details on the syntax, see “CREATE TABLE” in Chapter 1: “SQL Data Definition Language Statement Syntax” of *Teradata RDBMS SQL Reference, Volume 4*.

Also see "Constraint Specifications" in the chapter "Designing for Database Integrity" in *Teradata RDBMS Database Design*.

Examples

```
CHECK (Sex = 'F' OR Sex = 'M')  
CHECK (EdLev BETWEEN 0 AND 22 AND YrsExp BETWEEN 0 and 70)
```

Referential Constraints

Purpose

Use referential constraints to indicate relationships between columns of different tables. There are three specific types of referential constraints:

Referential Constraint Type	DDL Definition	Does It Enforce Referential Integrity?	Level of Referential Integrity Enforcement
Referential constraint	REFERENCES WITH NO CHECK OPTION	No	None
Batch referential integrity constraint	REFERENCES WITH CHECK OPTION	Yes	Implicit transaction
Referential integrity constraint	REFERENCES	Yes	Row

Notes

You cannot specify REFERENCES constraints for Identity columns.

For additional information about REFERENCES constraints, see:

- “ALTER TABLE” and “CREATE TABLE” in the chapter “SQL Data Definition Language Statement Syntax” of *Teradata RDBMS SQL Reference, Volume 4*.
- “Referential Integrity” in the chapter “SQL Data Definition, Control, and Manipulation” in *Teradata RDBMS SQL Reference, Volume 1*.
- “Constraint Specifications” in the chapter “Designing for Database Integrity” in *Teradata RDBMS Database Design*.



Chapter 2:

Data Literals

Literals are values coded directly in the text of an SQL statement, view or macro definition text, or CHECK constraint definition text. In general, the system is able to determine the type of a literal by its form.

Topics described in this chapter include:

- “Numeric Literals”
- “Date and Time Literals”
- “Interval Literals”
 - Year-Month Interval Literals
 - Day-Time Interval Literals
- “Character Data Literals”
- “Graphic Literals”
- “Hexadecimal Literals”

Numeric Literals

Definition

A numeric literal is a character string of 1 to 20 characters selected from the following:

- digits 0 through 9
- plus sign
- minus sign
- decimal point

Literals are also referred to as constants.

Types of Numeric Literals

There are three kinds of numeric literals:

- Integers (see [“Integer Literals”](#))
- Decimals (see [“Decimal Literals”](#))
- Floating point numbers (see [“Floating Point Literals”](#))

Examples

The following are examples of valid numeric literals.

BYTEINT	SMALLINT	INTEGER	DECIMAL	FLOATING POINT
127	32767	32768	0.0	1E1
-36	-12000	-60400	-235.	1.4E6
-128	-32768	2147483647	2147483650	18E-3

Differences Among BYTEINT, SMALLINT, and INTEGER

The differences among BYTEINT, SMALLINT, and INTEGER are the range of values allowed and the internal storage sizes.

This numeric literal type ...	Occupies this many bytes of storage space ...	With this range of values ...	
		Low	High
BYTEINT	1	-128	127
SMALLINT	2	-32768	32767
INTEGER	4	-2147483648	2147483647

Depending on the kind of operation performed or the type of column in which a value is to be stored, SQL may convert numeric literal constants from one data type to another or convert character strings that represent numeric values to a numeric data type.

Converting Numeric Literals

If a literal is outside the range for its required type, an error is reported during conversion. An error also is reported if a character string that is assigned to a numeric field does not represent a valid numeric value.

Converting Character Strings to Numeric Literals

Character data can contain a string that is intended to be interpreted as a numeric value (for example, '15'). In such cases, the Teradata RDBMS attempts to convert the string to a numeric value whenever the context makes such a conversion necessary.

Note that this can result in repeated conversions during data access. Thus, when a character field must be compared with a numeric field, the character field is converted to numeric before the comparison is made.

If a numeric field is defined and stored as a character field, and a SELECT operation uses a full table scan to compare the character field with a numeric literal, then the character field is converted to numeric in every row of the table.

Examples of Non-Valid Numeric Literals

The following are examples of non-valid numeric literals.

This literal is not valid ...	Because it contains ...
1234567890123456789012	more than 18 digits.
\$20,000.00	a dollar sign and a comma.
-38.7E2945	four digits following the E.

Integer Literals

Purpose

Declares literal strings of integer numbers in an expression, view or macro definition text, or CHECK constraint text.

Syntax

_____ ± n _____
KR01A141

where:

Syntax element ...	Specifies ...
±	an optional sign. The default is +.
n	any valid integer.

Components of Integer Literals

Integer literals consist of an optional sign followed by a sequence of digits.

Integer Literal Data Types

The data type of an integer literal is determined by the range of the literal value. The type used is the smallest that can contain the value.

Integer literals include the following data types:

- INTEGER
- SMALLINT
- BYTEINT

Examples

The following numbers are examples of integer literals.

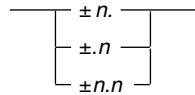
12
0
-5

Decimal Literals

Purpose

Declares literal strings of decimal numbers in an expression, view or macro definition text, or CHECK constraint text.

Syntax



KR01A143

where:

Syntax element ...	Specifies ...
±	an optional sign. The default is +.
n	any valid integer.

Components of Decimal Literals

Decimal literals consist of the following components, reading from left-to-right:

- 1 Optional sign
- 2 Sequence of digits (including none)
- 3 Decimal point
- 4 Optional sequence of digits.

The scale and precision of a decimal literal are determined by the total number of digits in the literal and the number of digits to the right of the decimal point, respectively.

Decimal Literal Data Types

Decimal literals include the following types:

- DECIMAL
- NUMERIC

Examples

The following numbers are examples of decimal literals.

3.14159

253.

-88.234

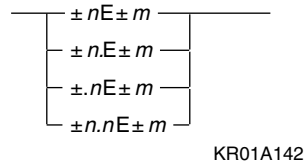
.29

Floating Point Literals

Purpose

Declares literal strings of floating point numbers in an expression, view or macro definition text, or CHECK constraint text.

Syntax



where:

Syntax element ...	Specifies ...
±	an optional sign. The default is +.
n	any valid integer representing the whole and, optionally, fractional component of the mantissa.
E	the symbol indicating that what follows is the exponent for the number.
m	any valid integer number representing the exponent for the number.

Components of Floating Point Literals

Floating point literals consist of the following components, reading from left-to-right:

- 1 Optional sign
- 2 Sequence of digits (including none) representing the whole number portion of the mantissa
- 3 Optional decimal point
- 4 Sequence of digits (including none) representing the fractional portion of the mantissa
- 5 Literal character E
- 6 Optional sign
- 7 Sequence of digits (including none) representing the exponent

Floating Point Literal Data Types

Floating point literals include the following types:

- FLOATING POINT
- REAL
- DOUBLE PRECISION

Examples

The following numbers are examples of floating point literals.

1E100
3.14E-10
6.023E23

Date and Time Literals

Date and time literals declare date, time, or timestamp values in an SQL expression, view or macro definition text, or CONSTRAINT definition text.

ANSI DateTime Literals

ANSI SQL provides DateTime literals to represent date, time, and timestamp values. There are three types of ANSI SQL DateTime literals:

- DATE
- TIME
- TIMESTAMP

Teradata Date and Time Literals

Teradata also provides a non-ANSI extension to DateTime functionality.

Teradata literals used with date or time values are simple quote strings and are interpreted as character data. These are converted to a date or time value based on the context, which is usually provided by a FORMAT clause.

The existing Teradata operations on character string literals used to represent date and time values are supported as a non-ANSI extension to DateTime functionality.

While Teradata date and time literals come in a variety of formats for the character representations of date and time values (both typed REAL), and so cannot be restricted to a default or standard format, ANSI DateTime literals are restricted to a strict set of formats.

Existing functionality for Teradata date literals is preserved.

For example, suppose a field is defined as DATE with a format of 'YY/MM/DD'. If the value '97/12/31' is encountered as a character literal assigned to a DATE field, the process works just as it did prior to the introduction of ANSI DateTime formats because the character literal is implicitly cast to the DATE format.

Unless explicitly stated, this chapter only discusses ANSI DateTime literals.

DateTime Literals

Introduction

The ANSI SQL DateTime literals provide DateTime values in source text, supporting a means for declaring values for DATE, TIME, and TIMESTAMP data types.

ANSI DateTime Literals Require a Keyword

ANSI SQL DateTime literals differ from other SQL literals in that they are always prefaced by a keyword or keywords.

Here is an example showing the TIMESTAMP keyword prefacing a timestamp literal:

```
TIMESTAMP '1999-07-01 15:00:00-08:00'
```

Literals expressed in this format are interpreted exclusively as ANSI DateTime values.

For More Information

For more detailed descriptions of ANSI DateTime literals, see “SQL Lexicon” in *Teradata RDBMS SQL Reference, Volume 1*.

Date Literals

Purpose

Declares a date value in ANSI DATE format in an expression, view or macro definition text, or CHECK constraint text.

ANSI DATE literal is the preferred format for date constants. All date operations accept this format.

Syntax

DATE — ' *string* '

1101A036

where:

Syntax element ...	Specifies ...								
<i>string</i>	a 10-character string enclosed in single quotes in the following form: YYYY-MM-DD								
	<table><tr><th>This string ...</th><th>Represents ...</th></tr><tr><td>YYYY</td><td>year. The valid range is 0001 through 9999, inclusive. You must specify all four digits.</td></tr><tr><td>MM</td><td>month. The valid range is 01 through 12, inclusive. You must specify both digits.</td></tr><tr><td>DD</td><td>day. The valid range is 01 through 31, inclusive, constrained by Gregorian calendar definitions. You must specify both digits.</td></tr></table>	This string ...	Represents ...	YYYY	year. The valid range is 0001 through 9999, inclusive. You must specify all four digits.	MM	month. The valid range is 01 through 12, inclusive. You must specify both digits.	DD	day. The valid range is 01 through 31, inclusive, constrained by Gregorian calendar definitions. You must specify both digits.
This string ...	Represents ...								
YYYY	year. The valid range is 0001 through 9999, inclusive. You must specify all four digits.								
MM	month. The valid range is 01 through 12, inclusive. You must specify both digits.								
DD	day. The valid range is 01 through 31, inclusive, constrained by Gregorian calendar definitions. You must specify both digits.								

Usage Notes

Date literals consist of the word DATE followed by a character string literal. This character string specifies the date value.

The year, month, and day components of the quotestring literal must be separated by hyphens.

Type

DATE

Example

The following example selects all classes from the Classes table that start on January 6, 1998.

```
SELECT *  
FROM CLASSES  
WHERE startdate = DATE '1998-01-06';
```

Time Literals

Purpose

Declares a time value in an expression, view or macro definition text, or CHECK constraint text.

Syntax

— TIME — ' *string* ' —

1101A021

where *string* specifies an 8- to 21-character string enclosed in single quotes in one of four formats.

This format ...	Represents ...	
hh:mi:ss	time with no fractional seconds digits or Time Zone. The TYPE is TIME(0).	
	This string ...	Represents ...
	hh	hour of the day. The valid range is 00–23, inclusive. You must specify both digits.
	mi	minute of the hour. The valid range is 00–59, inclusive. You must specify both digits.
	ss	seconds. The valid range is 00–59. You must specify both digits.

This format ...	Represents ...										
hh:mi:ss ^{sign} hh:mi	time with a specified Time Zone, but no fractional seconds digits. The TYPE is TIME(0) WITH TIME ZONE. <table> <tr> <th>This string ...</th><th>Represents ...</th></tr> <tr> <td>hh</td><td> hour of the day. The valid range is 00–23, inclusive, and you must specify both digits. </td></tr> <tr> <td>mi</td><td> minute of the hour. The valid range is 00–59, inclusive, and you must specify both digits. </td></tr> <tr> <td>ss</td><td> seconds. The valid range is 00–59, and you must specify both digits. </td></tr> <tr> <td>^{sign}hh:mi</td><td> hours and minutes in the Time Zone offset. The valid range is -12:59 through +13:00, inclusive. ^{sign} is + or - </td></tr> </table>	This string ...	Represents ...	hh	hour of the day. The valid range is 00–23, inclusive, and you must specify both digits.	mi	minute of the hour. The valid range is 00–59, inclusive, and you must specify both digits.	ss	seconds. The valid range is 00–59, and you must specify both digits.	^{sign} hh:mi	hours and minutes in the Time Zone offset. The valid range is -12:59 through +13:00, inclusive. ^{sign} is + or -
This string ...	Represents ...										
hh	hour of the day. The valid range is 00–23, inclusive, and you must specify both digits.										
mi	minute of the hour. The valid range is 00–59, inclusive, and you must specify both digits.										
ss	seconds. The valid range is 00–59, and you must specify both digits.										
^{sign} hh:mi	hours and minutes in the Time Zone offset. The valid range is -12:59 through +13:00, inclusive. ^{sign} is + or -										
hh:mi:ss.ssssss	time with up to six fractional seconds digits, but no Time Zone. The TYPE is TIME(n), where n is the number of fractional seconds digits in <i>string</i>. <table> <tr> <th>This string ...</th><th>Represents ...</th></tr> <tr> <td>hh</td><td> hour of the day. The valid range is 00–23, inclusive, and you must specify both digits. </td></tr> <tr> <td>mi</td><td> minute of the hour. The valid range is 00–59, inclusive, and you must specify both digits. </td></tr> <tr> <td>ss.ssssss</td><td> seconds. The valid range for the first two digits is 00–59. You must specify both digits. You can specify from one to six fractional digits. </td></tr> </table>	This string ...	Represents ...	hh	hour of the day. The valid range is 00–23, inclusive, and you must specify both digits.	mi	minute of the hour. The valid range is 00–59, inclusive, and you must specify both digits.	ss.ssssss	seconds. The valid range for the first two digits is 00–59. You must specify both digits. You can specify from one to six fractional digits.		
This string ...	Represents ...										
hh	hour of the day. The valid range is 00–23, inclusive, and you must specify both digits.										
mi	minute of the hour. The valid range is 00–59, inclusive, and you must specify both digits.										
ss.ssssss	seconds. The valid range for the first two digits is 00–59. You must specify both digits. You can specify from one to six fractional digits.										

This format ...	Represents ...										
hh:mi:ss.ssssssignhh:mi	time with up to six fractional seconds and a Time Zone offset. The TYPE is TIME(n) WITH TIME ZONE, where n is the number of fractional seconds digits in <i>string</i>. <table> <tr> <th>This string ...</th><th>Represents ...</th></tr> <tr> <td>hh</td><td> hour of the day. The valid range is 00–23, inclusive. You must specify both digits. </td></tr> <tr> <td>mi</td><td> minute of the hour. The valid range is 00–59, inclusive. You must specify both digits. </td></tr> <tr> <td>ss.sssss</td><td> seconds. The valid range for the first two digits is 00–59. You must specify both digits. You can specify from one to six fractional digits. </td></tr> <tr> <td>signhh:mi</td><td> hours and minutes in the Time Zone offset. The valid range is -12:59 through +13:00, inclusive. <i>sign</i> is + or - </td></tr> </table>	This string ...	Represents ...	hh	hour of the day. The valid range is 00–23, inclusive. You must specify both digits.	mi	minute of the hour. The valid range is 00–59, inclusive. You must specify both digits.	ss.sssss	seconds. The valid range for the first two digits is 00–59. You must specify both digits. You can specify from one to six fractional digits.	signhh:mi	hours and minutes in the Time Zone offset. The valid range is -12:59 through +13:00, inclusive. <i>sign</i> is + or -
This string ...	Represents ...										
hh	hour of the day. The valid range is 00–23, inclusive. You must specify both digits.										
mi	minute of the hour. The valid range is 00–59, inclusive. You must specify both digits.										
ss.sssss	seconds. The valid range for the first two digits is 00–59. You must specify both digits. You can specify from one to six fractional digits.										
signhh:mi	hours and minutes in the Time Zone offset. The valid range is -12:59 through +13:00, inclusive. <i>sign</i> is + or -										

Usage Notes

The colons are required between the first three elements, and the decimal point is required if fractional seconds are specified.

Example 1: hh:mi:ss Format

The following example selects all classes from the Classes table that start at 3:30 PM.

```
SELECT *
FROM CLASSES
WHERE start_time = TIME '15:30:00';
```

Example 2: hh:mi:ss Format

```
TIME '10:37:12-08:00'
```

Example 3: hh:mi:ss.ssssss Format

```
TIME '10:36:02.123456'
```

Example 4: hh:mi:ss.ssssssignhh:mi Format

```
TIME '21:17:35.123456+07:30'
```

Timestamp Literals

Purpose

Declares a timestamp value in an expression, view or macro definition text, or CHECK constraint text.

Syntax

—— TIMESTAMP —— ' *string* ' ——

1101A022

where *string* specifies a character string in single quotes in one of four formats.

This format ...	Represents ...														
YYYY-MM-DD hh:mi:ss	timestamp with no time zone or fractional seconds digits. The TYPE is TIMESTAMP(0). <table><tr><th>This string ...</th><th>Represents ...</th></tr><tr><td>YYYY</td><td>year. The valid range is from 0001–9999, and you must specify all four digits.</td></tr><tr><td>MM</td><td>month. The valid range is from 01–12, and you must specify both digits.</td></tr><tr><td>DD</td><td>day. The valid range is from 01–31, constrained by Gregorian calendar definitions. You must specify both digits, and follow them by a single pad character.</td></tr><tr><td>hh</td><td>hour of the day. The valid range is from 00–23, and you must specify both digits.</td></tr><tr><td>mi</td><td>minute of the hour. The valid range is from 00–59, and you must specify both digits.</td></tr><tr><td>ss</td><td>seconds. The valid range is 00–59, and you must specify both digits.</td></tr></table>	This string ...	Represents ...	YYYY	year. The valid range is from 0001–9999, and you must specify all four digits.	MM	month. The valid range is from 01–12, and you must specify both digits.	DD	day. The valid range is from 01–31, constrained by Gregorian calendar definitions. You must specify both digits, and follow them by a single pad character.	hh	hour of the day. The valid range is from 00–23, and you must specify both digits.	mi	minute of the hour. The valid range is from 00–59, and you must specify both digits.	ss	seconds. The valid range is 00–59, and you must specify both digits.
This string ...	Represents ...														
YYYY	year. The valid range is from 0001–9999, and you must specify all four digits.														
MM	month. The valid range is from 01–12, and you must specify both digits.														
DD	day. The valid range is from 01–31, constrained by Gregorian calendar definitions. You must specify both digits, and follow them by a single pad character.														
hh	hour of the day. The valid range is from 00–23, and you must specify both digits.														
mi	minute of the hour. The valid range is from 00–59, and you must specify both digits.														
ss	seconds. The valid range is 00–59, and you must specify both digits.														

This format ...	Represents ...	
YYYY-MM-DD hh:mi:sssignhh:mi	a timestamp with a specified time zone offset, but no fractional seconds digits. The TYPE is TIMESTAMP(0) WITH TIME ZONE.	
	This string ...	Represents ...
	YYYY	year. The valid range is from 0001–9999, and you must specify all four digits.
	MM	month. The valid range is from 01–12, and you must specify both digits.
	DD	day. The valid range is from 01–31, constrained by Gregorian calendar definitions. You must specify both digits, and follow them by a single pad character.
	hh	hour of the day. The valid range is 00–23, inclusive, and you must specify both digits.
	mi	minute of the hour. The valid range is 00–59, inclusive, and you must specify both digits.
	ss	seconds. The valid range is 00–59, and you must specify both digits.
	signhh:mi	hours and minutes in the Time Zone offset. The valid range is -12:59 through +13:00, inclusive. <i>sign</i> is + or -

This format ...	Represents ...														
YYYY-MM-DD hh:mi:ss.ssssss	a timestamp with up to six fractional seconds digits, but no Time Zone. The TYPE is TIMESTAMP(n), where n is the number of fractional seconds digits in <i>string</i>. <table> <tr> <th>This string ...</th><th>Represents ...</th></tr> <tr> <td>YYYY</td><td> year. The valid range is from 0001–9999, and you must specify all four digits. </td></tr> <tr> <td>MM</td><td> month. The valid range is from 01–12, and you must specify both digits. </td></tr> <tr> <td>DD</td><td> day. The valid range is from 01–31, constrained by Gregorian calendar definitions. You must specify both digits, and follow them by a single pad character. </td></tr> <tr> <td>hh</td><td> hour of the day. The valid range is 00–23, inclusive, and you must specify both digits. </td></tr> <tr> <td>mi</td><td> minute of the hour. The valid range is 00–59, inclusive, and you must specify both digits. </td></tr> <tr> <td>ss.ssssss</td><td> seconds. The valid range for the first two digits is 00–59. You must specify both digits. You can specify from one to six fractional digits. </td></tr> </table>	This string ...	Represents ...	YYYY	year. The valid range is from 0001–9999, and you must specify all four digits.	MM	month. The valid range is from 01–12, and you must specify both digits.	DD	day. The valid range is from 01–31, constrained by Gregorian calendar definitions. You must specify both digits, and follow them by a single pad character.	hh	hour of the day. The valid range is 00–23, inclusive, and you must specify both digits.	mi	minute of the hour. The valid range is 00–59, inclusive, and you must specify both digits.	ss.ssssss	seconds. The valid range for the first two digits is 00–59. You must specify both digits. You can specify from one to six fractional digits.
This string ...	Represents ...														
YYYY	year. The valid range is from 0001–9999, and you must specify all four digits.														
MM	month. The valid range is from 01–12, and you must specify both digits.														
DD	day. The valid range is from 01–31, constrained by Gregorian calendar definitions. You must specify both digits, and follow them by a single pad character.														
hh	hour of the day. The valid range is 00–23, inclusive, and you must specify both digits.														
mi	minute of the hour. The valid range is 00–59, inclusive, and you must specify both digits.														
ss.ssssss	seconds. The valid range for the first two digits is 00–59. You must specify both digits. You can specify from one to six fractional digits.														

This format ...	Represents ...												
YYYY-MM-DD hh:mi:ss.ssssssignhh:mi	a timestamp with up to six fractional seconds and a time zone offset. The TYPE is TIMESTAMP(n) WITH TIME ZONE, where n is the number of fractional seconds digits in <i>string</i>. <table><tr><th>This string ...</th><th>Represents ...</th></tr><tr><td>YYYY</td><td> year. The valid range is from 0001–9999, and you must specify all four digits. </td></tr><tr><td>MM</td><td> month. The valid range is from 01–12, and you must specify both digits. </td></tr><tr><td>DD</td><td> day. The valid range is from 01–31, constrained by Gregorian calendar definitions. You must specify both digits. DD must always be followed by a single pad character. </td></tr><tr><td>hh</td><td> hour of the day. The valid range is from 00–23, and you must specify both digits. </td></tr><tr><td>mi</td><td> minute of the hour. The valid range is from 00–59, and you must specify both digits. </td></tr></table>	This string ...	Represents ...	YYYY	year. The valid range is from 0001–9999, and you must specify all four digits.	MM	month. The valid range is from 01–12, and you must specify both digits.	DD	day. The valid range is from 01–31, constrained by Gregorian calendar definitions. You must specify both digits. DD must always be followed by a single pad character.	hh	hour of the day. The valid range is from 00–23, and you must specify both digits.	mi	minute of the hour. The valid range is from 00–59, and you must specify both digits.
This string ...	Represents ...												
YYYY	year. The valid range is from 0001–9999, and you must specify all four digits.												
MM	month. The valid range is from 01–12, and you must specify both digits.												
DD	day. The valid range is from 01–31, constrained by Gregorian calendar definitions. You must specify both digits. DD must always be followed by a single pad character.												
hh	hour of the day. The valid range is from 00–23, and you must specify both digits.												
mi	minute of the hour. The valid range is from 00–59, and you must specify both digits.												

This format ...	Represents ...	
YYYY-MM-DD hh:mi:ss.ssssssignhh:mi (continued)	This string ...	Represents ...
	ss.ssssss	seconds. The valid range for the first two digits is 00–59. You must specify both digits. You can specify from one to six fractional digits.
	signhh:mi	hours and minutes in the Time Zone offset. The valid range is -12:59 through +13:00, inclusive. <i>sign</i> is + or -

Example 1: YYYY-MM-DD hh:mi:ss Format

The following example selects all classes from the Classes table that are timestamped November 23 1998 at 3:30:23 PM.

```
SELECT *
FROM CLASSES
WHERE TIME_STAMP = 'TIMESTAMP 1998-11-23 15:30:23';
```

Example 2: YYYY-MM-DD hh:mi:sssignhh:mi Format

```
TIMESTAMP '2002-01-01 10:37:12-08:00'
```

Example 3: YYYY-MM-DD hh:mi:ss.ssssss Format

```
TIMESTAMP '1995-07-31 10:36:02.123456'
```

Example 4: YYYY-MM-DD hh:mi:ss.ssssssignhh:mi Format

```
TIMESTAMP '1492-10-27 21:17:35.456123+07:30'
```

Interval Literals

Introduction

Interval literals provide a means for declaring interval values in expressions, view or macro definition text, or CHECK constraint text.

Interval literals differ from other SQL literals in that keywords introduce and follow them.

General Syntax

— INTERVAL sign ' string ' — interval_qualifier —

1101A034

where:

Syntax element ...	Specifies ...
sign	an optional minus sign to designate a negative interval. The default is a positive interval. Note that the sign must be <i>outside</i> the quotation marks that enclose <i>string</i> .
string	a quoted character string.
interval_qualifier	a keyword or keywords from the list of Interval data types describing the literal. You cannot specify precision with the <i>interval_qualifier</i> .

Interval Literal Categories

There are two mutually exclusive categories of interval literals:

- Year-Month
- Day-Time

You cannot mix the types.

Interval Literal Data Types

The data type for a literal is derived directly from the *interval_qualifier*.

Note: You cannot specify precision with any of the *interval_qualifier* types.

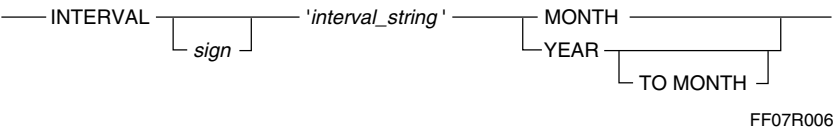
Interval Qualifiers

The following list describes the available keywords to indicate the value for the *interval_qualifier* variable, where the components are all unsigned decimal integers of not more than four digits for a YEAR component and not more than two digits for all other components.

- Year-Month types
 - YEAR
 - YEAR TO MONTH
 - MONTH
- Day-Time types
 - DAY
 - DAY TO HOUR
 - DAY TO MINUTE
 - DAY TO SECOND
 - HOUR
 - HOUR TO MINUTE
 - HOUR TO SECOND
 - MINUTE
 - MINUTE TO SECOND
 - SECOND

Interval Year-Month Literals

Syntax



where:

Syntax element ...	Specifies ...
INTERVAL	that the text to follow defines an Interval literal.
<i>string</i>	an optionally signed string defining either a 4-digit year, a 2-digit month, or both (in which case the year and date are separated by a hyphen character).

ANSI Compliance

Interval Year-Month literals are ANSI SQL-99-compliant.

Note: The Teradata implementation places the optional sign for the interval outside the single quotes, while the ANSI definition places the optional sign *within* the quotes.

Further Information

For descriptions of Year-Month interval literals, see:

- [“INTERVAL YEAR Literals”](#)
- [“INTERVAL YEAR TO MONTH Literals”](#)
- [“INTERVAL MONTH Literals”](#)

INTERVAL YEAR Literals

Purpose

Declares an Interval Year value in an expression, view or macro definition text, or CHECK constraint text.

Syntax

— INTERVAL sign 'string' — YEAR —

1101A023

where:

Syntax element ...	Specifies ...
<i>sign</i>	an optional minus sign to indicate a negative interval. The default is a positive interval. Note that the sign must be <i>outside</i> the quotation marks that enclose <i>string</i> .
<i>string</i>	one to four digits representing years.

ANSI Compliance

INTERVAL YEAR literals are ANSI SQL-99-compliant.

Type

INTERVAL YEAR

Example

This example adds an interval of -2 years to the current system date. (For this example, assume the current system date is 11-03-1999.)

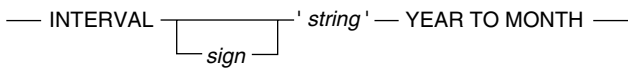
```
SELECT INTERVAL - '2' YEAR + CURRENT_DATE;  
  
      (-2+Date)  
-----  
1997/11/03
```

INTERVAL YEAR TO MONTH Literals

Purpose

Declares an Interval Year to Month value in an expression, view or macro definition text, or CHECK constraint text.

Syntax



1101A024

where:

Syntax element ...	Specifies ...
<i>sign</i>	an optional minus sign to indicate a negative interval. The default is a positive interval. Note that the sign must be <i>outside</i> the quotation marks that enclose <i>string</i> .
<i>string</i>	one to four digits representing years, optionally followed by a hyphen and two digits representing 00 to 11 months.

ANSI Compliance

INTERVAL YEAR TO MONTH literals are ANSI SQL-99-compliant.

Type

INTERVAL YEAR TO MONTH

Example

This example adds an interval of two years and six months to the current system date. (For this example, assume the current system date is 1999-11-03.)

```
SELECT CURRENT_DATE + INTERVAL '2-06' YEAR TO MONTH;  
      ( 2-06+Date)  
-----  
      2002/05/03
```

INTERVAL MONTH Literals

Purpose

Declares an Interval Month value in an expression, view or macro definition text, or CHECK constraint text.

Syntax

— INTERVAL sign 'string' — MONTH —

1101A025

where:

Syntax element ...	Specifies ...
<i>sign</i>	an optional minus sign to indicate a negative interval. The default is a positive interval. Note that the sign must be <i>outside</i> the quotation marks that enclose <i>string</i> .
<i>string</i>	one to four digits representing months.

ANSI Compliance

INTERVAL MONTH literals are ANSI SQL-99-compliant.

Type

INTERVAL MONTH

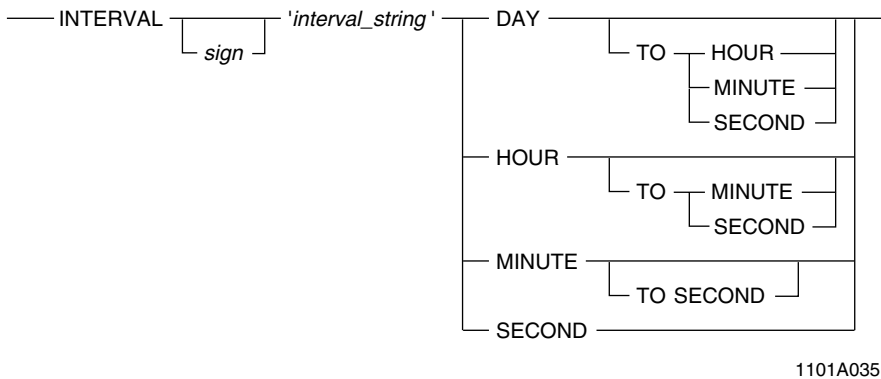
Example

This example adds an interval of six months to the current system date. (For this example, assume the current system date is 1999-11-03.)

```
SELECT (INTERVAL '6' MONTH) + CURRENT_DATE;  
  
      ( 6+Date)  
-----  
2000/05/03
```

Interval Day-Time Literals

Syntax



where:

Syntax element ...	Specifies ...
INTERVAL	that the text to follow defines an interval literal.
<i>string</i>	an optionally signed string composed of a sequence of 2-digit day, 2-digit hour, 2-digit minute, and 2-digit second (with optional 10-digit precision) as required by the specific Day-Time Interval literal.

ANSI Compliance

Interval Day-Time literals are ANSI SQL-99-compliant.

Note: In the ANSI definition, the optional sign for the interval is within the single quotes; in the Teradata implementation, the sign is outside the quotes.

Further Information

For descriptions of Day-Time interval literals, see:

- [“INTERVAL DAY Literals”](#)
- [“INTERVAL DAY TO HOUR Literals”](#)
- [“INTERVAL DAY TO MINUTE Literals”](#)
- [“INTERVAL DAY TO SECOND Literals”](#)
- [“INTERVAL HOUR Literals”](#)
- [“INTERVAL HOUR TO MINUTE Literals”](#)
- [“INTERVAL HOUR TO SECOND Literals”](#)
- [“INTERVAL MINUTE Literals”](#)
- [“INTERVAL MINUTE TO SECOND Literals”](#)
- [“INTERVAL SECOND Literals”](#)

INTERVAL DAY Literals

Purpose

Declares an Interval Day value in an expression, view or macro definition text, or CHECK constraint text.

Syntax

— INTERVAL sign 'string' — DAY —

1101A026

where:

Syntax element ...	Specifies ...
<i>sign</i>	an optional minus sign to indicate a negative interval. The default is a positive interval. Note that the sign must be <i>outside</i> the quotation marks that enclose <i>string</i> .
<i>string</i>	one to four digits representing the number of days.

ANSI Compliance

INTERVAL DAY literals are ANSI SQL-99-compliant.

Type

INTERVAL DAY

Example

This example adds an interval of -30 days to the current system date. (For this example, assume the current system date is 1999-11-03).

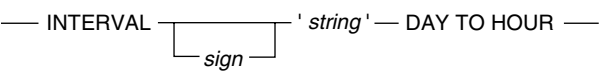
```
SELECT INTERVAL -'30' DAY + CURRENT_DATE;  
  
(-30+Date)  
-----  
1999/10/04
```

INTERVAL DAY TO HOUR Literals

Purpose

Declares an Interval Day to Hour value in an expression, view or macro definition text, or CHECK constraint text.

Syntax



1101A027

where:

Syntax element ...	Specifies ...
<i>sign</i>	an optional minus sign to indicate a negative interval. The default is a positive interval. Note that the sign must be <i>outside</i> the quotation marks that enclose <i>string</i> .
<i>string</i>	one to four digits representing the number of days, optionally followed by a pad character and two digits representing 00 to 23 hours.

ANSI Compliance

INTERVAL DAY TO HOUR literals are ANSI SQL-99-compliant.

Type

INTERVAL DAY TO HOUR

Example

This example adds an interval of 1 day and 12 hours to the current system timestamp.

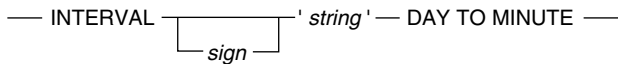
```
SELECT INTERVAL '1 12' DAY TO HOUR + CURRENT_TIMESTAMP;  
  
      ( 1 12+Current TimeStamp)  
-----  
1999-11-05 02:32:19.770000+00:00
```

INTERVAL DAY TO MINUTE Literals

Purpose

Declares an Interval Day to Minute value in an expression, view or macro definition text, or CHECK constraint text.

Syntax



1101A028

where:

Syntax element ...	Specifies ...
sign	an optional minus sign to indicate a negative interval. The default is a positive interval. Note that the sign must be <i>outside</i> the quotation marks that enclose <i>string</i> .
string	one to four digits representing the number of days, optionally followed by a pad character and two digits representing 00 to 23 hours, optionally followed by a colon and two digits representing 00 to 59 minutes.

ANSI Compliance

INTERVAL DAY TO MINUTE literals are ANSI SQL-99-compliant.

Type

INTERVAL DAY TO MINUTE

Example

This example adds an interval of 30 days, 12 hours, and 30 minutes to the current system timestamp.

```
SELECT INTERVAL '30 12:30' DAY TO MINUTE + CURRENT_TIMESTAMP;  
  
      ( 30 12:30+Current TimeStamp)  
-----  
1999-12-04 03:14:26.330000+00:00
```

INTERVAL DAY TO SECOND Literals

Purpose

Declares an Interval Day to Second value in an expression, view or macro definition text, or CHECK constraint text.

Syntax

— INTERVAL sign 'string' — DAY TO SECOND —

1101A029

where:

Syntax element ...	Specifies ...
sign	an optional minus sign to indicate a negative interval. The default is a positive interval. Note that the sign must be <i>outside</i> the quotation marks that enclose <i>string</i> .
string	one to four digits representing the number of days, optionally followed by a pad character and two digits representing 00 to 23 hours, optionally followed by a colon and two digits representing 00 to 59 minutes, optionally followed by a colon and two digits representing 00 to 59 seconds and zero to six digits representing fractional seconds.

ANSI Compliance

INTERVAL DAY TO SECOND literals are ANSI SQL-99-compliant.

Type

INTERVAL DAY TO SECOND

Example

This example adds an interval of 30 days, 12 hours, 30 minutes, and 30.5 seconds to the current system timestamp.

```
SELECT INTERVAL '30 12:30:30.5' DAY TO SECOND +
CURRENT_TIMESTAMP;

( 30 12:30:30.5+Current TimeStamp)
-----
1999-12-04 03:19:27.780000+00:00
```

INTERVAL HOUR Literals

Purpose

Declares an Interval Hour value in an expression, view or macro definition text, or CHECK constraint text.

Syntax

— INTERVAL sign ' string ' — HOUR —

1101A030

where:

Syntax element ...	Specifies ...
<i>sign</i>	an optional minus sign to indicate a negative interval. The default is a positive interval. Note that the sign must be <i>outside</i> the quotation marks that enclose <i>string</i> .
<i>string</i>	one to four digits representing the number of hours.

ANSI Compliance

INTERVAL HOUR literals are ANSI SQL-99-compliant.

Type

INTERVAL HOUR

Example

This example subtracts the current system timestamp from an interval of 2000 hours.

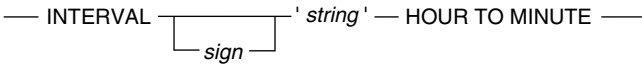
```
SELECT INTERVAL '2000' HOUR - CURRENT_TIMESTAMP;  
  
      ( 2000-Current TimeStamp)  
-----  
5210-83-01 05:61:34.502492+12:91
```

INTERVAL HOUR TO MINUTE Literals

Purpose

Declares an Interval Hour to Minute value in an expression, view or macro definition text, or CHECK constraint text.

Syntax



1101A037

where:

Syntax element ...	Specifies ...
<i>sign</i>	an optional minus sign to indicate a negative interval. The default is a positive interval. Note that the sign must be <i>outside</i> the quotation marks that enclose <i>string</i> .
<i>string</i>	one to four digits representing the number of hours, optionally followed by a colon and two digits representing 00 to 59 minutes.

ANSI Compliance

INTERVAL HOUR TO MINUTE literals are ANSI SQL-99-compliant.

Type

INTERVAL HOUR TO MINUTE

Example

This example adds an interval of 12 hours and 37 minutes to the current system time.

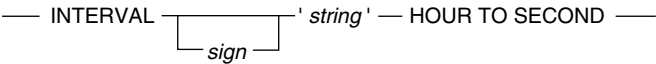
```
SELECT INTERVAL '12:37' HOUR TO MINUTE + CURRENT_TIME;  
  
( 12:37+Current Time)  
-----  
03:34:03+00:00
```

INTERVAL HOUR TO SECOND Literals

Purpose

Declares an Interval Hour to Second value in an expression, view or macro definition text, or CHECK constraint text.

Syntax



1101A038

where:

Syntax element ...	Specifies ...
<i>sign</i>	an optional minus sign to indicate a negative interval. The default is a positive interval. Note that the sign must be <i>outside</i> the quotation marks that enclose <i>string</i> .
<i>string</i>	one to four digits representing the number of hours, optionally followed by a colon and two digits representing 00 to 59 minutes, optionally followed by a colon and two digits representing 00 to 59 seconds and zero to six digits representing fractional seconds.

ANSI Compliance

INTERVAL HOUR TO SECOND literals are ANSI SQL-99-compliant.

