TINYINT -> ShortInt or Byte

Exact numeric value with precision 3 and scale 0

(signed: –128 <= n <= 127, unsigned: 0 <= n <= 255)

SMALLINT -> SmallInt or Word

Exact numeric value with precision 5 and scale 0

(signed: –32,768 <= n <= 32,767, unsigned: 0 <= n <= 65,535)

An application uses SQLGetTypeInfo or SQLColAttribute to

determine whether a particular data type or a particular column in a result set is unsigned.

INTEGER -> INTEGER = LONGINT or Cardinal = LongWord

Exact numeric value with precision 10 and scale 0

(signed: –2[31] <= n <= 2[31] – 1, unsigned: 0 <= n <= 2[32] – 1)

BIGINT -> Int64

Exact numeric value with precision 19 (if signed) or 20 (if unsigned) and scale 0

(signed: –2[63] <= n <= 2[63] – 1, unsigned: 0 <= n <= 2[64] – 1)

This data type has no corresponding data type in SQL-92.

CHAR(n) -> String[n]

Character string of fixed string length n.

VARCHAR(n) -> String[n]

Variable-length character string with a maximum string length n.

LONG VARCHAR -> String

Variable length character data. Maximum length is data source–dependent.

This data type has no corresponding data type in SQL-92.

WCHAR(n) -> WideString(n)

Unicode character string of fixed string length n

VARWCHAR(n) -> WideString(n)

Unicode variable-length character string with a maximum string length n

LONGWVARCHAR -> WideString

Unicode variable-length character data. Maximum length is data source–dependent

Char: Holds a single character, small alphabet

AnsiChar: Holds a single character, small alphabet

WideChar: Holds a single character, International alphabet

ShortString: Holds a string of up to 255 Char's

String: Holds strings of Char's of any size desired

AnsiString: Holds strings of AnsiChar's any size desired

WideString: Holds strings of WideChar's of any size desired

DECIMAL(p,s) = DEC(p[,s]) -> Extended

Signed, exact, numeric value with a precision of at least p and scale s.

(The maximum precision is driver-defined.) (1 <= p <= 15; s <= p)

NUMERIC(p,s) -> Currency if s<= 4 or Extended

Signed, exact, numeric value with a precision p and scale s (1 <= p <= 15; s <= p)

Currency; // 50+ significant digits, fixed 4 decimal places

Double; // 15 significant digits, exponent -308 to +308

Extended; // 19 significant digits, exponent -4932 to +4932

SQL\_DECIMAL and SQL\_NUMERIC data types differ only in their precision.

The precision of a DECIMAL(p,s) is an implementation-defined decimal precision

that is no less than p, whereas the precision of a NUMERIC(p,s) is exactly equal to p.

REAL -> Single

Signed, approximate, numeric value with a binary precision 24

(zero or absolute value 10[–38] to 10[38]).

Single; // 7 significant digits, exponent -38 to +38

FLOAT(p)

Signed, approximate, numeric value with a binary precision of at least p.

(The maximum precision is driver-defined.)

Depending on the implementation, the precision of SQL\_FLOAT can be either 24 or 53:

if it is 24, the SQL\_FLOAT data type is the same as SQL\_REAL; if it is 53,

the SQL\_FLOAT data type is the same as SQL\_DOUBLE.

DOUBLE PRECISION -> Double

Signed, approximate, numeric value with a binary precision 53

(zero or absolute value 10[–308] to 10[308]).

BIT

Single bit binary data.

The SQL\_BIT data type has different characteristics than the BIT type in SQL-92.

BIT

BIT(n)

BINARY(n)

Binary data of fixed length n.

This data type has no corresponding data type in SQL-92.

VARBINARY(n)

Variable length binary data of maximum length n. The maximum is set by the user

This data type has no corresponding data type in SQL-92.

LONG VARBINARY

Variable length binary data. Maximum length is data source–dependent.

DATE

Year, month, and day fields, conforming to the rules of the Gregorian calendar

TIME

Hour, minute, and second fields, with valid values for hours of 00 to 23,

valid values for minutes of 00 to 59, and valid values for seconds of 00 to 61.

Precision p indicates the seconds precision.

TIMESTAMP(p)

Year, month, day, hour, minute, and second fields, with valid values as defined

for the DATE and TIME data types.

