SQL Dialects Reference/Print version

SQL Dialects Reference

Contents

Introduction

Preamble

SQL (Structured Query Language) is one of the oldest programming languages in existence, first versions of which date back to 1969. Unfortunately, despite SQL being standardized since 1986, a lot of different implementations exist. They deviate more or less from each other, making developing applications that would work with a range of different SQL servers particularly difficult.

This wikibook is a compact comparative reference for several SQL language dialects. It lists particular common tasks and problems resolved in terms of several popular SQL server implementations. When

possible, it tries to emphasize an universal solution. When it's not possible, it tries to list best practices.

Two main goals for the book are **compactness** and **completeness**. Obvious information, e.g. that a particular function or query does the same thing on all implementations, will not be listed. However, when some implementation(s) have important (even minor) deviations that can be a major pitfalls for developers, such information definitely should be listed.

Who should use this book

Target audience is:

- Developers who know a single dialect of SQL and want to develop for other SQL implementation right away (it's a simple way to).
- Developers who port a ready project from one SQL implementation to another or a bunch of SQL implementations.
- Developers who want their applications to be portable on a set of SQL servers (it's particularly useful for developers of open source projects that want to ensure that their application can be used on a variety of platforms).
- System administrators who have to support a wide range of SQL servers and don't want to memorize all possible queries and nuances for every server.
- Project managers or lead developers that want to estimate what SQL server is best for their project, given a list of requirements.

Who shouldn't use this book

This book is not a general learning course, not a comprehensive manual - it's a quick and compact reference that assumes that the reader knows basic concepts of SQL and precisely what is needed. The book discusses fairly advanced topics and minor, but important differences in SQL implementations that beginners / learners generally don't have to worry about.

Technical notes

Always where it is possible, a book tries to reference a web-published documentation, so a reader can always check particular detail, so a page is usually full of external documentation links.

As the full names of implementations are pretty long and inconvenient to spell out thoroughly every time, we'll use common abbreviations (shown in **bold**) to distinguish SQL dialects. Also, we only provide links to corresponding Wikipedia articles here, not on every occasion.

Many pages in this book include comparison tables that can be pretty long. It is recommended to install a comparison table extension to view these tables comfortably.

SQL implementations covered in this book

Several SQL dialects are covered in this book. Note that in all cases of particular implementations, only the latest stable version is described.

- **Standard**. There are several versions of SQL standard (SQL-86, SQL-89, SQL-92, SQL:1999, SQL:2003, SQL:2011). The book would list all popular practices to do a job in

all implementations, particularly emphasizing SQL versions that a particular solution works for (i.e. if some feature became standard since version X, it would be stated that such solution works since version X).

- **DB2** (IBM DB2).
- **Firebird** (Firebird 2.5). An open-source RDBMS forked from sources of InterBase that were released by Borland on July 25, 2000. Firebird differs from Interbase and tries to be close (http://www.firebirdsql.org/index.php?op=devel&sub=engine&id=SQL_conformance) to SQL standard as much as possible
- **MonetDB** An open-source column-store. (MonetDB. V11)
- **MySQL** (MySQL 5.0)
- **MSSQL** (Microsoft SQL Server 2008). A proprietary RDBMS produced by Microsoft, targeting enterprise market. Original codebase was derived from Sybase SQL Server, but was mostly rewritten. Documentation is available in MSDN in a well-structured form, quite easy to reference.
- **Oracle** (Oracle Database 11g2)
- **PostgreSQL** (PostgreSQL 8.4)
- **SQLite**.
- **Virtuoso** (OpenLink Virtuoso running on Virtuoso Universal Server).
- **Linter**

Data structure definition/Data types/Numeric types

SQL version	Feature	Standard SQL:2011	DB2	Firebird	
?	Integer (1 byte)	[1]	N/A	N/A	tinyint integer1 (/Ingres /9.3/SQL% /numericc
?	Integer (2 bytes)	[1]	SMALLINT (http://publib.boulder.ibm.com /infocenter/db2luw /v9/index.jsp?topic= /com.ibm.db2.udb.admin.doc /doc/r0008469.htm)	SMALLINT	smallint integer2 (/Ingres /9.3/SQL% /numericc
?	Integer (3 bytes)	[1]	N/A	N/A	
?	Integer (4 bytes)	[1]	INTEGER (http://publib.boulder.ibm.com /infocenter/db2luw /v9/index.jsp?topic= /com.ibm.db2.udb.admin.doc /doc/r0008469.htm)	INTEGER	integer integer4 (/Ingres /9.3/SQL% /numericc
?	Integer (8 bytes)	[1]	BIGINT (http://publib.boulder.ibm.com /infocenter/db2luw /v9/index.jsp?topic= /com.ibm.db2.udb.admin.doc /doc/r0008469.htm)	BIGINT	bigint integer8 (/Ingres /9.3/SQL% /numericc
?	Float (single precision, 4 bytes)	[3]	REAL (http://publib.boulder.ibm.com /infocenter/db2luw /v9/index.jsp?topic= /com.ibm.db2.udb.admin.doc /doc/r0008469.htm)	FLOAT	float4 real float(23) (/Ingres /9.3/SQL% /numericc
?	Float (double precision, 8 bytes)	[3]	DOUBLE FLOAT (http://publib.boulder.ibm.com /infocenter/db2luw /v9/index.jsp?topic= /com.ibm.db2.udb.admin.doc /doc/r0008469.htm)	DOUBLE PRECISION	float float8 double pr float(53) (/Ingres /9.3/SQL% /numericc
?	Fixed precision (*p*) and scale (*s*) number	[1]	DECIMAL(p, s) NUMERIC(p, s) (http://publib.boulder.ibm.com /infocenter/db2luw /v9/index.jsp?topic=	DECIMAL(p, s) NUMERIC(p, s)	DECIMAL(NUMERIC(

			/com.ibm.db2.udb.admin.doc /doc/r0008469.htm)		
?	Fixed precision *p* and *s* limits	no limits	p ≤ 31, 0 ≤ s ≤ p (http://publib.boulder.ibm.com /infocenter/db2luw /v9/index.jsp?topic= /com.ibm.db2.udb.admin.doc /doc/r0008469.htm)	p ≤ 18, 0 ≤ s ≤ p	p ≤ 31, 0
?	UNSIGNED support	No	No	No	
?	Set display width in integer	No	No	No	