Type

INTERVAL HOUR TO SECOND

Example

This example subtracts the current system time from an interval of 12 hours, 37 minutes, and 12.123456 seconds.

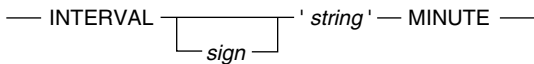
```
SELECT INTERVAL '12:37:12.123456' HOUR TO SECOND - CURRENT_TIME;  
  
( 12:37:12.123456-Current Time)  
-----  
33:71:34+06:91
```

INTERVAL MINUTE Literals

Purpose

Declares an Interval Minute value in an expression, view or macro definition text, or CHECK constraint text.

Syntax



1101A031

where:

Syntax element ...	Specifies ...
<i>sign</i>	an optional minus sign to indicate a negative interval. The default is a positive interval. Note that the sign must be <i>outside</i> the quotation marks that enclose <i>string</i> .
<i>string</i>	one to four digits representing the number of minutes.

ANSI Compliance

INTERVAL MINUTE literals are ANSI SQL-99-compliant.

Type

INTERVAL MINUTE

Example

This example adds an interval of 600 minutes to the current system time.

```
SELECT INTERVAL '600' MINUTE + CURRENT_TIME;  
  
( 600+Current Time)  
-----  
01:14:19+00:00
```

INTERVAL MINUTE TO SECOND Literals

Purpose

Declares an Interval Minute to Second value in an expression, view or macro definition text, or CHECK constraint text.

Syntax

— INTERVAL sign ' string ' — MINUTE TO SECOND —

1101A032

where:

Syntax element ...	Specifies ...
<i>sign</i>	an optional minus sign to indicate a negative interval. The default is a positive interval. Note that the sign must be <i>outside</i> the quotation marks that enclose <i>string</i> .
<i>string</i>	one to four digits representing the number of minutes, optionally followed by a colon and two digits representing 00 to 59 seconds and zero to six digits representing fractional seconds.

ANSI Compliance

INTERVAL MINUTE TO SECOND literals are ANSI SQL-99-compliant.

Type

INTERVAL MINUTE TO SECOND

Example

This example subtracts the current system timestamp from an interval of 9 minutes and 30 seconds.

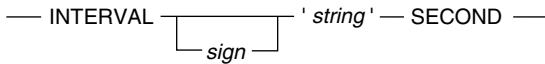
```
SELECT INTERVAL '9:30' MINUTE TO SECOND - CURRENT_TIMESTAMP;  
  
      ( 9:30-Current Timestamp)  
-----  
5210-83-01 05:61:34.502492+12:91
```

INTERVAL SECOND Literals

Purpose

Declares an Interval Second value in an expression, view or macro definition text, or CHECK constraint text.

Syntax



1101A033

where:

Syntax element ...	Specifies ...
<i>sign</i>	an optional minus sign to indicate a negative interval. The default is a positive interval. Note that the sign must be <i>outside</i> the quotation marks that enclose <i>string</i> .
<i>string</i>	an unsigned number that represents 0 to 9999.999999 seconds.

ANSI Compliance

INTERVAL SECOND literals are ANSI SQL-99-compliant.

Type

INTERVAL SECOND

Example

This example adds an interval of 0.000001 seconds to the current system time.

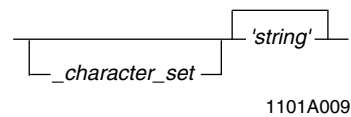
```
INTERVAL '0.000001' SECOND + CURRENT_TIME;  
  
( 0.0+Current Time)  
-----  
15:21:23+00.00
```

Character Data Literals

Purpose

Declares a character string literal value in an expression, view or macro definition text, or CHECK constraint definition text.

Syntax



where:

This syntax element ...	Specifies ...										
<i>_character_set</i>	the name of the character data type to be associated with the string literal. This is an optional “nice to have” feature that is defined by the ANSI SQL2 standard, but you can ignore for virtually all applications. <table><tr><th>THIS character data type identifier ...</th><th>Indicates that each character in the string is with in the repertoire of this character set ...</th></tr><tr><td>_Latin</td><td>LATIN</td></tr><tr><td>_Unicode</td><td>UNICODE</td></tr><tr><td>_KanjiSJIS</td><td>KANJISJIS</td></tr><tr><td>_Graphic</td><td>GRAPHIC</td></tr></table> A character literal without an explicitly declared character data type (of the form <i>'string'</i>) is equivalent to _Unicode <i>'string'</i>. You cannot define a character string literal type of _Kanji1, and if you attempt to do so, an error is returned.	THIS character data type identifier ...	Indicates that each character in the string is with in the repertoire of this character set ...	_Latin	LATIN	_Unicode	UNICODE	_KanjiSJIS	KANJISJIS	_Graphic	GRAPHIC
THIS character data type identifier ...	Indicates that each character in the string is with in the repertoire of this character set ...										
_Latin	LATIN										
_Unicode	UNICODE										
_KanjiSJIS	KANJISJIS										
_Graphic	GRAPHIC										
<i>string</i>	the quoted character string literal to be qualified with a character set declaration.										

ANSI Compliance

Character literals are ANSI SQL-99-compliant.

Definition

Character literals consist of zero or more alphanumeric characters enclosed in single quotes. The total length of a character literal is 0 to 31000 bytes. A zero-length character literal is represented by two consecutive single quotes ('').

A character literal that is split into multiple, quote-delimited character strings is treated as if the strings were concatenated.

Character Literal Data Types

Character literals are treated as VARCHAR(*n*) fields, where *n* is the length of the literal.

Note that trailing blanks can be trimmed from a VARCHAR field.

In a *Japanese character site*, all character data is assumed to be mixed single byte characters/multibyte characters.

Case Specificity

A character string literal is defined as NOT CASESPECIFIC or CASESPECIFIC by default, depending on the mode of the session parsing the string for execution.

In this mode ...	The default case specificity for strings is ...
ANSI	CASESPECIFIC
Teradata	NOT CASESPECIFIC

The CASESPECIFIC attribute of character strings determines the rules for string comparisons.

Client Character Set Assignment

This association does *not* have any effect on the assignment of character set defined by the BTEQ command SET SESSION CHARSET. Each character is encoded in the client form-of-use for character data as defined by SET SESSION CHARSET, so you cannot enter characters outside the repertoires defined by that session character set. Similarly, these literal assignments do *not* affect RDBMS coding used for the literal, which is always UNICODE (with the exception of the KANJI1 character data type).

For example, `_Latin '年'`, which cannot be entered from a single byte character-only session, is an error, because 年 is not in the Latin repertoire. The literal `_KanjiSJIS '年'` would be stored internally as 0x5E74 and not 0x944E.

Character Literal Identifiers and KANJI1 Data

The following character data type literal identifiers, unlike internal data, can be assigned to a KANJI1 field.

- `_Latin 'string'`
- `_Unicode 'string'`
- `_Graphic 'string'`
- `_KanjiSJIS 'string'`

Conversion is based on the current session character set, which is established using the BTEQ command `SET SESSION CHARSET`. The assignability of these literals is explained in the topic “CHARACTER Data Type Assignability Rules” in Chapter 13: “Data Type Conversions” of *Teradata RDBMS SQL Reference, Volume 5*.

Example 1: Simple Character Strings

The following are examples of valid character literals:

```
'Los Angeles'
''
```

Example 2: Character Strings Containing a Single Quoted Literal

To enter an apostrophe or single quote in a character literal, double the embedded apostrophe.

```
'He said ''yes'' to her question'
```

Example 3: Character Literals Composed of Segments

Multiple quote-delimited strings are treated as though they were concatenated into one character literal.

```
('AA' 'BB') is ('AA' || 'BB')
```

Graphic Literals

Purpose

Declares a graphic character string literal in an expression, view, or macro definition text or CHECK constraint definition text.

Syntax

— G — '< ABC >' —
HH01A090

where:

Syntax Element ...	Specifies ...
G	that the string is graphic data.
'	string delimiter for a graphic character string. Each ' (single quote mark) is a single byte character.
<ABC>	a quoted string of valid KanjiEBCDIC multibyte characters surrounded by Shift-Out and Shift-In characters.

ANSI Compliance

Graphic literals are Teradata extensions to the ANSI SQL-99 standard.

Rules

The following rules apply to the use of graphic literals.

- On a *Japanese character site*, the G '*string*' syntax for graphic string literals can only be used when the KanjiEBCDIC character set is in effect for the session. It is equivalent to _Graphic '*string*'.
- The _Graphic '*string*' form can be used from any session character set, and is the recommended form for specifying literals required to be within the graphic repertoire.
- For other character sets, you can use the hexadecimal representation of the graphic string (see [“Hexadecimal Literals” on page 2-44](#)).
- The literal must be enclosed within Shift-Out/Shift-In characters, as normal KanjiEBCDIC encoding requires.

- The maximum length is 15500 logical (double byte) characters.
Note that the Shift-Out/Shift-In characters are *not* included in the byte count and are not stored as part of the graphic literal.
- The contents must be valid graphic data (multibyte characters that are printable under the KanjiEBCDIC character set in effect for the current session).
- The internal representation must be an even number of bytes.

Case Sensitivity of Character String Literals

Character string literals, including graphic string literals, can occur in the following places:

- SQL query text
- View and macro definition text
- CHECK constraint text

In all of these instances, execution of queries using these literals involves parsing the literals and assigning case sensitivity attributes to them at that time.

IF the session is in this mode ...	THEN all character string literals are ...
ANSI	CASESPECIFIC
Teradata	NOT CASESPECIFIC

This means that for views with WHERE constraints and for CHECK, if character string literals are included, they compare differently when executed by users with sessions in ANSI and Teradata mode.

You should modify *existing* views having WHERE constraints to explicitly control case sensitivity for comparisons.

Further Information

See “ALTER TABLE” and “CREATE TABLE” in Chapter 1: “SQL Data Definition Language Statement Syntax” in *Teradata RDBMS SQL Reference, Volume 4* for explanations on the use of CHECK.

Also see “Case Sensitivity in Comparisons” in Chapter 3: “Comparison Operators” in *Teradata RDBMS SQL Reference, Volume 5*.

You should explicitly control case sensitivity in character string comparisons. For information about how to explicitly control case sensitivity, see “CASESPECIFIC Phrase” and “UPPERCASE Phrase” in [Chapter 5: “Character Data Types”](#) of this volume.

Hexadecimal Literals

Purpose

Declares a hexadecimal literal value that can represent characters, bytes, or integers.

Syntax: Client Form

The syntax for the client form of a hexadecimal literal is as follows:

```
_____X_____'hexadecimal digits'_____  
HH01A081
```

where:

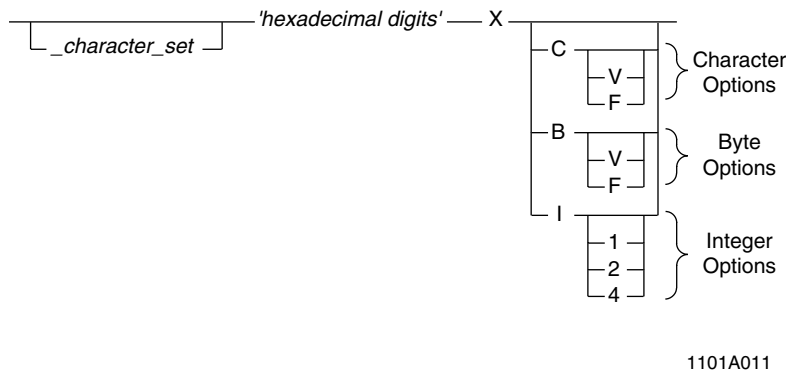
Syntax Element ...	Specifies ...
<i>hexadecimal digits</i>	a string of hexadecimal digits, where a hexadecimal digit is a character from 0 to 9, a to f, or A to F.

The client form syntax specifies a hexadecimal literal that represents characters within the repertoire of the session character set.

For example, a string in this form is assumed to be EBCDIC when submitted from an IBM mainframe. It is assumed to be ASCII when submitted from the Teradata RDBMS console.

Syntax: Internal DBS Form

The syntax for the internal DBS form of a hexadecimal literal is as follows:



where:

for hexadecimal character literals:

This syntax element ...	Specifies ...												
<i>_character_set</i>	that the characters represented by the hexadecimal literal are within the repertoire of this (optional) character set. The default is the user default character set.												
	<table> <tr> <th>This identifier ...</th><th>Specifies this character set ...</th></tr> <tr> <td><i>_Latin</i></td><td>LATIN</td></tr> <tr> <td><i>_Unicode</i></td><td>UNICODE</td></tr> <tr> <td><i>_Graphic</i></td><td>GRAPHIC</td></tr> <tr> <td><i>_KanjiSJIS</i></td><td>KANJISJIS</td></tr> <tr> <td><i>_Kanji1</i></td><td>KANJI1</td></tr> </table>	This identifier ...	Specifies this character set ...	<i>_Latin</i>	LATIN	<i>_Unicode</i>	UNICODE	<i>_Graphic</i>	GRAPHIC	<i>_KanjiSJIS</i>	KANJISJIS	<i>_Kanji1</i>	KANJI1
This identifier ...	Specifies this character set ...												
<i>_Latin</i>	LATIN												
<i>_Unicode</i>	UNICODE												
<i>_Graphic</i>	GRAPHIC												
<i>_KanjiSJIS</i>	KANJISJIS												
<i>_Kanji1</i>	KANJI1												
<i>hexadecimal digits</i>	a string of hexadecimal digits, where a hexadecimal digit is a character from 0 to 9, a to f, or A to F.												
C	that the hexadecimal literal represents characters.												
V	that the hexadecimal literal is in character variable format.												
F	that the hexadecimal literal is in character fixed format. This is the default if F or V is not specified.												

for hexadecimal byte literals:

This syntax element ...	Specifies ...
<i>hexadecimal digits</i>	a string of hexadecimal digits, where a hexadecimal digit is a character from 0 to 9, a to f, or A to F.
B	that the hexadecimal literal represents bytes.
V	the hexadecimal literal is in byte variable format.
F	the hexadecimal literal is in byte fixed format. This is the default if F or V is not specified.

for hexadecimal integer literals:

This syntax element ...	Specifies ...
<i>hexadecimal digits</i>	a string of hexadecimal digits, where a hexadecimal digit is a character from 0 to 9, a to f, or A to F.
I	that the hexadecimal literal represents integers. This is the default if C, B, or I is not specified.

This syntax element ...	Specifies ...
1	that the hexadecimal literal represents integers with a data type of BYTEINT.
2	that the hexadecimal literal represents integers with a data type of SMALLINT.
4	that the hexadecimal literal represents integers with a data type of INTEGER. This is the default if 1, 2, or 4 is not specified.

ANSI Compliance

Hexadecimal literals are Teradata extensions to the ANSI SQL-99 standard.

Definition

Hexadecimal literals consist of 0 to 62000 hexadecimal digits delimited by a matching pair of single quotes, where a hexadecimal digit is a character from 0 to 9, a to f, or A to F.

Examples of Hexadecimal Literal Data Types

The modifiers following the X determine the hexadecimal literal data type.

This hexadecimal literal...	Specifies this data type...
'41'XCV	VARCHAR
'41'XC	CHAR[1]
'41'XCF	
'55'XBV	VARBYTE
'55'XB	BYTE[1]
'55'XBF	
'5'X	INTEGER
'5'XI	
'5'XI4	
'5'XI2	SMALLINT
'5'XI1	BYTEINT

Rules

The following rules apply to hexadecimal literals:

- Hexadecimal character or byte literals are represented by an even number of hexadecimal digits. Hexadecimal literals are extended on the right with zeros when semantically required. For example:

```
_Latin 'C1C'XC = 'C1C0'XC
```

- As with other character literals, hexadecimal character literals are translated to UNICODE internal form, unless the character set is KANJI1, and then the hexadecimal literal is kept in KANJI1 internal form.
- Hexadecimal integer literals are represented by an odd or even number of hexadecimal digits. The hexadecimal literal is right-justified. For example, the value 1000 can be expressed as any of the following:

```
'3e8'X
```

```
'0003e8'X
```

```
'000003e8'X
```

Example 1: Using the XC Form

The XC form is useful for inserting character strings, such as TAB or BACKSPACE, that cannot generally be entered directly on the terminal keyboard.

For this example, assume the user default character set is Latin and the client system represents data in EBCDIC. If the terminal keyboard does not include the brace characters {}, use the following hexadecimal literal to represent "{12}":

```
'7B31327D'XC
```

where 7B represents the brace "{" and 7D represents the brace "}".

Example 2: Using the XB Form

XB is the only form for entering a BYTE string.

For example, suppose the column CodeVal has been defined as BYTE(2).

To insert a BYTE string as a hexadecimal literal, use the following form:

```
... (CodeVal) VALUES ('7879'XB) ;
```

To select those rows from CodeVal, specify the conditional like this:

```
... WHERE CodeVal = '7879'XB ;
```




Chapter 3:

Decimal/Numeric Data Types

This chapter describes the numeric data types, including:

- DECIMAL/NUMERIC
- BYTEINT
- SMALLINT
- INTEGER
- FLOAT/REAL/DOUBLE PRECISION

Each of these types can be combined with a range constraint when the data type is defined or modified.

The chapter also describes several usage considerations for the system default DECIMAL size definition. The relevant topics to the DECIMAL(18) issue covered in this chapter are the following.

- ANSI-Teradata correspondences for naming numeric data types
- Rounding and numeric/decimal data types
- Usage considerations for DECIMAL(18)
- DECIMAL(18) and the MaxDecimal DBSControl Flag
- Behavior of SQL operations independent of MaxDecimal
- Behavior for different values of MaxDecimal
- CAST and the DECIMAL data type

Note Following the ANSI standard, DATE is not classified as a numeric data type. All dates, times, timestamps, and DateTime intervals have their own data types and are defined in [Chapter 4: “DateTime and Interval Data Types.”](#)

ANSI/Teradata Correspondences for Naming Numeric Data Types

The following table lists Teradata-ANSI synonyms for numeric data types.

ANSI Name	Teradata Equivalent
NUMERIC	DECIMAL
REAL	FLOAT
DOUBLE PRECISION	

Rounding

Rounding DECIMAL/NUMERIC Data Types

If a value being inserted does not fit into a field defined as a DECIMAL or NUMERIC data type, the value is rounded. The way DECIMAL/NUMERIC types are rounded is user-selectable. Teradata RDBMS rounds using the first non-stored value.

IF the value is ...	THEN it is ...
< 5	rounded down
> 5	rounded up

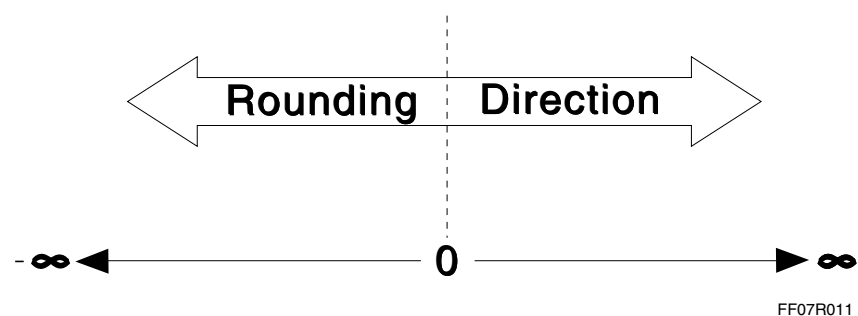
Rounding When a Value Is Exactly 5

Additional considerations come into play when the value is exactly five.

When the DBSControl Record RoundHalfwayMagUp flag is set to FALSE, then rounding is performed as explained in the following table. This is the default.

IF the first non-stored value is exactly 5 and ...	THEN ...						
there are no trailing non-zero digits	the last stored value is checked <table> <tr> <th>IF that value is ...</th><th>THEN the non-stored value is rounded ...</th></tr> <tr> <td>odd</td><td>up</td></tr> <tr> <td>even</td><td>down</td></tr> </table>	IF that value is ...	THEN the non-stored value is rounded ...	odd	up	even	down
IF that value is ...	THEN the non-stored value is rounded ...						
odd	up						
even	down						
there are trailing non-zero digits	rounding behaves as if the first non-stored value is greater than five.						

When the DBSControl Record RoundHalfwayMagUp flag is set to TRUE, then the magnitude of all halfway values, both negative and positive, is rounded up, away from zero (see the illustration).



While this is the most common rounding semantics for business applications, it is also nontrivially biased in the upward direction.

See *Teradata RDBMS Utilities* for additional information about the DBS Control Record and the RoundHalfwayMagUp flag.

Rounding FLOAT, REAL, DOUBLE PRECISION Data Types

If a value being inserted does not fit into a field defined as a FLOAT, REAL, DOUBLE PRECISION data type, the same rounding rule applies as applies to the DECIMAL/NUMERIC data type, but the rounding rule is based on the value of the actual number that is stored internally.

For example, if the number 0.995 is stored as 0.9949999, it is rounded down to 0.99; if it is stored as 0.9950001, it is rounded up to 1.0.

The following table shows the results of inserting values into column NUMBER, which is defined as DECIMAL(3,2):

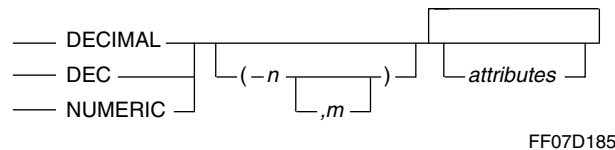
WHEN the value of INSERT is ...	AND the data value in NUMBER is ...	THEN the value is rounded ...	Because the non-stored value is ...
.014	.01	Down	< 5
.015	.02	Up	5 and last stored position is odd
.0151	.02	Up	> 5
.024	.02	Down	< 5
.025	.02	Down	5 and last stored position is even.
.0251	.03	Up	> 5

DECIMAL/NUMERIC Data Types

Purpose

Represents a decimal number of n digits, with m of those n digits to the right of the decimal point.

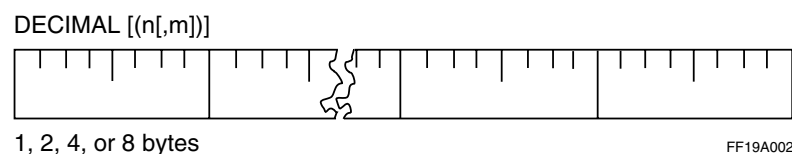
Syntax



where:

Syntax element ...	Specifies ...						
<i>n</i>	the precision. The range is from 1 through 18.						
<i>m</i>	the scale. The range is from 0 through <i>n</i> .						
	<table border="1"> <thead> <tr> <th>WHEN no value is specified for ...</th><th>THEN the default is ...</th></tr> </thead> <tbody> <tr> <td><i>n</i> and <i>m</i></td><td>DECIMAL(5,0)</td></tr> <tr> <td><i>m</i></td><td>DECIMAL(<i>n</i>,0)</td></tr> </tbody> </table>	WHEN no value is specified for ...	THEN the default is ...	<i>n</i> and <i>m</i>	DECIMAL(5,0)	<i>m</i>	DECIMAL(<i>n</i> ,0)
WHEN no value is specified for ...	THEN the default is ...						
<i>n</i> and <i>m</i>	DECIMAL(5,0)						
<i>m</i>	DECIMAL(<i>n</i> ,0)						
<i>attributes</i>	appropriate data type, column storage, or column constraint attributes. See “Core Data Type Attributes” on page 1-8 and “Additional Data Type Attributes” on page 1-9 for specific information.						

Scale



Decimal numbers are scaled by the power of ten equal to the number of fractional digits. The number is stored as a two's complement binary number in one, two, four, or eight bytes. The number of bytes used for a decimal value depends on the total number of digits in that value.

The following list shows the number of bytes used to store decimal values.

Number of Digits	Number of Bytes
1 to 2	1
3 to 4	2
5 to 9	4
10 to 18	8

External Representation of DECIMAL/NUMERIC Numbers

The following table lists the client representations for the DECIMAL/NUMERIC data type.

Determining the application definitions and client data types is the responsibility of the application programmer.

Client CPU Architecture	Client Internal Data Format						
IBM mainframe	Eight bytes (maximum) n-digit (where n represents the precision of the number), signed, packed decimal EBCDIC numbers. The rightmost nibble represents the sign. <table> <tr> <th>This sign ...</th><th>Has this hexadecimal representation ...</th></tr> <tr> <td>+</td><td>X'A' X'C' X'E' X'F'</td></tr> <tr> <td>-</td><td>X'B' X'D'</td></tr> </table> The remaining nibbles represent the digits X'0' - X'9', left-padded with 0 digits when n is even, giving a total of $(n+2)/2$ bytes, or 8 bytes maximum.	This sign ...	Has this hexadecimal representation ...	+	X'A' X'C' X'E' X'F'	-	X'B' X'D'
This sign ...	Has this hexadecimal representation ...						
+	X'A' X'C' X'E' X'F'						
-	X'B' X'D'						

Client CPU Architecture	Client Internal Data Format										
UTS	Eight bytes (maximum) n-digit (where n represents the precision of the number), signed, packed decimal ASCII numbers. The rightmost nibble represents the sign. <table> <tr> <th>This sign ...</th><th>Has this hexadecimal representation ...</th></tr> <tr> <td>+</td><td>X'A' X'C' X'E' X'F'</td></tr> <tr> <td>-</td><td>X'B' X'D'</td></tr> </table> The remaining nibbles represent the digits X'0' - X'9', left-padded with 0 digits when n is even, giving a total of $(n+s)/2$ bytes, or 8 bytes maximum.	This sign ...	Has this hexadecimal representation ...	+	X'A' X'C' X'E' X'F'	-	X'B' X'D'				
This sign ...	Has this hexadecimal representation ...										
+	X'A' X'C' X'E' X'F'										
-	X'B' X'D'										
- RISC - Motorola 68000 - WE 32000	One, 2, 4, or 8 byte signed 2's complement binary ASCII number, most significant byte first. <table> <tr> <th>For $n \leq$...</th><th>The number has this many bits ...</th></tr> <tr> <td>2</td><td>8</td></tr> <tr> <td>4</td><td>16</td></tr> <tr> <td>9</td><td>32</td></tr> <tr> <td>19</td><td>64</td></tr> </table>	For $n \leq$...	The number has this many bits ...	2	8	4	16	9	32	19	64
For $n \leq$...	The number has this many bits ...										
2	8										
4	16										
9	32										
19	64										
Intel	One, 2, 4, or 8 byte signed 2's complement binary ASCII number, least significant byte first. <table> <tr> <th>For $n \leq$...</th><th>The number has this many bits ...</th></tr> <tr> <td>2</td><td>8</td></tr> <tr> <td>4</td><td>16</td></tr> <tr> <td>9</td><td>32</td></tr> <tr> <td>19</td><td>64</td></tr> </table>	For $n \leq$...	The number has this many bits ...	2	8	4	16	9	32	19	64
For $n \leq$...	The number has this many bits ...										
2	8										
4	16										
9	32										
19	64										

Display Format Rule

The default display format rule is:

Data Type	Default Format
DECIMAL(<i>n</i> , <i>m</i>) NUMERIC(<i>n</i> , <i>m</i>)	--(I).9(F)

where:

This variable ...	Represents the ...
<i>n</i>	total number of digits.
<i>m</i>	decimal portion of the number.
I	<i>n - m</i>
F	<i>m</i>

Number of Bytes Stored	Default Display Formats
$n + 2$	- (n) 9 (m) - n (-) . (m)

Total length is $n+2$, where +2 represents one additional position for the sign, and one additional position for the decimal point.

Example 1

Consider the number 256.78. Its type is DECIMAL (5,2) and its default format is expressed either as - (4) . 9 (2) or the equivalent - - - - . 99

The default DECIMAL or NUMERIC display formats are described in [“Data Display Default Formats” on page 8-2](#).

Example 2

In the following partial table definition, column Salary is assigned the type DECIMAL.

```
...
Salary DECIMAL(8,2) FORMAT 'ZZZ,ZZ9.99'
CHECK (Salary BETWEEN 1.00 AND 999000.00),
...
```

Usage Considerations for DECIMAL(18)

Introduction

Release 2.0 introduced an optional default maximum size for DECIMAL numbers of 18. The previous default was always 15.

This option causes the listed behavior for the following specific cases:

WHEN the result is this ...	the system used to return this ...	but now returns this ...
SUM(DECIMAL(n,m))	DECIMAL(15,m)	DECIMAL(18,m)
an expression with at least one decimal argument	DECIMAL(15,m) as the maximum TotalDigits	DECIMAL(18,m) as the maximum TotalDigits

Advisories

A number of changes and advisories have been developed for upgrading from an R1 release to R2.0 (and higher) or for a migration from R1 to R2.0 (and higher).

These changes and advisories are as follows:

- A new MaxDecimal DBSControl Flag
- Advisories for operations that are independent of the MaxDecimal flag
- Behavioral advisories for MaxDecimal=15 and MaxDecimal=18
- Recommendations on the use of CAST
- Compatibility advisories for MaxDecimal=15 and MaxDecimal=18

These issues are discussed more fully on the following pages.

DECIMAL(18) and the MaxDecimal DBSControl Flag

Introduction

The MaxDecimal flag in DBSControl determines whether the maximum decimal default for expression evaluation is 15 or 18.

More information on the MaxDecimal field of the DBSControl record can be found in *Teradata RDBMS Utilities*.

Valid MaxDecimal Values

There are three valid values for the MaxDecimal flag, as detailed in the following table.

This value ...	Results in a default DECIMAL maximum value of ...
0	15
15	
18	18

Usage Notes

MaxDecimal is set to zero after a sysinit operation when the sysinit reinitializes DBSControl and after an upgrade from an R1 release.

The value for MaxDecimal is ...	After ...
0	a sysinit when the sysinit reinitializes DBSControl and also after an upgrade from a R1 release.
18	an upgrade or update from a R2 release containing this feature.

Subsequent upgrades or updates do not change the value of MaxDecimal.

To change the value of MaxDecimal, you must perform a TPA reset for the change to become effective.

When MaxDecimal has been set to 18, a sysinit may reset the value for MaxDecimal. If this happens, set the flag back to 18 and perform a TPA reset.

A value of 0 or 15 provides compatibility with releases prior to R2.0.0 and Version 1 (all releases), while allowing the introduction of cases of the higher limit.

A value of 18 allows full usage of the higher limit and compatibility with R2 releases prior to R2.0.0.

Behavior of SQL Operations Independent of MaxDecimal

Introduction

There are a number of SQL operations for which behavior is independent of the value specified for MaxDecimal in DBSControl. This section describes those operations and their behavior.

Column Definitions and Type Declarations for Casting

The maximum value for n in DECIMAL(n,m) remains 18.

Simple Selection, Insertion, and Updating

Selection, insertion, or updating of decimal values into columns defined with $n > 15$ continue to work as originally defined for R2.0.

This means that a user who defines a column as DECIMAL(18,2) must also be able to support this type of returned result.

Take special care for applications that generate scripts that access tables with decimal columns and that assume no tables exist with $n > 15$. If such an application is used against a new table with a value for $n > 15$, the application may still fail if it does not also generate the proper target host data type to receive the result.

When any decimal argument of an expression has a TotalDigits value greater than 15 (either because of its original type definition or by being CAST into a new type), then the type of the expression result has a TotalDigits value greater than 15 and the default maximum decimal digits value of 18 applies.

COUNT

Internally, the COUNT function uses DECIMAL(15,0) if MaxDecimal is 15 or 0, or to DECIMAL(18,0) if MaxDecimal is 18.

When COUNT is used in a query, the results are as follows:

In Teradata mode, results are always provided as an INTEGER, which may be CAST as DECIMAL($n,0$) for cases where the value of the result exceeds the range of INTEGER. In such a case, a definition for n of 15 should work well.

In ANSI mode, COUNT provides a result as DECIMAL(15,0) by default. In this case, the FORMAT of the value is 'Z(14)9' so that if the value is returned in Field mode, it is a set of digits with no decimal point in a 15 character field.

Behavior for Different Values of MaxDecimal

Introduction

This section discusses the behavior of operations where one argument in a computation is a decimal value, depending on how MaxDecimal is defined.

MaxDecimal=15

The current behavior for decimal arithmetic when MaxDecimal=15 is as described in the following table. For all these cases, the result for releases prior to R2.0.0 was always Decimal(15,n). Values not representable in that Type gave an overflow or, if the user was in Field Mode, a field of asterisks was retrieved.

Expression	Special Conditions
Integer + Decimal	None.
Decimal + Integer	
Integer - Decimal	
Decimal - Integer	
Decimal + Decimal	Only if TotalDigits of one argument is 15 or more.
Decimal - Decimal	
Integer * Decimal	None.
Decimal * Integer	
Decimal * Decimal	Only if Sum of TotalDigits of the arguments is 16 or more.
Integer/Decimal	None.
Decimal/Decimal	
SUM(Decimal)	
MOD(Decimal)	

Release 2.0.0 changed the maximum TotalDigits from 15 to 18. The table above provides results when TotalDigits=18. With the addition of the MaxDecimal control feature, you can change this behavior for installations that do not wish to support 18-digit decimal as the default maximum.

The behavior when MaxDecimal is defined as 15 is as follows:

IF one of the operands is a DECIMAL(<i>n</i> , <i>m</i>) where <i>n</i> is . . .	THEN the upper limit on the <i>n</i> for the result is . . .
> 15	18
anything else	15

The system algorithms for computing the values for *n* and *m* in DECIMAL(*n*,*m*) results assume that when *n* increases over the *n* value(s) of the argument(s), that values of arguments will reach the maxima allowed, the only restriction being the defined maximum values the system supports.

The restriction made when MaxDecimal is set to 15 is that unless you use CAST or unless columns, parameters, or constants are used with *n* > 15, then the largest result or computed DECIMAL values will be of TYPE DECIMAL(15,*m*).

Consider the following example:

```
v1{DECIMAL(8,2)} * v2{DECIMAL(8,2)}
```

With *n* restricted to 15 (when MaxDecimal=15), the result is DECIMAL(15,4). For some argument values, overflow can result.

Note that the product of DECIMAL(*n*,2) with DECIMAL(*n*,2) results in (2*n*,4).

Without the restriction of *n* to 15 (in other words, with MaxDecimal=18), the expression provides a result of (v1*v2){DECIMAL(16,4)} and no overflow results.

Defining MaxDecimal as 15 provides compatibility with previous releases, where the host interface expects to see (v1*v2){DECIMAL(15,4)}. This compatibility comes with a cost of requiring many CAST expressions when using the extended DECIMAL range.

MaxDecimal=18

Behavior when MaxDecimal is defined as 18 is as described for the R2.0.0 release.

CAST and the DECIMAL Data Type

Introduction

Use CAST to decrease n to 15 or less for installations that operate with MaxDecimal=15 or 18 to provide compatible results for applications that require n to be less than or equal to 15, for example

$(A*B)/C$

where A, B, and C are all defined as DECIMAL(8,2).

The resulting value may be less than 15 digits, but A*B could be up to 18.

CAST ((A*B)/C AS DECIMAL(8,2)), for instance, to ensure a result of less than 16 digits.

Unexpected Behavior For Large Expected Values

Because of the way the Teradata SQL compiler works, it is essential that you CAST the arguments of your expressions whenever large values are expected.

For example, suppose f1 is defined as DECIMAL(14,2) and you are going to multiply by an integer or get SUM(f1). The result is expected to be above the default value of 15.

In this case, the following operations:

`CAST (f1 AS DECIMAL (18,2)) *100`

or

`SUM (CAST (f1 AS DECIMAL (18,2)) )`

are proper techniques for ensuring correct answer sets.

On the other hand, if you were to cast the *results* of the expressions, such as the following:

`CAST (f1*100 AS DECIMAL (18,2))`

or

`CAST (SUM (f1) AS DECIMAL (18,2))`

then you will likely experience overflow during the computations (and before the CAST is made)—not the desired result.

DECIMAL Data Type Compatibility Issues

Introduction

This section discusses the compatibility issues that arise with MaxDecimal=15 and MaxDecimal=18.

MaxDecimal=15

This mode is compatible with all releases prior to R2.0.0.

With respect to the results of operations with the DECIMAL data type as discussed in previous sections, MaxDecimal=15 (or its equivalent, MaxDecimal=0) is not compatible with prior R2 releases when the result would have had n greater than 15.

Compatibility can be accomplished by explicitly casting results to DECIMAL(n,m) with n being 16, 17, or 18 as determined in previous R2 releases for specific cases where compatibility is required.

MaxDecimal=18

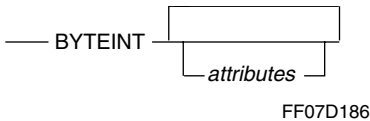
With respect to the DECIMAL data type as discussed in prior sections, this mode is compatible with previous R2 releases, but incompatible with releases prior to R2.0.0.

BYTEINT Data Type

Purpose

Represents a signed binary integer value in the range -128 to 127.

Syntax



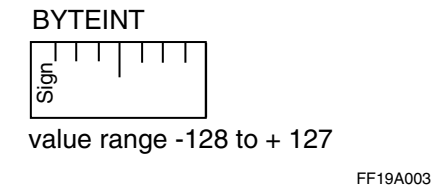
where:

Syntax element ...	Specifies ...
<i>attributes</i>	appropriate data type, column storage, or column constraint attributes. See “Core Data Type Attributes” on page 1-8 and “Additional Data Type Attributes” on page 1-9 for specific information.

ANSI Compliance

BYTEINT is a Teradata extension to the ANSI SQL-99 standard.

Scale



External Representation of BYTEINT

The following table lists the client representations for the Teradata BYTEINT data type.

Determining the application definitions and client data types is the responsibility of the application programmer.

Client CPU Architecture	Client Internal Data Format
IBM mainframe	One byte 8-bit signed 2's complement EBCDIC binary number.
• UTS • RISC • Motorola 68000 • WE 32000 • Intel	One byte 8-bit signed 2's complement ASCII binary number.

Usage Notes

BYTEINT format is described in [“Data Display Default Formats” on page 8-2](#).

Number of Bytes Stored	Default Display Formats
1	- (3) 9 - - - 9

Example

In the following partial table definition, column EdLev is assigned the BYTEINT data type:

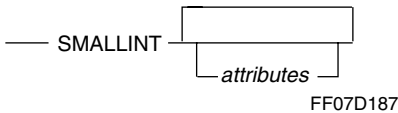
```
...  
EdLev BYTEINT FORMAT 'Z9' CHECK (EdLev BETWEEN 0 AND 22) NOT NULL,  
...
```

SMALLINT Data Type

Purpose

Represents a signed binary integer value in the range -32768 to 32767.

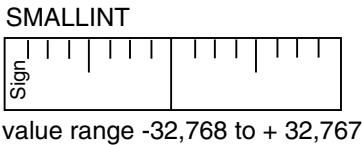
Syntax



where:

Syntax element ...	Specifies ...
<i>attributes</i>	appropriate data type, column storage, or column constraint attributes. See “Core Data Type Attributes” on page 1-8 and “Additional Data Type Attributes” on page 1-9 for specific information.

Scale



FF19A004

External Representation of SMALLINT

The following table lists the client representations for the Teradata SMALLINT data type.

Determining the application definitions and client data types is the responsibility of the application programmer.

Client CPU Architecture	Client Internal Data Format
IBM mainframe	Two byte 16-bit signed 2’s complement EBCDIC binary number, most significant byte first.
- UTS - RISC - Motorola 68000 - WE 32000	Two byte 16-bit signed 2’s complement ASCII binary number, most significant byte first.
Intel	Two byte 16-bit signed 2’s complement ASCII binary number, least significant byte first.

Usage Notes

SMALLINT values are stored with the least significant byte first.