UTCDATETIME

Year, month, day, hour, minute, second, utchour, and utcminute fields.

The utchour and utcminute fields have 1/10 microsecond precision

UTCTIME

Hour, minute, second, utchour, and utcminute fields.

The utchour and utcminute fields have 1/10 microsecond precision..

INTERVAL MONTH(p)

Number of months between two dates; p is the interval leading precision.

INTERVAL YEAR(p)

Number of years between two dates; p is the interval leading precision.

INTERVAL YEAR(p) TO MONTH

Number of years and months between two dates; p is the interval leading precision.

INTERVAL DAY(p)

Number of days between two dates; p is the interval leading precision.

INTERVAL HOUR(p)

Number of hours between two date/times; p is the interval leading precision.

INTERVAL MINUTE(p)

Number of minutes between two date/times; p is the interval leading precision.

INTERVAL SECOND(p,q)

Number of seconds between two date/times; p is the interval leading precision and q is the interval seconds precision.

INTERVAL DAY(p) TO HOUR

Number of days/hours between two date/times; p is the interval leading precision.

INTERVAL DAY(p) TO MINUTE

Number of days/hours/minutes between two date/times; p is the interval leading precision.

INTERVAL DAY(p) TO SECOND(q)

Number of days/hours/minutes/seconds between two date/times; p is the interval leading precision

and q is the interval seconds precision.

INTERVAL HOUR(p) TO MINUTE

Number of hours/minutes between two date/times; p is the interval leading precision.

INTERVAL HOUR(p) TO SECOND(q)

Number of hours/minutes/seconds between two date/times; p is the interval leading precision

and q is the interval seconds precision.

INTERVAL MINUTE(p) TO SECOND(q)

Number of minutes/seconds between two date/times; p is the interval leading precision

and q is the interval seconds precision.

GUID Data Type

Assigns a unique identifying number to any database object. The GUID data type is a 16

byte binary data type. This data type is used for the global identification of objects,

programs, records, and so on. The important property of a GUID is that each value

is globally unique. The value is generated by an algorithm, developed by Microsoft,

which assures this uniqueness.

The GUID is a 16-byte binary data type that can be logically grouped into the following

subgroups:

4byte-2byte-2byte-2byte-6byte.

The standard textual representation is {12345678-1234-1234-1234-1234567890AB}.

Using triggers makes a call to a procedure in Delphi.

An include file have the procedures implementation of all triggers.

Create trigger send a text file with the Delphi instruction of the procedure.

Column Constraints: 0 for NULL

0 for UNIQUE

0 for PRIMARY KEY

0 for FOREIGN KEY

For column reference, you can use int64 and column position to save in constraint; u can use 8 int64 for 512 columns.

A table can have only one primary key that is not null and unique. The others are called surrogate or candidate keys.

Column constraint:

1. Not Null
2. Unique
3. Primary Key
4. Check
5. References

Table constraint:

1. FOREIGN KEY
2. UNIQUE
3. PRIMARY KEY
4. CHECK
5. REFERENCES

Column:

* Bit for Null, implemented as Null: array [0..3] of int64 for columns
* Unique could be on many columns, implemented as Unique: array of array of int64 note that if one column is unique it is guaranteed that in many columns would be unique
* Primary Key: could be more than one implemented as pk: array of array of int64
* Check could be implemented as set of instruction for a stack
* References could be implement as triggers

The ON UPDATE and ON DELETE options can have the following actions:

**NO ACTION**

UPDATEs and DELETEs to the primary key are prohibited if referenced by a foreign key row. This is the default.

**CASCADE**

UPDATEs to the primary key update all foreign key columns that reference it. DELETEs on the primary key cause the deletion of all foreign key rows that reference it.

**SET NULL**

UPDATEs and DELETEs to the primary key row cause the foreign key to be set to NULL.