1. SMALLINT, INTEGER, INT, BIGINT, DECIMAL and NUMERIC. See: Wikibook SQL
2. uses dynamic typing (http://www.sqlite.org/datatype3.html); allows any type name
3. FLOAT, REAL, DOUBLE PRECISION. See: Wikibook SQL

Data structure definition/Data types/Character types

Character types

SQL version	Feature	Standard SQL:2011	DB2	Firebird	Ingres	Linter
?	Variable-length with limit	VARCHAR(n)	VARCHAR(n)	VARCHAR(n)	VARCHAR(n) NVARCHAR(n)	VARCHAR(r NVARCHAR
?	Variable-length with limit, max size	no limit	?	32765	?	?
?	Fixed-length, blank padded string	CHAR(n)	CHAR(n)	CHAR(n)	CHAR(n) NCHAR(n)	CHAR(n) NCHAR(n)
?	Fixed-length, blank padded string, max size	no limit	?	32767	?	?

Notes:

1. uses dynamic typing (http://www.sqlite.org/datatype3.html); allows any type name

Data structure definition/Data types/Date and time types

Date and time types

SQL version	Feature	Standard SQL:2011	DB2	Firebird	Ingres
?	Date only	DATE	DATE	DATE	DATE ANSIDATE INGRESDATE (http://docs.ingres.com/Ingres/9.3/SQL%20Reference%20Gui/datetimedatatypes.htm#o205
?	Time only	TIME	TIME	TIME	TIME [WITHOUT TIME ZONE] (http://docs.ingres.com/Ingres/9.3/SQL%20Reference%20Gui/datetimedatatypes.htm#o209
?	Time with time zone	TIME WITH TIME ZONE	?	?	TIME WITH TIME ZONE (http://docs.ingres.com/Ingres/9.3/SQL%20Reference%20Gui/datetimedatatypes.htm#o209
?	Date and time without time zone	TIMESTAMP	TIMESTAMP	TIMESTAMP	DATE TIMESTAMP ANSIDATE INGRESDATE
?	Date and time with time zone	TIMESTAMP WITH TIME ZONE	?	?	?
?	Time interval	INTERVAL DAY TO SECOND(n) INTERVAL YEAR TO MONTH	?	?	INTERVAL DAY TO SECOND(n) INTERVAL YEAR TO MONTH (http://docs.ingres.com/Ingres/9.3/SQL%20Reference%20Gui/datetimedatatypes.htm#o207

Data structure definition/Data types/Large object types

Large object types

SQL version	Feature	Standard SQL:2011	DB2	Firebird	Ingres	Linter	MS
?	Character LOBs	CLOB(n)	CLOB(n)	BLOB SUB_TYPE TEXT	CLOB, TEXT, LONG VARCHAR	BLOB	varchar(max) (http://msdn2 /en-us/library /ms176089.a nvarchar(max (http://msdn2 /en-us/library /ms186939.a text (http://msdn2 /en-us/library /ms187993.a ntext (http://msdn2 /en-us/library /ms187993.a
?	Character LOBs max size, bytes	no limit	?	about 4 GB	?	?	$2^{31}-1$ (http://msdn2 /en-us/library /ms178158.a
?	Binary LOBs	BLOB(n)	BLOB(n)	BLOB SUB_TYPE BINARY	BLOB LONG BYTE	BLOB	varbinary(ma (http://msdn2 /en-us/library /ms188362.a image (http://msdn2 /en-us/library /ms187993.a
?	Binary LOBs max size, bytes	no limit	?	about 4 GB	?	?	$2^{31}-1$ (http://msdn2 /en-us/library /ms178158.a

Notes:

1. the built-in date/time functions (http://www.sqlite.org/lang_datefunc.html) can use these types for storing values

Data structure definition/Data types/Misc types

Misc types

SQL version	Feature	Standard SQL:2011	DB2	Firebird	Ingres	Linter	MSSQL
?	Boolean	boolean	N/A	N/A	?	BOOLEAN	BIT (http://msdn2.micr /en-us/library /ms177603.aspx)
?	Boolean storage size	N/A	?	?	?	?	8 columns in a sing = 1 byte (http://msdn2.micr /en-us/library /ms177603.aspx)
?	UUID	N/A	N/A	?	?	?	uniqueidentifier (http://msdn2.micr /en-us/library /ms187942.aspx)
?	Row version	N/A	?	?	?	?	TIMESTAMP ROWVERSION (http://msdn2.micr /en-us/library /ms182776.aspx)
?	Any data type	N/A	?	?	?	?	SQL_VARIANT (http://msdn2.micr /en-us/library /ms173829.aspx)

Data structure definition/Delimited identifiers

Delimited identifiers

There are **regular identifiers** and **delimited identifiers** to denominate database objects like tables or

columns. You must apply delimited identifiers if you want to use SQL keywords like 'FROM' as identifiers or special characters within its name. Delimited identifiers are labeled by the following techniques.

Dialect	Sample	Comment
Standard	`"identifier"`	The SQL standard calls these "delimited identifiers". The SQL standard requires that delimited identifiers are case-sensitive, although not all servers enforce this.
DB2 (http://publib.boulder.ibm.com/infocenter/db2luw/v9/index.jsp?topic=/com.ibm.db2.udb.admin.doc/doc/r0000720.htm)	`"identifier"`	Case sensitive per standard and also allows special characters.
Firebird	`"identifier"`	Case sensitive per standard.
Ingres	`"identifier"`	Case sensitivity for delimited identifiers is specified when a database is created. For compliance with ANSI/ISO Entry SQL-92, delimited identifiers must be case sensitive.
Linter	`"identifier"`	Case sensitive per standard.
MonetDB	`"identifier"`	Case sensitive per standard.
MSSQL	`[identifier]` `"identifier"`[1]	Case sensitivity is set by a per-database option.
MySQL	`` `identifier` `` `"identifier"`[2]	
Oracle	`"identifier"`	Case sensitive per standard. Case insensitive when not double-quoted: They are converted to uppercase before the execution of any statement, e.g: 'CrEAtE table MyTable' is converted to 'CREATE TABLE MYTABLE'. This conforms to the standard.
PostgreSQL	`"identifier"`	Case sensitive per standard. Case insensitive when not double-quoted (converted to lower case - while it's upper case per standard).
SQLite (http://www.sqlite.org/lang_keywords.html)	`[identifier]` `"identifier"` `'identifier'` `identifier`	Never case sensitive, even when quoted. Bracketed keywords are always understood as identifiers. Double-quoted keywords are understood as identifiers if previously seen as such, but are otherwise

		interpreted as string literals. Single-quoted keywords are interpreted as string literals where such are allowed, but otherwise as identifiers. Some keywords may be used as identifiers even when not quoted.
Virtuoso	`"identifier"`	