Number of Bytes Stored	Default Display Formats
2	- (5) 9
	-----9

Example

In the following partial table definition, column DeptNo is assigned the SMALLINT data type:

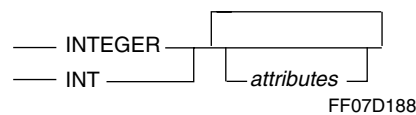
```
...
DeptNo SMALLINT FORMAT '999' BETWEEN 100 AND 900,
...
```

INTEGER Data Type

Purpose

Represents a signed, binary integer value from -2,147,483,648 to 2,147,483,647.

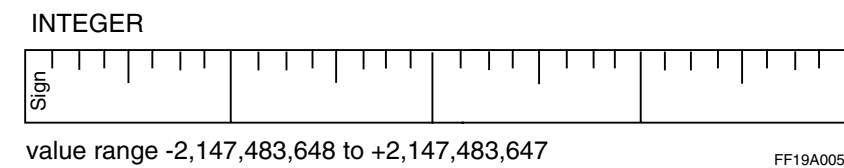
Syntax



where:

Syntax element ...	Specifies ...
attributes	appropriate data type, column storage, or column constraint attributes. See “Core Data Type Attributes” on page 1-8 and “Additional Data Type Attributes” on page 1-9 for specific information.

Scale



External Representation of INTEGER

The following table lists the client representations for the Teradata INTEGER data type.

Determining the application definitions and client data types is the responsibility of the application programmer.

Client CPU Architecture	Client Internal Data Format
IBM mainframe	Four byte 32-bit signed 2’s complement EBCDIC binary number, most significant byte first.
- UTS - RISC - Motorola 68000 - WE 32000	Four byte 32-bit signed 2’s complement ASCII binary number, most significant byte first.
Intel	Two byte 16-bit signed 2’s complement ASCII binary number, least significant byte first.

Usage Notes

INTEGER values are stored with the least significant byte first.

Number of Bytes Stored	Default Display Formats
4	- (10) 9
	-----9

Example

In the following partial table definition, column TelNo is assigned the INTEGER data type:

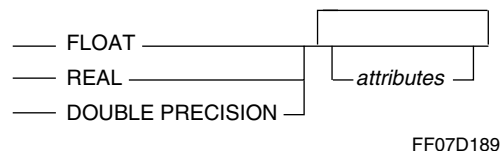
```
...
TelNo INTEGER,
...
```

FLOAT/REAL/DOUBLE PRECISION Data Types

Purpose

Represent a value in sign/magnitude form.

Syntax



where:

Syntax element ...	Specifies ...
<i>attributes</i>	appropriate data type, column storage, or column constraint attributes. See “Core Data Type Attributes” on page 1-8 and “Additional Data Type Attributes” on page 1-9 for specific information.

Scale

FLOAT

E D S	Exponent 11 bits	Mantissa 52 bits
-------------	---------------------	---------------------

8 bytes; value range $2 * 10^{-307}$ to $2 * 10^{+308}$

FF19A006

External Representation of FLOAT/REAL/DOUBLE PRECISION

The following table lists the client representations for the Teradata FLOAT/REAL/DOUBLE PRECISION data type.

Determining the application definitions and client data types is the responsibility of the application programmer.

Client CPU Architecture	Client Internal Data Format										
IBM mainframe	Eight byte 64-bit (double precision) EBCDIC floating point number, most significant byte first, with the following attributes: <table> <tr> <th>This many bits ...</th><th>Represent this attribute of the number ...</th></tr> <tr> <td>1</td><td>Sign of the fraction.</td></tr> <tr> <td>7</td><td>Unsigned power of 16 exponent stored as actual plus X'40'.</td></tr> <tr> <td>56</td><td>Unsigned fraction.</td></tr> </table>	This many bits ...	Represent this attribute of the number ...	1	Sign of the fraction.	7	Unsigned power of 16 exponent stored as actual plus X'40'.	56	Unsigned fraction.		
This many bits ...	Represent this attribute of the number ...										
1	Sign of the fraction.										
7	Unsigned power of 16 exponent stored as actual plus X'40'.										
56	Unsigned fraction.										
UTS	Eight byte 64-bit (double precision) ASCII floating point number, most significant byte first, with the following attributes: <table> <tr> <th>This many bits ...</th><th>Represent this attribute of the number ...</th></tr> <tr> <td>1</td><td>Sign of the fraction.</td></tr> <tr> <td>7</td><td>Unsigned power of 16 exponent stored as actual plus X'40'.</td></tr> <tr> <td>56</td><td>Unsigned fraction.</td></tr> </table>	This many bits ...	Represent this attribute of the number ...	1	Sign of the fraction.	7	Unsigned power of 16 exponent stored as actual plus X'40'.	56	Unsigned fraction.		
This many bits ...	Represent this attribute of the number ...										
1	Sign of the fraction.										
7	Unsigned power of 16 exponent stored as actual plus X'40'.										
56	Unsigned fraction.										
• RISC • Motorola 68000 • WE 32000	Eight byte 64-bit (double precision) ASCII floating point number, most significant byte first, with the following attributes: <table> <tr> <th>This many bits ...</th><th>Represent this attribute of the number ...</th></tr> <tr> <td>1</td><td>Implied extra preceding bit.</td></tr> <tr> <td>1</td><td>Sign of the fraction.</td></tr> <tr> <td>11</td><td>Unsigned power of 2 exponent stored as actual plus X'3FFH'.</td></tr> <tr> <td>52</td><td>Unsigned fraction.</td></tr> </table>	This many bits ...	Represent this attribute of the number ...	1	Implied extra preceding bit.	1	Sign of the fraction.	11	Unsigned power of 2 exponent stored as actual plus X'3FFH'.	52	Unsigned fraction.
This many bits ...	Represent this attribute of the number ...										
1	Implied extra preceding bit.										
1	Sign of the fraction.										
11	Unsigned power of 2 exponent stored as actual plus X'3FFH'.										
52	Unsigned fraction.										

Client CPU Architecture	Client Internal Data Format										
Intel	Eight byte 64-bit (double precision) floating point number, least significant byte first with the following attributes: <table> <tr> <th>This many bits ...</th><th>Represent this attribute of the number ...</th></tr> <tr> <td>1</td><td>Implied extra preceding bit.</td></tr> <tr> <td>1</td><td>Sign of the fraction.</td></tr> <tr> <td>11</td><td>Unsigned power of 16 exponent stored as actual plus X'3FFH'.</td></tr> <tr> <td>52</td><td>Unsigned fraction.</td></tr> </table>	This many bits ...	Represent this attribute of the number ...	1	Implied extra preceding bit.	1	Sign of the fraction.	11	Unsigned power of 16 exponent stored as actual plus X'3FFH'.	52	Unsigned fraction.
This many bits ...	Represent this attribute of the number ...										
1	Implied extra preceding bit.										
1	Sign of the fraction.										
11	Unsigned power of 16 exponent stored as actual plus X'3FFH'.										
52	Unsigned fraction.										

Usage Notes

This data type represents values that range from 2.226×10^{-308} to 1.797×10^{308} .

The mantissa sign is in the most significant bit position; the exponent sign is a part of the exponent field (excess-1024 notation, in which $(1024 - \text{exponent}) = \text{sign}$).

An example of a FLOAT format is 9.99E99, where 1095 is displayed as 1.095E03.

Number of Bytes Stored	Default Display Format
8	-9.999999999999999E-999

Floating Point Value Storage and Manipulation

Floating point values are stored and manipulated internally in IEEE floating point format. Floating point values are stored with the least significant byte first, with one bit for the mantissa sign, 11 bits for the exponent, and 52 bits for the mantissa. Eight bytes are used to hold a floating point value. Negative numbers differ from positive numbers of the same magnitude only in the sign bit.

FLOAT is synonymous with REAL and DOUBLE PRECISION.

The Teradata RDBMS supports normalized, but *not* non-normalized, client floating point values.

Answers for identical computations in floating point arithmetic may produce slightly different results on different machines because internal precision differs from computer to computer, and from model to model in the same series of computer.

The range of IEEE floating point values may be wider than that supported by the client system. Therefore, values can be created and stored in Teradata RDBMS that return error messages when converted for delivery to the client system.

Example

In the following partial table definition, column SalaryFactor is assigned a FLOAT data type:

```
...  
SalaryFactor FLOAT BETWEEN .1 AND 1E1  
...
```



Chapter 4:

DateTime and Interval Data Types

Introduction

Teradata supports DATE, TIME, TIMESTAMP, TIME WITH TIME ZONE, TIMESTAMP WITH TIME ZONE, and Interval data types for defining periods of time.

Note: DATE, TIME, TIME WITH TIME ZONE, TIMESTAMP, and TIMESTAMP WITH TIME ZONE data types are collectively referred to as DateTime data types.

Topics described in this chapter include:

- DateTime fields
- Time zones
- Individual DateTime data types
- Year-Month Interval data types
- Day-Time Interval data types
- CALENDAR system view

DateTime Fields

Introduction

The basic components of all DateTime and Interval data types are components called DateTime fields. All DateTime fields are represented internally as logical records of integer numbers, though the values are defined as type DateTime, not NUMERIC.

The components, ranked from most to least significant, are:

- YEAR
- MONTH
- DAY
- HOUR
- MINUTE
- SECOND

DateTime and Interval Data and Character Data Types

Because Unicode is supported, references to all character data lengths and format lengths should be interpreted as logical lengths and not as the number of bytes.

DateTime data exported to host processes are handled as character data. The physical size of values depends on the host character set.

Year 2000 and Teradata Format Compliance

While it is true that Teradata RDBMS has always had the capability to represent a DATE format as 'YYYY/MM/DD', the default has been to represent dates as 'YY/MM/DD'. This remains the default for dates (defined under INTEGERDATE), but the recommended practice is to define dates under the preferred ANSI DateTime form (defined under ANSIDATE).

New applications written under the ANSIDATE form default to ANSI-compliant 'YYYY-MM-DD' format, and ANSI date compliance for such applications is achieved automatically.

While 'YYYY-MM-DD' is the default input format (with the ANSI-defined separator being the '-' character), any character except a digit is a valid separator for inputting DATE values.

Older applications or new applications that are written under the INTEGERDATE form, but that explicitly declare date formats with four digit year fields, achieve ANSI date compliance at the application level.

See also the discussion of the CenturyBreak field in the DBSControl record in *Teradata RDBMS Utilities, Volume 1*.

Time Zones

Introduction

All TIME and TIMESTAMP data is associated with Time Zones either explicitly or implicitly. A Time Zone is represented by a signed displacement from Universal Coordinated Time (UTC). All TIME and TIMESTAMP values are stored internally as UTC.

TIME and TIMESTAMP values submitted for storage can be literals that include Time Zone information or they can be explicitly specified with syntax that provides Time Zone displacement information with the submitted value.

WHEN Time Zone values are defined ...	THEN ...
implicitly	the default time zone displacement of the SQL session is assigned to them.
explicitly	the submitted Time Zone displacement is stored with the values.

For information on defining time zones, see “SET TIME ZONE” in Chapter 1: “SQL Data Definition Language Statement Syntax” in *Teradata RDBMS SQL Reference, Volume 4*.

All Values Are Stored and Manipulated Using Their UTC Form

Teradata converts all TIME and TIMESTAMP values to UTC prior to storing them. All operations, including hashing, collation, and comparisons that act on TIME and TIMESTAMP values are performed using their UTC forms.

For an example of hashing, consider the following TIMESTAMP literals:

```
TIMESTAMP '1999-07-01 15:00:00-08:00'
TIMESTAMP '1999-07-01 18:00:00-05:00'
```

Both values refer to the same time, which is expressed in UTC as follows:

```
TIMESTAMP '1999-07-01 23:00:00'
```

Because they are equal to one another, both literals hash identically.

For an example of collation, consider the following TIME WITH TIME ZONE literals:

```
TIME '08:00:00-08:00'
TIME '12:00:00-08:00'
TIME '15:00:00-08:00'
TIME '20:00:00-08:00'
```

The correct collation of these values is as follows:

```
TIME '20:00:00-08:00'  
TIME '08:00:00-08:00'  
TIME '12:00:00-08:00'  
TIME '15:00:00-08:00'
```

This nonintuitive outcome becomes more apparent when the values are converted to UTC:

Local Time Value	UTC Time Value
TIME '08:00:00-08:00'	TIME '16:00:00'
TIME '12:00:00-08:00'	TIME '20:00:00'
TIME '15:00:00-08:00'	TIME '23:00:00'
TIME '20:00:00-08:00'	TIME '04:00:00'

As you can see, the TIME WITH TIME ZONE literal '20:00:00-08:00' is actually the lowest value in the sequence.

The counterargument to this explanation is that the time value '20:00:00-08:00' is really '04:00:00' the next day. This is a correct assessment; however, the TIME data type has no concept of day adjustment. Because of this property, you should consider using the TIMESTAMP data type for situations that might have outcomes like the one presented by this example.

Implicit Time Zone Assignment

When no explicit time zone is specified for TIME or TIMESTAMP data, the time zone for the current session is assigned to the data by default.

The current Time Zone for a session is defined relative to UTC.

For example, Eastern Standard Time (EST) is five hours earlier than UTC, so EST is indicated by the signed value -05:00. By the same token, Eastern Daylight Time (EDT) is only four hours earlier than UTC, so EDT is indicated by the signed value -04:00. European time lags UTC by one hour, so it is represented by +01:00.

WHEN a TIME or TIMESTAMP column is defined this way ...	THEN information about its time zone is ...
WITH TIME ZONE	stored explicitly using the fields TIMEZONE_HOUR and TIMEZONE_MINUTE to indicate the offset applicable to the data.
without WITH TIME ZONE	not stored with the data, so it is not possible to know what time zone was in effect at the time the data was stored.

Behavior of Time Values With UTC Offsets

Storing time values using UTC offsets results in the following standard behavior:

Suppose an installation is in the PST time zone and it is New Years Eve, 1998-12-31 20:30 local time.

The system `TIMESTAMP WITH TIME ZONE` for the indicated time is '1999-01-01 04:30-08:00' internally.

When you perform the `CURRENT_TIMESTAMP` function, it is in the form that includes `TIME ZONE`, and any external display converts the values into the appropriate values for the indicated time zone.

Should you want to return this value without the time zone, you could use `CAST` to convert the value to `TIMESTAMP` (without time zone). In PST, the result would be '1999-12-31 20:30', while the identical query performed with the time zone offset for EST returns the result '1999-12-31 23:30'.

It is very important to realize that making a time zone adjustment can change the values for Year, Month, and Day fields when the result is displayed.

When timestamps are compared or used in an `ORDER BY` clause, a time zone adjustment does *not* change the comparison or ordering.

`TIME` adjustments to the time zone can also change what is displayed, just as `TIMESTAMP` adjustments do.

Note that the effect of adding or subtracting a time zone can change the comparison and ordering behavior because there are no higher order fields above the `HOUR` field to mark the crossover into what would be the previous (or next) day.

Reload Tables With Indexes On a ‘... WITH TIME ZONE’ Field

Because TIME WITH TIME ZONE and TIMESTAMP WITH TIME ZONE values hash differently in release 4.0 than they did previously, you must reload any tables that are defined with any indexes on values having a data type of either TIME WITH TIME ZONE or TIMESTAMP WITH TIME ZONE.

The following SELECT statement reports any such tables.

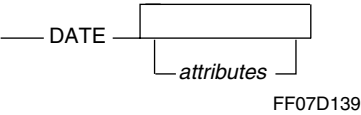
```
SELECT DBase.DatabaseName (CHAR(12)), TVM.TVMName (CHAR(12)),  
TVFields.FieldName (CHAR(12)), TVFields.FieldType  
FROM DBC.DBase, DBC.TVM, DBC.TVFields, DBC.Indexes  
WHERE TVFields.FieldType IN ('TZ', 'SZ')  
AND TVFields.TableID = Indexes.TableId  
AND TVFields.FieldID = Indexes.FieldID  
AND TVFields.TableID = TVM.TVMId  
AND TVM.DatabaseId = DBase.DatabaseID  
GROUP BY 1, 2, 3, 4  
ORDER BY 1, 2, 3, 4;
```

DATE

Purpose

Identifies a field as a DATE value and simplifies handling and formatting of date variables.

Syntax



where:

Syntax element ...	Specifies ...
<i>attributes</i>	appropriate data type, column storage, or column constraint attributes. See “Core Data Type Attributes” on page 1-8 and “Additional Data Type Attributes” on page 1-9 for specific information.

ANSI Compliance

DATE with a DateForm of ANSIDate is ANSI SQL-99-compliant.

DATE with a DateForm of IntegerDate is a Teradata extension to the ANSI SQL-99 standard.

Internal Representation of DATE

Internally, Teradata RDBMS stores each DATE value as a four-byte signed integer using the following formula:

$$(YEAR - 1900) * 10000 + (MONTH * 100) + DAY$$

where the YEAR, MONTH, and DAY components, defined appropriately for the Gregorian calendar, have the following range of values:

Component	Range of values	
	Minimum	Maximum
YEAR	1	9999
MONTH	1	12
DAY	1	28 29 30 31

Note that the maximum value for day depends on the month and year.

Entering Dates

You can specify dates in SQL statements in three ways:

- String
- Number
- ANSI date literal

The preferred way to specify dates is as an ANSI date literal. A valid ANSI date literal requires no additional format validation for SQL date operations. For example, the following SQL statement works regardless of the DATE format of the referenced column:

```
INSERT INTO DATETAB VALUES (DATE '2001-12-20');
```

Specifying the date as just a string requires matching the DATE format of the referenced column. SQL requires that date string comparisons have the same format. For example, the following statement would result in an error if the DATE format of the column DOB were 'YYYY-MM-DD':

```
SELECT Name FROM Employee WHERE DOB = 'Jan 31 1948';
```

For details on the format characters that make up a valid DATE format, see [“Formatting Characters for Date Information” on page 8-29](#).

You can also specify a date as a number. A date as a number requires no format checking, but entering the number for years outside of the 1900's is not obvious and error prone. See [“Numeric Date Validation” on page 4-11](#).

The following is an example of 2001-12-20 entered as a number.

```
INSERT INTO DATETAB VALUES (1011220);
```

String Date Validations

When converting a string into a date value, Teradata RDBMS validates the components of the string as described in the following table.

Component	Requirement	Example
Date string - LATIN character set	Enclosed in single quotes.	'Jan 28, 1960'
Date string - Unicode hexadecimal format	Enclosed in single quotes and followed by xc.	'79797979C7AF6D6DB7EE6464C6FC'xc
Separator	Any non-numeric character can serve as a separator. Separators within a date do not have to match.	/
YY	The year as two <i>or four</i> numeric characters that are valid for the calendar. To support the century change transition, Teradata RDBMS accepts four-digit years in two-digit year date formats. Note: The Century Break feature determines the century of a two-digit year. For more information, see <i>Teradata RDBMS Utilities</i> .	05 2003
YYYY Y4	The year as four numeric characters that are valid for the calendar.	2003
MM	The month as two numeric characters that are valid for the calendar.	02
MMM M3	An abbreviated month name that matches one of the names specified by <i>ShortMonths</i> in the current Specification for Data Formatting (SDF) file.	Jan
MMMM M4	A full month name that matches one of the names specified by <i>LongMonths</i> in the current SDF.	January
DD	The day of the month as two numeric characters that are valid for the calendar.	03
DDD D3	The day of the year as three numeric characters that are valid for the calendar. If the date includes a month, the day must fall in that month.	056
EEE E3	An abbreviated day of the week name that matches one of the names specified by <i>ShortDays</i> in the current SDF.	Fri
EEEE E4	A day of the week name that matches one of the names specified by <i>LongDays</i> in the current SDF.	Friday

For more information on CHARACTER to DATE conversions, see Chapter 13: “Data Type Conversions” in *Teradata RDBMS SQL Reference, Volume 5*.

Numeric Date Validation

Although not recommended, you can enter dates as numbers in the Teradata RDBMS storage format. Teradata RDBMS stores each DATE value as a four-byte integer using the following formula:

$$(\text{year} - 1900) * 10000 + (\text{month} * 100) + \text{day}$$

Allowable date values range from AD January 1, 0001 to AD December 31, 9999. For example, December 31, 1985 would be stored as the integer 851231; July 4, 1776 stored as -1239296; and March 30, 2041 stored as 1410330.

The Century Break feature does not affect numeric dates.

Teradata RDBMS exports dates in this numeric format if the current DateForm setting is set to IntegerDate.

The following table demonstrates how numeric dates are interpreted when inserted into a column. Note the translation of the third date, which was probably intended to be 1990-12-01.

This raw date value ...	Translates to this value ...
901201	1990-12-01
1001201	2000-12-01
19901201	3890-12-01

Notice that this formula best fits two-digit dates in the 1900's. Because of the difficulty of using this format outside of the 1900's, we recommend specifying dates as ANSI date literals instead.

For more information on NUMERIC to DATE conversions, see Chapter 13: "Data Type Conversions" in *Teradata RDBMS SQL Reference, Volume 5*.

Export Data Type of DATE Values

The external DATE format for the client depends on whether the DateForm option is set to IntegerDate or ANSIDate mode.

For an overview of the operations that set the DATEFORM option, see ["Hierarchy of Date Formats" on page 4-20](#).

For details of the respective external representations, see ["Export Data Type of ANSI Date" on page 4-11](#) and ["Export Data Type of IntegerDate" on page 4-12](#).

Export Data Type of ANSI Date

The export data type of ANSI DATE values is CHARACTER(10) in the client character set with format 'YYYY-MM-DD'.

Export Data Type of IntegerDate

The following table lists the export data characteristics for the Teradata DATE type.

Determining the application definitions and client data types is the responsibility of the application programmer.

Client CPU Architecture	Client Internal Data Format
IBM mainframe	Four byte 32-bit signed 2's complement EBCDIC integer, most significant byte first.
• UTS • RISC • Motorola 68000 • WE 32000	Four byte 32-bit signed 2's complement ASCII integer, most significant byte first.
Intel	Four byte 32-bit signed 2's complement ASCII integer, least significant byte first.

Arithmetic Operations on DATE

A field that is declared as DATE can be operated on using addition and subtraction. The MIN, MAX, AVERAGE, and COUNT aggregate operators can also be used with DATE values.

You can perform arithmetic operations on DATE data. For example, a number of days can be added to or subtracted from a DATE field, or from the DATE built-in value. The result of such operations is a value that is also a DATE data type.

The number of days between two dates can be calculated by subtracting one DATE value from another DATE value. The result is an integer.

Teradata RDBMS handles month-to-month, year-to-year, and leap year arithmetic automatically; it does not store invalid dates, such as 1998-02-30.

In the following column definition, DOB is assigned the DATE data type.

```
DOB DATE FORMAT 'YYYY-MM-DD' NOT NULL
```

The following operations can be performed on data defined as DATE.

- Arithmetic (addition and subtraction)
Addition and subtraction of numeric types such as SMALLINT or INTEGER is still permitted, though these are non-ANSI operations because DATE is not a numeric data type in ANSI.
Such arithmetic operations are treated as if the numeric value had been typed as INTERVAL DAY.
For example, DATE - 12 is equivalent to the ANSI expression DATE - INTERVAL '12' DAY.

- Comparison
 - =
 - <>
 - >
 - >=
 - <
 - <=
 - OVERLAPS
- Format conversion
- ADD_MONTHS function
- EXTRACT function

For more information on arithmetic operations on DATE types, see Chapter 5: “DateTime and Interval Functions and Expressions” in *Teradata RDBMS SQL Reference, Volume 5*.

Example 1

To list all male employees currently over 50 years of age, the following statement might be entered:

```
SELECT name, dob
FROM employee
WHERE CURRENT_DATE > ADD_MONTHS (dob, 12*50)
AND sex = 'M' ;
```

The system returns:

Name	DOB
-----	-----
Russell S	1932-06-05
Carter J	1935-03-12
Inglis C	1938-03-07

Note that CURRENT_DATE is used in the expression to get the current date from the system.

To project a date three months from the date of birth of employee Russell, enter:

```
SELECT name, ADD_MONTHS (dob, 3)
FROM employee
WHERE name = 'Russell S' ;
```

The system returns:

Name	ADD_MONTHS (dob, 3)
-----	-----
Russell S	1932-09-05

Note: Three months is *not* a specific number of days. In this particular case, “three months” is 92 days.

Example 2

To list employees who were born between March 7, 1938, and August 25, 1942, you could specify the date information as follows:

```
SELECT name, dob
FROM employee
WHERE dob BETWEEN 380307
AND DATE '1942-08-25'
ORDER BY dob ;
```

In this example, the first date string is integer, in the form yymmdd, and needs no qualification. The second date is in the preferred DATE literal form. The result returns the date of birth information as specified for the Employee table:

Name	DOB
-----	-----
Inglis C	Mar 07 1938
Peterson J	Mar 27 1942

The display of DOB is controlled by the format for Personnel.Employee.DOB: `FORMAT 'MMM DD YYYY'`.

Example 3

To change the date format displayed above to an alternate form, change the SELECT to:

```
SELECT name, dob (FORMAT '99-99-99')
FROM employee
WHERE dob BETWEEN 380307 AND DATE '1942-08-25'
ORDER BY dob ;
```

This format specification changes the display to the following:

Name	DOB
-----	-----
Inglis C	38-03-07
Peterson J	42-03-27

Example 4

To change the display from date format to integer format, change the statement in Example 3 to:

```
SELECT name, dob (INTEGER)
FROM employee
WHERE dob BETWEEN 380307 AND 420825
ORDER BY dob ;
```

This format specification changes the display to the following:

Name	DOB
-----	-----
Inglis C	380307
Peterson J	420327

Further examples illustrating arithmetic operations on DATE are listed below:

For this statement ...	When the system date is Jan 24, 2001, the system returns this value ...
SELECT CURRENT_DATE;	Current_Date ----- 01/01/24
SELECT CURRENT_DATE +3;	(Current_Date+3) ----- 01/01/27
SELECT CURRENT_DATE -3;	(Current_Date-3) ----- 01/01/21
SELECT CURRENT_DATE - CURRENT_DATE;	(Current_Date-Current_Date) ----- 0

DATE Data Type Extensions with CALENDAR System View

Introduction

The SYS_CALENDAR.CALENDAR system view helps to extend the properties of a DATE data type column by means of a join.

CALENDAR defines the attributes of any date, which permits access to the table by means of an index on the DATE column.

The CALENDAR system view is defined for the years 1900 - 2100.

You can easily join CALENDAR to other tables, for example as a dimension of a star schema.

Authorization

To access SYS_CALENDAR.CALENDAR, you must have SELECT privileges.

CALENDAR Definition

The definition of CALENDAR is as follows:

```
CREATE VIEW CALENDAR (  
  calendar_date          //the ANSI DATE data type  
  day_of_week            //1-7, calculated assuming 1/1/1900 is Monday  
  day_of_month           //1-31  
  day_of_year            //1-366  
  day_of_calendar        //1- , Julian days since 1/1/1900  
  weekday_of_month       //1-5, the nth occurrence of day_of_week in  
                           month  
  week_of_month          //0-5, the partial week in the beginning is 0  
  week_of_year           //0-53, the partial week in the beginning is 0  
  week_of_calendar       //0- , the partial week in the beginning is 0  
  month_of_year          //1-12  
  month_of_quarter       //1-4  
  month_of_calendar      //1- , months since 1/1/1900  
  quarter_of_year        //1-4:Jan-Mar, Apr-Jun, Jul-Sep, Oct-Dec  
  quarter_of_calendar    //1- , quarters since Q1/1900.  
  year_of_calendar       //the year (assuming year starts Jan. 1)  
) AS ...  
;
```

Defining Views on CALENDAR

You are encouraged to define views on the CALENDAR system view because of its convenience.

A useful view to define on CALENDAR is TODAY, defined as follows.

```
CREATE VIEW Today AS (
  SELECT *
  FROM SYS_CALENDAR.Calendar
  WHERE SYS_CALENDAR.Calendar.calendar_date = DATE
);
```

How to Use CALENDAR

CALENDAR permits easy specification of arithmetic expressions and aggregation. This is particularly useful in OLAP environments where it is common to request values aggregated by weeks, months, year-to-date, years, and so on.

The following examples illustrate use of CALENDAR and the TODAY view in aggregates.

Example 1

What are the dollar sales for this week, last week, and the same weeks last year for all items in the sportswear class for women?

```
SELECT a2.week_of_calendar, SUM(a1.price)
FROM Sales a1, SYS_CALENDAR.CALENDAR a2, Item a3, Class a4,
TODAY a5
WHERE a1.calendar_date=a2.calendar_date
AND (a2.week_of_calendar=a5.week_of_calendar
OR a2.week_of_calendar=a5.week_of_calendar - 1
OR a2.week_of_calendar=a5.week_of_calendar - 52
OR a2.week_of_calendar=a5.week_of_calendar - 53
)
AND a1.itemID=a3.itemID
AND a3.classID=a4.classID
AND a4.classDesc='Women's Sportswear'
GROUP BY a2.week_of_calendar
ORDER BY a2.week_of_calendar
```

DATE Formats

Introduction

To simplify your work with dates, Teradata assumes that you want dates in a preset format. Your database administrator sets that format for the system with the DateForm setting and the default date format setting in the specification for data formatting (SDF) file.

Those settings determine the following:

- Export data type of DATE values
- Data entry format for DATE values
- Display format of DATE values in macros being executed
- Date format for string-to-DATE comparisons and conversions
- Display format of DATE columns in newly created tables and views

Changing or Overriding the DATE Format Settings

You can change or override the DATE format settings in any of the following ways:

- Change the DateForm at the system level
- Set the DateForm at the user or session level
- Change the system default date format in the SDF
- Specify a format in a FORMAT phrase

Each of these methods involves consequences that are explained in the following topics and summarized in [“Hierarchy of Date Formats” on page 4-20](#).

About DateForm

Before you change the DateForm setting, you need to consider the effect the change has on the following:

- Export data type of DATE values.
Changing the DateForm setting changes the export data type of DATE values.
- Data-entry format for dates.
Changing the DateForm setting changes the data entry format of dates for new tables, but not for existing tables. To avoid data entry problems, however, you can enter all dates as ANSI DATE literals.
- Display format of DATE columns.

Changing the DateForm setting changes the display of DATE values in new tables, but not in existing tables. So DATE values display differently for new tables than for old tables. Because views are based on the underlying table, this is true for views of tables as well.

- String-to-DATE comparisons and conversions.

To compare or convert strings to DATE values, date strings must have the same format as the DATE values. Such comparisons can fail after you change the DateForm setting, because the DATE format for new tables, views of new tables, and existing macros changes. To avoid format errors in string-to-DATE comparisons, however, you can specify dates as ANSI DATE literals.

- Format of DATE values in macros being executed.

Changing the DateForm setting can change the format of DATE values in macros, which can cause the macro to fail.

Changing the DateForm setting does *not* change the DATE format for the following:

- Tables created prior to the format change
- Tables created by users currently logged on
- Macros executed by users currently logged on
- Views based on tables that were created prior to the change

About the System Default DATE Format

When the value of DateForm is ANSIDate, the system default DATE format is YYYY-MM-DD. When the value of DateForm is IntegerDate, the system default DATE format is YY/MM/DD.

Your system administrator can change the default format that applies to DATE data types when the DateForm is set to IntegerDate. Changing the default DATE format involves changing the value of the *DATE* element in a custom Specification for Data Formatting (SDF) file, and using the *tdlocaledef* utility to convert the information into an internal form usable by Teradata RDBMS.

Setting or changing the default DATE format in the SDF has the same consequences as those for DateForm, except that the default DATE format in the SDF does not change the export data type of DATE values.

To view the value of the system default DATE format, use `HELP SESSION`.

About the FORMAT Phrase

You can use the `FORMAT` phrase to set the format for a specific DATE column or DATE value. A `FORMAT` phrase overrides the DATE format of the system, user, and session.

Setting a different DATE format for a column with the `FORMAT` phrase has the same consequences as with any other method:

- Data entry for that column must be in the new format or the ANSI DATE literal format.
- To compare or convert strings to DATE values in the column, date strings must match that format.

A FORMAT phrase does not change the export data type of DATE values.

For details about using the FORMAT phrase, see [“FORMAT Phrase and DateTime Formats” on page 8-29](#) or [“FORMAT Phrase, DateTime Formats, and Japanese Character Sets” on page 8-37](#).

Valid DATE Formats

The following table describes the valid DATE formats. The components that make up a valid date format are the same components that make up a DATE string validation. See [“String Date Validations” on page 4-10](#).

Using a format with a four-digit year, such as YYYY-MM-DD, is recommended to avoid the Year 2000 problem. To work with two-digit year formats in strings, see the Century Break feature in *Teradata RDBMS Utilities*.

	ANSIDate	IntegerDate	User-defined
Field Mode Display Format	YYYY-MM-DD	YY/MM/DD, or the value of the DATE element in a custom SDF	As defined by format
Allowed Input Separators	Any non-numeric character	Any non-numeric character	Any non-numeric character
Export Data Type	Character	Numeric	Depends on DateForm setting
Description	ANSI-specified date format.	Teradata legacy date format.	Custom date format for the system or a DATE column or value.
Example	1999-11-30	99/11/30	Jan 31, 2000

Hierarchy of Date Formats

The operations that set the DATE format are best described as a hierarchy, where one setting overrides another.

At the lowest level, the system comes with a DateForm of IntegerDate ('YY/MM/DD'). Your system administrator can override that date format by changing the DateForm to ANSIDate ('YYYY-MM-DD') or creating a custom SDF that changes the default DATE format (any valid date format).

You can override the system-level date format for a user, session, individual column, or individual value.

The following table summarizes the hierarchy of date formats.

For this operation ...	The format is set for ...	And using this session date setting ...	And field mode display format ...	The export data type is ...
SELECT with FORMAT phrase	an individual column	user-defined display format	as defined by FORMAT	CHAR(10) if DateForm is ANSIDate; ELSE four-byte integer
CREATE / ALTER TABLE, with FORMAT phrase				
SET SESSION DateForm	the session	ANSIDate	YYYY-MM-DD	CHAR(10)
CLI SessionOptions parcel DateForm		IntegerDate	YY/MM/DD, or the value of the DATE element in a custom SDF	four-byte integer
CREATE / MODIFY USER DateForm	the user	ANSIDate	YYYY-MM-DD	CHAR(10)
		IntegerDate	YY/MM/DD, or the value of the DATE element in a custom SDF	four-byte integer
System DBS Control utility DateForm	the system	ANSIDate	YYYY-MM-DD	CHAR(10)
		IntegerDate	YY/MM/DD, or the value of the DATE element in a custom SDF	four-byte integer
Default DATE format in custom SDF	the system	IntegerDate	as defined by the DATE format in the SDF	four-byte integer

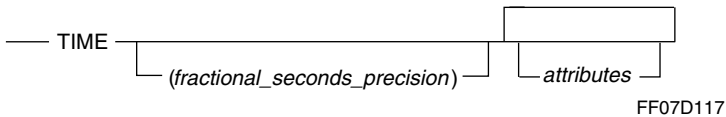
TIME

Purpose

Identifies a field as a TIME value.

This is not the TIME value returned by the TIME function, which is formatted as a REAL number.

Syntax



where:

Syntax element ...	Specifies ...
<i>fractional_seconds_precision</i>	a single digit representing the number of significant digits in the fractional portion of the SECOND field. Values for <i>fractional_seconds_precision</i> range from 0 through 6 inclusive. The default precision is 6.
<i>attributes</i>	appropriate data type, column storage, or column constraint attributes. See “Core Data Type Attributes” on page 1-8 and “Additional Data Type Attributes” on page 1-9 for specific information.

ANSI Compliance

TIME is ANSI SQL-99-compliant.

Specification of *fractional_seconds_precision* is full-level ANSI-compliant.

Internal Representation of TIME

Conceptually, TIME is treated as a record composed of three fields. The record is not treated as a numeric value, though it is composed of numeric fields.

The length of the internal stored form is six bytes.

The component fields, defined appropriately for the Gregorian calendar, are:

Field name	Range of values		Storage Format
	Minimum	Maximum	
HOUR	00	23	BYTEINT
MINUTE	00	59	BYTEINT
SECOND	00.000000	61.999999	DECIMAL(8,6)

TIME Formats

For information on changing the TIME format setting, and importing and exporting TIME data types, see [“TIME and TIMESTAMP Formats” on page 4-31](#).

Implicit and Explicit TIME Conversion

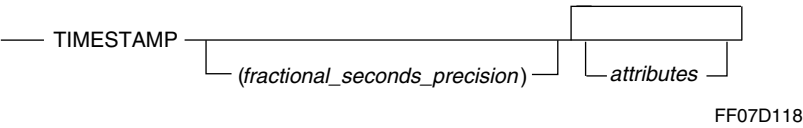
Teradata performs implicit conversion from CHARACTER to TIME types during assignment and comparison. You can also use CAST or the Teradata conversion syntax to explicitly convert CHARACTER to TIME types. For more information, see Chapter 13: “Data Type Conversions” in *Teradata RDBMS SQL Reference, Volume 5*.

TIMESTAMP

Purpose

Identifies a field as a TIMESTAMP value.

Syntax



where:

Syntax element ...	Specifies ...
<i>fractional_seconds_precision</i>	a single digit representing the number of significant digits in the fractional portion of the SECOND field. Values for <i>fractional_seconds_precision</i> range from zero through six inclusive. The default precision is six.
<i>attributes</i>	appropriate data type, column storage, or column constraint attributes. See “Core Data Type Attributes” on page 1-8 and “Additional Data Type Attributes” on page 1-9 for specific information.

ANSI Compliance

TIMESTAMP is ANSI SQL-99-compliant.

Internal Representation of TIMESTAMP

Conceptually, TIMESTAMP is treated as a record composed of six fields. The record is not treated as a numeric value, though it is composed of numeric fields.

The length of the internal stored form is 10 bytes.

The component fields, defined appropriately for the Gregorian calendar, are:

Field name	Range of values		Storage Format
	Minimum	Maximum	
YEAR	0001	9999	SMALLINT
MONTH	01	12	BYTEINT

Field name	Range of values		Storage Format
	Minimum	Maximum	
DAY	01	28 29 30 31	BYTEINT
HOUR	00	23	BYTEINT
MINUTE	00	59	BYTEINT
SECOND	00.000000	61.999999	DECIMAL(8,6)

TIMESTAMP Formats

For information on changing the TIMESTAMP format setting, and importing and exporting TIMESTAMP data types, see [“TIME and TIMESTAMP Formats” on page 4-31](#).

Implicit and Explicit TIMESTAMP Conversion

Teradata performs implicit conversion from CHARACTER to TIMESTAMP types during assignment and comparison. You can also use CAST or the Teradata conversion syntax to explicitly convert CHARACTER to TIMESTAMP types. For more information, see Chapter 13: “Data Type Conversions” in *Teradata RDBMS SQL Reference, Volume 5*.

Example 1

The following example shows TIMESTAMP used in a query.

```
SELECT item, quantity, saletime
FROM sales
WHERE saletime > TIMESTAMP '2000-08-25 10:14:59'
AND saletime < TIMESTAMP '2000-08-25 10:30:01';
```

Example 2

The difference between two TIMESTAMP types is an Interval type.

First define a table:

```
CREATE TABLE BillDateTime
(phone_no CHARACTER(10)
, start_time TIMESTAMP(0)
, end_time TIMESTAMP(0));
```

Now, determine the difference, specifying an Interval unit of DAY TO SECOND for the result:

```
SELECT (end_time - start_time) DAY(4) TO SECOND
FROM BillDateTime;
```

The DAY(4) specifies four digits of precision, and allows for a maximum of 9999 days, or approximately 27 years. The result looks like:

```
5 16:49:20.340000
```

Example 3

The following example compares two TIMESTAMP numbers to find out if they are within 30 minutes of each other.