**SET DEFAULT**

UPDATEs and DELETEs to the primary key row cause the foreign key to be set to its DEFAULT.

create database pippo

PUSH NAME pippo (119 {$77}, 0, 'pippo', 'PUSH DBNAME pippo')

Push ‘pippo’ pippo

DATABASE NAME (2, 0, '', 'DATABASE NAME')

Opr DBN -> Pop and return DBN pippo pippo -> DBN pippo

CREATE DATABASE (1, 0, '', 'CREATE DATABASE')

Opr CDB -> Pop and execute Create Database EMPT STACK

At this point check in databases if exist this database, if no create it

In DataDictionary:

DATABASE metadata(DBName)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

create table t (c char(2) not null, v int default 10 primary key, l int default

5 constraint y null, constraint a primary key (c,v))

PUSH DBNAME t

Push ‘t’ t

TABLE NAME (4, 0, '', 'TABLE NAME')

Opr TBN -> Pop and return TBN t t -> TBN t

PUSH NAME c

Push ‘c’ TBN t -> c ---TBN t

COLUMN NAME (6, 0, '', 'COLUMN NAME')

Opr CLN -> Pop and return CLN c c ---TBN t -> CLN c ---TBN t

PUSH 2

Push 2 (7, 0, '', 'CHAR') CLN c ---TBN t -> 2 --- CLN c ---TBN t

Opr TYPE -> Pop and return CHAR 2 2 --- CLN c ---TBN t -> CHAR 2 --- CLN c ---TBN t

NOT NULL

Push NOT NULL CHAR 2 --- CLN c ---TBN t -> NOT NULL --- CHAR 2 --- CLN c ---TBN t

CONSTRAINT COLUMN

Opr CONSTRAINT -> Pop, insert into Data dictionary and return CONSTRAINT NOT NULL

NOT NULL --- CHAR 2 --- CLN c ---TBN t -> CONSTRAINT NOT NULL --- CHAR 2 --- CLN c ---TBN t

OR

Opr CONSTRAINT -> Pop, insert into Data dictionary and DON’T RETURN CONSTRAINT NOT NULL but keep that this constraint is for the new column. (Maybe with some structure)

As there is no name for the constraint put it automatically.

NEW COLUMN

Opr NCN -> Pop till CLN, insert into Data Dictionary and return NCN c CHAR 2 CONSTRAINT NOT NULL

Pop till CLN, so the stack is empty but that this column is for the new table. (Maybe with some structure)

PUSH DBNAME v

Push ‘v’

COLUMN NAME

Opr CLN -> Pop and return CLN v

INT

Opr TYPE -> Pop and return INT

PUSH 10

Push 10

DEFAULT

Opr DFLT -> Pop, insert into Data Dictionary and return DFLT 10

PRIMARY KEY

Push PRK

CONSTRAINT

Opr CONSTRAINT -> Pop, insert into Data dictionary and return CONSTRAINT PRK

Constraint\_metaData(constraint\_name,column\_name,type\_constraint)

NEW COLUMN

Opr NCN -> Pop till CLN, insert into Data Dictionary and return NCN v INT DFLT 10 CONSTRAINT PRK

PUSH DBNAME l

Push ‘l’

COLUMN NAME

Opr CLN -> Pop and return CLN l

INT

Opr TYPE -> Pop and return INT

PUSH 5

Push 5

DEFAULT

Opr DFLT -> Pop, insert into Data Dictionary and return DFLT 5

PUSH DBNAME y

CONSTRAINT NAME

NULL

CONSTRAINT

NEW COLUMN

PUSH DBNAME c

COLUMN NAME

PUSH DBNAME v

COLUMN NAME

PRIMARY KEY

PUSH DBNAME a

CONSTRAINT NAME

TABLE CONSTRAINT

CREATE TABLE

delete from tablename

PUSH DBNAME tablename

TABLE NAME

DELETE FROM

DBNAME tablename

DBNAME tablename

TABLE NAME -> TABLE NAME tablename

TABLENAME tablename

DELETE FROM -> opr Empty the table (Fast: Remove the table, Create it)

Delete from tablename where (a=b) or (c=d) and (e=f)

PUSH DBNAMEtablename

TABLE NAME

PUSH DBNAMEa

PUSH DBNAMEb

EQ

PUSH DBNAMEc

PUSH DBNAMEd

EQ

PUSH DBNAMEe

PUSH DBNAMEf

EQ

AND

OR

DELETE FROM

The following table lists valid SQL type identifiers for all SQL data types. The table also lists the name and description of the corresponding data type from SQL-92 (if one exists).