Notes:

1. If the **quoted_identifier** option is set **on** — otherwise, `"identifier"` would be parsed as a string in single quotes.
2. If running in ANSI mode.

Data structure definition/Auto-increment column

A short hint: In most cases auto-increment columns are used as Primary Key columns. In the SQL standard the junction of the two concepts is not mandatory.

SQL Standard

The SQL standard defines two ways to generate auto-increment values. First, there are *identity columns* as an extention to exact numeric types. The syntax is: "GENERATED { ALWAYS | BY DEFAULT } AS IDENTITY". Second, the use of sequences in combination with triggers is standardized.

```
CREATE TABLE t1 (col1 DECIMAL GENERATED ALWAYS AS IDENTITY);
```

DB2

Identity columns or sequences combined with triggers (comparison (http://publib.boulder.ibm.com/infocenter/db2luw/v8/topic/com.ibm.db2.udb.doc/admin/c0004994.htm) of both techniques).

```
CREATE TABLE t1 (col1 INT GENERATED ALWAYS AS IDENTITY);

--  or:

CREATE TABLE t1 (col1 INT);
CREATE SEQUENCE sequence_name;
CREATE TRIGGER insert_trigger
       NO CASCADE BEFORE INSERT ON t1
       REFERENCING NEW AS n
       FOR EACH ROW
```

```
  SET n.col1 = NEXTVAL FOR sequence_name;
```

Firebird

Is recommended to use sequences (http://www.firebirdsql.org/manual/generatorguide-sqlsyntax.html#generatorguide-sqlsyntax-overview) combined with triggers . From 3.0 there is Identity (https://github.com/FirebirdSQL/firebird/blob/master/doc/sql.extensions/README.identity_columns.txt) support.

```
SET TERM  ^;
CREATE TABLE t1(col1 INTEGER NOT NULL PRIMARY KEY)^
CREATE SEQUENCE sequence_name^
ALTER  SEQUENCE sequence_name RESTART WITH 0^

CREATE TRIGGER trigger_name FOR t1
BEFORE INSERT
AS
BEGIN
  NEW.col1 = NEXT VALUE FOR sequence_name;
END^
```

Linter

AUTOINC columns (maybe with RANGEs) or sequences combined with triggers.

```
CREATE TABLE t1 (col1 SMALLINT AUTOINC);
CREATE TABLE t2 (col1 INTEGER AUTOINC);
CREATE TABLE t3 (col1 BIGINT AUTOINC);
```

MonetDB

```
CREATE SEQUENCE sequence_name AS INT START WITH 1 INCREMENT BY 1;
CREATE TABLE t1 (
  col1 INT PRIMARY KEY DEFAULT NEXT VALUE FOR sequence_name,
  col2 INT AUTO_INCREMENT,
  col1 INT GENERATED ALWAYS AS IDENTITY (
           START WITH 100 INCREMENT BY 2
           NO MINVALUE MAXVALUE 1000
           CACHE 2 CYCLE)
);
```

MSSQL

```
CREATE TABLE t1 (col1 INT IDENTITY(1,1));
```

MySQL

```
CREATE TABLE t1 (col1 INT NOT NULL PRIMARY KEY AUTO_INCREMENT);
```

OpenLink Virtuoso

```
IDENTITY (start with 1, increment by 1);
CREATE TABLE t1 (col1 INTEGER IDENTITY);

-- or:

CREATE TABLE t1 (col1 INTEGER IDENTITY (start with 1));
```

Oracle

```
CREATE TABLE t1 (col1 NUMBER PRIMARY KEY);
CREATE SEQUENCE sequence_name START WITH 1 INCREMENT BY 1;
CREATE OR REPLACE TRIGGER trigger_name BEFORE INSERT ON t1 FOR EACH ROW
DECLARE
  max_id NUMBER;
  cur_seq NUMBER;
BEGIN
IF :NEW.col1 IS NULL THEN
  -- normal assignment of the next value in the sequence
  :NEW.col1 := sequence_name.NEXTVAL;
ELSE
  -- or allow the user to specify the value, so must advance the sequence to match
  SELECT GREATEST(COALESCE(MAX(col1), 0), :NEW.col1) INTO max_id FROM t1;
  WHILE cur_seq < max_id LOOP
    SELECT sequence_name.NEXTVAL INTO cur_seq FROM DUAL;
  END LOOP;
END IF;
END;

-- since Oracle 12.1:
CREATE TABLE t1 (col1 NUMBER GENERATED BY DEFAULT AS IDENTITY);
```

PostgreSQL

```
CREATE TABLE t1 (col1 SERIAL PRIMARY KEY);
```

SQLite

Both create an autoincrementing column (http://www.sqlite.org/autoinc.html); the AUTOINCREMENT keyword only prevents reusing deleted values.

```
CREATE TABLE t1 (col1 INTEGER PRIMARY KEY);
CREATE TABLE t1 (col1 INTEGER PRIMARY KEY AUTOINCREMENT);
```