First define a table:

```
CREATE TABLE PhoneTime
(phone_no CHARACTER(10)
,start_time TIMESTAMP(0)
,end_time TIMESTAMP(0));
```

Note that the difference between two TIMESTAMP types is an Interval type:

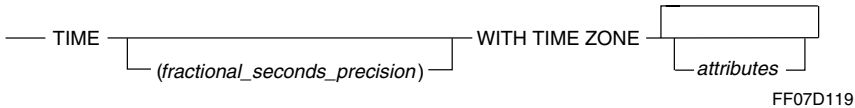
```
SELECT phone_no
FROM PhoneTime
WHERE (end_time - start_time) DAY(4) TO MINUTE > INTERVAL '30'
MINUTE;
```

TIME WITH TIME ZONE

Purpose

Identifies a field as a TIME value with a displacement from UTC as defined for the system.

Syntax



where:

Syntax element ...	Specifies ...
<i>fractional_seconds_precision</i>	a single digit representing the number of significant digits in the fractional portion of the SECOND field. Values for <i>fractional_seconds_precision</i> range from zero through six inclusive. The default precision is six.
<i>attributes</i>	appropriate data type, column storage, or column constraint attributes. See “Core Data Type Attributes” on page 1-8 and “Additional Data Type Attributes” on page 1-9 for specific information.

ANSI Compliance

TIME WITH TIME ZONE is ANSI SQL-99-compliant.

Internal Representation of TIME WITH TIME ZONE

Conceptually, TIME WITH TIME ZONE is treated as a record composed of five fields. The record is not treated as a numeric value, though it is composed of numeric fields.

The length of the internal stored form is eight bytes.

The component fields, defined appropriately for the Gregorian calendar, are:

Field name	Range of values		Storage Format
	Minimum	Maximum	
HOUR	00	23	BYTEINT
MINUTE	00	59	BYTEINT
SECOND	00.000000	61.999999	DECIMAL(8,6)
TIMEZONE_HOUR	-12:59	+13.00	BYTEINT
TIMEZONE_MINUTE			BYTEINT

TIME WITH TIME ZONE Formats

For information on changing the TIME WITH TIME ZONE format setting, and importing and exporting TIME WITH TIME ZONE data types, see [“TIME and TIMESTAMP Formats” on page 4-31](#).

Implicit and Explicit TIME WITH TIME ZONE Conversion

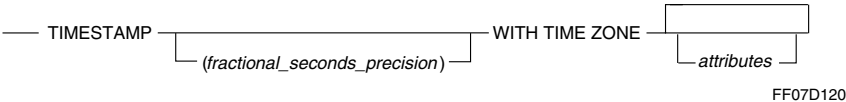
Teradata performs implicit conversion from CHARACTER to TIME WITH TIME ZONE types during assignment and comparison. You can also use CAST or the Teradata conversion syntax to explicitly convert CHARACTER to TIME WITH TIME ZONE types. For more information, see Chapter 13: “Data Type Conversions” in *Teradata RDBMS SQL Reference, Volume 5*.

TIMESTAMP WITH TIME ZONE

Purpose

Identifies a field as a `TIMESTAMP` value with a displacement from UTC as defined for the system.

Syntax



where:

Syntax element ...	Specifies ...
<i>fractional_seconds_precision</i>	a single digit representing the number of significant digits in the fractional portion of the <code>SECOND</code> field. Values for <i>fractional_seconds_precision</i> range from zero through six inclusive. The default precision is six.
<i>attributes</i>	appropriate data type, column storage, or column constraint attributes. See “Core Data Type Attributes” on page 1-8 and “Additional Data Type Attributes” on page 1-9 for specific information.

ANSI Compliance

`TIMESTAMP WITH TIME ZONE` is ANSI SQL-99-compliant.

Internal Representation of `TIMESTAMP WITH TIME ZONE`

Conceptually, `TIMESTAMP WITH TIME ZONE` is treated as a record composed of eight fields. The record is not treated as a numeric value, though it is composed of numeric fields.

The length of the internal stored form is 12 bytes.

The component fields, defined appropriately for the Gregorian calendar, are:

Field name	Range of values		Storage Format
	Minimum	Maximum	
YEAR	0001	9999	SMALLINT
MONTH	01	12	BYTEINT
DAY	01	28 29 30 31	BYTEINT
HOUR	00	23	BYTEINT
MINUTE	00	59	BYTEINT
SECOND	00.000000	61.999999	DECIMAL(8,6)
TIMEZONE_HOUR	-12:59	+13.00	BYTEINT
TIMEZONE_MINUTE			BYTEINT

TIMESTAMP WITH TIME ZONE Formats

For information on changing the TIMESTAMP WITH TIME ZONE format setting, and importing and exporting TIMESTAMP WITH TIME ZONE data types, see [“TIME and TIMESTAMP Formats” on page 4-31](#).

Implicit and Explicit TIMESTAMP WITH TIME ZONE Conversion

Teradata performs implicit conversion from CHARACTER to TIMESTAMP WITH TIME ZONE types during assignment and comparison. You can also use CAST or the Teradata conversion syntax to explicitly convert CHARACTER to TIMESTAMP WITH TIME ZONE types. For more information, see Chapter 13: “Data Type Conversions” in *Teradata RDBMS SQL Reference, Volume 5*.

TIME and TIMESTAMP Formats

Introduction

Teradata SQL uses a default set of formats for the output of TIME and TIMESTAMP expressions and column data, and for comparison and conversion of TIME and TIMESTAMP data types.

Changing or Overriding the TIME and TIMESTAMP Format Settings

You can change or override the TIME and TIMESTAMP format settings for field mode in the following ways:

- Change the system default TIME and TIMESTAMP formats in the specification for data formatting (SDF) file.
- Specify a format in a FORMAT phrase.

The FORMAT phrase sets the format for a specific TIME or TIMESTAMP column or value. A FORMAT phrase overrides the system format.

The TIME and TIMESTAMP format setting pertains to data in report form, as is the case in BTEQ. The format does not control internal storage representation of data or data returned in record or indicator variable mode.

For details about changing the system default TIME format and using the FORMAT phrase, see [Chapter 8: “Output Format Phrases.”](#)

About the System Default TIME and TIMESTAMP Format

The system default formats are as follows:

Data Type	System Default Format
TIME TIME WITH TIME ZONE	HH:MI:SS.S(F)Z Time zone information is displayed for TIME WITH TIME ZONE, but not for TIME.
TIMESTAMP TIMESTAMP WITH TIME ZONE	YYYY-MM-DDHH:MI:SS.S(F)Z Time zone information is displayed for TIMESTAMP WITH TIME ZONE, but not for TIMESTAMP.

Your system administrator can change the default format for TIME and TIMESTAMP data types. Changing the default TIME and TIMESTAMP formats involves changing the values of the *TIME* and *TIMESTAMP* elements in the Specification for Data Formatting (SDF) file, and using the `tdlocaledef` utility to convert the information into an internal form usable by Teradata RDBMS.

Before changing the default format, consider the effect on the following:

- Data-entry format.
Changing the default formats changes the data entry format of TIME and TIMESTAMP data for new tables, but not for existing tables. To avoid data entry problems, however, you can enter all times and timestamps as ANSI TIME and TIMESTAMP literals.
- Display format.
Changing the default format changes the display of TIME and TIMESTAMP values in new tables, but not in existing tables. So values display differently for new tables than for old tables. Because views are based on the underlying table, this is true for views of tables as well.
- Character-to-TIME and character-to-TIMESTAMP comparisons and conversions.
To compare or convert strings to TIME or TIMESTAMP values, the strings must have the same format as the TIME or TIMESTAMP values. Such comparisons can fail after you change the default format, because the format changes for new tables, views of new tables, and existing macros. To avoid format errors in character-to-TIME and character-to-TIMESTAMP comparisons, however, you can specify times and timestamps as ANSI TIME and TIMESTAMP literals.
- Format of TIME and TIMESTAMP values in macros being executed.
Changing the default format can change the format of TIME and TIMESTAMP values in macros, which can cause the macro to fail.

Changing the default format does *not* change the TIME or TIMESTAMP format for the following:

- Tables created prior to the format change
- Tables created by users currently logged on
- Macros executed by users currently logged on
- Views based on tables that were created prior to the change

To view the values of the system default TIME and TIMESTAMP formats, use `HELP SESSION`.

About the FORMAT Phrase

You can use the FORMAT phrase to set the format for a specific TIME or TIMESTAMP column or value. A FORMAT phrase overrides the TIME or TIMESTAMP system format.

Setting a different TIME or TIMESTAMP format for a column with the FORMAT phrase has the same consequences as with any other method:

- Data entry for that column must be in the new format or the ANSI TIME or TIMESTAMP literal format.
- To compare or convert strings to TIME or TIMESTAMP values in the column, strings must match that format.

A FORMAT phrase does *not* change the export data type of TIME or TIMESTAMP values.

For details about using the FORMAT phrase, see [“FORMAT Phrase and DateTime Formats” on page 8-29](#) or [“FORMAT Phrase, DateTime Formats, and Japanese Character Sets” on page 8-37](#).

Valid TIME and TIMESTAMP Formats

For details on the components that make up a valid TIME and TIMESTAMP format, see [“Formatting Characters for Time Information” on page 8-31](#).

Hierarchy of TIME and TIMESTAMP Formats

The operations that set the TIME and TIMESTAMP formats are best described as a hierarchy, where one setting overrides another.

At the lowest level, the system comes with a default TIME and TIMESTAMP format. Your system administrator can override the formats by creating a custom SDF that changes the default TIME and TIMESTAMP formats.

You can override the system-level TIME or TIMESTAMP format for an individual column or individual value.

The following table summarizes the hierarchy of TIME and TIMESTAMP formats.

For this operation ...	The format is set for ...	And field mode display format ...
SELECT with FORMAT phrase	an individual column	as defined by FORMAT
CREATE / ALTER TABLE, with FORMAT phrase		
Default TIME or TIMESTAMP format in custom SDF	the system	as defined by the TIME and TIMESTAMP format in the SDF

Import and Export Formats for Record and Indicator Mode

TIME and TIMESTAMP types are imported and exported in record and indicator modes as CHARACTER data using the ANSI format string and the site-defined client character set.

FOR this data type ...	WHEN fractional seconds precision is this value ...	THEN the import and export type and format is this value ...
TIME	0	CHAR(8) 'hh:mi:ss'
	<i>n</i> where <i>n</i> is 1 - 6	CHAR(9+ <i>n</i>) 'hh:mi:ss.ss...'
TIME WITH TIME ZONE	0	CHAR(14) 'hh:mi:ss-hh:mi' or 'hh:mi:ss+hh:mi'
	<i>n</i> where <i>n</i> is 1 - 6	CHAR(15+ <i>n</i>) 'hh:mi:ss.ss...-hh:mi' or 'hh:mi:ss.ss...+hh:mi'
TIMESTAMP	0	CHAR(19) 'yyyy-mm-dd hh:mi:ss'
	<i>n</i> where <i>n</i> is 1 - 6	CHAR(20+ <i>n</i>) 'yyyy-mm-dd hh:mi:ss.ss...'
TIMESTAMP WITH TIME ZONE	0	CHAR(25) 'yyyy-mm-dd hh:mi:ss-hh:mi' or 'yyyy-mm-dd hh:mi:ss+hh:mi'
	<i>n</i> where <i>n</i> is 1 - 6	CHAR(26+ <i>n</i>) 'yyyy-mm-dd hh:mi:ss.ss...-hh:mi' or 'yyyy-mm-dd hh:mi:ss.ss...+hh:mi'

Example 1: TIMESTAMP WITH TIME ZONE Export Format

The following table shows examples of how TIMESTAMP WITH TIME ZONE types are exported in record and indicator modes.

WHEN <i>fractional_seconds_precision</i> is set to this value ...	THEN length is this value ...	FOR example ...
6	32	'2000-01-01 11:37:58.123456+08:00'
0	25	'2000-01-01 11:37:58-08:00'

Example 2: TIME Export Format

The following table shows examples of how TIME types are exported in record and indicator modes.

WHEN <i>fractional_seconds_precision</i> is set to this value ...	THEN the length is this value ...	FOR example ...
6	15	'11:37:58.123456'
0	8	'11:37:58'

Example 3: TIME WITH TIME ZONE Export Format

The following table shows examples of how TIME WITH TIME ZONE types are exported in record and indicator modes.

WHEN <i>fractional_seconds_precision</i> is set to this value ...	THEN length is this value ...	FOR example ...
6	21	'11:37:58.123456+08:00'
0	14	'11:37:58-08:00'

Example 4: TIMESTAMP Export Format

The following table shows examples of how TIMESTAMP types are exported in record and indicator modes.

WHEN <i>fractional_seconds_precision</i> is set to this value ...	THEN length is this value ...	FOR example ...
6	26	'1999-01-01 23:59:59.999999'
0	19	'1999-01-01 23:59:59'

Example 5: TIME Output Format in Field Mode

The following table is a nonexhaustive list of formats that can be used to present an output time in field mode, where the data is 13:20:53.64+03:00:

FORMAT Phrase	Result
FORMAT 'HH:MIBT'	01:20 PM
FORMAT 'HH:MI'	13:20
FORMAT 'HH.MI.SS'	13.20.53
FORMAT 'HH:MI:SSBT'	01:20:53 nachm ^a
FORMAT 'HH:MI:SSDS(F)'	13:20:53.64
FORMAT 'HH:MI:SSDS(F)Z'	13:20:53.64+03:00
FORMAT 'HHhMI mSSs'	13h20m53s

a. nachm is German for PM.

Example 6: TIMESTAMP Output Format in Field Mode

The following table is a nonexhaustive list of formats for presenting an output timestamp in field mode, where the data is 85/09/12 13:20:53.64+03:00:

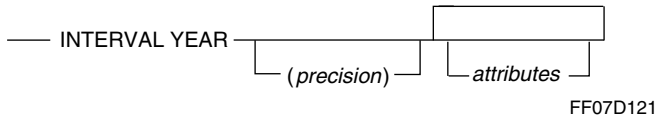
FORMAT Phrase	Result
FORMAT 'MM/DD/YYBHH:MIBT'	85/09/12 01:20 PM
FORMAT 'MMMBDD,BYYBHH:MI:SS'	Sep 12, 85 13:20:53
FORMAT 'E3,BM4BDD,BY4BHH:MI:SSDS(F)'	Thu, September 12, 1985 13:20:53.64
FORMAT 'YYYY-MM-DDBHH:MI:SSDS(F)Z'	1985-09-12 13:20:53.64+03:00

INTERVAL YEAR

Purpose

Identifies a field as an INTERVAL value defining a period of time in years.

Syntax



where:

Syntax element ...	Specifies ...
<i>precision</i>	the permitted range of digits for YEAR, ranging from 1 to 4. The default is 2. For example, if you specify a <i>precision</i> of 4, the range for YEAR is - '9999' through '9999'.
<i>attributes</i>	appropriate data type, column storage, or column constraint attributes. See “Core Data Type Attributes” on page 1-8 and “Additional Data Type Attributes” on page 1-9 for specific information.

ANSI Compliance

INTERVAL YEAR is ANSI SQL-99-compliant.

Internal Representation of INTERVAL YEAR

Storage Format	Length
SMALLINT	2 bytes

External Representation of INTERVAL YEAR

Storage Format	Length
CHARACTER(n)	<i>precision</i> + 1 byte

The value includes a leading blank or minus sign followed by the indicated number of digits.

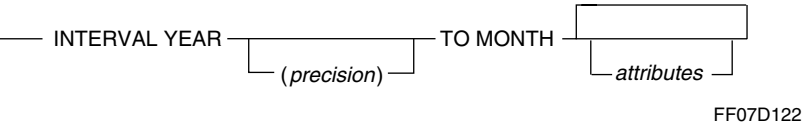
For example, for INTERVAL YEAR(3), the value might be something like '-125'.

INTERVAL YEAR TO MONTH

Purpose

Identifies a field as an INTERVAL value defining a period of time in years and months.

Syntax



where:

Syntax element ...	Specifies ...
<i>precision</i>	the permitted range of digits for YEAR, ranging from one to four. The default is two.
<i>attributes</i>	appropriate data type, column storage, or column constraint attributes. See “Core Data Type Attributes” on page 1-8 and “Additional Data Type Attributes” on page 1-9 for specific information.

ANSI Compliance

INTERVAL YEAR TO MONTH is ANSI SQL-99-compliant.

Internal Representation of INTERVAL YEAR TO MONTH

Storage Format	Length
SMALLINT SMALLINT	4 bytes

The value for MONTH is always two digits, ranging from 00 to 11, inclusive.

External Representation of INTERNAL YEAR TO MONTH

Storage Format	Length
CHARACTER(n)	<i>precision</i> + 4 bytes

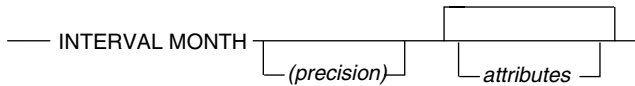
For example, for INTERVAL YEAR(3) TO MONTH, the value might be something like ‘013-11’.

INTERVAL MONTH

Purpose

Identifies a field as an INTERVAL value defining a period of time in months.

Syntax



1101A013

where:

Syntax element ...	Specifies ...
<i>precision</i>	the permitted range of digits for MONTH, ranging from one to four. The default is two.
<i>attributes</i>	appropriate data type, column storage, or column constraint attributes. See “Core Data Type Attributes” on page 1-8 and “Additional Data Type Attributes” on page 1-9 for specific information.

ANSI Compliance

INTERVAL MONTH is ANSI SQL-99-compliant.

Internal Representation of INTERVAL MONTH

Storage Format	Length
SMALLINT	2 bytes

External Representation of INTERVAL MONTH

Storage Format	Length
CHARACTER(n)	<i>precision</i> + 1 byte

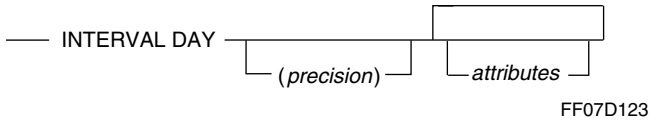
For example, for INTERVAL MONTH(4), the value might be something like '1300'.

INTERVAL DAY

Purpose

Identifies a field as an INTERVAL value defining a period of time in days.

Syntax



where:

Syntax element ...	Specifies ...
<i>precision</i>	the permitted range of digits for DAY, ranging from one to four. The default is two.
<i>attributes</i>	appropriate data type, column storage, or column constraint attributes. See “Core Data Type Attributes” on page 1-8 and “Additional Data Type Attributes” on page 1-9 for specific information.

ANSI Compliance

INTERVAL DAY is ANSI SQL-99-compliant.

Internal Representation of INTERVAL DAY

Storage Format	Length
SMALLINT	2 bytes

External Representation of INTERVAL DAY

Storage Format	Length
CHARACTER(n)	<i>precision</i> + 1 byte

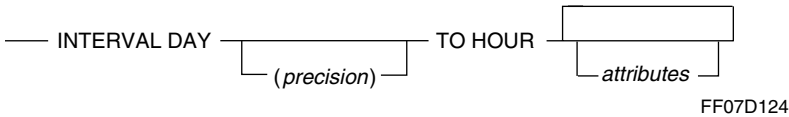
For example, for INTERVAL DAY(2), the value might be something like ‘-39’.

INTERVAL DAY TO HOUR

Purpose

Identifies a field as an INTERVAL value defining a period of time in days and hours.

Syntax



where:

Syntax element ...	Specifies ...
<i>precision</i>	the permitted range of digits for DAY, ranging from one to four. The default is two.
<i>attributes</i>	appropriate data type, column storage, or column constraint attributes. See “Core Data Type Attributes” on page 1-8 and “Additional Data Type Attributes” on page 1-9 for specific information.

ANSI Compliance

INTERVAL DAY TO HOUR is ANSI SQL-99-compliant.

Internal Representation of INTERVAL DAY TO HOUR

Storage Format	Length
SMALLINT	4 bytes
SMALLINT	

The value for HOUR is always two digits, ranging from 00 to 23, inclusive.

External Representation of INTERVAL DAY TO HOUR

Storage Format	Length
CHARACTER(n)	<i>precision</i> + 4 bytes

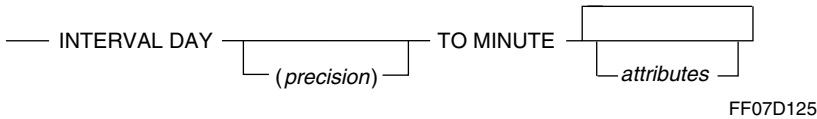
For example, for INTERVAL DAY (2) TO HOUR, the value might be something like ‘17 11’.

INTERVAL DAY TO MINUTE

Purpose

Identifies a field as an INTERVAL value defining a period of time in days and minutes.

Syntax



where:

Syntax element ...	Specifies ...
<i>precision</i>	the permitted range of digits for DAY, ranging from one to four. The default is two.
<i>attributes</i>	appropriate data type, column storage, or column constraint attributes. See “Core Data Type Attributes” on page 1-8 and “Additional Data Type Attributes” on page 1-9 for specific information.

ANSI Compliance

INTERVAL DAY TO MINUTE is ANSI SQL-99-compliant.

Internal Representation of INTERVAL DAY TO MINUTE

Storage Format	Length
SMALLINT SMALLINT SMALLINT	8 bytes (with 2 pad bytes)

The value for DAY is always two digits, ranging from 01 to 31, inclusive, and the value for MINUTE is always 2 digits, ranging from 00 to 59, inclusive.

External Representation of INTERVAL DAY TO MINUTE

Storage Format	Length
CHARACTER(n)	<i>precision</i> + seven

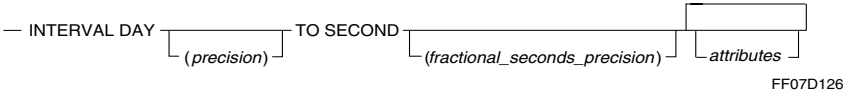
For example, for INTERVAL DAY (3) TO MINUTE, the value might be something like '-127 12:37'.

INTERVAL DAY TO SECOND

Purpose

Identifies a field as an INTERVAL value defining a period of time in days and seconds.

Syntax



where:

Syntax element ...	Specifies ...
<i>precision</i>	the permitted range of digits for DAY, ranging from one to four. The default is two.
<i>fractional_seconds_precision</i>	the fractional precision for the values of SECOND. The default is six.
<i>attributes</i>	appropriate data type, column storage, or column constraint attributes. See “Core Data Type Attributes” on page 1-8 and “Additional Data Type Attributes” on page 1-9 for specific information.

ANSI Compliance

INTERVAL DAY TO SECOND is ANSI SQL-99-compliant.

Internal Representation of INTERVAL DAY TO SECOND

Storage Format	Length
SMALLINT SMALLINT SMALLINT DECIMAL(8,6)	10 bytes

External Representation of INTERVAL DAY TO SECOND

Storage Format	Length	
	WHEN <i>fractional_seconds_precision</i> is this value ...	THEN the length is this value ...
CHARACTER(n)	0	<i>precision</i> + 10 bytes
	1 - 6	<i>precision</i> + <i>fractional_seconds_precision</i> + 11 bytes

For example, for INTERVAL DAY (4) TO SECOND(3), the value might be something like '125 12:27:31.125'.

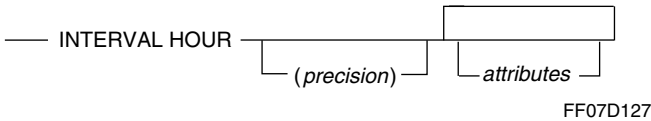
The DAY field has two leading blanks. The first is for a sign, which is not present for positive intervals. The second is to preserve the fixed size of the DAY field at two digits.

INTERVAL HOUR

Purpose

Identifies a field as an INTERVAL value defining a period of time in hours.

Syntax



where:

Syntax element ...	Specifies ...
<i>precision</i>	the permitted range of digits for HOUR, ranging from one to four. The default is two.
<i>attributes</i>	appropriate data type, column storage, or column constraint attributes. See “Core Data Type Attributes” on page 1-8 and “Additional Data Type Attributes” on page 1-9 for specific information.

ANSI Compliance

INTERVAL HOUR is ANSI SQL-99-compliant.

Internal Representation of INTERVAL HOUR

Storage Format	Length
SMALLINT	2 bytes

External Representation of INTERVAL HOUR

Storage Format	Length
CHARACTER(n)	<i>precision</i> + 1 byte

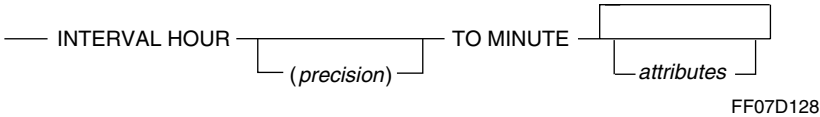
For example, for INTERVAL HOUR(1), the value might be something like ‘-5’.

INTERVAL HOUR TO MINUTE

Purpose

Identifies a field as an INTERVAL value defining a period of time in hours and minutes.

Syntax



where:

Syntax element ...	Specifies ...
<i>precision</i>	the permitted range of digits for HOUR, ranging from one to four. The default is two.
<i>attributes</i>	appropriate data type, column storage, or column constraint attributes. See “Core Data Type Attributes” on page 1-8 and “Additional Data Type Attributes” on page 1-9 for specific information.

ANSI Compliance

INTERVAL HOUR TO MINUTE is ANSI SQL-99-compliant.

Internal Representation of INTERVAL HOUR TO MINUTE

Storage Format	Length
SMALLINT SMALLINT	4 bytes

External Representation of INTERVAL HOUR TO MINUTE

Storage Format	Length
CHARACTER(n)	<i>precision</i> + 4 bytes

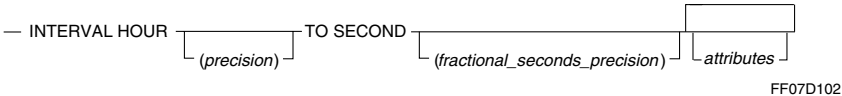
For example, for INTERVAL HOUR(2) TO MINUTE, the value might be something like '17:37'.

INTERVAL HOUR TO SECOND

Purpose

Identifies a field as an INTERVAL value defining a period of time in hours and seconds.

Syntax



where:

Syntax element ...	Specifies ...
<i>precision</i>	the permitted range of digits for HOUR, ranging from one to four. The default is two.
<i>fractional_seconds_precision</i>	the fractional precision for the values of SECOND, ranging from zero to six. The default is six.
<i>attributes</i>	appropriate data type, column storage, or column constraint attributes. See “Core Data Type Attributes” on page 1-8 and “Additional Data Type Attributes” on page 1-9 for specific information.

ANSI Compliance

INTERVAL HOUR TO SECOND is ANSI SQL-99-compliant.

Internal Representation of INTERVAL HOUR TO SECOND

Storage Format	Length
SMALLINT SMALLINT DECIMAL(8,6)	8 bytes

External Representation of INTERVAL HOUR TO SECOND

Storage Format	Length	
	WHEN <i>fractional_seconds_precision</i> is this value ...	THEN the length is this value ...
CHARACTER(n)	0	<i>precision</i> + 7 bytes
	1 - 6	<i>precision</i> + <i>fractional_seconds_precision</i> + 8 bytes

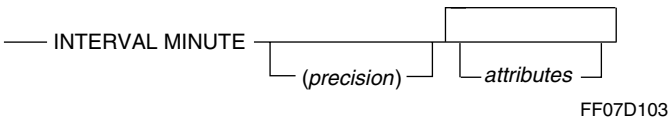
For example, for INTERVAL HOUR(2) TO SECOND(2), the value might be something like '-13:27:58.17'.

INTERVAL MINUTE

Purpose

Identifies a field as an INTERVAL value defining a period of time in minutes.

Syntax



where:

Syntax element ...	Specifies ...
<i>precision</i>	the permitted range of digits for MINUTE, ranging from one to four. The default is two.
<i>attributes</i>	appropriate data type, column storage, or column constraint attributes. See “Core Data Type Attributes” on page 1-8 and “Additional Data Type Attributes” on page 1-9 for specific information.

ANSI Compliance

INTERVAL MINUTE is ANSI SQL-99-compliant.

Internal Representation of INTERVAL MINUTE

Storage Format	Length
SMALLINT	2 bytes

External Representation of INTERVAL MINUTE

Storage Format	Length
CHARACTER(n)	<i>precision</i> + 1 byte

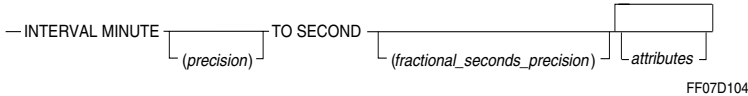
For example, for INTERVAL MINUTE(2), the value might be something like '49'.

INTERVAL MINUTE TO SECOND

Purpose

Identifies a field as an INTERVAL value defining a period of time in minutes and seconds.

Syntax



where:

Syntax element ...	Specifies ...
<i>precision</i>	the permitted range of digits for MINUTE, ranging from one to four. The default is two.
<i>fractional_seconds_precision</i>	the fractional precision for the values of SECOND, ranging from zero to six. The default is six.
<i>attributes</i>	appropriate data type, column storage, or column constraint attributes. See “Core Data Type Attributes” on page 1-8 and “Additional Data Type Attributes” on page 1-9 for specific information.

ANSI Compliance

INTERVAL MINUTE TO SECOND is ANSI SQL-99-compliant.

Internal Representation of INTERVAL MINUTE TO SECOND

Storage Format	Length
SMALLINT DECIMAL(8,6)	6 bytes

External Representation of INTERVAL MINUTE TO SECOND

Storage Format	Length	
	WHEN <i>fractional_seconds_precision</i> is this value ...	THEN the length is this value ...
CHARACTER(n)	0	<i>precision</i> + 4 bytes
	1 - 6	<i>precision</i> + <i>fractional_seconds_precision</i> + 5 bytes

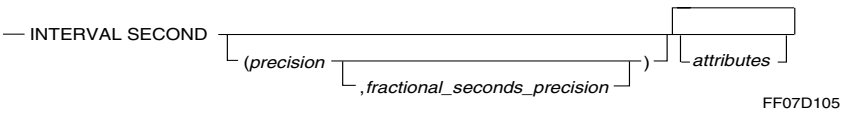
For example, for INTERVAL MINUTE(2) TO SECOND(0), the value might be something like ' 17:39'.

INTERVAL SECOND

Purpose

Identifies a field as an INTERVAL value defining a period of time in seconds.

Syntax



where:

Syntax element ...	Specifies ...
<i>precision</i>	the permitted range of digits for SECOND, ranging from one to four. The default is two.
<i>fractional_seconds_precision</i>	the fractional precision for the values of SECOND, ranging from zero to six. The default is six.
<i>attributes</i>	appropriate data type, column storage, or column constraint attributes. See “Core Data Type Attributes” on page 1-8 and “Additional Data Type Attributes” on page 1-9 for specific information.

ANSI Compliance

INTERVAL SECOND is ANSI SQL-99-compliant.

Internal Representation of INTERVAL SECOND

Storage Format	Length
DECIMAL(10,6)	6 bytes

External Representation of INTERVAL SECOND

Storage Format	Length	
	WHEN <i>fractional_seconds_precision</i> is this value ...	THEN the length is this value ...
CHARACTER(n)	0	<i>precision</i> + 1 byte
	1 - 6	<i>precision</i> + <i>fractional_seconds_precision</i> + 2 bytes

For example, for INTERVAL SECOND(3,2), the value might be something like '-127.35'.



Chapter 5:

Character Data Types

Topics described in this chapter include:

- Character data
- Client character set
- Teradata CHARACTER strings and client physical bytes
- Character data types
 - CHARACTER
 - VARCHAR
 - LONG VARCHAR
- Defining case sensitivity for table columns
 - CASESPECIFIC phrase
 - UPPERCASE phrase
- CHARACTER SET clause
 - LATIN server character set
 - UNICODE server character set
 - GRAPHIC server character set
 - KANJISJIS server character set
 - KANJI1 server character set

Character Data

Introduction

In general, CHARACTER, VARCHAR, and LONG VARCHAR data types are used to represent character data. Each of these can be combined with a:

- Case option ([NOT]CASESPECIFIC or UPPERCASE).
- Server character set literal when the field data type is declared or modified. Character literals can be prefixed with a server character set to ensure conformity to the given server character set.

Character data is automatically translated between the client and the RDBMS. Its form-of-use is determined by the client character set or session character set.

The form of character data *internal* to the RDBMS is determined by the server character set attribute of the column.

Terminology

Note the definition of the following expressions, used throughout this chapter:

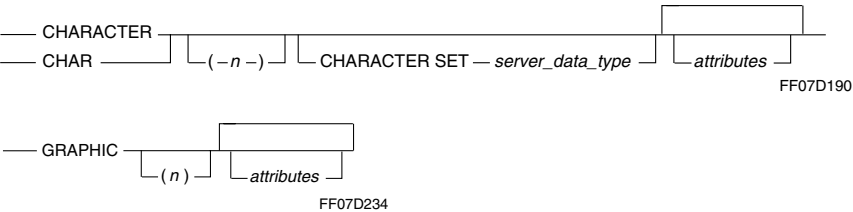
Expression	Definition
Character data type	The definition for the type of data stored in a column. See: • “CHARACTER Type” on page 5-3 • “VARCHAR Type” on page 5-9 • “LONG VARCHAR Type” on page 5-14
Client character set	The character set used to communicate between the client and the Teradata RDBMS. See “Client Character Set” on page 5-23 .
Server character set	The internal character storage handling implied by the ANSI CHARACTER SET clause. See “CHARACTER SET Clause” on page 5-28 .

CHARACTER Type

Purpose

Represents a fixed length character string for Teradata RDBMS internal character storage.

Syntax



where:

Syntax element ...	Specifies ...		
n	the number of characters or bytes allotted to the column defined with this server character set.		
	IF the server character set is one of these ...	THEN it is defined in terms of ...	AND its maximum length is ...
	LATIN	characters	64000
	UNICODE		32000
	GRAPHIC		32000
	KANJ11	bytes	64000
	KANJISJIS		32000

Syntax element ...	Specifies ...												
<i>server_data_type</i>	the server character set for the character column being defined. See “CHARACTER SET Clause” on page 5-28. If the CHARACTER SET <i>server_data_type</i> clause is omitted, the default <i>server_data_type</i> depends on how the user is defined in the DEFAULT CHARACTER SET clause of the CREATE USER statement. See “CREATE USER” in Chapter 1 – “SQL Data Definition Language Statement Syntax” in <i>Teradata RDBMS SQL Reference, Volume 4</i>. <table> <tr> <th>This value for <i>server_data_type</i> ...</th><th>Represents ...</th></tr> <tr> <td>LATIN</td><td> fixed 8-bit characters from the ASCII ISO 8859 Latin1 or ISO 8859 Latin9 repertoires. See “LATIN Server Character Set” on page 5-29. </td></tr> <tr> <td>UNICODE</td><td> fixed 16-bit characters from the UNICODE ISO 10646 Level1 repertoire. See “UNICODE Server Character Set” on page 5-30. </td></tr> <tr> <td>GRAPHIC</td><td> fixed 16-bit UNICODE characters defined by IBM Corporation for DB2. See “GRAPHIC Server Character Set” on page 5-31. </td></tr> <tr> <td>KANJISJIS</td><td> mixed single byte / multibyte characters intended for Japanese applications that rely on KanjiShiftJIS characteristics. See “KANJIISJIS Server Character Set” on page 5-32. </td></tr> <tr> <td>KANJI1</td><td> mixed single byte / multibyte characters as defined by KanjiEBCDIC, KanjiShift-JIS, or KanjiEUC, depending on the current character set for the session. See “KANJI1 Server Character Set” on page 5-33. </td></tr> </table>	This value for <i>server_data_type</i> ...	Represents ...	LATIN	fixed 8-bit characters from the ASCII ISO 8859 Latin1 or ISO 8859 Latin9 repertoires. See “LATIN Server Character Set” on page 5-29.	UNICODE	fixed 16-bit characters from the UNICODE ISO 10646 Level1 repertoire. See “UNICODE Server Character Set” on page 5-30.	GRAPHIC	fixed 16-bit UNICODE characters defined by IBM Corporation for DB2. See “GRAPHIC Server Character Set” on page 5-31.	KANJISJIS	mixed single byte / multibyte characters intended for Japanese applications that rely on KanjiShiftJIS characteristics. See “KANJIISJIS Server Character Set” on page 5-32.	KANJI1	mixed single byte / multibyte characters as defined by KanjiEBCDIC, KanjiShift-JIS, or KanjiEUC, depending on the current character set for the session. See “KANJI1 Server Character Set” on page 5-33.
This value for <i>server_data_type</i> ...	Represents ...												
LATIN	fixed 8-bit characters from the ASCII ISO 8859 Latin1 or ISO 8859 Latin9 repertoires. See “LATIN Server Character Set” on page 5-29.												
UNICODE	fixed 16-bit characters from the UNICODE ISO 10646 Level1 repertoire. See “UNICODE Server Character Set” on page 5-30.												
GRAPHIC	fixed 16-bit UNICODE characters defined by IBM Corporation for DB2. See “GRAPHIC Server Character Set” on page 5-31.												
KANJISJIS	mixed single byte / multibyte characters intended for Japanese applications that rely on KanjiShiftJIS characteristics. See “KANJIISJIS Server Character Set” on page 5-32.												
KANJI1	mixed single byte / multibyte characters as defined by KanjiEBCDIC, KanjiShift-JIS, or KanjiEUC, depending on the current character set for the session. See “KANJI1 Server Character Set” on page 5-33.												

Syntax element ...	Specifies ...
<i>attributes</i>	appropriate data type, column storage, or column constraint attributes. See “ Core Data Type Attributes ” on page 1-8 and “ Additional Data Type Attributes ” on page 1-9 for specific information.

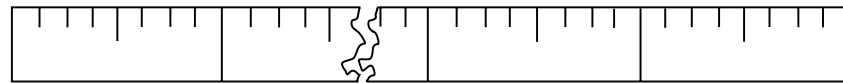
ANSI Compliance

CHARACTER is ANSI SQL-99-compliant.

GRAPHIC is a Teradata extension to the ANSI SQL-99 standard.

Scale

CHAR (n)



$n = 1$ to 64000 bytes depending on the character data type

FF07D283

External Representation of CHARACTER

Whenever a client application communicates with the Teradata RDBMS, it indicates its character set (form-of-use for character data). The Teradata server returns all character data to the client application in that form.

Whenever multibyte characters are involved, their representation has the length $(n-2)/2$ multibyte characters exclusive of the preceding Shift-Out and following Shift-In characters.

The following table lists the client representations for the Teradata CHARACTER data type.

Client CPU Architecture	Client Internal Data Format
IBM mainframe	n EBCDIC characters.
• UTS • RISC • Motorola 68000 • WE 32000 • Intel	n ASCII characters.

General Usage Notes

CHARACTER data defaults to 8-bit LATIN type for U.S. and European character support, and KANJI1 for Japanese language support.

If a value for n is not specified, the default is CHARACTER(1).

The maximum length of a CHARACTER string constant is 32 Kbytes.