|  |  |  |
| --- | --- | --- |
| **SQL type identifier[1]** | **Typical SQL data type[2]** | **Typical type description** |
| SQL\_CHAR | CHAR(*n*) | Character string of fixed string length *n*. |
| SQL\_VARCHAR | VARCHAR(*n*) | Variable-length character string with a maximum string length *n*. |
| SQL\_LONGVARCHAR | LONG VARCHAR | Variable length character data. Maximum length is data source–dependent.[9] |
| SQL\_WCHAR | WCHAR(*n*) | Unicode character string of fixed string length *n* |
| SQL\_WVARCHAR | VARWCHAR(*n*) | Unicode variable-length character string with a maximum string length *n* |
| SQL\_WLONGVARCHAR | LONGWVARCHAR | Unicode variable-length character data. Maximum length is data source–dependent |
| SQL\_DECIMAL | DECIMAL(*p*,*s*) | Signed, exact, numeric value with a precision of at least *p* and scale *s.* (The maximum precision is driver-defined.) (1 <= *p* <= 15; *s* <= *p*).[4] |
| SQL\_NUMERIC | NUMERIC(*p*,*s*) | Signed, exact, numeric value with a precision *p* and scale *s*  (1 <= *p* <= 15; *s* <= *p*).[4] |
| SQL\_SMALLINT | SMALLINT | Exact numeric value with precision 5 and scale 0 (signed: –32,768 <= *n* <= 32,767, unsigned: 0 <= *n* <= 65,535)[3]. |
| SQL\_INTEGER | INTEGER | Exact numeric value with precision 10 and scale 0 (signed: –2[31] <= *n* <= 2[31] – 1, unsigned: 0 <= *n* <= 2[32] – 1)[3]. |
| SQL\_REAL | REAL | Signed, approximate, numeric value with a binary precision 24 (zero or absolute value 10[–38] to 10[38]). |
| SQL\_FLOAT | FLOAT(*p*) | Signed, approximate, numeric value with a binary precision of at least *p*. (The maximum precision is driver-defined.)[5] |
| SQL\_DOUBLE | DOUBLE PRECISION | Signed, approximate, numeric value with a binary precision 53 (zero or absolute value 10[–308] to 10[308]). |
| SQL\_BIT | BIT | Single bit binary data.[8] |
| SQL\_TINYINT | TINYINT | Exact numeric value with precision 3 and scale 0 (signed: –128 <= *n* <= 127, unsigned: 0 <= *n* <= 255)[3]. |
| SQL\_BIGINT | BIGINT | Exact numeric value with precision 19 (if signed) or 20 (if unsigned) and scale 0 (signed: –2[63] <= *n* <= 2[63] – 1, unsigned: 0 <= *n* <= 2[64] – 1)[3],[9]. |
| SQL\_BINARY | BINARY(*n*) | Binary data of fixed length *n*.[9] |
| SQL\_VARBINARY | VARBINARY(*n*) | Variable length binary data of maximum length *n*. The maximum is set by the user.[9] |
| SQL\_LONGVARBINARY | LONG VARBINARY | Variable length binary data. Maximum length is data source–dependent.[9] |
| SQL\_TYPE\_DATE[6] | DATE | Year, month, and day fields, conforming to the rules of the Gregorian calendar. (See [Constraints of the Gregorian Calendar](http://msdn.microsoft.com/en-us/library/ms712480(VS.85).aspx), later in this appendix.) |
| SQL\_TYPE\_TIME[6] | TIME(*p*) | Hour, minute, and second fields, with valid values for hours of 00 to 23, valid values for minutes of 00 to 59, and valid values for seconds of 00 to 61. Precision *p* indicates the seconds precision. |
| SQL\_TYPE\_TIMESTAMP[6] | TIMESTAMP(*p*) | Year, month, day, hour, minute, and second fields, with valid values as defined for the DATE and TIME data types. |
| SQL\_TYPE\_UTCDATETIME | UTCDATETIME | Year, month, day, hour, minute, second, utchour, and utcminute fields. The utchour and utcminute fields have 1/10 microsecond precision. |
| SQL\_TYPE\_UTCTIME | UTCTIME | Hour, minute, second, utchour, and utcminute fields. The utchour and utcminute fields have 1/10 microsecond precision.. |
| SQL\_INTERVAL\_MONTH[7] | INTERVAL MONTH(*p*) | Number of months between two dates; *p* is the interval leading precision. |
| SQL\_INTERVAL\_YEAR[7] | INTERVAL YEAR(*p*) | Number of years between two dates; *p* is the interval leading precision. |
| SQL\_INTERVAL\_YEAR\_TO\_MONTH[7] | INTERVAL YEAR(*p*) TO MONTH | Number of years and months between two dates; *p* is the interval leading precision. |
| SQL\_INTERVAL\_DAY[7] | INTERVAL DAY(*p*) | Number of days between two dates; *p* is the interval leading precision. |
| SQL\_INTERVAL\_HOUR[7] | INTERVAL HOUR(*p*) | Number of hours between two date/times; *p* is the interval leading precision. |
| SQL\_INTERVAL\_MINUTE[7] | INTERVAL MINUTE(*p*) | Number of minutes between two date/times; *p* is the interval leading precision. |
| SQL\_INTERVAL\_SECOND[7] | INTERVAL SECOND(*p*,*q*) | Number of seconds between two date/times; *p* is the interval leading precision and *q* is the interval seconds precision. |
| SQL\_INTERVAL\_DAY\_TO\_HOUR[7] | INTERVAL DAY(*p*) TO HOUR | Number of days/hours between two date/times; *p* is the interval leading precision. |
| SQL\_INTERVAL\_DAY\_TO\_MINUTE[7] | INTERVAL DAY(*p*) TO MINUTE | Number of days/hours/minutes between two date/times; *p* is the interval leading precision. |
| SQL\_INTERVAL\_DAY\_TO\_SECOND[7] | INTERVAL DAY(*p*) TO SECOND(*q*) | Number of days/hours/minutes/seconds between two date/times; *p* is the interval leading precision and *q* is the interval seconds precision. |
| SQL\_INTERVAL\_HOUR\_TO\_MINUTE[7] | INTERVAL HOUR(*p*) TO MINUTE | Number of hours/minutes between two date/times; *p* is the interval leading precision. |
| SQL\_INTERVAL\_HOUR\_TO\_SECOND[7] | INTERVAL HOUR(*p*) TO SECOND(*q*) | Number of hours/minutes/seconds between two date/times; *p* is the interval leading precision and *q* is the interval seconds precision. |
| SQL\_INTERVAL\_MINUTE\_TO\_SECOND[7] | INTERVAL MINUTE(*p*) TO SECOND(*q*) | Number of minutes/seconds between two date/times; *p* is the interval leading precision and *q* is the interval seconds precision. |
| SQL\_GUID | GUID | Fixed length GUID. |