Functions and expressions/Math functions/Trigonometric functions

Trigonometric functions

SQL version	Feature	Standard SQL:2011	DB2	Firebird	Ingres (/9.3/SQL% /num
?	Arc sine	N/A	ASIN(x) (http://publib.boulder.ibm.com/infocenter/db2luw/v9/index.jsp?topic=/com.ibm.db2.udb.admin.doc/doc/r0000771.htm)	ASIN(x)	ASIN(X)
?	Arc cosine	N/A	ACOS(x) (http://publib.boulder.ibm.com/infocenter/db2luw/v9/index.jsp?topic=/com.ibm.db2.udb.admin.doc/doc/r0000769.htm)	ACOS(x)	ACOS(x)
?	Arc tangent of x	N/A	ATAN(x) (http://publib.boulder.ibm.com/infocenter/db2luw/v9/index.jsp?topic=/com.ibm.db2.udb.admin.doc/doc/r0000772.htm)	ATAN(x)	ATAN(x)
?	Arc tangent of x and y	N/A	ATAN2(x, y) (http://publib.boulder.ibm.com/infocenter/db2luw/v9/index.jsp?topic=/com.ibm.db2.udb.admin.doc/doc/r0000773.htm)	ATAN2(x,y)	ATAN2(x, y
?	Sine	N/A	SIN(x) (http://publib.boulder.ibm.com/infocenter/db2luw/v9/index.jsp?topic=/com.ibm.db2.udb.admin.doc/doc/r0000849.htm)	SIN(x)	SIN(x)
?	Cosine	N/A	COS(x) (http://publib.boulder.ibm.com/infocenter/db2luw/v9/index.jsp?topic=/com.ibm.db2.udb.admin.doc/doc/r0000782.htm)	COS(x)	COS(x)
?	Tangent	N/A	TAN(x) (http://publib.boulder.ibm.com/infocenter/db2luw/v9/index.jsp?topic=/com.ibm.db2.udb.admin.doc/doc/r0000857.htm)	TAN(x)	TAN(x)
?	Cotangent	N/A	COT(x) (http://publib.boulder.ibm.com/infocenter/db2luw/v9/index.jsp?topic=	COT(x)	

			/com.ibm.db2.udb.admin.doc /doc/r0000783.htm)	
?	Hyperbolic sine	N/A	SINH(x) (http://publib.boulder.ibm.com /infocenter/db2luw /v9/index.jsp?topic= /com.ibm.db2.udb.admin.doc /doc/r0007105.htm)	SINH(x)
?	Hyperbolic cosine	N/A	COSH(x) (http://publib.boulder.ibm.com /infocenter/db2luw /v9/index.jsp?topic= /com.ibm.db2.udb.admin.doc /doc/r0007104.htm)	COSH(x)
?	Hyperbolic tangent	N/A	TANH(x) (http://publib.boulder.ibm.com /infocenter/db2luw /v9/index.jsp?topic= /com.ibm.db2.udb.admin.doc /doc/r0007106.htm)	TANH(x)
?	Hyperbolic arctangent	N/A	ATANH(x) (http://publib.boulder.ibm.com /infocenter/db2luw /v9/index.jsp?topic= /com.ibm.db2.udb.admin.doc /doc/r0007103.htm)	ATANH(x)

Functions and expressions/Math functions/Numeric functions

Numeric functions

SQL version	Feature	Standard SQL:2011	DB2	Firebird	Ingres (http://docs /Ingres /9.3/SQL%20Referer /numericfunctio
?	Absolute value of *x*	ABS(*x*)	ABS(*x*)	ABS(*x*)	ABS(*x*)
?	Sign of number *x*	N/A	SIGN(*x*)	SIGN(*x*)	SING(*x*)
?	Modulus (remainder) of x/y	MOD(*x*, *y*)	MOD(*x*, *y*)	MOD(*x*, *y*)	MOD(*x*, *y*)
?	Smallest integer >= *x*	CEILING(*x*) CEIL(*x*)	CEILING(*x*) CEIL(*x*)	CEILING(*x*) CEIL(*x*)	CEIL(*x*) CEILING(*x*)
?	Largest integer <= *x*	FLOOR(*x*)	FLOOR(*x*)	FLOOR(*x*)	FLOOR(*x*)
?	Round *x* (to precision of *d* digits)	N/A	ROUND(*x*, *d*)	ROUND(*x*, *d*)	ROUND(*x*, *d*)
?	Truncate *x* to *n* decimal places	N/A	TRUNCATE(x, n) TRUNC(x, n)	TRUNC(x[, n])	TRUNCATE(x, n) TRUNC(x, n)
?	Square root of *x* ($\sqrt{x}$)	SQRT(*x*)	SQRT(*x*)	SQRT(*x*)	SQRT(*x*)
?	Exponent of *x* (e^x)	EXP(*x*)	EXP(*x*)	EXP(*x*)	EXP(*x*)
?	Power (x^y)	POWER(*x*, *y*)	POWER(*x*, *y*)	POWER(*x*, *y*)	POWER(*x*, *y*) *x* ** *y*

?	Natural logarithm of *x*	LN(*x*)	LN(*x*)	LN(*x*)	LOG(*x*) LN(*x*)
?	Logarithm of *x*, base *b*	N/A	LOG(*b*, *x*)	LOG(*b*, *x*)	N/A
?	Logarithm, base 10	N/A	LOG10(*x*)	LOG10(*x*)	N/A
?	Randomize, set seed to *x*	N/A	RAND(*x*)	N/A	SET RANDOM_SEED *x* (http://docs.ingres.com/9.3/SQL%20Reference/randomnumberfunctio
?	Generate floating-point random number between 0 and 1	N/A	RAND()	RAND()	RANDOMF() (http://doc/Ingres/9.3/SQL%20Reference/randomnumberfunctio
?	Highest number in a *list*	MAX(*list*)	N/A	MAXVALUE(*list*)	?
?	Lowest number in a *list*	MIN(*list*)	N/A	MINVALUE(*list*)	?

Notes

1. The random() function in Oracle can be found in the built-in DBMS package dbms_random.

Functions and expressions/Math functions/Aggregate functions

Aggregate functions

Aggregate function operate on a group of values, returning single scalar value. The standard specifies that with the exception of VAR_POP, VAR_SAMP, STDDEV_POP and STDDEV_SAMP all aggregate function should be able to handle one of two additional quantifier before an argument: **ALL** (feature ID E091-06)

and **DISTINCT** (feature ID E091-07). **ALL** is the default and can be omitted and **DISTINCT** means that only unique values would be passed in aggregate function. To make presentation more compact, these two quantifiers aren't discussed separately for every function, but specified as **[DISTINCT|ALL]** if function supports both of them.