In the following column definitions, the column named Sex is assigned the CHARACTER data type with a length of 1, and the column named Frgn_Lang is assigned the CHARACTER data type with a length of seven.

```
Sex CHARACTER NOT NULL UPPERCASE,  
Frgn_Lang CHARACTER(7) NULL UPPERCASE,
```

The default display format of CHARACTER(n) is X(n). For example, X(5), where data 'HELLO' displays as 'HELLO'.

Character data is allocated either in terms of characters or in terms of bytes, depending on the server character set used.

The number of bytes of storage per character also varies depending on the server character set, as illustrated by the following table.

Server Character Set	Sharable Among Heterogeneous Clients?	Server Space Allocation	RDBMS Form-of-Use
LATIN	Yes	Character-based	Fixed 8-bit LATIN
UNICODE	Yes	Character-based	Fixed 16-bit UNICODE
KANJISJIS	Yes	Byte-based	Mixed single and multibyte KANJISJIS
GRAPHIC	Yes	Character-based	Fixed 16-bit UNICODE
KANJI1	No	Byte-based	Mixed single and multibyte KanjiEBCDIC, KanjiShift-JIS, or KanjiEUC

Any conversion to or from the client system data types is done by the Teradata RDBMS.

Note that the CHARACTER SET clause is a form of explicit translation.

For information on explicit translation of CHARACTER data, see “TRANSLATE Function” in Chapter 7: “Character String Functions” in *Teradata RDBMS SQL Reference, Volume 5*.

If the clause is applied to a character expression, the expression is first translated using the implicit translation rules (see “Implicit Character-to-Character Translation” in Chapter 13: “Data Type Conversions” in *Teradata RDBMS SQL Reference, Volume 5*), and then truncated or extended as appropriate.

FOR this <i>character_data_type</i> ...	<i>n</i> indicates the number of these in the translation result ...
• UNICODE • LATIN • GRAPHIC	characters
• KANJISJIS • KANJI1	bytes

Usage Notes: GRAPHIC Data

You should define all GRAPHIC(*n*) data as CHARACTER(*n*) CHARACTER SET GRAPHIC.

The following information describes properties of GRAPHIC data (defined as CHARACTER(*n*) CHARACTER SET GRAPHIC).

IF ...	THEN ...
the length is not specified	the default is GRAPHIC(1)—1 logical character (2 bytes).
a graphic string is shorter than the specified length of the column	the remaining space is filled with the graphic pad character.
a graphic string is longer than the specified length of the column	the extra characters are truncated.

On a Japanese language supported site, the following data types can be used to represent multibyte character data.

- GRAPHIC
- VARGRAPHIC (see [“VARCHAR Type” on page 5-9](#))
- LONG VARGRAPHIC (see [“LONG VARCHAR Type” on page 5-14](#))

Each multibyte character in a graphic string is stored assuming 2 bytes per logical character. Therefore, a graphic data string must always represent an *even* multiple of bytes.

Graphic data types accommodate the following character sets:

- KanjiEBCDIC double byte graphic data
- KanjiShift-JIS for double byte Shift-JIS codes
- KanjiEUC for fixed-length, double byte EUC characters

Use the following syntax for KanjiEBCDIC graphic data:

```
— G — '< graphic_characters >' —
```

HH01A098

Under a KanjiEBCDIC character set, the multibyte characters comprising the graphic string constant must be delimited with the Shift-Out/Shift-In characters; for example:

```
INSERT INTO TableEBCDIC (ColGRAPH)
VALUES (G'<AB>');
```

where **AB** is a valid string of KanjiEBCDIC multibyte characters, G specifies the string must be in the Graphic repertoire, and each '(quote)' is a single byte character.

External Representation of GRAPHIC

The following table lists the client representation for the IBM DB2 GRAPHIC type.

Determining the application definitions and client data types is the responsibility of the application programmer.

Client CPU Architecture	Client Internal Data Format
IBM mainframe	2 <i>n</i> bytes of <i>n</i> DB2 GRAPHIC characters.

Japanese Character Graphic Literals

The graphic literal G'<**ABC**>' is a Teradata extension to the ANSI SQL-99 standard.

Example

Assume that column_1 is defined as CHARACTER(3) CHARACTER SET GRAPHIC and that it contains the following data:

```
457F4577456D
```

Under a KanjiEBCDIC session in record or indicator mode, the contents of column_1 are returned to the user as follows:

```
457F4577456D
```

Under a KanjiEBCDIC session in field mode, the contents of column_1 are returned to the user in proper format, as follows:

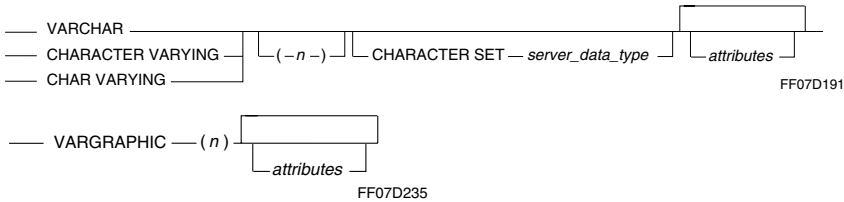
```
0E457F4577456D0F
```

VARCHAR Type

Purpose

Represents a variable length character string of length *n* for Teradata RDBMS internal character storage.

Syntax



where:

Syntax element ...	Specifies ...															
<i>n</i>	the number of characters or bytes allotted to the column defined with this server character set. <table><tr><th>IF <i>server_data_type</i> is one of these ...</th><th>THEN it is defined in terms of ...</th><th>AND its maximum length is ...</th></tr><tr><td>LATIN</td><td rowspan="3">characters</td><td>64000</td></tr><tr><td>UNICODE</td><td>32000</td></tr><tr><td>GRAPHIC VARCHAR</td><td>32000</td></tr><tr><td>KANJI1</td><td rowspan="2">bytes</td><td>64000</td></tr><tr><td>KANJISJIS</td><td>32000</td></tr></table>	IF <i>server_data_type</i> is one of these ...	THEN it is defined in terms of ...	AND its maximum length is ...	LATIN	characters	64000	UNICODE	32000	GRAPHIC VARCHAR	32000	KANJI1	bytes	64000	KANJISJIS	32000
IF <i>server_data_type</i> is one of these ...	THEN it is defined in terms of ...	AND its maximum length is ...														
LATIN	characters	64000														
UNICODE		32000														
GRAPHIC VARCHAR		32000														
KANJI1	bytes	64000														
KANJISJIS		32000														

Syntax element ...	Specifies ...												
<i>server_data_type</i>	the server character set for the character being defined. If the CHARACTER SET <i>server_data_type</i> clause is omitted, the default <i>server_data_type</i> depends on how the user is defined in the DEFAULT CHARACTER SET clause of the CREATE USER statement. See “CREATE USER” in Chapter 1: “SQL Data Definition Language Statement Syntax” in <i>Teradata RDBMS SQL Reference, Volume 4</i>. <table> <tr> <th>This value for <i>server_data_type</i> ...</th><th>Represents ...</th></tr> <tr> <td>LATIN</td><td> fixed 8-bit characters from the ASCII ISO 8859 Latin1 or ISO 8859 Latin9 repertoires. See “LATIN Server Character Set” on page 5-29. </td></tr> <tr> <td>UNICODE</td><td> fixed 16-bit characters from the UNICODE ISO 10646 Level1 repertoire. See “UNICODE Server Character Set” on page 5-30. </td></tr> <tr> <td>GRAPHIC</td><td> fixed 16-bit UNICODE characters defined by IBM Corporation for DB2. See “GRAPHIC Server Character Set” on page 5-31. </td></tr> <tr> <td>KANJISJIS</td><td> mixed single byte/multibyte characters intended for Japanese applications that rely on KANJISJIS characteristics. See “KANJISJIS Server Character Set” on page 5-32. </td></tr> <tr> <td>KANJI1</td><td> mixed single byte/multibyte characters as defined by KanjiEBCDIC, KanjiShift-JIS, or KanjiEUC, depending on the current character set for the session. See “KANJI1 Server Character Set” on page 5-33. </td></tr> </table>	This value for <i>server_data_type</i> ...	Represents ...	LATIN	fixed 8-bit characters from the ASCII ISO 8859 Latin1 or ISO 8859 Latin9 repertoires. See “LATIN Server Character Set” on page 5-29.	UNICODE	fixed 16-bit characters from the UNICODE ISO 10646 Level1 repertoire. See “UNICODE Server Character Set” on page 5-30.	GRAPHIC	fixed 16-bit UNICODE characters defined by IBM Corporation for DB2. See “GRAPHIC Server Character Set” on page 5-31.	KANJISJIS	mixed single byte/multibyte characters intended for Japanese applications that rely on KANJISJIS characteristics. See “KANJISJIS Server Character Set” on page 5-32.	KANJI1	mixed single byte/multibyte characters as defined by KanjiEBCDIC, KanjiShift-JIS, or KanjiEUC, depending on the current character set for the session. See “KANJI1 Server Character Set” on page 5-33.
This value for <i>server_data_type</i> ...	Represents ...												
LATIN	fixed 8-bit characters from the ASCII ISO 8859 Latin1 or ISO 8859 Latin9 repertoires. See “LATIN Server Character Set” on page 5-29.												
UNICODE	fixed 16-bit characters from the UNICODE ISO 10646 Level1 repertoire. See “UNICODE Server Character Set” on page 5-30.												
GRAPHIC	fixed 16-bit UNICODE characters defined by IBM Corporation for DB2. See “GRAPHIC Server Character Set” on page 5-31.												
KANJISJIS	mixed single byte/multibyte characters intended for Japanese applications that rely on KANJISJIS characteristics. See “KANJISJIS Server Character Set” on page 5-32.												
KANJI1	mixed single byte/multibyte characters as defined by KanjiEBCDIC, KanjiShift-JIS, or KanjiEUC, depending on the current character set for the session. See “KANJI1 Server Character Set” on page 5-33.												

Syntax element ...	Specifies ...
<i>attributes</i>	appropriate data type, column storage, or column constraint attributes. See “Core Data Type Attributes” on page 1-8 and “Additional Data Type Attributes” on page 1-9 for specific information.

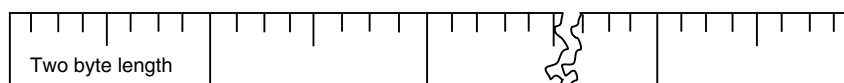
ANSI Compliance

VARCHAR is ANSI SQL-99-compliant.

VARGRAPHIC is a Teradata extension to the ANSI SQL-99 standard.

Scale

VARCHAR (n)



n = 1 to 64000

FF07D284

External Representation of VARCHAR

Whenever a client application talks to the Teradata RDBMS, it indicates its character set (form-of-use for character data). The RDBMS returns all character data to the application in that form.

Whenever multibyte characters are involved, their representation has the length $(n-2)/2$ multibyte characters exclusive of the preceding Shift-Out and following Shift-In characters.

The following table lists the client representations for the Teradata VARCHAR type.

Define the length of the VARCHAR string as k , where $0 \leq k \leq n$.

Client CPU Architecture	Client Internal Data Format
IBM mainframe	Two byte SMALLINT length count k followed by k bytes of EBCDIC CHARACTER data.

Client CPU Architecture	Client Internal Data Format
• UTS • RISC • Motorola 68000 • WE 32000	Two byte 16-bit SMALLINT (most significant byte first) length count <i>k</i> followed by <i>k</i> bytes of ASCII CHARACTER data for a total of <i>k</i> +2 bytes.
Intel	Two byte 16-bit SMALLINT (least significant byte first) length count <i>k</i> followed by <i>k</i> bytes of ASCII CHARACTER data for a total of <i>k</i> +2 bytes.

Usage Notes

You should define all VARGRAPHIC(*n*) data as VARCHAR(*n*) CHARACTER SET GRAPHIC.

The maximum length for *n* in a VARCHAR(*n*) CHARACTER SET GRAPHIC definition is 32 000. There is no default—omitting the length specification results in an error.

The default display format for VARCHAR(*n*) is X(*n*).

For example, X(6) displays data 'GOOD' as 'GOOD ' with two trailing blanks.

Character data is allocated either in terms of characters or in terms of bytes, depending on the server character set used. The number of bytes of storage per character also varies depending on the server character set, as illustrated by the following table.

Server Character Set	Sharable Among Heterogeneous Clients?	Server Space Allocation	Server Form-of-Use
LATIN	Yes	Character-based	Fixed 8-bit LATIN
UNICODE	Yes	Character-based	Fixed 16-bit UNICODE
KANJISJIS	Yes	Byte-based	Mixed single and multibyte KANJISJIS
GRAPHIC	Yes	Character-based	Fixed 16-bit UNICODE
KANJI1	No	Byte-based	Mixed single and multibyte KanjiEBCDIC, KanjiShift-JIS, or KanjiEUC

Any conversion to or from the client system data types is done by the Teradata RDBMS. This data type supports national character sets.

Note that the CHARACTER SET clause is a form of explicit translation.

For information on explicit translation of VARCHAR data, see “TRANSLATE Function” in Chapter 7: “Character String Functions” in *Teradata RDBMS SQL Reference, Volume 5*.

If the clause is applied to a character expression, the expression is first translated using the implicit translation rules (see “Implicit Character-To-Character Translation” in Chapter 13: “Data Type Conversions” in *Teradata RDBMS SQL Reference, Volume 5*), and then truncated or extended as appropriate.

FOR this <i>server_data_type</i> ...	<i>n</i> indicates the number of these in the translation result ...
UNICODE LATIN GRAPHIC	characters
KANJISJIS KANJI1	bytes

External Representation of VARGRAPHIC

The following table lists the client representation for the IBM DB2 VARGRAPHIC type.

Determining the application definitions and client data types is the responsibility of the application programmer.

Define the length of the VARGRAPHIC string as k , where $0 \leq k \leq n$

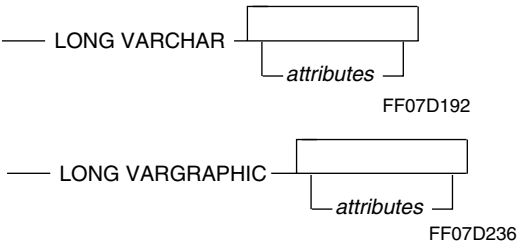
Client CPU Architecture	Client Internal Data Format
IBM mainframe	Two byte SMALLINT length count k followed by k DB2 GRAPHIC characters for a total of $2+2k$ EBCDIC bytes.

LONG VARCHAR Type

Purpose

LONG VARCHAR specifies the longest permissible variable length character string for Teradata internal character storage.

Syntax



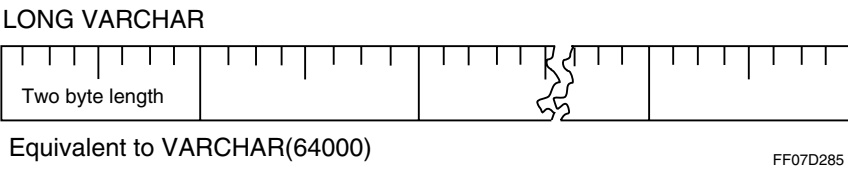
where:

Syntax element ...	Specifies ...
<i>attributes</i>	appropriate data type, column storage, or column constraint attributes. See “Core Data Type Attributes” on page 1-8 and “Additional Data Type Attributes” on page 1-9 for specific information.

ANSI Compliance

LONG VARCHAR is a Teradata extension to the ANSI SQL-99 standard.

Scale



External Representation of LONG VARCHAR

Whenever a client application talks to the Teradata RDBMS, it indicates its character set (form-of-use for character data). The RDBMS returns all character data to the application in that form.

See [“External Representation of VARCHAR” on page 5-11](#).

The following table shows how LONG VARCHAR data is represented for the various server character sets. Apart from these definitions, LONG VARCHAR strings behave identically to VARCHAR strings.

FOR this server character set ...	The external representation for LONG VARCHAR is equivalent to ...
• LATIN • KANJI1	VARCHAR(64000)
• UNICODE • GRAPHIC • KANJISJIS	VARCHAR(32000)

Usage Notes

LONG VARCHAR is equivalent to VARCHAR(64000) for server character sets LATIN and KANJI1 or VARCHAR(32000) for server character sets UNICODE, GRAPHIC, and KANJISJIS.

You should define all LONG VARGRAPHIC data as LONG VARCHAR CHARACTER SET GRAPHIC.

LONG VARCHAR CHARACTER SET GRAPHIC is equivalent to specifying a data type of VARCHAR(32000) CHARACTER SET GRAPHIC.

LONG VARCHAR data is stored in extended ASCII for U.S. and European character support, and extended JIS X 0201 for Japanese language support. Character data is allocated in terms of bytes. Except for the multibyte characters available with Japanese language support, each character requires one byte of storage.

The default display format for LONG VARCHAR(n) is either X(32000) or X(64000) depending on the server character set in effect.

Defining Case Sensitivity for Table Columns

Introduction

Default case sensitivity varies with mode.

ANSI Mode

All character data columns defined in ANSI mode are defined as CASESPECIFIC by default. This means that the values 'aa' and 'AA' are not treated as equals.

An option of NOT CASESPECIFIC is implemented to allow columns to be case *insensitive*. NOT CASESPECIFIC is an ANSI extension.

Teradata Mode

All character data columns defined in Teradata mode (with the exception of character data defined as GRAPHIC) are defined as NOT CASESPECIFIC by default.

Character data columns for tables created under prior releases continue to have the same data type attribute set when as when the columns were defined.

Because Latin characters have canonical representation in all Teradata RDBMS character sets, the effect of the UPPERCASE function is the same for each character set (with the exception of KANJI1, where multibyte characters are not converted to uppercase).

The CASESPECIFIC and UPPERCASE phrases are described in detail in:

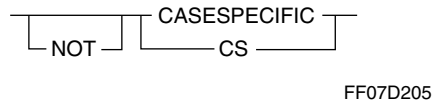
- [“CASESPECIFIC Phrase” on page 5-17](#)
- [“UPPERCASE Phrase” on page 5-21](#)

CASESPECIFIC Phrase

Purpose

Specifies case for character data comparisons and collations.

Syntax



ANSI Compliance

CASESPECIFIC is a Teradata extension to the ANSI SQL-99 standard.

Rules: All Modes

The following set of rules applies to CASESPECIFIC and NOT CASESPECIFIC in both ANSI and Teradata modes.

- CASESPECIFIC specifies that character data be stored as typed and then compared case-specifically.
- NOT CASESPECIFIC specifies that character data be stored as typed, but compares it non-case-specifically.
- If neither CASESPECIFIC nor NOT CASESPECIFIC is specified as part of the column definition, then you can specify the option in an SQL request to ensure that the statement behaves as intended. By including an explicit CASESPECIFIC (or CS) qualifier in your SQL statement, you override the stored case specificity assignment for a column.
- In character string comparisons, if either of the strings being compared is CASESPECIFIC, then the comparison is always CASESPECIFIC.
- A column typed as GRAPHIC always defaults to CASESPECIFIC when it is created.
- To create a non-CASESPECIFIC GRAPHIC column, you must specify a NOT CASESPECIFIC clause in the column definition. You can also use NOT CASESPECIFIC in a comparison predicate.
- KANJI1 data only handles NOT CASESPECIFIC for characters in the range A–Z. All other server character sets handle all data correctly.

Note that the performance of views, macros, and CHECK table constraints can involve parsing character string literals at execution time as well as the more common processing of character string literals in queries.

See also Chapter 1: “The SELECT Statement” in *Teradata RDBMS SQL Reference, Volume 6*.

Rules: Teradata Mode

The following set of rules applies to CASESPECIFIC and NOT CASESPECIFIC in Teradata mode.

- All character types except GRAPHIC default to NOT CASESPECIFIC. You can override this default in a query by specifying (CASESPECIFIC) immediately following the CHARACTER data to be compared. For example, the following query is not case specific in Teradata mode.

```
SELECT DataBaseName
FROM DBC.Databases
WHERE DataBaseName = 'dbc' ;
```

You can make it case specific by qualifying the predicate with (CASESPECIFIC) as follows.

```
SELECT DataBaseName
FROM DBC.Databases
WHERE DataBaseName = 'dbc' (CASESPECIFIC) ;
```

Note that the first example returns 1 row, while the second example returns none.

- CASESPECIFIC supports all letters in the ISO 10646 repertoire. See also [“UPPERCASE Phrase” on page 5-21](#).

Rules: ANSI Mode

The following set of rules applies to CASESPECIFIC and NOT CASESPECIFIC in ANSI mode.

- All character data defaults to CASESPECIFIC.
- To make case blind comparisons using ANSI SQL-99 syntax, you *must* apply the UPPER function to any character string value that may contain lower case Latin letters.
- To make case blind comparisons using Teradata syntax, you can apply the NOT CASESPECIFIC specification to the appropriate character string. UPPER is preferable because it is defined by the ANSI SQL-99 standard, while NOT CASESPECIFIC is a Teradata extension to the standard.
- UPPER is an ANSI-defined function and *is not* the same as declaring a value UPPERCASE, which is a Teradata extension to the ANSI standard. See “UPPER” in Chapter 8: “Character String Functions” in *Teradata RDBMS SQL Reference, Volume 5* for further information.

- CASESPECIFIC supports all letters in the ISO 10646 repertoire.
If you declare a character column to be NOT CASESPECIFIC and type 'ᾱ' (GREEK SMALL LETTER ALPHA TONOS), then that letter is stored as 'ᾱ' but during comparison compares equal to 'Α' (GREEK CAPITAL LETTER ALPHA TONOS).
The same applies to the (NOT CASESPECIFIC) qualifier in a SQL predicate.
The following SQL predicate evaluates to TRUE.

```
'ᾱ' (NOT CASESPECIFIC) = 'Α'
```

Without the (NOT CASESPECIFIC) qualifier, the same predicate evaluates to FALSE.

Usage Rules

The following table summarizes case criteria and issues.

Case Specification	Usage Rules
CASESPECIFIC (CS)	Character data is stored in the case in which it was entered unless UPPERCASE is also specified. In sorts or comparisons, lowercase characters are not converted to uppercase and are not equal to uppercase characters. • If a column is defined as CASESPECIFIC, characters entered as 'aaa' are not equivalent to 'AAA' when used in a unique index. • Applications that conform to ANSI should define columns for ANSI mode and use the UPPER function to force case blind comparisons. See “Example 1” on page 5-20.
NOT CASESPECIFIC (NOT CS)	In sorts or comparisons, lowercase characters are converted to uppercase. 'aaa' is equivalent to 'AAA'. Because the stored characters were not converted from lowercase to uppercase, data entered as 'aaa' is not equivalent to 'AAA' when used in a unique index.
Neither CASE SPECIFIC nor NOT CASESPECIFIC	The session mode determines the attribute assigned by default. • In ANSI mode, CASESPECIFIC is set. • In Teradata mode (except for GRAPHIC data), NOT CASESPECIFIC is set.
UPPERCASE <i>not</i> specified NOT CASESPECIFIC specified	AAA, aaa, AaA, and aAA are all equivalent as unique indexes.

Example 1

The following query returns a result only if a case specific comparison of the literal 'Leidner P' finds a match.

```
SELECT Name
FROM Employee
WHERE Name(CS) = 'Leidner P' ;
```

The literal 'Leidner P' might default to CS or NOT CS, depending on the current mode, but since the type modifier of the comparison is CS, the comparison is case specific irrespective of the session mode.

To ensure a case blind comparison irrespective of the session mode, specify the query as follows.

```
SELECT Name
FROM Employee
WHERE Name (NOT CS) = 'Leidner P' (NOT CS) ;
```

Alternatively, you can specify the query using ANSI-compatible syntax as follows.

```
SELECT Name
FROM Employee
WHERE UPPER (Name) = 'LEIDNER P' ;
```

Example 2

CASESPECIFIC comparison and collation on mixed case data can produce unintended results.

```
SELECT Last_Name
FROM SalesReps
ORDER BY Last_Name(CS) ;
```

might return one of the following sorted lists depending on the collation in effect for the session:

EBCDIC	ASCII	MULTINATIONAL
bart	ACME	ACME
fernandez	ALBERT	Albert
hill	Albert	ALBERT
Albert	FARRAH	bart
ACME	Kimble	FARRAH
ALBERT	bart	fernandez
FARRAH	fernandez	hill
Kimble	hill	Kimble

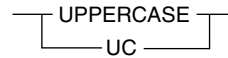
UPPERCASE Phrase

Purpose

Specifies that character data for a column is stored as uppercase.

Also used to specify case for character data comparisons and collations.

Syntax



FF07D206

ANSI Compliance

UPPERCASE is a Teradata extension to the ANSI SQL-99 standard.

Rules: All Modes

The following set of rules applies to UPPERCASE in both ANSI and Teradata modes.

- If UPPERCASE is not part of the column definition, you can specify the option in a SQL request to override the stored case specificity assignment for a column.
- If a column is declared to be UPPERCASE, then lowercase letters are converted and stored as their uppercase equivalents
Note that the KANJI1 server character set only converts the 26 letters a-z to uppercase.
- UPPERCASE supports all letters in the ISO 10646 repertoire.
If you declare a character column to be UPPERCASE and type 'ᾱ' (GREEK SMALL LETTER ALPHA TONOS), then that character is translated and stored as 'A' (GREEK CAPITAL LETTER ALPHA TONOS).

The same applies to the (UC) qualifier.

The following SQL predicate evaluates to TRUE.

```
'ᾱ' (UC) = 'A'
```

UPPERCASE Is Not the Same As UPPER

The UPPER function is defined by the ANSI SQL-99 standard and is *not* the same as declaring a value to be UPPERCASE. See “UPPER” in Chapter 8: “Character String Functions” in *Teradata RDBMS SQL Reference, Volume 5* for further information.

Usage Rules

The following table summarizes case criteria and issues.

Case Specification	Usage Rules
UPPERCASE <i>not</i> specified NOT CASESPECIFIC specified	The following are all equivalent as unique indexes: • AAA • aaa • AaA • aAA
UPPERCASE (UC)	Character data is stored in uppercase regardless of the case in which it was typed. Because of the conversion, if a column is defined as UPPERCASE, characters typed as 'aaa' are equivalent to 'AAA' when used in a unique index.

Example 1

The Sex column is created using the UPPERCASE option.

```
Sex CHAR UPPERCASE
```

If Sex data is entered in lowercase as shown in the following INSERT statement, then it is stored and returned in uppercase.

```
INSERT INTO Employee (Name, EmpNo, Sex ...  
VALUES ('Smith', 10021, 'f', ...);
```

```
SELECT Sex  
FROM Employee;
```

```
Sex  
F
```

Example 2

If used with the concatenation (||) operator, UC must be enclosed in parentheses, and be placed immediately after the column name.

```
SELECT (City_name (UC)) || ', ' || State, Population  
FROM World  
WHERE Country = 'USA'  
ORDER BY Population ;
```

Client Character Set

Definition

A client character set is the standard character form-of-use defined for a client user session.

Note: Teradata has always used the phrase “character set” to refer to the session character set—the character set used between the client and the Teradata RDBMS. The ANSI SQL-92 standard defines a CHARACTER SET clause to specify the handling of character data stored in the RDBMS.

Output Conversion by Teradata

The Teradata RDBMS translates output characters (internal-to-external, or E2I) to one of the following:

- The normal character set of the logged on client.
- The character set defined in the system tables as the default for the logon client.

This is typically one of the following:

- EBCDIC
- ASCII
- Predefined Japanese character set such as KATAKANAEBCDIC or KanjiEUC
- User-defined character set

Query the DBC.HostsInfo and DBC.CharSets views to see the contents of the system tables. If your application relies on the DBC.Hosts table for the definition of the default character set name, you must ensure that the initial logon from the channel-attached (IBM) side is in EBCDIC and from *all other platforms* ASCII; otherwise, the default character set cannot be known to the application before logon.

- The character set specified for the current session with the BTEQ .SET SESSION CHARSET command or the CLIV2 CHARSET call.

Uppercase Conversion

On a Japanese language site, only the lowercase simple Latin letters (a...z) are converted to uppercase.

The following are excluded from this function:

- Non-Latin single byte characters
- Multibyte characters
- Any byte indicating a transition between single byte characters and multibyte characters

Teradata Character Strings and Client Physical Bytes

Problem Overview

The server character sets LATIN and UNICODE are declared within Teradata SQL as strings of characters.

For this server character set ...	The character responds to this type of code point ...
UNICODE	UCS-2
LATIN	LATIN

Within a client environment, those character data types are expected to be specified as strings of physical bytes rather than as characters.

For example, within client application programs, space is allocated in terms of bytes, the layout of the import files for data loading utilities is expressed in terms of bytes, and so on.

How Teradata Solves the Problem

To provide a reasonable bridge between characters in SQL declarations and physical bytes in client environments, clients can specify character strings in bytes even though the destination columns are declared as characters.

Whenever a client is informed about the length of the character expression (as in the HELP COLUMN statement, PclPrepInfo parcel, and so on), Teradata specifies the length in terms of physical bytes.

General Rules for Server Character Set Declarations

The rules for declaring data types are as follows:

-

Declaration of character types using these server character sets within Teradata SQL ...	Are defined in terms of ...
LATIN UNICODE GRAPHIC	characters
KANJI1 KANJI2	bytes

- The length of the imported CHARACTER data described by the USING clause is in terms of bytes (the CHARACTER data description in the USING clause is in bytes—see “USING Row Descriptor” in Chapter 3: “SQL Data Manipulation Language Statement Syntax” in *Teradata RDBMS SQL Reference, Volume 6*).
- The definition of the CHARACTER data in the following list of response and request parcels is in bytes.
 - PclDataInfo
 - PclSessInfo
 - PclPrepInfo
- Client Load (FastLoad and MultiLoad) and Export (FastExport) utilities use the MultiLoad and FastExport DEFINE or LAYOUT command to specify the layout of the file on the client.

The layout of the CHARACTER data within the DEFINE command is in terms of bytes.

The layout of the GRAPHIC data within the DEFINE command is in terms of characters. This layout is then translated into the PclDataInfo parcel or the USING clause for communication with Teradata SQL (see “USING Row Descriptor” in Chapter 3: “SQL Data Manipulation Language Statement Syntax” in *Teradata RDBMS SQL Reference, Volume 6*).

Exported Character Data

Export width determines the appropriate number of bytes to export for a client application. Three system-wide export width tables define the export width of characters during export to a client. The values used for the export width are set based on the active export width table, and the server character set and client character set used by the application.

At the system level, DBSControl field 23, the Export Width Table ID field, identifies which export width table to use to determine the export width.

The following table describes the number of bytes exported from the various server character sets to the various client character sets for the three export width tables (identified by their Export-IDs).

IF the client character set is...	AND the server character set is...	AND the Export-ID is...	THEN the number of bytes exported for a CHARACTER(n) column is...
any single-byte character set	Latin	0, 1, or 2	n
	Unicode	0, 1, or 2	n
	KanjiSJIS	0, 1, or 2	n

Teradata Character Strings and Client Physical Bytes

IF the client character set is...	AND the server character set is...	AND the Export-ID is...	THEN the number of bytes exported for a CHARACTER(n) column is...
KanjiEUC	Latin	0 or 1	n
		2	3n
	Unicode	0	2n
		1	n
		2	3n
	KanjiSJIS	0, 1, or 2	n
	Graphic	0 or 1	2n
		2	3n
KanjiEBCDIC	Latin	0, 1, or 2	n
	Unicode	0	2n+2
		1	n
		2	3n+1
	KanjiSJIS	0 or 1	n
		2	2n
	Graphic	0, 1, or 2	• 2n (Record and Indicator modes) • 2n+2 (Field Mode)
KanjiSJIS	Latin	0 or 1	n
		2	2n
	Unicode	0 or 2	2n
		1	n
	KanjiSJIS	0, 1, or 2	n
	Graphic	0, 1, or 2	2n
UTF8	Latin	0 or 1	2n
		2	3n
	Unicode	0, 1, or 2	3n
	KanjiSJIS	0 or 1	n
		2	3n
	Graphic	0, 1, or 2	3n

HELP Statements

HELP TABLE and HELP COLUMN statement results are determined by the preceding export width rules.

CHARACTER SET Clause

Using CHARACTER SET to Define Internal Storage of a Column

You can define the server character set for a character column by using the CHARACTER SET clause of a character column type definition in the CREATE TABLE statement as defined by the ANSI SQL-92 standard.

This optional clause defines the internal handling of character data stored on the Teradata server and has nothing to do with the client, or session, character set. See [“Client Character Set” on page 5-23](#).

If you omit the CHARACTER SET clause, then the default server character set for the column depends on how the user accessing the data in the table is defined in the DEFAULT CHARACTER SET clause of the CREATE USER statement (see Chapter 1: “SQL Data Definition Language Statement Syntax” in *Teradata RDBMS SQL Reference, Volume 4*).

Supported Server Character Sets for CHARACTER SET

Teradata supports five server character sets for use in the CHARACTER SET clause of a CHARACTER data type definition.

The CHARACTER SET clause is entry-level ANSI-compliant.

Implementation-defined character sets are intermediate-level ANSI-compliant.

Each server character set is discussed in detail in:

- [“LATIN Server Character Set” on page 5-29](#)
- [“UNICODE Server Character Set” on page 5-30](#)
- [“GRAPHIC Server Character Set” on page 5-31](#)
- [“KANJI SJIS Server Character Set” on page 5-32](#)
- [“KANJI1 Server Character Set” on page 5-33](#)

LATIN Server Character Set

Intended Use

U.S. and European applications using only those characters from ASCII, ISO 8859 Latin1, or proposed ISO 8859 Latin9 repertoires.

Pad Character

SPACE (0x20)

N-Character SQL Declaration

CHARACTER(*n*) CHARACTER SET LATIN

Maximum Value for *n*

64000

Notes

The semantics of the 8-bit LATIN server character set determine what characters are considered letters for the purpose of uppercasing, and how to translate between server character sets. RDBMS storage space for this type is allocated on a character basis.

UNICODE Server Character Set

Intended Use

International applications using the UNICODE repertoire or a subset thereof.

Pad Character

SPACE (U+0020)

N-Character SQL Declaration

CHARACTER(*n*) CHARACTER SET UNICODE

Maximum Value for *n*

32000

This also is the size of LONG VARCHAR CHARACTER SET UNICODE.

Notes

The domain of the 16-bit UNICODE server character set repertoire is the domain of ISO 10646. The notion of character follows ISO 10646 Level 1.

Each code point represents a distinct character, including non-spacing characters such as diacritical marks and joiners. All characters named as letters in UNICODE are considered as such, and are candidates for uppercasing.

Undefined code points in the standard (including surrogates) may not be stored as UNICODE character data.

RDBMS storage space for UNICODE is allocated on a character basis.

GRAPHIC Server Character Set

Intended Use

DB2 compatibility.

Pad Character

IDEOGRAPHIC SPACE (U+3000)

N-Character SQL Declaration

CHARACTER(*n*) CHARACTER SET GRAPHIC

or

GRAPHIC(*n*)

Maximum Value for *n*

32000

Notes

GRAPHIC data can also be defined as GRAPHIC(*n*) or VARGRAPHIC(*n*).

All characters named as letters in this subset of UNICODE are considered as such, and are candidates for uppercasing.

KANJIIS Server Character Set

Intended Use

Japanese applications that rely on the KANJIIS characteristics (for example, KANJIIS physical space allocation).

Pad Character

ASCII SPACE (0x20)

N-Character SQL Declaration

CHARACTER(*n*) CHARACTER SET KANJIIS

Maximum Value for *n*

32000

This also is the size of LONG VARCHAR CHARACTER SET KANJIIS.

Notes

The KANJIIS type has a mixed single byte/multibyte character form-of-use of the KANJIIS client character set.

The RDBMS storage space allocation for this type is on a byte basis rather than character basis. When updating the field, truncation occurs based on this physical space.

All SQL string functions on this type operate on a logical character basis.

This type is included for third party tools that rely on the semantics of KANJIIS, such as the string length or physical space allocation.

KANJI1 Server Character Set

Intended Use

Japanese applications that must remain compatible with previous Teradata Kanji releases.

Caution: This type will not support future character sets, and new string functions cannot be used with this type. Because of these limitations, you should limit use of this type, and you should convert its data to the UNICODE type. A set of SQL translation functions is provided for that purpose.

This character set will be removed in a subsequent release.

Pad Character

ASCII SPACE (0x20)

N-Byte SQL Declaration

CHARACTER(n) CHARACTER SET KANJI1

Maximum Value for *n*

64000

This also is the size of LONG VARCHAR CHARACTER SET KANJI1.

Notes

KANJI1 contains mixed single- and multibyte characters in KanjiEBCDIC, KANJISJIS, or KanjiEUC as determined by the current character set for the session.

KANJI1 inherits the semantics and limitations on CHARACTER type of prior releases. For example, it is possible to generate non-valid strings with the *substr* function.

Note the following peculiarity of KANJI1:

- The single byte character portion of the type is stored in the canonical form defined by JIS X 0201.
- The multibyte character portion of the type is stored in the client form-of-use and thus cannot be shared among heterogeneous clients.

The maximum length in bytes of the exported CHARACTER data for Kanji1 character data type is always *n* for CHARACTER(n) and VARCHARACTER(n).

On a Japanese language site, the Teradata RDBMS assumes that *all* character data is mixed single and multibyte characters. Mixed single and multibyte character data is associated with any column defined as CHAR, VARCHAR, or LONG VARCHAR.

Encoding of the KANJI1 character set is based on KanjiEBCDIC, KanjiEUC, or KanjiShift-JIS, depending on the current character set for the session.

Depending on the character set of the session, single byte and multibyte characters are distinguished as described by the following table.

Character Set	Definition
KanjiEBCDIC	Characters are assumed to be single byte until a shift-out character is encountered. Subsequent characters are assumed to be multibyte characters until a Shift-In character is encountered. If the end of the string is reached without finding Shift-In, an error condition occurs.
KanjiEUC	The first byte of a multibyte character always has the most significant bit on. Multibyte characters are two bytes except KanjiEUC cs3 characters, which require three bytes.
KanjiShift-JIS	

If the CASESPECIFIC option is defined, conversion to uppercase is not performed, and simple Latin letters (A...Z, a...z) are considered matched only if they are the same letters and the same case.

Because simple Latin letters always have the same canonical representation, the effect of converting to uppercase is the same across all the character sets supported by the Teradata RDBMS.

The effect of the UPPERCASE function on various Japanese characters is illustrated in the following table.

Character Set	Character String	Conversion Result
KanjiEBCDIC	mN< abc >b	MN< abc >B
KanjiEUC	mn a ss2B ss3 c	MN a ss2B ss3 c
KanjiShift-JIS	mn/ abc	MN/ abc

Fixed Length KanjiEBCDIC: Example

Assume that a fixed-length column is to contain the following KanjiEBCDIC data:

```
< S T R I N G > 12
```

The column definition must be at least 16 bytes (CHAR(16) CHARACTER SET KANJI1) to accommodate the internal representation of the data plus the Shift-Out and Shift-In characters, which is as follows:

```
0E 42E2 42E3 42D9 42C9 42D5 42D7 0F 31 32
```

Note that each of the three client representations of multibyte character data could require a different length for the same sequence of symbols.

Fixed Length KanjiShift-JIS: Example

The same string in KanjiShift-JIS, is as follows:

```
S T R I N G 12
```

and requires a length of only 14 bytes (CHAR(14) CHARACTER SET KANJISJIS). The internal equivalent for the KanjiShift-JIS string is as follows:

```
8272 8273 8271 8268 826D 8266 31 32
```

Character Expressions Assigned to Shorter or Longer CHAR Columns

If a character expression of some length is assigned to a CHAR column of a longer length, the field is padded with the SPACE character.

In Teradata mode, if a character expression is assigned to a CHAR column of a shorter length, the extra bytes are truncated. This may result in an improper string. You are *not* informed that this truncation has occurred.