[1]   This is the value returned in the DATA\_TYPE column by a call to **SQLGetTypeInfo**.

[2]   This is the value returned in the NAME and CREATE PARAMS column by a call to **SQLGetTypeInfo**. The NAME column returns the designation—for example, CHAR—whereas the CREATE PARAMS column returns a comma-separated list of creation parameters such as precision, scale, and length.

[3]   An application uses **SQLGetTypeInfo** or **SQLColAttribute** to determine whether a particular data type or a particular column in a result set is unsigned.

[4]   SQL\_DECIMAL and SQL\_NUMERIC data types differ only in their precision. The precision of a DECIMAL(*p*,*s*) is an implementation-defined decimal precision that is no less than *p*, whereas the precision of a NUMERIC(*p*,*s*) is exactly equal to *p*.

[5]   Depending on the implementation, the precision of SQL\_FLOAT can be either 24 or 53: if it is 24, the SQL\_FLOAT data type is the same as SQL\_REAL; if it is 53, the SQL\_FLOAT data type is the same as SQL\_DOUBLE.

[6]   In ODBC 3*.x*, the SQL date, time, and timestamp data types are SQL\_TYPE\_DATE, SQL\_TYPE\_TIME, and SQL\_TYPE\_TIMESTAMP, respectively; in ODBC 2.*x*, the data types are SQL\_DATE, SQL\_TIME, and SQL\_TIMESTAMP.

[7]   For more information about the interval SQL data types, see the [Interval Data Types](http://msdn.microsoft.com/en-us/library/ms716506(VS.85).aspx) section, later in this appendix.

[8]   The SQL\_BIT data type has different characteristics than the BIT type in SQL-92.

[9]   This data type has no corresponding data type in SQL-92.