SQL version	Feature	Standard SQL:2011	DB2	Firebird
?	Count all rows in a group	COUNT(*) ... GROUP BY <grouping criterion>	COUNT(*) (http://publib.boulder.ibm.com/infocenter/db2luw/v9/index.jsp?topic=/com.ibm.db2.udb.admin.doc/doc/r0000759.htm)	COUNT(*)
?	Count non-NULL values in *x*	COUNT(x)	COUNT([DISTINCT] x)	COUNT([DISTIN x)
?	Concatenate non-NULL values in *x* using *y* as separator	N/A	?	LIST([DISTINCT y)
?	Sum of *x*	SUM(x)	SUM([DISTINCT] x)	SUM(x)
?	Average of *x*	AVG(x)	AVG([DISTINCT] x)	AVG(x)
?	Minimum value in *x*	MIN(x)	MIN(x)	MIN(x)
?	Maximum value in *x*	MAX(x)	MAX(x)	MAX(x)
?	Standard deviation	STDDEV_POP(x) STDDEV_SAMP(x)	STDDEV([DISTINCT] x)	STDDEV_POP(x STDDEV_SAMP(
?	Variance	VAR_POP(x) VAR_SAMP(x)	VARIANCE([DISTINCT] x)	VAR_POP(x) VAR_SAMP(x)

?	Population covariance of *x* and *y*	COVAR_POP(x, y)	COVARIANCE(x, y) COVAR(x, y) (http://publib.boulder.ibm.com/infocenter/db2luw/v9/index.jsp?topic=/com.ibm.db2.udb.admin.doc/doc/r0002320.htm)	COVAR_POP(x,
?	Sample covariance of *x* and *y*	COVAR_SAMP(x, y)	N/A	COVAR_SAMP(x

1. Can be implemented using user-defined aggregator functions or several other approaches[1] (http://www.simple-talk.com/sql/t-sql-programming/concatenating-row-values-in-transact-sql/)
2. Can be implemented using user-defined aggregator function, undocumented **wmsys.wm_concat** function and using several other approaches[2] (http://www.williamrobertson.net/documents/one-row.html)
3. Can be implemented for older versions using PostgreSQL arrays and user-defined aggregator functions[3] (http://mssql-to-postgresql.blogspot.com/2007/12/cool-groupconcat.html)

Functions and expressions/Date and time functions

Date and time functions

SQL version	Feature	Standard SQL:2011	DB2	Fireb
?	Current date	CURRENT_DATE	CURRENT DATE CURRENT_DATE (http://publib.boulder.ibm.com/infocenter/db2luw/v9/index.jsp?topic=/com.ibm.db2.udb.admin.doc/doc/r0005870.htm)	CURRENT_D
?	Current time	CURRENT_TIME	CURRENT TIME CURRENT_TIME (http://publib.boulder.ibm.com/infocenter/db2luw/v9/index.jsp?topic=/com.ibm.db2.udb.admin.doc/doc/r0005885.htm)	CURRENT_TI
?	Current date and time	CURRENT_TIMESTAMP	CURRENT TIMESTAMP CURRENT_TIMESTAMP (http://publib.boulder.ibm.com/infocenter/db2luw/v9/index.jsp?topic=/com.ibm.db2.udb.admin.doc/doc/r0005886.htm)	CURRENT_TI 'NOW'
?	Date addition	date + arg	date + arg	arg1 + arg2
?	Date subtraction	date - arg	date - arg	arg1 - arg2
?	Date difference	(date1 - date2) field	date1 - date2	arg1 - arg2 DATEDIFF
?	Last day of month	?	date + 1 MONTH - DAY(date) DAYS	LASTDAYMO
?	Time zone conversion	date AT TIME ZONE offset	?	N/A
?	First weekday after date	?	?	N/A
?	Convert date to string	CAST (x AS STRING)	TO_CHAR(value, format) VARCHAR_FORMAT(value, format)	CAST(value, DATETOSTR

?	Convert date to number	N/A	INT(date)	EXTRACT
?	Convert string to date	CAST (x AS DATE)	DATE(value) TIMESTAMP(value)	CAST
?	Extract year from DATE or DATETIME *x*	EXTRACT(YEAR FROM *x*)	?	EXTRACT(YE *x*)
?	Extract month from DATE or DATETIME *x*	EXTRACT(MONTH FROM *x*)	?	EXTRACT(M(FROM *x*)
?	Extract day of month from DATE or DATETIME *x*	EXTRACT(DAY FROM *x*)	?	EXTRACT(DA *x*)
?	Extract hour (0…23) from TIME or DATETIME *x*	EXTRACT(HOUR FROM *x*)	?	EXTRACT(HC *x*)

1. These functions are only applicable to **ingresdate** data type in Ingres; normal dates, represented by **ansidate** datatype should be converted to **ingresdate** first.

Functions and expressions/String functions

String functions

Function	Since SQL	Standard	DB2	SQ (http://ww /corefu
Convert character *x* to ASCII	N/A	N/A	ASCII(x) (http://publib.boulder.ibm.com /infocenter/db2luw /v9/index.jsp?topic= /com.ibm.db2.udb.admin.doc /doc/r0000770.htm)	UNICODE(x
Convert ASCII *x* to character	N/A	N/A	CHR(x) (http://publib.boulder.ibm.com /infocenter/db2luw /v9/index.jsp?topic= /com.ibm.db2.udb.admin.doc /doc/r0000778.htm)	CHAR(x)
String concatenate	92	arg1 || arg2	arg1 || arg2 arg1 CONCAT arg2	arg1 || arg2
Find first occurrence of substring *search* in *str*, starting from *start*	92	SUBSTRING_REGEX (search IN str FROM start)	LOCATE(search, str[, start]) POSSTR(str, search)	INSTR(str, s
Find first occurrence of pattern *search* in string *str*	2003	SUBSTRING_REGEX (search IN str)		
Convert *x* to lowercase	92	LOWER(x)	LOWER(x) LCASE(x) (http://publib.boulder.ibm.com /infocenter/db2luw /v9/index.jsp?topic= /com.ibm.db2.udb.admin.doc /doc/r0002411.htm)	LOWER(x)