In ANSI mode, an error occurs if a nonblank character is truncated.

Shorter strings are padded with single-byte spaces, regardless of whether the mode is Teradata or ANSI. Only truncation differs between the two modes.

Character Data Validation and Storage

Teradata extensions to JIS X 0201 are indicated in the following table.

Character	Description	Hexadecimal Value
¢	Cent sign	0x10
£	Pound Sterling currency symbol	0x11
¬	IBM logical NOT sign	0x12
\	Backslash	0x13
~	Tilde	0x14

Translation and storage of validated multibyte character data on the Teradata server depends on the character set of the current session, as explained in the following table.

For this client character set ...	Multibyte characters are translated and stored on the Teradata server ...	
KanjiEBCDIC	as received.	
KanjiShift-JIS	They are not translated and remain in the client encoding. For KanjiEBCDIC only, the Shift-Out and Shift-In characters are stored as part of the string after being translated to the same encoding (0x0E and 0x0F, respectively).	
KanjiEUC	according to the client encoding of the current session, as follows	
	For each character of this code set ...	The first byte is translated to ...
	cs0	all characters, from EUC to the internal representation (based on JIS X 0201) and stored as single byte characters.
	cs1	KanjiShift-JIS character data only (based on JIS X 0208), as illustrated in the translation map in Chapter 4 – “Japanese Language Support” and in Appendix E – “Japanese Character Sets” in <i>Teradata RDBMS SQL Reference, Volume 1</i>. GRAPHIC data is stored without translation. Subsequent characters are also translated to KanjiShift-JIS.
	cs2	character data only. The first byte (ss2=0x8E) is translated to 0x80 and the second byte is left unmodified.
	cs3	character data only. The first byte (ss3=0x8F) is translated to 0xFF and the remaining 2 bytes are left unmodified. GRAPHIC data is stored without translation.

Chapter 6:

Byte Data Types

This chapter describes:

- BYTE data type
- VARBYTE data type

BYTE and VARBYTE Data

Introduction

The BYTE and VARBYTE data types are stored in the client system format—they are never translated by the Teradata RDBMS.

The BYTE and VARBYTE data types store raw data as logical bit streams. For any machine, BYTE and VARBYTE data is transmitted directly from the memory of the client system. The sort order is logical, and values are compared as if they were *n*-byte, unsigned binary integers suitable for digitized images.

Logical bit streams are not translated. However, the interpretation of a hexadecimal string in EBCDIC form depends on the client system from which that string is submitted.

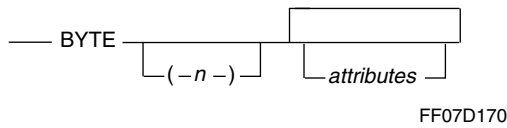
Both are Teradata extensions to the ANSI standard.

BYTE

Purpose

Represents a fixed-length binary string.

Syntax



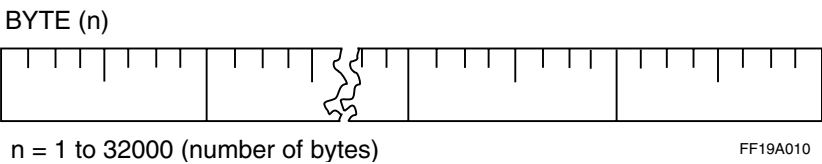
where:

Syntax element ...	Specifies ...
<i>n</i>	the number of bytes in the string.
<i>attributes</i>	appropriate data type, column storage, or column constraint attributes. See “Core Data Type Attributes” on page 1-8 and “Additional Data Type Attributes” on page 1-9 for specific information.

ANSI Compliance

BYTE is a Teradata extension to the ANSI SQL-99 standard.

Scale



External Representation of BYTE

The following table lists the client representations for the Teradata BYTE data type.

Determining the application definitions and client data types is the responsibility of the application programmer.

Client CPU Architecture	Client Internal Data Format
IBM mainframe	<i>n</i> EBCDIC bytes.
• UTS • RISC • Motorola 68000 • WE 32000 • Intel	<i>n</i> ASCII bytes.

Usage Notes

If no value is specified for *n*, the default is BYTE(1).

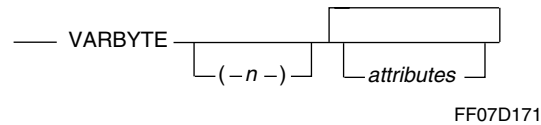
The maximum length for *n* is 64000.

VARBYTE

Purpose

Represents a variable-length binary string.

Syntax



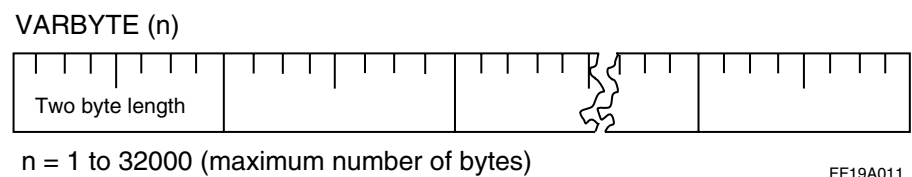
where:

Syntax element ...	Specifies ...
<i>n</i>	the number of bytes in the string.
<i>attributes</i>	appropriate data type, column storage, or column constraint attributes. See “Core Data Type Attributes” on page 1-8 and “Additional Data Type Attributes” on page 1-9 for specific information.

ANSI Compliance

VARBYTE is a Teradata extension to the ANSI SQL-99 standard.

Scale



External Representation of VARBYTE

The following table lists the client representations for the Teradata VARBYTE data type.

Define the length of the VARBYTE string as k , where $0 \leq k \leq n$.

Client CPU Architecture	Client Internal Data Format
IBM mainframe	$k + 2$ bytes 16-bit SMALLINT (2 byte) length count k followed by k bytes of EBCDIC BYTE data.
• UTS • RISC • Motorola 68000 • WE 32000 • Intel	$k + 2$ bytes 16-bit SMALLINT (2 byte) length count k followed by k bytes of ASCII BYTE data.

Usage Note

The maximum length for n is 64000.

Example

In the following example, the column named BinaryData is assigned the VARBYTE data type and has an upper limit of 12 bytes:

```
BinaryData VARBYTE(12),
```


Chapter 7:

Default Value Control Phrases

This chapter describes the following topics:

- Rules and guidelines for using default value control phrases
- COMPRESS phrase
- NOT NULL phrase
- DEFAULT phrase
- WITH DEFAULT phrase

Using Default Value Control Phrases

Rules and Guidelines

A default value control phrase determines the action to take when you do not supply a value for a field. The following rules and guidelines apply to default value control phrases:

- Default value control phrases are only valid with the following data definitions.
 - Fields defined in CREATE TABLE and ALTER TABLE statements
 - Parameters defined in CREATE MACRO and REPLACE MACRO statements
- Default value controls are *not* effective for views and expressions.
- The default value for a field is null unless you specify NOT NULL.
- The presence of nulls can affect query performance negatively; therefore, consider using the NOT NULL default value phrase unless you intend to have nulls in the column.

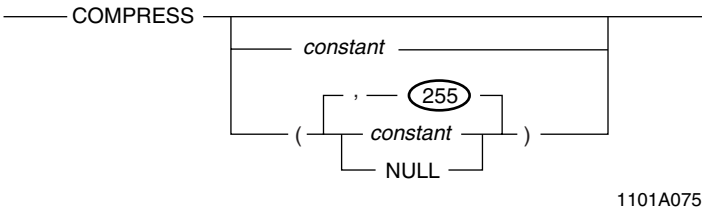
The default control value phrases are documented individually in the remaining topics in this chapter.

COMPRESS

Purpose

Compresses specified values and nulls in one or more columns of a table to zero space. When the data in a column matches a value specified in the COMPRESS phrase, then that value is stored only once in the table header regardless of how many times it occurs as a field value for the column, thus saving disk storage space.

Syntax



where:

Syntax element ...	Specifies ...
no argument	to compress nulls only.
<i>constant</i>	a value set to be compressed. You cannot specify COMPRESS for any column that is a member of the primary index column set for a table. You can specify a single <i>constant</i> value, or a multivalued, comma-separated list of up to 255 distinct <i>constant</i> values enclosed in parentheses. You can specify COMPRESS for the following data types: • BYTE • CHARACTER • DATE To compress a DATE value, you must specify the value as a string preceded by the word DATE and enclosed in parentheses. For example: COMPRESS (DATE '2000-06-15'). Otherwise, you cannot compress DATE values. • Any numeric data type.

Syntax element ...	Specifies ...
<i>constant</i> (continued)	You cannot specify COMPRESS for the following data types: • DATE (unless it is formatted as (DATE '<i>date_value</i>') • INTERVAL • TIME • TIMESTAMP • VARBYTE • VARCHAR You can compress fixed-length CHARACTER columns of up to 255 characters. You can compress fixed-length BYTE columns of up to 255 bytes.
NULL	that nulls are to be compressed. NULL can be specified alone, but it must be enclosed in parentheses. This is the same as specifying COMPRESS without an argument. NULL can be specified with up to 255 <i>constant</i> values in a multivalued, comma-separated list enclosed in parentheses.

ANSI Compliance

COMPRESS is a Teradata extension to the ANSI SQL-99 standard.

No Limit To The Number of Columns That Can Be Compressed

As many columns as is practical can be compressed for any table.

IF you specify COMPRESS ...	THEN ...
• with no argument • with NULL enclosed in parentheses	nulls are compressed for nullable columns. NULL cannot be specified in a COMPRESS phrase for a column that is declared NOT NULL.
<i>constant</i>	the indicated value is compressed. For nullable columns, nulls are also compressed.
• as a multivalued, comma-separated list of <i>constant</i> values enclosed in parentheses • as a multivalued comma-separated list of <i>constant</i> values and the NULL phrase enclosed in parentheses	nulls and the specified distinct values are compressed for nullable columns. Nulls are compressed for nullable columns regardless of whether NULL appears in the list. NULL cannot appear in a COMPRESS phrase for a column that is declared NOT NULL.

Using COMPRESS with CHARACTER Data Type

If you specify *constant* for a CHARACTER data type without the UPPERCASE option, a data value must match the case of *constant* to be compressed.

For example, if a column named “Doc_Type” is defined as follows:

```
Doc_Type CHARACTER(6) COMPRESS 'Manual'
```

then each value must be entered as 'Manual' in order to be compressed. A value in a different case, such as 'MANUAL' or 'manual,' is not compressed.

COMPRESS Cannot Be Specified for Identity Columns

You cannot specify the COMPRESS attribute for identity columns.

Achieving Maximum Benefit With COMPRESS

Maximum benefit is achieved when the COMPRESS phrase is applied to a column under the following conditions:

- Enough rows contain a compressible value (null, zero, blank, or constant value) in the compressed field to exceed the break even point. For details, see "Guideline for Break Even Point on Compression" in Chapter 12: "Database-Level Considerations" in *Teradata RDBMS Database Design*.
- The shortened row length results in the elimination of one or more data blocks.

Limits for the Value of *constant* in Column Compression

The compressible value of a fixed-length CHARACTER column can be up to 255 characters. The compressible value of a fixed-length BYTE column can be up to 255 bytes.

Additionally, the value to be compressed must be in the repertoire of the character set defined for the column and session.

Limits for a List of Distinct Values in Column Compression

The following limitations apply to a COMPRESS phrase that specifies a comma-separated list of *constant* values:

- The maximum number of distinct *constant* values in the list is 255 plus nulls.
- Values can appear in any order.
- NULL can appear at any point in the multivalued constant list.
- The value list cannot contain duplicate values.
- NULL cannot appear in the list if the column is declared NOT NULL.

Example: *constant* DATE Data Type

The following example shows a DATE column specified with a COMPRESS phrase:

```
CREATE TABLE Duration
(DurationID INTEGER,
 Start      DATE COMPRESS (DATE '2000-01-01'));
```

Example: Multivalued *constant* DATE Data Type

The following example shows a DATE column specified with a multivalued COMPRESS phrase:

```
CREATE TABLE Duration
(DurationID INTEGER,
 Start      DATE COMPRESS ((DATE '2000-01-01'),
                           (DATE '2000-01-02'),
                           (DATE '2000-01-03'))
);
```

Example: *constant* Numeric Data Type

The following example shows a numeric column specified with a COMPRESS phrase:

```
CREATE TABLE ID
(IDNum INTEGER,
 Post  INTEGER COMPRESS (44, 45, 63));
```

For Further Information

For information on ...	See ...
the performance and storage capacity savings benefits of value compression	the chapter "Database-Level Considerations" in <i>Teradata RDBMS Database Design</i> .
using COMPRESS when defining or modifying columns in a table	"CREATE TABLE" and "ALTER TABLE" in <i>Teradata RDBMS SQL Reference, Volume 4</i> .

NOT NULL

Purpose

Specifies that the fields of a column must contain a value; they cannot be null.

Syntax

—— NOT NULL ——

FF07D148

ANSI Compliance

NOT NULL is ANSI SQL-99-compliant.

Usage Notes

The NOT NULL specification ensures that every row in the example Employee table contains a value in the Name field. An error is returned if you attempt to insert row data without a name, or specify NULL in place of a name.

If a NOT NULL phrase is specified for a column in an ALTER TABLE statement, a DEFAULT or a WITH DEFAULT phrase must also be specified unless the table is empty. This ensures that any empty field in that column will be supplied with default values in compliance with the NOT NULL specification.

Example

In the following column definition, the Name column is defined as NOT NULL, thus ensuring that every row in the table has a value for Name:

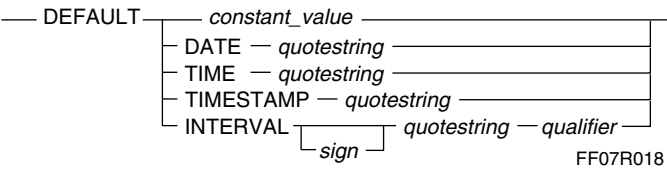
```
Name VARCHAR(12) NOT NULL,
```

DEFAULT

Purpose

Specifies that a user-defined default value is to be inserted in the field when a value is not specified for a column in an INSERT statement.

Syntax



where:

Syntax element ...	Specifies ...														
constant_value	a default value to be inserted when a column in the target table is omitted from the column_list specification of an INSERT statement. When no DEFAULT phrase is provided, a NULL is inserted into the field. A default value can be a keyword or built-in function. See Chapter 12: “Built-In Functions” in <i>Teradata RDBMS SQL Reference, Volume 5</i>. The following table defines the valid keywords and functions. <table><tr><th>This value ...</th><th>Signifies to insert this when no value is specified in an INSERT statement ...</th></tr><tr><td>number</td><td>a numeric literal value.</td></tr><tr><td>NULL</td><td>a null.</td></tr><tr><td>USER</td><td>the user name of the current user.</td></tr><tr><td>CURRENT_DATE</td><td>the current system date.</td></tr><tr><td>CURRENT_TIME</td><td>the current system time.</td></tr><tr><td>CURRENT_TIMESTAMP</td><td>the current system date and time.</td></tr></table>	This value ...	Signifies to insert this when no value is specified in an INSERT statement ...	number	a numeric literal value.	NULL	a null.	USER	the user name of the current user.	CURRENT_DATE	the current system date.	CURRENT_TIME	the current system time.	CURRENT_TIMESTAMP	the current system date and time.
This value ...	Signifies to insert this when no value is specified in an INSERT statement ...														
number	a numeric literal value.														
NULL	a null.														
USER	the user name of the current user.														
CURRENT_DATE	the current system date.														
CURRENT_TIME	the current system time.														
CURRENT_TIMESTAMP	the current system date and time.														
DATE	a date value in ANSI DATE literal format as the insert default.														
TIME	a time value in ANSI TIME literal format as the insert default.														
TIMESTAMP	a timestamp value in ANSI TIMESTAMP literal format as the insert default.														
INTERVAL	an interval value in ANSI INTERVAL format as the insert default.														

Syntax element ...	Specifies ...
<i>quotedstring</i>	the literal used as the insert default for the specified column.
<i>sign</i>	the sign of an interval value.
<i>qualifier</i>	the interval time period: • SECOND • MINUTE • MINUTE TO SECOND • HOUR • HOUR TO MINUTE • HOUR TO SECOND • DAY • DAY TO HOUR • DAY TO MINUTE • DAY TO SECOND • MONTH • YEAR • YEAR TO MONTH

ANSI Compliance

DEFAULT is ANSI SQL-99-compliant.

Also see the non-ANSI [“WITH DEFAULT” on page 7-13](#).

DEFAULT and INSERT

The only DML statement that can use a DEFAULT phrase is INSERT (see “INSERT” in Chapter 3: “SQL Data Manipulation Language Statement Syntax” in *Teradata RDBMS SQL Reference, Volume 6*).

If you attempt to insert a row expression of the form DEFAULT VALUES, an error is returned if any column in the table is not defined with a DEFAULT value.

If you attempt to insert an explicit *column_list* in which a value is omitted, an error is returned if no DEFAULT is defined for the omitted column.

Data Types Must Match DEFAULT Values

A default value must be compatible with the data type specified for the column it is inserted into. For example, the phrase INTEGER DEFAULT 3.5 is not valid and returns an error.

The default value (excepting keywords) for character columns must be in the repertoire of the character data type defined for the column and session character set.

DEFAULT Values Cannot Violate CHECK Constraints

A default value must not violate any CHECK constraints specified for the column. If a default value would violate a CHECK constraint, the conflict is not recognized at the time the table is defined or altered. The conflict is recognized the first time that an INSERT or UPDATE attempts to enter a NULL, which would be replaced by the default.

DEFAULT Values Cannot Be Specified for Identity Columns

You cannot specify the DEFAULT attribute for Identity columns.

DEFAULT Values Can Be Built-In Functions

A keyword used as a *constant_value* inserts a string that is already known to the system, such as the system date, the system time, or the name of the user defining the column.

Use of the Teradata built-in functions DATE and TIME as default values is deprecated. They are non-ANSI standard and continue in the language only to maintain backward compatibility. Use the ANSI-standard CURRENT_DATE and CURRENT_TIME instead.

Maximum Length for a DEFAULT Value

The maximum length of *constant_value* is 255 characters for character columns, and 255 bytes for byte columns.

In an ALTER TABLE statement, the DEFAULT phrase can be used with a keyword to override a previously-defined value. The column data type must agree with that of the inserted string.

System Values for DEFAULT Phrase

Unlike the Teradata WITH DEFAULT phrase, there are no system default values for the DEFAULT phrase.

Values for DEFAULT Phrase Keywords

The following table lists the DEFAULT phrase forms, the function performed by each variable or keyword, and the corresponding default data types.

This form ...	Inserts ...	Using this data type ...
DEFAULT <i>constant_value</i>	the value defined as the default for the column	the type defined for the table column.
DEFAULT DATE <i>quotestring</i>	the date value specified by <i>quotestring</i> as the default for the column	DATE.
DEFAULT NULL	a NULL	None.
DEFAULT CURRENT_DATE	the current system date	DATE.
DEFAULT CURRENT_TIME	the current system time	TIME.
DEFAULT CURRENT_TIMESTAMP	the current system date and time	TIMESTAMP.
DEFAULT USER Teradata extension to the ANSI standard	the user name of the current user	CHAR(n) CHARACTER SET UNICODE VARCHAR(n) CHARACTER SET UNICODE where <i>n</i> is the length of the longest permissible user name.

If DEFAULT NULL is specified with the NOT NULL phrase in CREATE TABLE or ALTER TABLE ADD statements, no error or warning messages are returned.

You can assign a literal value, such as a blank, as the DEFAULT for a column as follows:

```
CHAR(1) DEFAULT ' ' ...
```

Example: Numeric

Suppose the DeptNo column is defined as follows.

```
DeptNo SMALLINT DEFAULT 100 FORMAT '999' BETWEEN 100 AND 900,
```

When the value for the DeptNo field is not provided in an INSERT statement, then the value 100 with type SMALLINT is inserted automatically.

Example: Date Literal

The following example shows a DATE column specified with a DEFAULT date literal:

```
F4 DATE DEFAULT DATE '2000-01-01'
```

Example: Current Time

The following example shows a TIME column specified with a DEFAULT of the current time:

```
Stage TIME(3) DEFAULT CURRENT_TIME(3)
```

Example: Interval

The following example shows an INTERVAL column specified with a DEFAULT interval:

```
Scale INTERVAL YEAR(2) DEFAULT INTERVAL -'10' YEAR
```

WITH DEFAULT

Purpose

Specifies that a user- or system-defined default value is to be inserted in the field when a value is not specified for a column in an INSERT statement.

Syntax

—— WITH DEFAULT ——
FF07D149

ANSI Compliance

WITH DEFAULT is a Teradata extension to the ANSI SQL-99 standard.

Usage Notes

The ANSI form of this phrase is documented in [“DEFAULT” on page 7-8](#).

If you use a WITH DEFAULT phrase with a numeric range constraint, the range must include the system default value assigned to the data type defined for the column.

When a CREATE TABLE statement is processed, all WITH DEFAULT phrases are converted to DEFAULT phrases in which the system default value becomes the default *constant_value*.

Incompatibility Note

The WITH DEFAULT phrase is mutually exclusive with the DEFAULT phrase.

System Values for WITH DEFAULT Phrase

The value of a system default is determined by the data type defined for the column. The data types and associated system default values appear in the following table.

Data Type	System Default
BYTE[(n)]	Zero if <i>n</i> omitted, or <i>n</i> binary zeros
BYTEINT	Zero

Data Type	System Default												
CHAR[(n)]	Depends on the server data type as shown in the following table: <table> <tr> <th>Character Set</th><th>System Default</th></tr> <tr> <td>Latin</td><td>ASCII SPACE (0x20)</td></tr> <tr> <td>Unicode</td><td>SPACE (U+0020)</td></tr> <tr> <td>KanjiSJIS</td><td>ASCII SPACE (0x20)</td></tr> <tr> <td>Kanji1</td><td>ASCII SPACE (0x20)</td></tr> <tr> <td>Graphic</td><td>IDEOGRAPHIC SPACE (U+3000)</td></tr> </table>	Character Set	System Default	Latin	ASCII SPACE (0x20)	Unicode	SPACE (U+0020)	KanjiSJIS	ASCII SPACE (0x20)	Kanji1	ASCII SPACE (0x20)	Graphic	IDEOGRAPHIC SPACE (U+3000)
Character Set	System Default												
Latin	ASCII SPACE (0x20)												
Unicode	SPACE (U+0020)												
KanjiSJIS	ASCII SPACE (0x20)												
Kanji1	ASCII SPACE (0x20)												
Graphic	IDEOGRAPHIC SPACE (U+3000)												
DATE	Current date												
DECIMAL NUMERIC	Zero												
DOUBLE PRECISION	Zero												
FLOAT	Zero												
INTEGER	Zero												
INTERVAL DAY	INTERVAL '0' DAY												
INTERVAL DAY TO HOUR	INTERVAL '0 00' DAY TO HOUR												
INTERVAL DAY TO MINUTE	INTERVAL '0 00:00' DAY TO MINUTE												
INTERVAL DAY TO SECOND	INTERVAL '0 00:00:00' DAY TO SECOND												
INTERVAL HOUR	INTERVAL '0' HOUR												
INTERVAL HOUR TO MINUTE	INTERVAL '00:00' HOUR TO MINUTE												
INTERVAL HOUR TO SECOND	INTERVAL '00:00:00' HOUR TO SECOND												
INTERVAL MINUTE	INTERVAL '0' MINUTE												
INTERVAL MINUTE TO SECOND	INTERVAL '00:00' MINUTE TO SECOND												
INTERVAL MONTH	INTERVAL '0' MONTH												
INTERVAL SECOND	INTERVAL '0' SECOND												
INTERVAL YEAR	INTERVAL '0' YEAR												
INTERVAL YEAR TO MONTH	INTERVAL '0-00' YEAR TO MONTH												
REAL	Zero												
SMALLINT	Zero												
TIME	CURRENT_TIME												

Data Type	System Default
TIMESTAMP	CURRENT_TIMESTAMP
TIMESTAMP WITH TIME ZONE	CURRENT_TIMESTAMP
TIME WITH TIME ZONE	CURRENT_TIME
VARBYTE (n)	'' (null string)

Example 1

If the EdLev column is defined as:

```
EdLev BYTEINT FORMAT 'Z9' NOT NULL WITH DEFAULT,
```

then the column definition for EdLev is stored as:

```
EdLev BYTEINT NOT NULL DEFAULT 0 FORMAT 'Z9',
```

Example 2

The following INSERT statement adds a row containing a zero in the EdLev field defined in Example 1:

```
INSERT INTO Employee
(Name, EmpNo, DeptNo, DOB, Sex, EdLev)
VALUES ('Newhire A', 10025, '49/10/17', 'M', ) ;
```

Example 3

The following example shows a TIME column specified with a default of the current time:

```
Stage TIME(3) WITH DEFAULT
```

Example 4

The following example shows an INTERVAL column specified with a default interval:

```
Scale INTERVAL HOUR(2) WITH DEFAULT
```




Chapter 8:

Output Format Phrases

This chapter describes the Teradata SQL output format phrases.

Topics include:

- Default data display formats
- FORMAT phrase
 - and CHARACTER formats
 - and NUMERIC formats
 - and DATE formats
 - and DATE formats for Japanese-language sites
 - and TIME formats
- Naming columns and expressions
 - AS phrase
 - NAMED phrase
 - TITLE phrase

Data Display Default Formats

Introduction

Teradata SQL uses a default set of formats to display expressions, column data, and conversions between data types.

You can override the default display format by using the FORMAT output format phrase in a CREATE TABLE, ALTER TABLE, or SELECT statement. See [“FORMAT” on page 8-6](#) for details.

You can also change the default display formats that Teradata SQL uses to display numeric, date, and time data types. See [“Changing the Default Display Formats” on page 8-4](#) for details.

Data Type Format Defaults

The following table lists the default display formats that Teradata SQL uses for each data type. For explanations of the formatting characters, see:

- [“FORMAT Phrase and CHARACTER Formats” on page 8-10](#)
- [“FORMAT Phrase and NUMERIC Formats” on page 8-12](#)
- [“FORMAT Phrase and DateTime Formats” on page 8-29](#)

Data Type	Default Format
BYTEINT	-(3)9
SMALLINT	-(5)9
INTEGER	-(10)9
DECIMAL[(n[,m])] NUMERIC[(n[,m])]	--(I).9(F), where $I = n - m$ and $F = m$.
FLOAT REAL DOUBLE PRECISION	-9.999999999999999E-999
CHARACTER[(n)] VARCHAR(n)	X(n)
LONG VARCHAR	X(64000)
BYTE[(n)] VARBYTE(n)	X(2n)

Data Type	Default Format
DATE	In ANSIDATE mode: YYYY-MM-DD. In INTEGERDATE mode: YY/MM/DD, or, for systems that migrate from pre-V2R5 releases, the format derived from the pre-V2R5 setting of the DefaultDateFormat field in the DBSControl record.
TIME TIME WITH TIME ZONE	HH:MI:SS.S(F)Z Time zone information is displayed for TIME WITH TIME ZONE, but not for TIME.
TIMESTAMP TIMESTAMP WITH TIME ZONE	YYYY-MM-DDBBHH:MI:SS.S(F)Z Time zone information is displayed for TIMESTAMP WITH TIME ZONE, but not for TIMESTAMP.
INTERVAL YEAR INTERVAL MONTH INTERVAL DAY INTERVAL HOUR INTERVAL MINUTE	$X(1+n)$, where n is the value of the optional defined precision.
INTERVAL YEAR TO MONTH INTERVAL DAY TO HOUR INTERVAL HOUR TO MINUTE	$X(4+n)$, where n is the value of the optional defined precision.
INTERVAL DAY TO MINUTE	$X(7+n)$, where n is the value of the optional defined precision.
INTERVAL DAY TO SECOND	$X(10+n)$ where n is value of the optional defined precision and there is no defined fractional precision. $X(11+n+m)$ for fractional precision, where n is the value of the optional defined precision and m is the value of the optional defined fractional precision.
INTERVAL HOUR TO SECOND	$X(7+n)$, where n is the value of the optional defined precision and there is no defined fractional precision. $X(8+n+m)$ for fractional precision, where n is the value of the optional defined precision and m is the value of the optional defined fractional precision.

Data Type	Default Format
INTERVAL MINUTE TO SECOND	$X(4+n)$, where n is the value of the optional defined precision and there is no defined fractional precision. $X(5+n+m)$ for fractional precision, where n is the value of the optional defined precision and m is the value of the optional defined fractional precision.
INTERVAL SECOND	$X(1+n)$, where n is the value of the optional defined precision and there is no defined fractional precision. $X(2+n+m)$ for fractional precision, where n is the value of the optional defined precision and m is the value of the optional defined fractional precision.

Changing the Default Display Formats

You can change the default display formats that Teradata SQL uses for the following data types:

- BYTEINT
- SMALLINT
- INTEGER
- NUMERIC and DECIMAL
- REAL, DOUBLE PRECISION, and FLOAT
- DATE
- TIME and TIME WITH TIME ZONE
- TIMESTAMP and TIMESTAMP WITH TIME ZONE

To change the data type default formats, you must create a specification for data formatting (SDF) file, and then use the `tdlocaledef` utility. For details, see *Teradata RDBMS Utilities*.

To view the settings of the preceding data type default formats for the current session, use the `HELP SESSION` statement. For more information, see “SQL Help and Database Object Definition Tools: HELP and SHOW” in *Teradata RDBMS SQL Reference, Volume 4*.

Output Format Phrases

Introduction

The output format phrases control how data appears on output, whether to an online user or to a printed report. These output formatting phrases have *no* effect on how data is input or stored.

The sections that follow describe output formatting phrases alphabetically.

FORMAT

Purpose

Controls the display of expressions and column data and conversions between data types.

Syntax

```
_____ FORMAT _____ 'format_string' _____  
FF07D250
```

where:

Syntax element ...	Specifies ...
'format_string'	a quoted, variable length string containing formatting characters that define the display format for the data type.

ANSI Compliance

FORMAT is a Teradata extension to the ANSI SQL-99 standard.

Usage Notes

- Formats can be specified for columns that have character, numeric, byte, or DateTime data types.
- FORMAT pertains to data exported in report form, as is the case in BTEQ. FORMAT does *not* control internal storage representation of data or data returned in record or indicator variable mode.
- Use the FORMAT phrase in a CREATE TABLE statement or ALTER TABLE statement to define the display format for a column. Or, use it in a retrieval statement to override the default format of a column or to define the display format of an expression. As such, it is both a Data Definition Language phrase and a Data Manipulation Language phrase.
- A FORMAT specification can contain a maximum of 30 characters.
- A FORMAT phrase can describe up to 18 digit positions (17 if the FORMAT contains an E). IEEE 64-bit floating numbers are accurate to about 15 digits.
- The output string that is produced as a result of a FORMAT phrase can have a maximum of 255 characters.

Relationship to SDF

In addition to defining the default display formats for numeric, date, time, and timestamp data types, the specification for data formatting (SDF) file defines strings that Teradata displays in place of certain formatting characters that appear in a FORMAT phrase.

For example, you can set the value of the *Currency* string in the SDF file to the currency symbol native to your locale. To include that currency symbol when you display numeric monetary information in an SQL SELECT statement, use the L formatting character in the FORMAT phrase.

For details on how to create an SDF file and convert the contents into an internal form usable by Teradata, see the description of the *tdlocaledef* utility in *Teradata RDBMS Utilities*.

Example Display Formats

Examples of character string, numeric, and date formats appear in the following table.

FORMAT	Data	Display Form
X(6)	'HELLO'	'HELLO '
XXXXXX	'HELLO'	'HELLO '
X	'HELLO'	'H '
\$\$9.99	.079	' \$0.08 '
\$\$9.99	1095	*****
ZZ,ZZ9.99	1095	' 1,095.00 '
9.99E99	1095	'1.09E03 '
999V99	123.456	'12346 '
\$(5).9(2)	1	'\$1.00 '
G--(8)D9(2)	-12345678.90	'-12 345 678,90 ' where the SDF defines the group separator as ' ' (blank) and the radix separator as ','.
Z(I)D9(F)	000000.42	' ,42 ' where the SDF defines the radix separator as ','.
YY.DDD	85.224	'85.224 '
MMMBDD,BYY	Sep 12, 85	'Sep 12, 85 '

FORMAT

FORMAT	Data	Display Form
YYYY-MM-DD	1996-02-14	'1996-02-14'
YYYYBMMMBDD	1985 Sep 12	'1985 Sep 12'
KatakanaEBCDIC with default format Note that KatakanaEBCDIC uppercases single byte data.	mN<ABC>b	MN<ABC>B

For More Information

For more information on ...	See ...
the display format of character data types	“FORMAT Phrase and CHARACTER Formats” on page 8-10
the display format of numeric data types	“FORMAT Phrase and NUMERIC Formats” on page 8-12
the display format of DateTime data types	“FORMAT Phrase and DateTime Formats” on page 8-29
the default data display formats	“Data Display Default Formats” on page 8-2
default format examples	the descriptions of each data type in this volume
using the FORMAT phrase in data conversions	“Data Type Conversions,” in <i>Teradata RDBMS SQL Reference, Volume 5</i>
using Kanji date and time markers to format DateTime data types	“FORMAT Phrase, DateTime Formats, and Japanese Character Sets” on page 8-37

Example 1

The following are simple examples of the FORMAT phrase:

```
FORMAT '9(5)', FORMAT 'Z999', FORMAT 'XX'
```

Example 2: Without FORMAT Clause

If the CREATE TABLE statement defined the format for the Salary column as 'ZZ9999.99', the result of the statement:

```
SELECT SUM(Salary) FROM Employee;
```

is:

```
SUM(Salary)
851100.00
```

Example 3: With FORMAT Clause

The result of the following statement:

```
SELECT SUM(Salary) (FORMAT '$$99,999.99')
FROM Employee;
```

is:

```
SUM(Salary)
$851,100.00
```

FORMAT phrases, by themselves, cannot cause conversion of character to numeric data, or numeric to character data. An error message is returned if a FORMAT phrase implies data conversion.

Example 4: Using FORMAT to Change the Format of Returned Data

Reconfigure the format of data returned:

```
SELECT empno(FORMAT '99-999'), name, deptno
FROM employee;
```

The system returns:

EmpNo	Name	DeptNo
-----	-----	-----
10-021	Smith T	700
10-007	Aguilar J	600
10-018	Russell S	300
10-011	Chin M	100

Example 5: Using FORMAT to Override Default Format

Another example of overriding the column default and formatting the output:

```
SELECT salary(FORMAT '$$$,$$9.99')
FROM employee WHERE empno = 10019 ;
```

The system returns:

```
Salary
$28,600.00
```

Example 6: Using FORMAT as Part of Derived Expression

You can use FORMAT as part of a derived expression. To determine the percent increase if employee 10019 is given a \$1000 raise:

```
SELECT (1000/salary)*100 (FORMAT 'zz9%')
(TITLE 'Percent Incr')
FROM employee
WHERE empno = 10019 ;
```

```
Percent Incr
-----
          3%
```

FORMAT Phrase and CHARACTER Formats

Introduction

If a CREATE TABLE, ALTER TABLE, or SELECT statement does not include the FORMAT phrase, the default is to return the full length specified for a fixed-length CHARACTER column (that is, pad characters are returned unless TRIM is included in the SELECT statement), and to return the actual character string for a VARCHAR or LONG VARCHAR column.

Character strings are padded or truncated on the right when they are shorter or longer than the positions defined in the FORMAT phrase.

Formatting Characters

Use the following characters in the FORMAT phrase to control formatting.

Character	Meaning
X X(<i>n</i>)	The letter X formats the display of character data. Each X represents one character. The data is returned in left-to-right order. For example, if a column is defined as FORMAT 'X', only the first (leftmost) character of each field value is displayed. Multiple character positions can be defined either by using the (<i>n</i>) notation, where <i>n</i> defines the length of the string, or by entering the equivalent number of X characters.

How FORMAT Applies to Server Character Sets

The following table describes how FORMAT 'X(*n*)' applies to server character sets.

FORMAT 'X(<i>n</i>)' returns this ...	WHEN applied to a string expression having this character data type ...
<i>n</i> logical characters	LATIN GRAPHIC UNICODE
<i>n</i> bytes	KANJI1 KANJI1SJIS

The character string in a FORMAT clause cannot be of type KANJI1. The form FORMAT _Kanji1 'hex digits'XC in particular always generates an error.

Truncation of Kanji1 Character Data Type Fields

It is possible to truncate the output of multibyte characters in an expression with Kanji1 character data type with any of the following character sets:

- KanjiEBCDIC
- KanjiEUC
- KanjiShift-JIS

In Teradata mode, the system does *not* tell you this has happened, just returning the number of bytes specified. The table below shows the hexadecimal returns. The actual return display depends on which system is used.

This character ...	With this format ...	Returns this values in hexadecimal ...	With this character set ...
B<AA>	'xx'	C20E	KanjiEBCDIC
BAA	'xx'	42A3	KanjiEUC

In ANSI mode, an error occurs if a non-blank character is truncated.

Example 1

Both of the following FORMAT phrases specify a five-position character string:

```
FORMAT 'X(5) '
FORMAT 'XXXXX'
```

Example 2

If your data is the character string 'HELLO', then:

```
FORMAT 'X(6) '
```

displays HELLO with trailing blank, and:

```
FORMAT 'X'
```

displays H.

FORMAT Phrase and NUMERIC Formats

Introduction

The formatting characters in a FORMAT phrase determine whether the output of numeric data is considered to be monetary or non-monetary. Numeric information is considered to be monetary if the FORMAT phrase contains a currency symbol.

Formatting characters are case insensitive.

The result of a formatted numeric string is right-justified.

Formatting Characters for Non-Monetary Numeric Information

Numeric information is considered to be non-monetary if the FORMAT phrase does not contain any currency or dual currency symbols. Use the following characters in the FORMAT phrase to control formatting of non-monetary numeric information:

Character	Meaning
G	Invokes the grouping rule defined by <i>GroupingRule</i> in the SDF. The value of <i>GroupSeparator</i> in the SDF is copied to the output string to separate groups of digits to the left of the radix separator, according to the grouping rule defined by <i>GroupingRule</i>. Grouping applies to the integer portion of REAL, FLOAT, or DOUBLE PRECISION data. The G must appear as the first character in a FORMAT phrase. If the FORMAT phrase does not contain the letter G, no grouping is done on the output string. The G cannot appear in a FORMAT phrase that contains any of the following characters: • , • . • / • : • S
D	Radix symbol. The value of <i>RadixSeparator</i> in the current SDF is copied to the output string whenever a D appears in the FORMAT phrase. The D cannot appear in a FORMAT phrase that contains any of the following characters: • . • , • / • : • S • V

Character	Meaning
/ : %	Insertion characters. Copied to output string where they appear in the FORMAT phrase. The % insertion character cannot appear in a FORMAT phrase that contains S, and cannot appear between digits in a FORMAT phrase that contains G, D, or E. For example, G9999D99% is valid, but 9(9)D99%E999 is not. The / and : insertion characters cannot appear in a FORMAT phrase that contains any of the following characters: • G • D • S
,	Grouping character. The comma is inserted only if a digit has already appeared. The comma is interpreted as the grouping character regardless of the value of <i>GroupSeparator</i> in the SDF. The comma cannot appear in a FORMAT phrase that contains any of the following characters: • G • D • S
.	Radix character. The period is interpreted as the radix character, regardless of the value of <i>RadixSeparator</i> in the SDF, and is copied to the output string. The period cannot appear in a FORMAT phrase that contains any of the following characters: • G • D • V • S
B	Insertion character. A blank is copied to the output string wherever a B appears in the FORMAT phrase. B cannot appear between digits in a FORMAT phrase that contains G, D, or E. For example, 999B99B9999 is valid, but G9(9)BD99 is not.

Character	Meaning
+ -	Sign characters. Can appear at the beginning or end of a format string, but cannot appear between Z or 9 characters. One sign character places the edit character in a fixed position for the output string. If two or more of these characters are present, the sign floats (moves to the position just to the left of the number as determined by the stated structure). Repeated sign characters must appear to the left of a combination of the radix and any 9 formatting characters. If a group of repeated sign characters appears in a FORMAT phrase with a group of repeated Z characters, the group of Z characters must immediately follow the group of sign characters. For example, --ZZZ. One trailing sign character can occur to the right of any digits, and can combine with B. For example, G9(I)B+. The trailing sign character for a mantissa cannot appear to the right of the exponent. For example, 999D999E+999+ is not valid. The + translates to + or - as appropriate; the - translates to - or blank.
V	Implied decimal point position. Internally, the V is recognized as a decimal point to align the numeric value properly for calculation. Because the decimal point is implied, it does not occupy any space in storage and is not included in the output. V cannot appear in a FORMAT phrase that contains the D radix symbol or the . radix character.
Z	Zero-suppressed decimal digit. Translates to blank if the digit is zero and preceding digits are also zero. A FORMAT phrase containing Z characters (including Z(I) and Z(F)), a combination of commas, dots, G, or D, and no other formatting characters means "blank when zero." For example, ZZZZZ, ZZ.Z, GZ(I)DZ(F), GZZZZZZDZZ and Z,ZZZ.ZZ print only blanks if the number is zero. A Z cannot follow a 9. Repeated Z characters must appear to the left of any combination of the radix and any 9 formatting characters. The characters to the right of the radix cannot be a combination of 9 and Z characters; they must be all 9s or all Zs. If they are all Zs, then the characters to the left of the radix must also be all Zs. If a group of repeated Z characters appears in a FORMAT phrase with a group of repeated sign characters, the group of Z characters must immediately follow the group of sign characters. For example, --ZZZ.
9	Decimal digit (no zero suppress).

Character	Meaning
<i>CHAR(n)</i>	For more than one occurrence of a character, where <i>CHAR</i> can be: • - (sign character) • + • Z • 9 and <i>n</i> can be: • an integer constant • I • F If <i>n</i> is F, <i>CHAR</i> can only be Z or 9. If <i>n</i> is an integer constant, the (<i>n</i>) notation means that the character repeats <i>n</i> number of times. For the meanings of I and F, see the definitions later in this table.
E	For exponential notation. Defines the end of the mantissa and the start of the exponent. The exponent consists of one optional + or - sign character followed by one or more 9 formatting characters.
I	The number of characters needed to display the integer portion of numeric and integer data. I can only appear as <i>n</i> in the <i>CHAR(n)</i> character sequence (see the definition of <i>CHAR(n)</i> earlier in this table), where <i>CHAR</i> can be: • - (sign character) • + • Z • 9 <i>CHAR(I)</i> can only appear once, and is valid for the following types: • DECIMAL/NUMERIC • BYTEINT • SMALLINT • INTEGER The value of I is resolved during the formatting of the numeric data. The value is obtained from the declaration of the data type. For example, I is eight for the DECIMAL(10,2) type. If <i>CHAR(F)</i> also appears in the FORMAT phrase, <i>CHAR(F)</i> must appear to the right of <i>CHAR(I)</i>, and one of the following characters must appear between <i>CHAR(I)</i> and <i>CHAR(F)</i>: • D • . • V

Character	Meaning									
F	The number of characters needed to display the fractional portion of numeric data. F can only appear as <i>n</i> in the <i>CHAR(n)</i> character sequence (see the definition of <i>CHAR(n)</i> earlier in this table), where <i>CHAR</i> can be: • Z • 9 <i>CHAR(F)</i> is valid for the DECIMAL/NUMERIC data type. The value of F is resolved during the formatting of the monetary numeric data. The value is obtained from the declaration of the data type. For example, F is two for the DECIMAL(10,2) type. A value of zero for F displays no fractional precision for the data; however, the value of <i>RadixSeparator</i> in the current SDF is copied to the output string if a D appears in the FORMAT phrase. <i>CHAR(F)</i> can appear only once. If <i>CHAR(I)</i> also appears in the FORMAT phrase, <i>CHAR(F)</i> must appear to the right of <i>CHAR(I)</i>, and one of the following characters must appear between <i>CHAR(I)</i> and <i>CHAR(F)</i>: • D • . • V									
-	Dash character. Used when displaying numbers such as telephone numbers, social security numbers, and account numbers. A dash appears after the first digit and before the last digit, and is treated as an embedded dash rather than a sign character. A dash cannot follow any of these characters: <table><tr><td>• .</td><td>• G</td><td>• S</td></tr><tr><td>• ,</td><td>• D</td><td>• V</td></tr><tr><td>• +</td><td>• E</td><td></td></tr></table>	• .	• G	• S	• ,	• D	• V	• +	• E	
• .	• G	• S								
• ,	• D	• V								
• +	• E									
時 分 秒	Kanji JL, FUN, and BYOU time markers. Each marker can appear only once. For more information on using Kanji time markers, see “FORMAT Phrase, DateTime Formats, and Japanese Character Sets” on page 8-37.									

Character	Meaning																																													
S	Signed Zoned Decimal character. Defines signed zoned decimal input as a numeric data type and displays numeric output as signed zone decimal character strings. The final character indicates the sign, as shown in the following table. <table><tr><th>IF the final data digit is ...</th><th>THEN if the result is a positive number, the final result digit is displayed as ...</th><th>ELSE if the result is a negative number, the final result digit is displayed as ...</th></tr><tr><td>0</td><td>{</td><td>}</td></tr><tr><td>1</td><td>A</td><td>J</td></tr><tr><td>2</td><td>B</td><td>K</td></tr><tr><td>3</td><td>C</td><td>L</td></tr><tr><td>4</td><td>D</td><td>M</td></tr><tr><td>5</td><td>E</td><td>N</td></tr><tr><td>6</td><td>F</td><td>O</td></tr><tr><td>7</td><td>G</td><td>P</td></tr><tr><td>8</td><td>H</td><td>Q</td></tr><tr><td>9</td><td>I</td><td>R</td></tr></table> The S must follow the last decimal digit in the FORMAT phrase. It cannot appear in the same phrase with the following characters. <table><tr><td>• %</td><td>• -</td><td>• G</td></tr><tr><td>• +</td><td>• ,</td><td>• Z</td></tr><tr><td>• :</td><td>• .</td><td>• F</td></tr><tr><td>• /</td><td>• D</td><td>• E</td></tr></table>	IF the final data digit is ...	THEN if the result is a positive number, the final result digit is displayed as ...	ELSE if the result is a negative number, the final result digit is displayed as ...	0	{	}	1	A	J	2	B	K	3	C	L	4	D	M	5	E	N	6	F	O	7	G	P	8	H	Q	9	I	R	• %	• -	• G	• +	• ,	• Z	• :	• .	• F	• /	• D	• E
IF the final data digit is ...	THEN if the result is a positive number, the final result digit is displayed as ...	ELSE if the result is a negative number, the final result digit is displayed as ...																																												
0	{	}																																												
1	A	J																																												
2	B	K																																												
3	C	L																																												
4	D	M																																												
5	E	N																																												
6	F	O																																												
7	G	P																																												
8	H	Q																																												
9	I	R																																												
• %	• -	• G																																												
• +	• ,	• Z																																												
• :	• .	• F																																												
• /	• D	• E																																												

Formatting Characters for Monetary Numeric Information

Numeric information is considered to be monetary if the FORMAT phrase contains a currency symbol. Use the following characters in the FORMAT phrase to control formatting of monetary numeric information:

Character	Meaning
G	Invokes the currency grouping rule defined by <i>CurrencyGroupingRule</i> in the SDF. The value of <i>CurrencyGroupSeparator</i> in the current SDF is copied to the output string to separate groups of digits to the left of the currency radix separator, according to the currency grouping rule defined by <i>CurrencyGroupingRule</i>. Grouping applies to the integer portion of REAL, FLOAT, or DOUBLE PRECISION data. The G must appear as the first character in a FORMAT phrase. A currency character, such as L or C, must also appear in the FORMAT phrase. If the FORMAT phrase does not contain the letter G, no grouping is done on the output string. The G cannot appear in a FORMAT phrase that contains any of the following characters: • , • . • / • : • S
/ : %	Insertion characters. Copied to output string where they appear in the FORMAT phrase. The % insertion character cannot appear in a FORMAT phrase that contains S, and cannot appear between digits in a FORMAT phrase that contains G, D, or E. For example, GL9999D99% is valid, but L9(9)D99%E999 is not. The / and : insertion characters cannot appear in a FORMAT phrase that contains any of the following characters: • G • D • S
B	Insertion character. A blank is copied to the output string wherever a B appears in the FORMAT phrase. B cannot appear between digits in a FORMAT phrase that contains G, D, or E. For example, GNB99D99 is valid, but G9(9)BD99 is not.

Character	Meaning												
+ -	Sign characters. Can appear at the beginning or end of a format string, but cannot appear between Z or 9 characters, or between repeated currency characters. One sign character places the edit character in a fixed position for the output string. If two or more of these characters are present, the sign floats (moves to the position just to the left of the number as determined by the stated structure). Repeated sign characters must appear to the left of formatting characters consisting of a combination of the radix and any 9 formatting characters. If a group of repeated sign characters appears in a FORMAT phrase with a group of repeated Z characters or a group of repeated currency characters or both, the groups must be contiguous. For example, +++\$\$\$ZZZ. One trailing sign character can occur to the right of any digits, and can combine with B and one currency character or currency sign. For example, G9(I)B+L. The trailing sign character for a mantissa cannot appear to the right of the exponent. For example, 999D999E+999+ is invalid. The + translates to + or - as appropriate; the - translates to - or blank.												
\$ £ ¥ ¤ €	Currency signs. <table border="1"> <thead> <tr> <th>Currency Sign</th><th>Meaning</th></tr> </thead> <tbody> <tr> <td>\$</td><td>Dollar sign</td></tr> <tr> <td>£</td><td>Pound sterling</td></tr> <tr> <td>¥</td><td>Yen sign</td></tr> <tr> <td>¤</td><td>General currency sign</td></tr> <tr> <td>€</td><td>Euro sign</td></tr> </tbody> </table> A currency sign cannot appear between Z or 9 formatting characters, or between repeated sign characters. One currency sign places the edit character in a fixed position for the output string. If a result is formatted using a single currency sign with Zs for zero-suppressed decimal digits (for example, £ZZ9.99), blanks can occur between the currency sign and the leftmost nonzero digit of the number. If the same currency sign appears more than once, the sign floats to the right, leaving no blanks between it and the leftmost digit.	Currency Sign	Meaning	\$	Dollar sign	£	Pound sterling	¥	Yen sign	¤	General currency sign	€	Euro sign
Currency Sign	Meaning												
\$	Dollar sign												
£	Pound sterling												
¥	Yen sign												
¤	General currency sign												
€	Euro sign												

Character	Meaning																					
\$ £ ¥ ¤ € (cont'd)	A currency sign cannot appear in the same phrase with a currency character, such as L. If + or - is present, the currency character cannot precede it. If a group of repeated currency signs appears in a FORMAT phrase with a group of repeated sign characters or a group of repeated Z characters or both, the groups must be contiguous. For example, +++\$\$\$ZZZ. One currency sign can occur to the right of any digits, and can combine with B and one trailing sign character. For example, G9(I)B+\$. A currency sign cannot appear to the right of an exponent. For example, 999D999E+999B+\$ is invalid.																					
L C N O U A	Currency characters. The value of the corresponding currency string in the current SDF is copied to the output string whenever the character appears in the FORMAT phrase. <table><tr><th>WHEN this character appears in a FORMAT phrase...</th><th>THEN it is interpreted as this symbol...</th><th>AND the value of this string in the SDF is copied to the output string...</th></tr><tr><td>L</td><td>currency</td><td><i>Currency</i></td></tr><tr><td>C</td><td>ISO currency</td><td><i>ISOCurrency</i></td></tr><tr><td>N</td><td>full currency name, such as Yen or Kroner</td><td><i>CurrencyName</i></td></tr><tr><td>O</td><td>dual currency</td><td><i>DualCurrency</i></td></tr><tr><td>U</td><td>dual ISO currency</td><td><i>DualISOCurrency</i></td></tr><tr><td>A</td><td>full dual currency name, such as Euro</td><td><i>DualCurrencyName</i></td></tr></table> A currency character cannot appear between Z or 9 formatting characters, or between repeated sign characters. If the same currency character appears more than once, the value that is copied to the output string floats to the right, leaving no blanks between it and the leftmost digit. Repeated characters must be contiguous, and must appear to the left formatting characters consisting of a combination of the radix and any 9 formatting characters. If a group of repeated currency characters appears in a FORMAT phrase with a group of repeated sign characters or a group of repeated Z characters or both, the groups must be contiguous. For example, +++LLLZZZ.	WHEN this character appears in a FORMAT phrase...	THEN it is interpreted as this symbol...	AND the value of this string in the SDF is copied to the output string...	L	currency	<i>Currency</i>	C	ISO currency	<i>ISOCurrency</i>	N	full currency name, such as Yen or Kroner	<i>CurrencyName</i>	O	dual currency	<i>DualCurrency</i>	U	dual ISO currency	<i>DualISOCurrency</i>	A	full dual currency name, such as Euro	<i>DualCurrencyName</i>
WHEN this character appears in a FORMAT phrase...	THEN it is interpreted as this symbol...	AND the value of this string in the SDF is copied to the output string...																				
L	currency	<i>Currency</i>																				
C	ISO currency	<i>ISOCurrency</i>																				
N	full currency name, such as Yen or Kroner	<i>CurrencyName</i>																				
O	dual currency	<i>DualCurrency</i>																				
U	dual ISO currency	<i>DualISOCurrency</i>																				
A	full dual currency name, such as Euro	<i>DualCurrencyName</i>																				

Character	Meaning
L C N O U A (cont'd)	A currency character cannot appear in the same phrase with any of the following characters: • other currency characters • a currency sign, such as \$ or £ • , • . One currency character can occur to the right of any digits, and can combine with B and one trailing sign character. For example, G9(I)B+L. A currency character cannot appear to the right of an exponent. For example, 999D999E+999B+L is invalid.
V	Implied decimal point position. Internally, the V is recognized as a decimal point to align the numeric value properly for calculation. Because the decimal point is implied, it does not occupy any space in storage and is not included in the output. V cannot appear in a FORMAT phrase that contains the D radix symbol or the . radix character.
Z	Zero-suppressed decimal digit. Translates to blank if the digit is zero and preceding digits are also zero. A Z cannot follow a 9. Repeated Z characters must appear to the left of any combination of the radix and any 9 formatting characters. The characters to the right of the radix cannot be a combination of 9 and Z characters; they must be all 9s or all Zs. If they are all Zs, then the characters to the left of the radix must also be all Zs. If a group of repeated Z characters appears in a FORMAT phrase with a group of repeated sign characters, the group of Z characters must immediately follow the group of sign characters. For example, --ZZZ.
9	Decimal digit (no zero suppress).
E	For exponential notation. Defines the end of the mantissa and the start of the exponent. The exponent consists of one optional + or - sign character followed by one or more 9 formatting characters.

Character	Meaning
<i>CHAR(n)</i>	For more than one occurrence of a character, where <i>CHAR</i> can be one of the following: • - (sign character) • + • Z • 9 • \$ • □ • € • ¥ • £ and <i>n</i> can be: • an integer constant • I • F If <i>n</i> is F, <i>CHAR</i> can only be Z or 9. If <i>n</i> is an integer constant, the (<i>n</i>) notation means that the character repeats <i>n</i> number of times. For the meanings of I and F, see the definitions later in this table.
,	Currency grouping character. The comma is inserted only if a digit has already appeared. The comma is interpreted as the currency grouping character regardless of the value of the <i>CurrencyGroupSeparator</i> in the SDF. The comma cannot appear in a FORMAT phrase that contains any of the following characters: • G • D • L • C • O • U • N • A • S
.	Currency radix character. The period is interpreted as the currency radix character, regardless of the value of the <i>CurrencyRadixSeparator</i> in the SDF, and is copied to the output string. The period cannot appear in a FORMAT phrase that contains any of the following characters: • G • D • L • V • C • O • U • N • A • S

Character	Meaning
D	Radix symbol. The value of <i>CurrencyRadixSeparator</i> in the current SDF is copied to the output string whenever a D appears in the FORMAT phrase. A currency symbol, such as a dollar sign or yen sign, must also appear in the FORMAT phrase. The D cannot appear in a FORMAT phrase that contains any of the following characters: • , • . • / • : • S • V
I	The number of characters needed to display the integer portion of numeric and integer data. I can only appear as <i>n</i> in the <i>CHAR(n)</i> character sequence (see the definition of <i>CHAR(n)</i> earlier in this table), where <i>CHAR</i> can be: • - (sign character) • + • Z • 9 • \$ • □ • € • ¥ • £ <i>CHAR(I)</i> can only appear once, and is valid for the following types: • DECIMAL/NUMERIC • BYTEINT • SMALLINT • INTEGER The value of I is resolved during the formatting of the monetary numeric data. The value is obtained from the declaration of the data type. For example, I is eight for the DECIMAL(10,2) type. If <i>CHAR(F)</i> also appears in the FORMAT phrase, <i>CHAR(F)</i> must appear to the right of <i>CHAR(I)</i>, and one of the following characters must appear between <i>CHAR(I)</i> and <i>CHAR(F)</i>: • D • . • V

Character	Meaning																					
F	The number of characters needed to display the fractional portion of numeric data. F can only appear as <i>n</i> in the <i>CHAR(n)</i> character sequence (see the definition of <i>CHAR(n)</i> earlier in this table), where <i>CHAR</i> can be: • Z • 9 <i>CHAR(F)</i> is valid for the DECIMAL/NUMERIC data type. The value of F is resolved during the formatting of the monetary numeric data. The value is obtained from the declaration of the data type. For example, F is two for the DECIMAL(10,2) type. A value of zero for F displays no fractional precision for the data; however, the value of <i>CurrencyRadixSeparator</i> in the current SDF is copied to the output string if a D appears in the FORMAT phrase. <i>CHAR(F)</i> can appear only once. If <i>CHAR(I)</i> also appears in the FORMAT phrase, <i>CHAR(F)</i> must appear to the right of <i>CHAR(I)</i>, and one of the following characters must appear between <i>CHAR(I)</i> and <i>CHAR(F)</i>: • D • . • V																					
-	Dash character. Used when storing numbers such as telephone numbers, social security numbers, and account numbers. A dash appears after the first digit and before the last digit, and is treated as an embedded dash rather than a sign character. A dash cannot follow any of these characters: <table><tr><td>• .</td><td>• L</td><td>• E</td></tr><tr><td>• ,</td><td>• O</td><td>• \$</td></tr><tr><td>• +</td><td>• U</td><td>• □</td></tr><tr><td>• G</td><td>• D</td><td>• £</td></tr><tr><td>• N</td><td>• V</td><td>• ¥</td></tr><tr><td>• A</td><td>• S</td><td>• €</td></tr><tr><td>• C</td><td></td><td></td></tr></table>	• .	• L	• E	• ,	• O	• \$	• +	• U	• □	• G	• D	• £	• N	• V	• ¥	• A	• S	• €	• C		
• .	• L	• E																				
• ,	• O	• \$																				
• +	• U	• □																				
• G	• D	• £																				
• N	• V	• ¥																				
• A	• S	• €																				
• C																						

Character	Meaning																																																									
S	Signed Zoned Decimal character. Defines signed zoned decimal input as a numeric data type and displays numeric output as signed zone decimal character strings. The final character indicates the sign, as shown in the following table. <table><tr><th>IF the final data digit is ...</th><th>THEN if the result is a positive number, the final result digit is displayed as ...</th><th>ELSE if the result is a negative number, the final result digit is displayed as ...</th></tr><tr><td>0</td><td>{</td><td>}</td></tr><tr><td>1</td><td>A</td><td>J</td></tr><tr><td>2</td><td>B</td><td>K</td></tr><tr><td>3</td><td>C</td><td>L</td></tr><tr><td>4</td><td>D</td><td>M</td></tr><tr><td>5</td><td>E</td><td>N</td></tr><tr><td>6</td><td>F</td><td>O</td></tr><tr><td>7</td><td>G</td><td>P</td></tr><tr><td>8</td><td>H</td><td>Q</td></tr><tr><td>9</td><td>I</td><td>R</td></tr></table> The S must follow the last decimal digit in the FORMAT phrase. It cannot appear in the same phrase with the following characters. <table><tr><td>• %</td><td>• E</td><td>• U</td></tr><tr><td>• +</td><td>• D</td><td>• O</td></tr><tr><td>• :</td><td>• G</td><td>• \$</td></tr><tr><td>• /</td><td>• F</td><td>• □</td></tr><tr><td>• -</td><td>• N</td><td>• £</td></tr><tr><td>• ,</td><td>• A</td><td>• ¥</td></tr><tr><td>• .</td><td>• C</td><td>• €</td></tr><tr><td>• Z</td><td>• L</td><td></td></tr></table>	IF the final data digit is ...	THEN if the result is a positive number, the final result digit is displayed as ...	ELSE if the result is a negative number, the final result digit is displayed as ...	0	{	}	1	A	J	2	B	K	3	C	L	4	D	M	5	E	N	6	F	O	7	G	P	8	H	Q	9	I	R	• %	• E	• U	• +	• D	• O	• :	• G	• \$	• /	• F	• □	• -	• N	• £	• ,	• A	• ¥	• .	• C	• €	• Z	• L	
IF the final data digit is ...	THEN if the result is a positive number, the final result digit is displayed as ...	ELSE if the result is a negative number, the final result digit is displayed as ...																																																								
0	{	}																																																								
1	A	J																																																								
2	B	K																																																								
3	C	L																																																								
4	D	M																																																								
5	E	N																																																								
6	F	O																																																								
7	G	P																																																								
8	H	Q																																																								
9	I	R																																																								
• %	• E	• U																																																								
• +	• D	• O																																																								
• :	• G	• \$																																																								
• /	• F	• □																																																								
• -	• N	• £																																																								
• ,	• A	• ¥																																																								
• .	• C	• €																																																								
• Z	• L																																																									

Display Limits

Up to 20 digits can be displayed. The number of digits after the decimal must *at least* equal the number by which the total digit count exceeds 18.

The following table demonstrates how this works.

IF the number of total digits is ...	THEN there must be at least this many digits after the decimal point ...
19	1
20	2

Although precision of up to two digits can be lost, this loss occurs only *after* the decimal point and never before it.

Note that the equivalent format composed of a string of 9s is *not* supported.

| Results of a FORMAT Phrase Defining Too Few Positions

A FORMAT phrase that defines fewer positions than are required by numeric values causes the data to be returned as follows:

- Asterisks appear when the integer portion cannot be accommodated.
- When only the integer portion can be accommodated, any digits to the right of the least significant digit are either truncated (for an integer value) or rounded (for a floating or decimal value).

Rounding is based on “Round to the Nearest” mode, as illustrated by the following process.

Stage	Process						
1	Let B represent the actual result.						
2	Let A and C represent the nearest bracketing values that can be represented, such that A < B < C.						
3	The determination as to whether A or C is the represented result is made as follows <table><tr><th>Stage</th><th>Process</th></tr><tr><td>1</td><td>When possible, the result is the value nearest to B.</td></tr><tr><td>2</td><td>If A and C are equidistant (for example, the fractional part is exactly .5), the result is the even number.</td></tr></table>	Stage	Process	1	When possible, the result is the value nearest to B.	2	If A and C are equidistant (for example, the fractional part is exactly .5), the result is the even number.
Stage	Process						
1	When possible, the result is the value nearest to B.						
2	If A and C are equidistant (for example, the fractional part is exactly .5), the result is the even number.						

Examples of Rounding

The following SELECT statement returns 48 and 48.

```
SELECT 47.5 (FORMAT 'zzzz'), 48.5 (FORMAT('zzzz')) ;
```

A FORMAT clause can cause the number to be rounded; for example, consider the following:

```
SELECT 1.3451 (FORMAT 'zz.z');
```

The FORMAT clause operates on 1.3451 and returns 1.3.

```
SELECT 13451 / 10000.000
```

1.345 is returned as expected.

```
SELECT 13451 / 10000.000 (FORMAT 'zz.z');
```

The arithmetic expression is evaluated first (yielding 1.345), then the FORMAT clause is applied, returning 1.3.

```
SELECT 13451 / 10000.00 (FORMAT 'zz.z');
```

Note that two 'roundings' occur; first to 1.35 when evaluating the expression, then a second rounding to 1.4 as a result of the FORMAT statement.

To avoid double 'roundings', extend the precision of the arithmetic expression two decimal places beyond that of the FORMAT clause.

Example Display Results

The display results of various FORMAT phrases for numeric data appear in the following table. For FORMAT phrases that contain G, D, L, and N formatting characters, assume that the related entries in the SDF are:

```
RadixSeparator { ",", " " }
GroupSeparator { " ", " " }
GroupingRule   { "3" }
Currency       { "$" }
CurrencyName    { "US Dollars" }
```

FORMAT Phrase	Data	Result
'\$\$9.99'	.069	\$0.07
'\$\$9.99'	1095	*****
'ZZ,ZZ9.99'	1095	1,095.00
'9.99E99'	1095	1.10E03
'999V99'	128.457	128.46
'\$(5).9(2)'	1	\$1.00
'999-9999'	8278777	827-8777
'ZZ,ZZ9.99-'	1095	1,095.00

FORMAT Phrase	Data	Result
'ZZ,ZZ9.99-'	-1095	1,095.00-
'99999S'	-1095	0109N
'99999S'	1095	0109E
'--(8)D9(2)'	0034567890	34567890,00
'G--(8)D9(2)'	-12345678.90	-12.345.678,90
'Z(I)D9(F)'	000000.42	,42
'G-(10)9'	1234567890	1.234.567.890
'-9D9999999999999999E-999'	1.74524064372835 e-2	1,74524064372835 e-002
'GLLZ(I)D9(F)'	9988.77	\$9.988,77
'-Z(I)BN'	998877.66	998878 US Dollars

If the FORMAT phrase does not include a sign character or a signed zoned decimal character, then the sign for a negative value is discarded and the output is displayed as a positive number.

FORMAT Phrase and DateTime Formats

Introduction

The date and time formatting characters in a FORMAT phrase determine the output of DATE, TIME, and TIMESTAMP information.

IF the data type is ...	THEN use formatting characters for ...
DATE	date information.
TIME TIME WITH TIME ZONE	time information.
TIMESTAMP TIMESTAMP WITH TIME ZONE	date and time information. Date formatting characters must be grouped separately from time formatting characters.

Formatting characters are case insensitive.

Formatting Characters for Date Information

Use the following characters in the FORMAT phrase to control formatting of date information in DATE and TIMESTAMP types:

Characters	Meaning
MM	Represent the month as two numeric digits.
MMM M3	Represent the month as an abbreviated month name, such as 'Apr' for April. Valid names are specified by <i>ShortMonths</i> in the current SDF. M3 is equivalent to MMM, and is preferable to allow for a shorter, unambiguous format string. You cannot specify MMM in a format that also has MM.
MMMM M4	Represent the month as a full month name, such as November. Valid names are specified by <i>LongMonths</i> in the current SDF. M4 is equivalent to MMMM, and is preferable to allow for a shorter, unambiguous format string. You cannot specify M4 in a format that also has M3 or MM.
DD	Represent the day of the month as two numeric digits.

Characters	Meaning
DDD D3	Represent the date as the sequential day in the year, using three numeric digits, such as '032' as February 1. D3 is equivalent to DDD, and allows for a shorter format string. You cannot specify DDD or D3 in a format that also has DD.
YY	Represent the year as two numeric digits.
YYYY Y4	Represent the year as four numeric digits. Y4 is equivalent to YYYY, and allows for a shorter format string. You cannot specify YYYY or Y4 in a format that also has YY.
EEE E3	Represent the day of the week as an abbreviated name, such as 'Mon' for Monday. Valid abbreviations are specified by <i>ShortDays</i> in the current SDF. E3 is equivalent to EEE, and allows for a shorter format string.
EEEE E4	Represent the day of the week using the full name, such as Thursday. Valid names are specified by <i>LongDays</i> in the current SDF. E4 is equivalent to EEEE, and allows for a shorter format string.
/	Slash separator. Copied to output string where it appears in the FORMAT phrase. This is the default separator for Teradata dates.
B b	Blank representation separator. Use this instead of a space to represent a blank.
,	Comma separator. Copied to output string where it appears in the FORMAT phrase.
'	Apostrophe separator. Copied to output string where it appears in the FORMAT phrase.
:	Colon separator. Copied to output string where it appears in the FORMAT phrase.
.	Period separator. Copied to output string where it appears in the FORMAT phrase.
-	Dash separator. Copied to output string where it appears in the FORMAT phrase. This is the default separator for ANSI dates.
9	Separator. Use this formatting character for separators less than 0x009F.

Formatting Characters for Time Information

Use the following characters in the FORMAT phrase to control formatting of time information in TIME and TIMESTAMP types:

Characters	Meaning
HH	Represent the hour as two numeric digits.
MI	Represent the minute as two numeric digits.
SS	Represent the second as two numeric digits.
S(<i>n</i>) S(F)	Number of fractional seconds. Replace <i>n</i> with a number between 0 and 6, or use F for the number of characters needed to display the fractional seconds precision. The value of F is resolved during the formatting of the TIME or TIMESTAMP data. The value is obtained from the fractional seconds precision in the declaration of the data type. For example, F is two for the TIME(2) type. A value of zero for F displays no radix symbol and no fractional precision for the data. The S(F) formatting characters must follow a D formatting character or a . separator character. A value of <i>n</i> that is less than the TIME or TIMESTAMP fractional second precision produces an error.
D	Radix symbol. The value of <i>RadixSeparator</i> in the current SDF is copied to the output string whenever a D appears in the FORMAT phrase. Separator characters, such as . or :, can also appear in the FORMAT phrase, but only if they do not match the value of <i>RadixSeparator</i>.
T	Represent time in 12-hour format instead of 24-hour format. The appropriate time of day, as specified by <i>AMPM</i> in the current SDF is copied to the output string where a T appears in the FORMAT phrase.
Z	Time zone. The Z controls the placement of the time zone in the output of TIME and TIMESTAMP data, and can only appear at the beginning or end of the time formatting characters. For example, the following statement uses a FORMAT phrase that includes a Z before the time formatting characters: <pre>SELECT CURRENT_TIMESTAMP (FORMAT 'YYYY-MM-DDBZBHH:MI:SS.S(6)') ;</pre> If the TIME or TIMESTAMP data contains time zone data, the time zone is copied to the output string. The time zone format is +HH:MI or -HH:MI, depending on the time zone hour displacement.

Characters	Meaning
:	Colon separator. Copied to output string where it appears in the FORMAT phrase. This is the default separator for ANSI time. This character cannot appear in the FORMAT phrase if the value of <i>RadixSeparator</i> in the current SDF is a colon.
.	Period separator. This can also be used to indicate the fractional seconds. Copied to output string where it appears in the FORMAT phrase. This character cannot appear in the FORMAT phrase if the value of <i>RadixSeparator</i> in the current SDF is a period.
-	Dash separator. Copied to output string where it appears in the FORMAT phrase.
h	Hour separator. A lowercase h character is copied to the output string. The h formatting character must follow the HH formatting characters. This character cannot appear in the FORMAT phrase if the value of <i>RadixSeparator</i> in the current SDF is a lowercase h character.
m	Minute separator. A lowercase m character is copied to the output string. The m formatting character must follow the MI formatting characters. This character cannot appear in the FORMAT phrase if the value of <i>RadixSeparator</i> in the current SDF is a lowercase m character.
s	Second separator. A lowercase s character is copied to the output string. The s formatting character must follow SS or SSDS(F) formatting characters. This character cannot appear in the FORMAT phrase if the value of <i>RadixSeparator</i> in the current SDF is a lowercase s character.
B b	Blank representation separator. Use this instead of a space to represent a blank. This character cannot appear in the FORMAT phrase if the value of <i>RadixSeparator</i> in the current SDF is a blank.

Kanji Date and Time Markers

For information on using Kanji date and time markers, see [“FORMAT Phrase, DateTime Formats, and Japanese Character Sets”](#) on page 8-37.

Example Date Formats

The following table is a nonexhaustive list of formats that can be used to present an output date, where the data is 85/09/12:

FORMAT Phrase	Result
FORMAT 'YY/MM/DD'	85/09/12
FORMAT 'DD-MM-YY'	12-09-85
FORMAT 'YYYY/MM/DD'	1985/09/12
FORMAT 'YYYY-MM-DD'	1985-09-12
FORMAT 'YYYY.DDD'	1985.225
FORMAT 'YYBDDD'	85 225
FORMAT 'DDBMMMBYYYY'	12 Sep 1985
FORMAT 'MMMBDD,BYYYY'	Sep 12, 1985
FORMAT 'YYYYBMMMBDD'	1985 Sep 12
FORMAT 'MMM'	Sep
FORMAT 'EEE,BM4BDD,BYYYY'	Thu, September 12, 1985
FORMAT 'E4,BMMMMBDD,BYYYY'	Thursday, September 12, 1985
FORMAT 'E4BDDBM4BYYYY'	jeudi 12 septembre 1985 ^a

a. jeudi is French for Thursday and septembre is French for September.

Example Time Formats

The following table is a nonexhaustive list of formats that can be used to present an output time, where the data is 13:20:53.64+03:00:

FORMAT Phrase	Result
FORMAT 'HH:MIBT'	01:20 PM
FORMAT 'HH:MI'	13:20
FORMAT 'HH.MI.SS'	13.20.53
FORMAT 'HH:MI:SSBT'	01:20:53 nachm ^a
FORMAT 'HH:MI:SSDS(F)'	13:20:53.64
FORMAT 'HH:MI:SSDS(F)Z'	13:20:53.64+03:00
FORMAT 'HHhMIImSSs'	13h20m53s

a. nachm is German for PM.

Example Timestamp Formats

The following table is a nonexhaustive list of formats that can be used to present an output timestamp, where the data is 85/09/12 13:20:53.64+03:00:

FORMAT Phrase	Result
FORMAT 'MM/DD/YYBHH:MI:SS'	85/09/12 01:20 PM
FORMAT 'MMMBDD,BYYBHH:MI:SS'	Sep 12, 85 13:20:53
FORMAT 'E3,BM4BDD,BY4BHH:MI:SSDS(F)'	Thu, September 12, 1985 13:20:53.64
FORMAT 'YYYY-MM-DDBHH:MI:SSDS(F)Z'	1985-09-12 13:20:53.64+03:00

Format Consistency Is Enforced

If a field is declared as type DATE and a FORMAT phrase is specified for it, the date must be entered in the specified format and it must be enclosed in apostrophes.

For example, data supplied to, or used in, a conditional WHERE or HAVING clause for the following column must be defined as 'OCT 24, 1985'.

```
DOB DATE FORMAT 'MMMBDD,bYYYY'
```

This format consistency restriction applies only to the date values themselves, and *not* to the date separators. The input date separators need not match the FORMAT clause date separators.

DATE Comparisons

If a field is to be inserted into a DATE column, the format must match either the ANSI DATE literal format or the format of the column. If a field is to be compared with a DATE value, the format must match one of the following:

- ANSI DATE literal format
- Format of the DATE value
- DATE format determined by DateForm and default DATE data type format in the SDF. See [“DATE Formats” on page 4-18](#).

For example, the comparison works if the data is CHAR(8) in the form YY/MM/DD and a DATE column format is YY/MM/DD. The comparison fails, however, if the column format is YYYY-MM-DD.

To perform comparisons that do not meet these qualifications, convert the values as described in Chapter 13: “Data Type Conversions” in *Teradata RDBMS SQL Reference, Volume 5*.

Examples of How to Query Using Date Formats

Use a statement like the following to display a date in uppercase:

```
SELECT DATE (FORMAT 'MMMbdd,bYYYY') (CHAR(12), UC);
```

Using 1985-09-12 for data, this statement returns:

```
SEP 12, 1985
```

The following query shows how to specify a date if, for example, the p_date column was specified as FORMAT 'DDBMMMBYYYY':

```
SELECT * FROM sales WHERE p_date = '30 Mar 1994'...
```

If the p_date column was specified as FORMAT 'YYYY-MM-DD', the query would be as follows:

```
SELECT * FROM sales WHERE p_date = '1994-03-30'...
```

Examples of Using Time Formats

Create a table using the FORMAT phrase to specify the output format:

```
CREATE TABLE t1
(f1 TIME(3) WITH TIME ZONE FORMAT 'HH:MI:SS.S(F)',
 f2 TIMESTAMP(4) FORMAT 'YYYY-MM-DDBHH:MI:SS.S(F)Z');
```

Populate the table using TIME and TIMESTAMP literals.

```
INSERT t1 (TIME '10:44:25.123-08:00',
TIMESTAMP '2000-09-20 10:44:25.1234');
```

Query the data:

```
SELECT f1, f2
FROM t1;
```

The query returns:

```

              f1                      f2
-----
10:44:25.123 2000-09-20 10:44:25.1234
```

Column f1 was defined as a TIME WITH TIME ZONE data type, but the FORMAT did not include the Z formatting character, so the time zone does not appear in the output. Column f2 was defined as a TIMESTAMP data type, and the FORMAT includes the Z formatting character. However the timestamp data is not associated with any time zone, so no time zone information appears in the output.

To display the time zone information for column f1, use the FORMAT phrase in the SELECT:

```
SELECT CAST (f1 AS TIME(3) WITH TIME ZONE
FORMAT 'HH:MI:SS.S(F)Z')
FROM t1;
```

The query returns:

```
              f1  
-----  
10:44:25.123-8:00
```

For more information on TIME and TIMESTAMP literals, see [Chapter 2: “Data Literals.”](#)

For more information on using the FORMAT phrase in data type conversions, see Chapter 13: “Data Type Conversions,” in *Teradata RDBMS SQL Reference, Volume 5*.

Record Mode Import Anomaly

When data is imported in Record Mode into a NULLABLE DATE field, and the source data is a binary integer of value zero, then the field is set to NULL (not zero).

FORMAT Phrase, DateTime Formats, and Japanese Character Sets

Introduction

The FORMAT phrase can present the date and time in Japanese format.

Separators

FORMAT allows symbols like SLASH and COMMA as separators between day, month, and year. Non-Kanji characters in the FORMAT clause are halfwidth.

For example, applying (FORMAT 'YY/MM/DD') to 920310 yields 92/03/10.

Kanji Date Markers

The FORMAT phrase can contain the following Kanji date markers to separate day, month, and year information in DATE and TIMESTAMP data types.

Use this ideograph ...	After these formatting characters ...	To describe ...
NEN (年)	'YYYY' 'YY'	Year
GATSU (月)	'MM'	Month
NICHI (日)	'DD'	Day

For example, apply (FORMAT 'YY年MM月DD日') to 920310 to yield 92年03月10日.

Because Kanji characters have no canonical form when represented by the KANJI1 character data type, the hexadecimal KANJI1 literal cannot be used for the FORMAT clause.

The Kanji year, month, and day designators can appear in the FORMAT clause in any order, for example, (FORMAT 'MM月YY年DD日').