Convert *x* to uppercase	92	UPPER(x)	UPPER(x) UCASE(x) (http://publib.boulder.ibm.com/infocenter/db2luw/v9/index.jsp?topic=/com.ibm.db2.udb.admin.doc/doc/r0000867.htm)	UPPER(x)
Pad left side	2003			
Pad right side	2003			
Remove leading blank spaces from *x*	92	TRIM(LEADING [' '] FROM x)	N/A	LTRIM(x)
Remove trailing blank spaces from *x*	92	TRIM(TRAILING [' '] FROM x)	N/A	RTRIM(x)
Remove leading and trailing blanks from *x*	92	TRIM(BOTH [' '] FROM x) TRIM(x)	LTRIM(RTRIM(x)) or TRIM(x)	TRIM(x)
Repeat *str n* times	2003		REPEAT(str, n)	
String of *n* spaces	2003		SPACE(n)	
Convert number to string	2003		CHAR(num)	CAST
Substring from string *str*, starting from *start*, length of *len*	92	SUBSTRING(str FROM start [FOR len])	SUBSTR(str, len[, start]) (http://publib.boulder.ibm.com/infocenter/db2luw/v9/index.jsp?topic=/com.ibm.db2.udb.admin.doc/doc/r0000854.htm)	SUBSTR(str
Replace characters			REPLACE(string, from, to)	REPLACE(st
Capitalize first letter of each word in string *x*	N/A	N/A	INITCAP(x)[1]	
Translate string			TRANSLATE(str, to, from) (http://publib.boulder.ibm.com/infocenter/db2luw/v9/index.jsp?topic=	

			/com.ibm.db2.udb.admin.doc /doc/r0000862.htm)	
Length of string *x* (in characters)	92	CHAR_LENGTH(x) CHARACTER_LENGTH(x)	LENGTH(x)	LENGTH(x)
Length of string *x* (in bytes)	92	OCTET_LENGTH(x)	LENGTH(x)	LENGTH(CA BLOB))
Greatest character string in list	2003			MAX
Least character string in list	2003			MIN
Quote SQL in string *x*				QUOTE(x)
Soundex index of string *x*			SOUNDEX(x) (http://publib.boulder.ibm.com /infocenter/db2luw /v9/index.jsp?topic= /com.ibm.db2.udb.admin.doc /doc/r0000851.htm)	SOUNDEX(x
Calculate MD5 hash from string *x*	N/A	N/A		
Calculate SHA1 hash from string *x*	N/A	N/A		
Generate UUID	N/A	N/A		

Notes

1. INITCAP is supported starting DB2 V9.7.
2. Soundex function is omitted from SQLite by default. Only available if SQLite is built with `-DSQLITE_SOUNDEX=1` compile-time option.

3. MySQL uses original Soundex algorithm.
4. Uses enhanced Soundex algorithm as defined in The Art of Computer Programming, Volume 3: Sorting and Searching, by Donald E. Knuth.

Functions and expressions/Misc expressions

Misc functions

SQL version	Feature	Standard SQL:2011	DB2	Firebird	Ingres	Li
?	Convert value *val* to data type *type*	CAST(*val* AS *type*)	?	CAST(*val* AS *type*)	?	CAST(*type*) CASTe as typ TO_CH TO_NU TO_DA
?	Replace NULL within a *val* with a *fallback* value, return *val* intact, if it's non-NULL	COALESCE(*val*, *fallback*)	?	?	?	NVL(*v fallbac*
?	Return the first non-NULL value from a list of values (*val1*, *val2*, ...)	COALESCE(*val1*, *val2*, ...)	COALESCE(val1, val2)	COALESCE(*val1*, *val2*, ...)	?	COALE val2, .
?	Return NULL if two values *a* and *b* are equal	NULLIF(*a*, *b*)	?	NULLIF(*a*, *b*)	?	NULLI

Select queries/Subqueries

Subqueries in FROM clause

Standard	?
DB2	?
Firebird	?
Ingres	?
Linter	?
MonetDB	?
MSSQL	▪ SELECT *expressions* FROM (SELECT *subquery expressions* FROM *subquery tables*) ***subquery_alias*** WHERE *conditions* ▪ Subquery alias is mandatory ▪ WITH *subquery_alias* AS (SELECT *subquery expressions* FROM *subquery tables*) SELECT *expressions* FROM subquery_alias WHERE *conditions*
MySQL	?
Oracle	SELECT *expressions* FROM (SELECT *subquery expressions* FROM *subquery tables*) WHERE *conditions*
PostgreSQL	?
SQLite	▪ SELECT ... FROM (SELECT ...) [AS *alias*] WHERE ... ▪ WITH (http://www.sqlite.org/lang_with.html) *cte* AS (SELECT ...) SELECT ... FROM *cte* WHERE ...
Virtuoso	?

Select queries/Select without tables

Select without tables

Sometimes one needs to execute SQL scalar expressions without table context, i.e. make a query that would act as normal SELECT operator, evaluate given comma-separated *expressions* and return a table with single row and one or multiple columns (one for every individual expression). Obviously, *expressions* can't reference any columns from the tables, as there are none.

An example is determining the value of a mathematical function, using Oracle syntax:

```
SQL> select 4*atan(1) as "Arc tangent of 1 times 4" from dual;

Arc tangent of 1 times 4
------------------------
              3.14159265
```

Standard	?
DB2	■ VALUES ■ SELECT *expressions* FROM sysibm.sysdummy1
Firebird	SELECT *expressions* FROM rdb$database
Ingres	SELECT *expressions*
Linter	SELECT *expressions*
MonetDB	SELECT *expressions*
MSSQL	SELECT *expressions*
MySQL	■ SELECT *expressions* ■ SELECT *expressions* FROM dual
Oracle	SELECT *expressions* FROM dual
PostgreSQL	■ SELECT *expressions* ■ VALUES (*expressions*) (http://www.postgresql.org/docs/8.2/interactive/sql-values.html)
SQLite	SELECT *expressions*
Virtuoso	?

Select queries/Limiting results of select query

Limiting results of select query

Note that *end_row = start_row + num_rows - 1*

Standard	SELECT columns FROM table FETCH FIRST num_rows ROWS ONLY SELECT columns FROM table OFFSET start_row FETCH FIRST num_rows ROWS (
DB2	SELECT columns FROM table FETCH FIRST num_rows ROWS ONLY

Firebird	Versions > 2.0
	`SELECT columns FROM table ROWS start_row TO end_row`
	All versions
	`SELECT FIRST num_rows SKIP start_row columns FROM table`

Ingres	■ SELECT FIRST *num_rows columns* FROM *table* (http://docs.ingres.com/Ingre /9.3/SQL%20Reference%20Guide/selectinteractive.htm#o1349) ■ SELECT *columns* FROM *table* OFFSET *start_row* FETCH {FIRST|NEXT} *num_r* (http://docs.ingres.com/Ingres/9.3/SQL%20Reference%20Guide/selectintera

Linter	`SELECT columns FROM table FETCH FIRST num_rows`
	`SELECT columns FROM table LIMIT start_row, num_rows`
	`(rows are numbered from 0)`

MonetDB	SELECT columns FROM table LIMIT num_rows OFFSET start_row

MSSQL	Simple version (when start_row = 1)
	`SELECT TOP num_rows columns FROM table`
	Complex version (full-featured, requires ordering)
	`SELECT * FROM (`
	`  SELECT TOP num_rows columns FROM (`
	`  SELECT TOP num_rows + start_row columns FROM table ORDER BY some_key ASC`
	`  ) AS newtable ORDER BY some_key DESC`
	`) AS newtable2 ORDER BY some_key ASC`

MySQL	Versions > 4.0.14
	`SELECT columns FROM table LIMIT num_rows OFFSET start_row`
	All versions
	`SELECT columns FROM table LIMIT start_row, num_rows`

Oracle	Simple query `-- Notice: Will not work, if start_row > 1, since the first row will return false,`

```
SELECT columns FROM table WHERE rownum >= start_row AND rownum <= end_row
```

This works fine without specifying start_row:

```
SELECT columns FROM table WHERE rownum <= end_row
```

This also works, but suffix num column in resultset:

```
SELECT * FROM (
  SELECT temp.*, rownum num FROM table ORDER BY columns
  ) WHERE num >= start_row and num <= end_row
```

Since 12.1, similar to standards:

```
SELECT columns FROM table FETCH FIRST num_rows ROWS ONLY
SELECT columns FROM table OFFSET start_row ROWS FETCH FIRST num_rows ROWS ONLY
```

Full syntax: (Can have OFFSET, or FETCH, or both clauses)

```
SELECT column [, column2 …] FROM table [ORDER BY column2]
[OFFSET start_row ROWS]
[FETCH [FIRST|NEXT] [num_rows|percent PERCENT] ROWS [ONLY|WITH TIES]]
```

PostgreSQL

SELECT columns FROM table LIMIT num_rows OFFSET start_row

or: The SQL standard syntax.

SQLite

SELECT columns FROM table LIMIT num_rows OFFSET start_row

Virtuoso

```
SELECT TOP num_rows columns FROM table
SELECT TOP skip_rows,num_rows columns FROM table
```

Standard	-
DB2	?
Firebird	?
Ingres	?
Linter	?
MonetDB	?
MSSQL	NOT IN()
MySQL	NOT IN()
Oracle	MINUS
PostgreSQL	?
SQLite	?
Virtuoso	?

Select queries/Hierarchical Queries

Hierarchical Queries

Hierarchical queries are a way to extract information from a table that is linked with itself.

Let's say we have the following table:

```
My example table:

id     of type numeric
father of type numeric, that references an id of other register of the same table
data   rest of fields, etc
```

If we have the following values:

```
id     father        data
50     null          The boss
51     50            The well positioned manager
52     50            Another well positioned manager
53     51            The worker
54     52            Another worker
5      null          Other node
10     5             The son of Other node
```

The values that "hang" from node 50 are the values 50, 51, 52, 53, 54 but not 5 nor 10.

- DB2

or

- Firebird / InterBase
- Ingres, MySQL, MSSQL[1]
- PostgreSQL
- SQLite (http://www.sqlite.org/lang_with.html)

```
WITH RECURSIVE t AS (
    SELECT id, father FROM "table" WHERE id = 50 AND father IS NULL
UNION ALL
    SELECT t1.id, t1.father FROM t JOIN "table" t1 ON (t1.father = t.id)
)
SELECT * FROM t;
```

- Oracle, Linter

```
SELECT * FROM table CONNECT BY id = PRIOR father START WITH id = 50
```

1. MS SQL does **not** allow the RECURSIVE keyword

Write queries/Replace query

Replace query

Replace query inserts new row if no row with such primary key exists or updates existing row if it does. SQL:2003 standard introduced a MERGE statement to implement such functionality, while other implementations provide similar queries named "REPLACE" or so-called "Upsert" query (a portmanteau of UPDATE and INSERT).

Standard	MERGE statement can be used to do a replace query:
	`MERGE INTO table_name1 USING table_name2 ON (condition)`
	`WHEN MATCHED THEN UPDATE SET column1 = value1 [, column2 = value2 ...]`
	`WHEN NOT MATCHED THEN INSERT columns VALUES (values)`
	Note that MERGE is much more powerful than just doing replace queries.

DB2	MERGE statement
	`MERGE INTO phonebook AS p`
	`  USING ( VALUES ('john doe', '1234' ) ) AS v(name, extension)`
	`  ON ( p.name = v.name )`
	`  WHEN MATCHED`
	`  UPDATE SET p.extension = v.extension`
	`  WHEN NOT MATCHED`
	`  INSERT VALUES ( v.name, v.extension )`

Firebird	MERGE statement
	`MERGE INTO phonebook B`
	`USING (`
	`  SELECT name`
	`  FROM phonebook`
	`  WHERE name = 'john doe') E`
	`ON (B.name = E.name)`
	`WHEN MATCHED THEN`
	`  UPDATE SET B.extension = '1234'`
	`WHEN NOT MATCHED THEN`
	`  INSERT (name, extension)`
	`  VALUES ('john doe', '1234');`
	Non-standard simplified form:
	`UPDATE OR INSERT INTO phonebook (name, extension)`
	`VALUES ('john doe', '1234')`
	`MATCHING (name)`

Ingres	?
Linter	?
MonetDB	?