Date formatting for DATE and TIMESTAMP data types also supports the last four Japanese Imperial Eras, as described in the following table.

Era	Kanji Ideographs	Start Date
Meiji	明治	1867/1/9
Taisho	大正	1912/7/30
Showa	昭和	1926/12/25
Heisei	平成	1989/1/8

To enable the Japanese Imperial Era formatting, the year designator ‘和暦YY年’ must be used in the format clause (和暦 indicates the WA and REKI ideographs). For example, (FORMAT ‘和暦YY年MM月DD日’), (FORMAT ‘和暦YY年DD日MM月’), and (FORMAT ‘和暦YY年DD日’) are valid, but (FORMAT ‘和暦MM月DD日’) is incorrect and an error is generated.

Applying (FORMAT ‘和暦YY年MM月DD日’) to 920310 yields 平成04年03月10日, where 平成 are the two ideographs that together indicate the Heisei era, and 年, 月 and 日 are the NEN, GATSU, and NICHI ideographs respectively (04 stands for fourth year of Heisei era).

Kanji Time Markers

Kanji characters are valid in the FORMAT clause as markers for time information in TIME and TIMESTAMP data types. The following table lists the appropriate markers, their Kanji characters, and their description.

Marker	Kanji Character	Description
JI	時	Hours
FUN	分	Minutes
BYOU	秒	Seconds

For example, applying (FORMAT ‘99時99分99秒’) to 010203 yields 01時02分03秒.

You can use Kanji characters with fractional seconds in TIME and TIMESTAMP formatting. For example:

```
CREATE TABLE t3
  (tstart TIME FORMAT 'HH時MI分SS.S(2)秒',
   tend TIME FORMAT 'HH時MI分SSDS(F)秒');
```

Time data must include the same Kanji time markers that appear in the FORMAT phrase. For example, the following CREATE TABLE statement uses the FORMAT phrase to define the display format for a TIMESTAMP column.

```
CREATE TABLE t2
  (ts TIMESTAMP FORMAT 'YY年MM月DD日HH時MI分SS秒');
```

The following INSERT statement is valid, because the character string data includes the Kanji time markers specified in the FORMAT phrase.

```
INSERT t2 ('92年03月10日18時06分23秒');
```

Using FORMAT for Input

FORMAT can also be used for input.

For example, the following statement creates a table table_2 with a single date type column:

```
CREATE TABLE table_2 (f1 DATE FORMAT 'YY年MM月DD日');
```

The following SQL statements are valid:

```
INSERT INTO table_2 ('92年03月10日');
INSERT INTO table_2 ('平成04年03月10日');
```

Note that the older FORMAT for input is still valid. For example, the following SQL statement is valid.

```
INSERT INTO table_2('92/03/10');
```

Each Imperial Era (except the current one) has an end, which is considered in the validation of input. Using Japanese Imperial Era formatting for dates previous to the Meiji era (before 1867/1/9) is considered a run-time error.

Normally on input, the Teradata RDBMS assigns default values for the year (current year), month (current month), and day (first day), if not specified.

Rules Applied for the First Year of an Era

For the first year of an era, the default month and day is affected by the starting date of the era. For other years, the standard default of the first month and first day are applied.

The following rules are applied for the first year of an era:

- If the day and month have not been indicated, the day is set to the first day of that era (for example, day 25 for Showa era) and the month is set to the first month of that era (for example, month 12 for Showa era).
- If the day is indicated but the month is not, the month is set to the first month of that era.

Note that this can produce an error.

For example, the date '平成01年06日' indicates the first year of the Heisei era and day 6. The month is defaulted to the first month of the Heisei era (January).

This yields the date 1989/1/6, which is not a valid date in the Heisei era.

- If the month is indicated but the day is not
 - If it is the first month of the era (for example, month 12 for the Showa era), then day is set to the first day of that era.
 - If it is not the first month of the era, the day is set to 1 (the beginning of the month).

Naming Columns and Expressions

Introduction

Each column in a SELECT result has a name that is derived from the list of specified column names (expressions) that generated the data. The name is typically the column name from which the data came.

For example, the columns in the result

```
SELECT EmpNo,Name FROM Employee;
```

are labeled EmpNo and Name by default.

In some cases, it is necessary to associate a column name that is different from the default column name with a result. Teradata RDBMS provides two ways to name a column.

This phrase for naming ...	Is ...
NAMED	a Teradata extension to the ANSI SQL-99 standard.
AS	ANSI SQL-99-compliant.

Naming Columns

Teradata RDBMS allows for naming derived columns and renaming existing columns. For every expression that is given a name, that name is entered in a NAMED list that associates the expression with the name. This is true for both forms of naming columns.

The AS and NAMED phrases are documented individually on the pages that follow.

| Rules for Columns Associated With an Explicit Table Name

If a column is associated with an explicit table name, the table is searched for the column name. Based on the result of the search, one of the following occurs:

IF this ...	IS ...	THEN ...
a matching column	found in the table	the requested information is returned.
	<i>not</i> found in the table	the NAMED list is searched for the column name. “Named” names are not valid when used to fully qualify a table name.
a column name	not found in the explicit table name or in the NAMED list	a nonexistent column error is reported.

Rules for Columns Not Associated With an Explicit Table Name

If a column is not associated with an explicit table name, tables named in the SQL statement are searched, and one of the following occurs.

IF this ...	IS ...	THEN ...
a matching column	found in only one table	the requested information is returned.
	found in two or more tables	an ambiguous column name error is reported.
	<i>not</i> found	the NAMED list is searched for the column name. “Named” names are not valid when used to fully qualify a table name.
a column name	not found in the SQL statement or in the NAMED list	a nonexistent column error is reported.

Why You Cannot Use Named Object “Columns” to Qualify a Table Name

When you qualify a column with a table name, the qualifying column cannot be a named object. The following examples demonstrate why this does not work.

Begin with table *t*, defined as:

```
CREATE TABLE T (a INT, b INT);
```

Using this base table, note how the SQL resolver parses and resolves the following statement, creating a view based on the base table:

```
CREATE VIEW V AS
SELECT T.a (NAMED X), X (NAMED Y);
```

Name order and resolution are extremely important because of the way the resolver treats this statement.

Step	IF Column x is ...	THEN ...	ELSE ...
1	in table <i>t</i>	proceed	
2	not in table <i>t</i>	find column <i>x</i> in view <i>v</i>	display an appropriate error message

The following example explains why named objects can be tricky to use in statements such as this:

```
CREATE VIEW V AS
SELECT a*5+3 (NAMED X), x*2 (NAMED Y)
FROM T;
```

The Resolver analyzes this statement as follows:

Stage	Process
1	Look for X as a column in table T.
2	If X is not found, then try to locate column X in view V (this is the so-called “named” object).

Usage of named objects is important, as shown in the following example:

```
CREATE VIEW V AS
SELECT a (NAMED b) , b (NAMED y)
FROM T;
```

Note that the phrase b (NAMED Y) resolves to T.b.

The following example does not work because the table named T does not have a column named X. X is a named object, not a column name. The erroneous example is:

```
CREATE VIEW v AS
SELECT t.a (NAMED X) , t.x (NAMED Y);
```

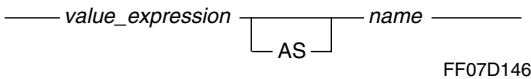
Because table T does not have a column named X, the statement fails. X is a named object. The rule is that when you qualify a column with a table name, the qualifying column cannot be a named object.

AS

Purpose

Assigns a temporary name to an expression.

Syntax



where:

Syntax element ...	Specifies ...
value_expression	the expression to which a temporary name is to be assigned.
AS	an optional keyword indicating that the variable that follows is the temporary new name for value_expression.
name	the new temporary name for value_expression.

ANSI Compliance

AS is ANSI SQL-99-compliant.

Usage Notes

The temporary name can be referenced elsewhere in the request.

Column headings default to the newly assigned name.

Example 1

This syntax allows the ORDER BY clause to include column names rather than positionally defining the sort columns. The AS keyword is not included when column_2 is renamed to second_value.

```
SELECT COLUMN1 +100 AS first_value, column_2 second_value
FROM table_1
WHERE second_value >100
ORDER BY first_value
```

Example 2

This syntax creates a temporary named column and uses it in a WHERE clause.

```
SELECT Name, ((Salary + (YrsExp * 200))/12) AS Projection
FROM Employee
WHERE DeptNo = 600
AND Projection < 2500;
```

NAMED

Purpose

Assigns a temporary name to an expression.

Syntax

—— (*expression*) —— (NAMED —— *name*) ——
FF07D147

where:

Syntax element ...	Specifies ...
<i>expression</i>	the expression to which a temporary name is to be assigned.
NAMED	a keyword indicating that the variable that follows is the temporary new name for <i>expression</i> .
<i>name</i>	the new temporary name for <i>expression</i> .

ANSI Compliance

NAMED is a Teradata extension to the ANSI SQL-99 standard.

Usage Notes

The temporary name can be referenced elsewhere in the request. But it should not be the same as another column name of any table used in a query.

Column headings default to the newly assigned name.

Example 1

In the following SELECT statement, a NAMED phrase associates an arithmetic expression with the name “Projection” so that the calculated column can be more easily referenced in the WHERE clause.

```
SELECT Name, ((Salary + (YrsExp * 200))/12) (NAMED Projection)
FROM Employee
WHERE DeptNo = 600 AND Projection < 2500;
```

The ANSI syntax for this example is the same as used for Example 2 for the “AS” on [page 8-43](#).

Naming Differences: NAMED Versus AS

NAMED is required in Teradata syntax, while AS is not.

The NAMED clause is enclosed in parentheses immediately following the renamed column.

The NAMED clause is supported as an extension. Do not use it for newly developed applications.

TITLE

Purpose

Defines a heading for displayed or printed results that is different from the column name, which is used by default.

Syntax

—— TITLE — *quotestring* ——

FF07D145

where:

Syntax element ...	Specifies ...
<i>quotestring</i>	a character string, delimited by single quotes, defining a column name heading.

ANSI Compliance

TITLE is a Teradata extension to the ANSI SQL-99 standard.

Usage Notes

A TITLE specification string is limited to 60 characters, and must be enclosed in single quotes.

In statements submitted via BTEQ, a TITLE specification can be up to three lines.

A double slash character (//) defines a line break. You can insert blanks to center lines.

You can use the TITLE phrase in a CREATE TABLE or ALTER TABLE statement to specify a standard heading.

Example 1

The following example shows a simple use of the TITLE phrase.

```
CREATE TABLE Charges, FALLBACK
  (EmpNo SMALLINT FORMAT '9(5)' TITLE 'Employee//Id' BETWEEN
    10001 AND 32001 NOT NULL,
   Proj_Id CHAR(8) TITLE 'Project// Id' NOT NULL,
   WkEnd DATE FORMAT 'YYYY/MM/DD' TITLE 'Week//Ending',
   Hours DECIMAL(4,1) FORMAT 'ZZ9.9' BETWEEN 0.5 AND 999.5)
PRIMARY INDEX (EmpNo, Proj_Id)
INDEX (Proj_Id);
```

Example 2

When a title is specified at column creation time, note that the defined column name, not the title name, must be specified in statements that access the column. For example, to retrieve the total hours worked on each project by user Peterson, the project column must be referenced:

```
SELECT Proj_Id, SUM(Hours)
FROM CHARGES
WHERE EmpNo = 10001
GROUP BY Proj_Id ORDER BY Proj_Id ;
```

The returned report, however, uses the following title headings:

Project	
Id	Sum(Hours)
-----	-----
PAY-0001	9.5
PAY-0002	34.5

A TITLE phrase also can be given in the retrieval statement to override any defined standard.

For example, in the following SELECT statement, a TITLE phrase provides a more descriptive heading for the DOB column.

```
SELECT Name, DOB (TITLE 'Birthdate')
FROM Employee;
```

This statement returns:

Name	Birthdate
-----	-----
Smith T	51/10/31
Newman P	56/08/29
Omura H	54/04/24
.	.
.	.

The multiline TITLE definition in this statement:

```
SELECT Name, DOB (TITLE 'Date//Of//Birth')
FROM Employee;
```

returns:

	Date
	Of
Name	Birth
-----	-----
Smith T	51/10/31
Newman P	56/08/29
Omura H	54/04/24
.	.
.	.

TITLE**TITLE Phrase Rules**

Teradata SQL uses the following rules to generate a title for expression x:

IF x is ...	THEN ...	Example
a column reference with an explicit TITLE phrase	that title value is returned.	The explicit title for Project.ProjId, as defined in the CREATE TABLE statement for the Project table, is Project//Id. The following statement returns the indicated result. <pre>SELECT Proj_Id FROM Project WHERE Description='O/E Batch System';</pre> <u>Project Id</u> OE2-0003
a column reference with no explicit TITLE	the column name is returned.	The Salary column has no explicit TITLE. The following statement returns the indicated result. <pre>SELECT Salary FROM Employee WHERE EmpNo = 10002;</pre> Salary 35,000.00
a constant	TITLE is a character representation of that constant. The character representation of the constant could be the identical characters or a normalized form of the constant.	The following statement returns 12 as the column heading for the value 12. <pre>SELECT Name, 12 FROM Employee WHERE EmpNo = 10003;</pre> Name 12 Leidner P12
of the form “operator y”, where operator is a unary (+ or -) or aggregate operator	TITLE is ‘operator’ followed by ‘(y)’	The following statement returns the name of the aggregate operator (SUM) as part of the column title. <pre>SELECT SUM(Salary) FROM Employee WHERE DeptNo = 700;</pre> Sum(Salary) 113,000.00

IF x is ...	THEN ...	Example
of the form “y operator z”	y followed by “operator” followed by z.	The following statement returns the headings of the column data for which the computation is performed and the operator. <pre>SELECT Salary_Loan - 1000 FROM Employee_Loan WHERE EmpNo = 10004;</pre> Salary_Loan - 1000 41000.00
an attribute function of the form “function (y)”	TITLE is function, followed by (y).	The following statement returns the indicated result. <pre>SELECT FORMAT (Employee.EmpNo) ; FORMAT (EmpNo)</pre> ZZZZ9
of the form “y (data_description)”, where data_description does <i>not</i> contain a TITLE phrase	TITLE is (y).	The following statement returns the indicated result. <pre>SELECT YrsExp (BYTEINT) FROM Employee WHERE EmpNo = 10016;</pre> YrsExp 20
of the form “y (data_description)”, where data_description contains a TITLE phrase	TITLE is the title that is specified by the clause.	The following statement returns the indicated result. <pre>SELECT Salary (INTEGER, TITLE 'Pay') FROM Employee WHERE EmpNo = 10018;</pre> Pay 65000

The number of dashes used to define column width is an attribute of BTEQ.

You can modify this display by including a FORMAT phrase in the SELECT statement.

Appendix A:

Notation Conventions

This appendix describes the notation conventions used in this book.

Introduction

Throughout this manual, three conventions are used to describe the SQL syntax and code:

- Syntax diagrams, used to describe SQL syntax form, including options. See [“Syntax Diagram Conventions” on page A-2](#).
- Square braces in the text, used to represent options. The indicated parentheses are required when you specify options.

For example:

- DECIMAL [(n[,m])] means the decimal data type can be defined optionally:
 - without specifying the precision value *n* or scale value *m*
 - specifying precision (n) only
 - specifying both values (n,m)
 - you cannot specify scale without first defining precision.
- CHARACTER [(n)] means that use of (n) is optional.

The values for *n* and *m* are integers in all cases

- Japanese character code shorthand notation, used to represent unprintable Japanese characters. See [“Character Shorthand Notation Used In This Manual” on page A-6](#).

Symbols from the predicate calculus are also used occasionally to describe logical operations. See [“Predicate Calculus Notation Used in This Manual” on page A-8](#).

Syntax Diagram Conventions

Notation Conventions

The following table defines the notation used in this section:

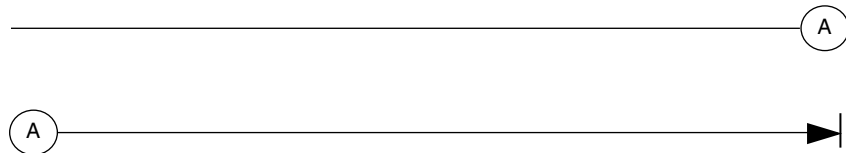
Item	Definition / Comments	
Letter	An uppercase or lowercase alphabetic character ranging from A through Z.	
Number	A digit ranging from 0 through 9. Do not use commas when entering a number with more than three digits.	
Word	Variables and reserved words.	
	IF a word is shown in ...	THEN it represents ...
	UPPERCASE LETTERS	a keyword. Syntax diagrams show all keywords in uppercase, unless operating system restrictions require them to be in lowercase. If a keyword is shown in uppercase, you may enter it in uppercase or mixed case.
	lowercase letters	a keyword that you must enter in lowercase, such as a UNIX command.
	lowercase italic letters	a variable such as a column or table name. You must substitute a proper value.
	lowercase bold letters	a variable that is defined immediately following the diagram that contains it.
	UNDERLINED LETTERS	the default value. This applies both to uppercase and to lowercase words.
Spaces	Use one space between items, such as keywords or variables.	
Punctuation	Enter all punctuation exactly as it appears in the diagram.	

Paths

The main path along the syntax diagram begins at the left, and proceeds, left to right, to the vertical bar, which marks the end of the diagram. Paths that do not have an arrow or a vertical bar only show portions of the syntax.

The only part of a path that reads from right to left is a loop.

Paths that are too long for one line use continuation links. Continuation links are small circles with letters indicating the beginning and end of a link:



FE0CA002

When you see a circled letter in a syntax diagram, go to the corresponding circled letter and continue.

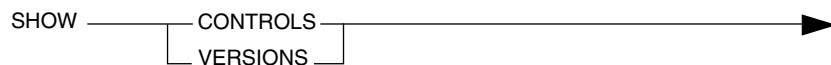
Required Items

Required items appear on the main path:



FE0CA003

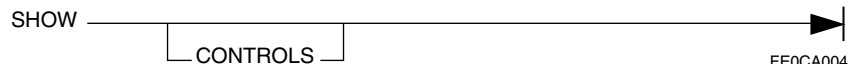
If you can choose from more than one item, the choices appear vertically, in a stack. The first item appears on the main path:



FE0CA005

Optional Items

Optional items appear below the main path:



FE0CA004

If choosing one of the items is optional, all the choices appear below the main path:

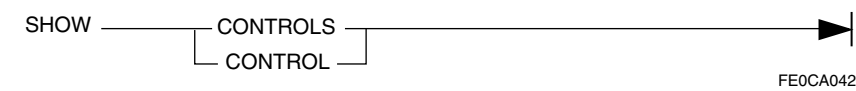


FE0CA006

You can choose one of the options, or you can disregard all of the options.

Abbreviations

If a keyword or a reserved word has a valid abbreviation, the unabbreviated form always appears on the main path. The shortest valid abbreviation appears beneath.

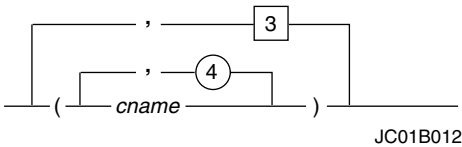


In the above syntax, the following formats are valid:

- `SHOW CONTROLS`
- `SHOW CONTROL`

Loops

A loop is an entry or a group of entries that you can repeat one or more times. Syntax diagrams show loops as a return path above the main path, over the item or items that you can repeat.



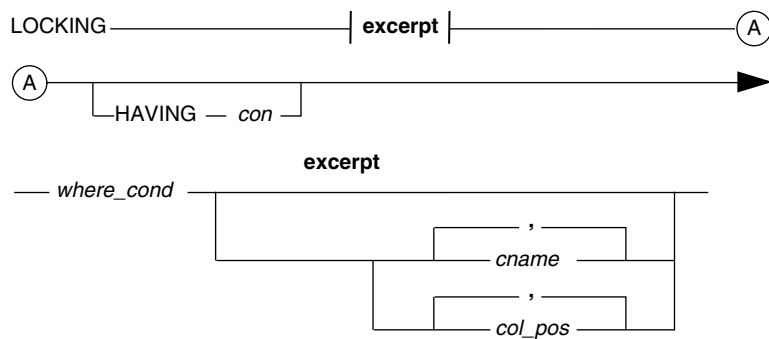
The following rules apply to loops:

IF ...	THEN ...
there is a maximum number of entries allowed	the number appears in a circle on the return path. In the example, you may enter <i>cname</i> a maximum of 4 times.
there is a minimum number of entries required	the number appears in a square on the return path. In the example, you must enter at least 3 groups of column names.
a separator character is required between entries	the character appears on the return path. If the diagram does not show a separator character, use one blank space. In the example, the separator character is a comma.
a delimiter character is required around entries	the beginning and end characters appear outside the return path. Generally, a space is not needed between delimiter characters and entries. In the example, the delimiter characters are the left and right parentheses.

Excerpts

Sometimes a piece of a syntax phrase is too large to fit into the diagram. Such a phrase is indicated by a break in the path, marked by | terminators on either side of the break. A name for the excerpted piece appears between the break marks in boldface type.

The named phrase appears immediately after the complete diagram, as illustrated by the following example.



JC01A014

Character Shorthand Notation Used In This Manual

Introduction

This manual uses the UNICODE naming convention for characters. For example, the lowercase character 'a' is more formally specified as either LATIN SMALL LETTER A or U+0041. The U+xxxx notation refers to a particular code point in the Unicode standard, where xxxx stands for the hexadecimal representation of the 16-bit value defined in the standard.

In parts of the manual, it is convenient to use a symbol to represent a special character, or a particular class of characters. This is particularly true in discussion of the following Japanese character encodings.

- KanjiEBCDIC
- KanjiEUC
- KanjiShift-JIS

These encodings are further defined in Appendix B: "Japanese Character Sets," in *Teradata RDBMS International Character Set Support*.

Symbols

The symbols and the character sets with which they are used are defined in the following table.

Symbol	Encoding	Meaning
a..z A..Z 0..9	Any	Any single byte Latin letter or digit.
<u>a</u> .. <u>z</u> <u>A</u> .. <u>Z</u> <u>0</u> .. <u>9</u>	Unicode compatibility zone	Any fullwidth Latin letter or digit.
<	KanjiEBCDIC	Shift Out [SO] (0x0E). Used to indicate transition from single to multibyte character in KanjiEBCDIC.
>	KanjiEBCDIC	Shift In [SI] (0x0F). Used to indicate transition from multibyte to single byte KanjiEBCDIC.

Symbol	Encoding	Meaning
T	Any	Any multibyte character. Its encoding depends on the current character set. For KanjiEUC, code set 3 characters are sometimes shown preceded by “ss ₃ ”.
I	Any	Any single byte Hankaku Katakana character. In KanjiEUC, it must be preceded by “ss ₂ ”, forming an individual multibyte character.
<u>Δ</u>	Any	Represents the graphic pad character
Δ	Any	Represents either a single or multibyte pad character, depending on context.
ss ₂	KanjiEUC	Represents the EUC code set 2 introducer (0x8E).
ss ₃	KanjiEUC	Represents the EUC code set 3 introducer (0x8F).

For example, string “TEST”, where each letter is intended to be a fullwidth character, is written as **TEST**. Occasionally, when encoding is important, hexadecimal representation is used.

For example, the following mixed single byte/multibyte character data in KanjiEBCDIC character set

LMN<**TEST**>QRS

is represented as:

D3 D4 D5 0E 42E3 42C5 42E2 42E3 0F D8 D9 E2

Pad Characters

The following table lists the pad characters for the various character data types.

Character Data Type	Pad Character Name	Pad Character Value
LATIN	SPACE	0x20
UNICODE	SPACE	U+0020
GRAPHIC	IDEOGRAPHIC SPACE	U+3000
KANJISJIS	ASCII SPACE	0x20
KANJII1	ASCII SPACE	0x20

Predicate Calculus Notation Used in This Manual

Relational databases are based on the theory of relations as developed in set theory. Predicate calculus is often the most unambiguous way to express certain relational concepts.

Occasionally this manual uses the following predicate calculus notation to explain concepts.

This symbol...	Represents this phrase...
iff	If and only if
$\forall$	For all
$\exists$	There exists

Index

A

AMPM, and FORMAT phrase 8–31
ANSI mode
 case sensitivity 5–16
 CASESPECIFIC 5–18
AS phrase 8–43

B

BYTE data type 6–3
BYTE default format 8–2
BYTEINT default format 8–2
BYTEINT numeric data type 3–17

C

CALENDAR system view 4–16
Case sensitivity, defining 5–16
Case, specifying 5–17
CASESPECIFIC, phrase described 5–17
CAST
 data type conversions 3–15
 DECIMAL data type 3–15
CHAR default format 8–2
CHARACTER data type 5–3
Character formats, FORMAT phrase and 8–10
Character set 5–23
CHECK constraints 1–14
Client character set. See Character set
Client representation. See External representation
Column constraints 1–11
Columns, naming 8–40
COMPRESS
 guideline for break even point 7–5
 limits on 7–5
 phrase 7–3
Constants
 character 2–40
 graphic 2–42
 hexadecimal 2–46
Constraints
 CHECK 1–14
 referential 1–15
 uniqueness 1–13

Control phrases

 DEFAULT 7–8
 NOT NULL 7–7
 WITH DEFAULT 7–13
COUNT
 used with DECIMAL(18) 3–12
Currency, and FORMAT phrase 8–20
CurrencyGroupingRule, and FORMAT phrase 8–18
CurrencyGroupSeparator, and FORMAT
 phrase 8–18, 8–22
CurrencyName, and FORMAT phrase 8–20
CurrencyRadixSeparator, and FORMAT
 phrase 8–23, 8–24

D

Data default formats
 changing 8–4
 specification 8–2
Data definition
 case sensitivity, defining 5–16
 character data
 case, specifying 5–17
 character data types
 BYTE 6–3
 CHARACTER 5–3
 GRAPHIC 2–39, 5–7
 LONG VARCHAR 5–14
 LONG VARGRAPHIC 5–7
 VARBYTE 6–5
 VARCHAR 5–9
 VARGRAPHIC 5–7
 character string data
 literals, case sensitivity 2–43
 column constraints 1–11
 constraints
 CHECK 1–14
 constraints, types of 1–12
 PRIMARY KEY 1–13
 referential 1–15
 UNIQUE 1–13
 uniqueness 1–13
 data default formats 8–2
 data type attributes
 ANSI mode 1–4
 Teradata mode 1–4
 data type phrases 1–6
 DateTime data types

- DATE 4-8
- DATE literals 2-11
- TIME 4-22
- TIME literals 2-13
- TIME WITH TIME ZONE 4-27
- TIMESTAMP 4-24
- TIMESTAMP literals 2-17
- TIMESTAMP WITH TIME ZONE 4-29
- DECIMAL data type 3-5
- example 1-3
- FORMAT phrase 8-6
 - CHARACTER format and 8-10
 - DATE formats and 8-29, 8-37
 - NUMERIC formats and 8-12
- Interval data types
 - INTERVAL SECOND literals 2-38
 - INTERVAL DAY 4-40
 - INTERVAL DAY literals 2-29
 - INTERVAL DAY TO HOUR 4-41
 - INTERVAL DAY TO HOUR literals 2-30
 - INTERVAL DAY TO MINUTE 4-42
 - INTERVAL DAY TO MINUTE literals 2-31
 - INTERVAL DAY TO SECOND 4-44
 - INTERVAL DAY TO SECOND literals 2-32
 - INTERVAL HOUR 4-46
 - INTERVAL HOUR literals 2-33
 - INTERVAL HOUR TO MINUTE 4-47
 - INTERVAL HOUR TO MINUTE literals 2-34
 - INTERVAL HOUR TO SECOND 4-48
 - INTERVAL HOUR TO SECOND literals 2-35
 - INTERVAL MINUTE 4-50
 - INTERVAL MINUTE literals 2-36
 - INTERVAL MINUTE TO SECOND 4-51
 - INTERVAL MINUTE TO SECOND
 - literals 2-37
 - INTERVAL MONTH 4-39
 - INTERVAL MONTH literals 2-27
 - INTERVAL SECOND 4-53
 - INTERVAL YEAR 4-37
 - INTERVAL YEAR literals 2-25
 - INTERVAL YEAR TO MONTH 4-38
 - INTERVAL YEAR TO MONTH literals 2-26
- Japanese character graphic data types 2-39
- naming expressions
 - AS phrase 8-43
 - NAMED phrase 8-44
 - TITLE phrase 8-46
- NUMERIC data type 3-5
- numeric data types
 - BYTEINT 3-17
 - DOUBLE PRECISION 3-23
 - FLOAT 3-23
 - INTEGER 3-21
 - REAL 3-23
 - SMALLINT 3-19
- phrases
 - COMPRESS 7-3
 - DEFAULT 7-8
 - NOT NULL 7-7
 - WITH DEFAULT 7-13
- table constraints 1-11
- use of 1-2
- Data type attributes 1-4
- Data type equivalents, ANSI/Teradata 3-2
- Data type phrases 1-6
- DATE
 - arithmetic operations on 4-12
 - DateForm setting 4-18, 4-21
 - default date format setting 4-19, 4-21
 - display format 8-29, 8-37
 - entering 4-9
 - export format 4-11
 - formats 4-18
 - formats allowed 4-20
 - formats hierarchy 4-20
 - numeric date validation 4-11
 - string date validations 4-10
 - system default date format setting 4-19
- DATE data type 4-8
 - CALENDAR view, extending with 4-16
- DATE default format 8-3
- DATE display format 8-29, 8-37
- DATE formats, FORMAT phrase and 4-19, 4-21, 8-29
- DATE literals 2-11
- DateTime fields
 - DateTime data types, components of 4-2
 - values, how represented 4-2
- DateTime literals
 - ANSI, keyword required for 2-10
 - defined 2-10
 - Teradata 2-9
- DBSControl record
 - DecimalMax flag 3-10
- DECIMAL data type 3-5
 - compatibility issues 3-16
- DECIMAL default format 3-8, 8-2
- DECIMAL(18)
 - behavior of SQL operations independent of
 - MaxDecimal 3-12
 - CAST to DECIMAL(18) with a DECIMAL(15)
 - default 3-15
 - DBSControl 3-10
 - DecimalMax flag 3-10
 - problem expressions for different values of
 - MaxDecimal 3-13
 - usage considerations 3-9
- Default DATE format 4-19
- DEFAULT phrase 7-8
 - values for 7-11

WITH DEFAULT phrase 7-13
 DOUBLE PRECISION default format 8-2
 DOUBLE PRECISION numeric data type 3-23
 DualCurrency, and FORMAT phrase 8-20
 DualCurrencyName, and FORMAT phrase 8-20
 DualISOCurrency, and FORMAT phrase 8-20

E

Export data type. See External representation
 Expressions, naming 8-40
 External representation
 ANSI DATE data type 4-11
 BYTE data type 6-4
 BYTEINT data type 3-17
 CHARACTER data type 5-5
 DECIMAL data type 3-6
 DOUBLE PRECISION data type 3-23
 FLOAT data type 3-23
 GRAPHIC data type 5-8
 INTEGER data type 3-21
 IntegerDate data type 4-12
 INTERVAL DAY data type 4-40
 INTERVAL DAY TO HOUR data type 4-41
 INTERVAL DAY TO MINUTE data type 4-43
 INTERVAL DAY TO SECOND data type 4-45
 INTERVAL HOUR data type 4-46
 INTERVAL HOUR TO MINUTE data type 4-47
 INTERVAL HOUR TO SECOND data type 4-49
 INTERVAL MINUTE data type 4-50
 INTERVAL MINUTE TO SECOND data type 4-52
 INTERVAL MONTH data type 4-39
 INTERVAL SECOND data type 4-54
 INTERVAL YEAR data type 4-37
 INTERVAL YEAR TO MONTH data type 4-38
 LONG VARCHAR data type 5-14
 NUMERIC data type 3-6
 REAL data type 3-23
 SMALLINT data type 3-19
 Teradata DATE data type 4-12
 TIMESTAMP data type 4-35
 TIMESTAMP WITH TIME ZONE data type 4-31
 VARBYTE data type 6-6
 VARCHAR data type 5-11
 VARGRAPHIC data type 5-13

F

FLOAT default format 8-2
 FLOAT numeric data type 3-23
 FORMAT phrase 8-6, 8-10, 8-12, 8-29, 8-37
 displaying character information 8-10
 displaying date information 8-29, 8-37
 displaying Japanese date markers 8-37
 displaying Japanese time markers 8-37
 displaying monetary numeric information 8-18
 displaying numeric information 8-12
 displaying time information 8-31
 relationship to SDF 8-7

G

Graphic constants 2-42
 GRAPHIC data type 5-7
 GRAPHIC server data type 5-31
 GroupingRule, and FORMAT phrase 8-12
 GroupSeparator, and FORMAT phrase 8-12, 8-13

H

Hexadecimal literals 2-44

I

INTEGER default format 8-2
 INTEGER numeric data type 3-21
 INTERVAL DAY data type 4-40
 INTERVAL DAY literals 2-29
 INTERVAL DAY TO HOUR data type 4-41
 INTERVAL DAY TO HOUR literals 2-30
 INTERVAL DAY TO MINUTE data type 4-42
 INTERVAL DAY TO MINUTE literals 2-31
 INTERVAL DAY TO SECOND data type 4-44
 INTERVAL DAY TO SECOND literals 2-32
 INTERVAL HOUR data type 4-46
 INTERVAL HOUR literals 2-33
 INTERVAL HOUR TO MINUTE data type 4-47
 INTERVAL HOUR TO MINUTE literals 2-34
 INTERVAL HOUR TO SECOND data type 4-48
 INTERVAL HOUR TO SECOND literals 2-35
 INTERVAL MINUTE data type 4-50
 INTERVAL MINUTE literals 2-36
 INTERVAL MINUTE TO SECOND data type 4-51

INTERVAL MINUTE TO SECOND literals 2–37
 INTERVAL MONTH data type 4–39
 INTERVAL MONTH literals 2–27
 INTERVAL SECOND data type 4–53
 INTERVAL SECOND literals 2–38
 INTERVAL YEAR data type 4–37
 INTERVAL YEAR default format 8–3
 INTERVAL YEAR literals 2–25
 INTERVAL YEAR TO MONTH data type 4–38
 INTERVAL YEAR TO MONTH literals 2–26
 ISOCurrency, and FORMAT phrase 8–20

J

Japanese character code notation, how to read A–6

K

Kanji
 time formatting characters 8–38
 KANJI1 server data type 5–33
 KANJI1SJIS server data type 5–32

L

LATIN server data type 5–29
 Literals
 case sensitivity of 2–43
 DateTime data types
 DATE 2–11
 TIME 2–13
 TIMESTAMP 2–17
 Interval data types
 INTERVAL DAY 2–29
 INTERVAL DAY TO HOUR 2–30
 INTERVAL DAY TO MINUTE 2–31
 INTERVAL DAY TO SECOND 2–32
 INTERVAL HOUR 2–33
 INTERVAL HOUR TO MINUTE 2–34
 INTERVAL HOUR TO SECOND 2–35
 INTERVAL MINUTE 2–36
 INTERVAL MINUTE TO SECOND 2–37
 INTERVAL MONTH 2–27
 INTERVAL SECOND 2–38
 INTERVAL YEAR 2–25
 INTERVAL YEAR TO MONTH 2–26
 LONG VARCHAR data type 5–14
 LONG VARCHAR default format 8–2

LONG VARCHAR data type 5–7
 LongDays, and FORMAT phrase 4–10, 8–30
 LongMonths, and FORMAT phrase 8–29

N

NAMED phrase 8–44
 NOT CASESPECIFIC, phrase described 5–17
 NOT NULL phrase 7–7
 NUMERIC data type 3–5
 NUMERIC default format 3–8, 8–2
 Numeric formats, FORMAT phrase and 8–12

P

Primary key, constraint 1–13

R

RadixSeparator, and FORMAT phrase 8–12, 8–13,
 8–16, 8–31, 8–32
 REAL default format 8–2
 REAL numeric data type 3–23
 Referential constraints 1–15
 Representation. See External representation
 Rounding
 DECIMAL/NUMERIC data 3–3
 FLOAT/REAL/DOUBLE PRECISION data 3–4

S

SDF
 AMPM 8–31
 and changing data type default formats 8–4
 and changing TIME data type default
 formats 4–31
 and changing TIMESTAMP data type default
 formats 4–31
 and DATE data type default format 4–18
 and tdllocaledef utility 4–19
 Currency 8–20
 CurrencyGroupingRule 8–18
 CurrencyGroupSeparator 8–18, 8–22
 CurrencyName 8–20
 CurrencyRadixSeparator 8–23, 8–24
 DualCurrency 8–20

- DualCurrencyName 8–20
- DualISOCurrency 8–20
- GroupingRule 8–12
- GroupSeparator 8–12, 8–13
- ISOCurrency 8–20
- LongDays 4–10, 8–30
- LongMonths 8–29
- RadixSeparator 8–12, 8–13, 8–16, 8–31, 8–32
- relationship to formatting characters 8–7
- setting the DATE data type default format 4–19
- ShortDays 4–10, 8–30
- ShortMonths 4–10, 8–29
- Server data types
 - GRAPHIC 5–31
 - KANJI1 5–33
 - KANJISJIS 5–32
 - LATIN 5–29
 - UNICODE 5–30
- ShortDays, and FORMAT phrase 4–10, 8–30
- ShortMonths, and FORMAT phrase 4–10, 8–29
- SMALLINT default format 8–2
- SMALLINT numeric data type 3–19
- Specification for data formatting, see SDF
- Syntax, how to read A–1
- SYS_CALENDAR system database
 - CALENDAR view 4–16

T

- Table constraints 1–11
- tdlocaledef utility 8–4, 8–7
 - and SDF 4–19
- Teradata mode, case sensitivity 5–16
- TIME
 - default time format setting 4–33
 - formats 4–31
 - formats allowed 4–33
 - formats hierarchy 4–33
- Time
 - Kanji formatting characters 8–38
- TIME data type 4–22
- TIME default format 8–3
- TIME display format 8–31
- TIME formats, FORMAT phrase and 4–32, 4–33
- TIME literals 2–13
- TIME values
 - operation on 4–4
 - storage 4–4
 - UTC offsets, effect on 4–6
- TIME WITH TIME ZONE data type 4–27

- TIME WITH TIMEZONE default format 8–3
- Time zones
 - explicit 4–4
 - implicit 4–5
 - representation of 4–4
 - UTC and 4–4
- TIMESTAMP
 - default time format setting 4–33
 - formats 4–31
 - formats allowed 4–33
 - formats hierarchy 4–33
- TIMESTAMP data type 4–24
 - arithmetic on 4–25, 4–26
- TIMESTAMP default format 8–3
- TIMESTAMP display format 8–29, 8–31, 8–37
- TIMESTAMP formats, FORMAT phrase and 4–32, 4–33
- TIMESTAMP literals 2–17
- TIMESTAMP values
 - operations on 4–4
 - storage 4–4
 - UTC offsets, effect on 4–6
- TIMESTAMP WITH TIME ZONE data type 4–29
- TIMESTAMP WITH TIMEZONE default format 8–3
- TITLE phrase 8–46

U

- Unicode
 - effect on references to character data length 4–2
 - effect on references to format length 4–2
- UNICODE server data type 5–30
- Unicode, notation A–6
- Unique, defined 1–13
- UPPERCASE, phrase described 5–22

V

- VARBYTE data type 6–5
- VARBYTE default format 8–2
- VARCHAR data type 5–9
- VARCHAR default format 8–2
- VARGRAPHIC data type 5–7

W

- WITH DEFAULT phrase 7–13

Y

Year 2000 compliance 4–2