MSSQL	MERGE statement (from version **SQL Server 2008**)
	`DECLARE @UnitMeasureCode nchar(3) = 'ABC'`
	`DECLARE @Name varchar(25) = 'Test name'`
	`MERGE INTO Production.UnitMeasure AS target`
	`  USING (SELECT @UnitMeasureCode, @Name) AS source (UnitMeasureCode, Name)`
	`  ON (target.UnitMeasureCode = source.UnitMeasureCode)`
	`  WHEN MATCHED THEN`
	`  UPDATE SET Name = source.Name`
	`  WHEN NOT MATCHED THEN`
	`  INSERT (UnitMeasureCode, Name)`
	`  VALUES (source.UnitMeasureCode, source.Name)`

MySQL

Allows 3 syntaxes: non-standard REPLACE query, (since 4.1) INSERT ... ON DUP
UPDATE, and a variant on IF EXISTS.

```
REPLACE [INTO] table [(columns)] VALUES (values)
```

```
INSERT INTO table (columns) VALUES (values) ON DUPLICATE KEY UPDATE column1=value1
```

```
IF EXISTS( SELECT * FROM phonebook WHERE name = 'john doe' )
THEN UPDATE phonebook SET extension = '1234' WHERE name = 'john doe'
ELSE INSERT INTO phonebook VALUES( 'john doe','1234' )
END IF
```

Oracle

MERGE statement

```
MERGE INTO phonebook B
USING (
  SELECT name_id
  FROM phonebook
  WHERE name = 'john doe') E
ON (B.name = E.name)
WHEN MATCHED THEN
  UPDATE SET B.extension = '1234'
WHEN NOT MATCHED THEN
  INSERT (B.name, B.extension)
  VALUES ('john doe', '1234');
```

- Multi-statement form:

```
IF EXISTS( SELECT * FROM phonebook WHERE name = 'john doe' )
UPDATE phonebook SET extension = '1234' WHERE name = 'john doe'
ELSE
INSERT INTO phonebook VALUES( 'john doe','1234' )
```

PostgreSQL

Since version 9.5 INSERT INTO...ON CONFLICT... syntax inspired by MySQL:

```
INSERT INTO distributors (did, dname)
    VALUES (5, 'Gizmo Transglobal'), (6, 'Associated Computing, Inc')
    ON CONFLICT (did) DO UPDATE SET dname = EXCLUDED.dname;
```

SQLite

REPLACE statement:

```
REPLACE [INTO] table [(columns)] VALUES (values)
```

(always deletes the old row)

Virtuoso

?

Write queries/Insert results from select query into existing table

Insert results from select query into existing table

Problem: Insert results from a given select query (*SELECT columns FROM table* ...) into existing table (*existing_table*) of the same or compatible structure (as query result).

Standard	INSERT INTO *existing_table SELECT columns FROM table ...*
DB2	INSERT INTO *existing_table SELECT columns FROM table ...*
Firebird	INSERT INTO *existing_table SELECT columns FROM table ...*
Ingres	?
Linter	INSERT INTO *existing_table SELECT columns FROM table ...*
MonetDB	INSERT INTO *existing_table SELECT columns FROM table ...*
MSSQL	INSERT INTO *existing_table SELECT columns FROM table ...*
MySQL	INSERT INTO *existing_table SELECT columns FROM table ...* (http://dev.mysql.com/doc/refman/5.0/en/insert-select.html)
Oracle	INSERT INTO *existing_table SELECT columns FROM table ...*
PostgreSQL	INSERT INTO *existing_table SELECT columns FROM table ...* (http://www.postgresql.org/docs/8.2/interactive/sql-insert.html)
SQLite	INSERT INTO *existing_table SELECT columns FROM table ...*
Virtuoso	INSERT INTO *existing_table SELECT columns FROM table ...*

Write queries/Save results from select query as new table

Save results from select query as new table

Problem: save results from an arbitrary SELECT query (*SELECT columns FROM table* ...) as a new permanent table *new_table*.

Standard	?
DB2	?
Firebird	?
Ingres	?
Linter	CREATE TABLE new_table AS *SELECT columns FROM table ...*
MonetDB	CREATE TABLE new_table AS SELECT columns FROM table ... WITH DATA (https://www.monetdb.org/Documentation/Manuals/SQLreference/Tables)
MSSQL	*SELECT columns* INTO new_table *FROM table ...*
MySQL	CREATE TABLE new_table [AS] *SELECT columns FROM table ...* (http://dev.mysql.com/doc/refman/5.0/en/create-table.html)
Oracle	CREATE TABLE new_table [AS] *SELECT columns FROM table ...*
PostgreSQL	▪ *SELECT columns* INTO new_table *FROM table ...* ▪ CREATE TABLE new_table [AS] *SELECT columns FROM table ...*
SQLite	CREATE TABLE new_table [AS] *SELECT columns FROM table ...*
Virtuoso	CREATE TABLE new_table [AS] *SELECT columns FROM table ...*

Write queries/Update returning

Update returning

This variant of `UPDATE` statement allows to return the values from the row(s) affected by the update.

Standard	No support in SQL:2003
DB2	`UPDATE ... RETURNING ... INTO ...` since 9.7. Only returns the updated values, see the documentation (https://www-01.ibm.com/support/knowledgecenter/SSEPGG_9.7.0/com.ibm.db2.luw.apdv.plsql.doc/doc/c0053868.html).
Firebird	`UPDATE ... RETURNING ...` since 2.1. Can return both old and new values using `old.` or `new.` prefixes but for at most one row, see the documentation (http://www.firebirdsql.org/refdocs/langrefupd21-update.html#langrefupd21-update-returning).
Ingres	?
Linter	?
MonetDB	?
MSSQL	`UPDATE ... OUTPUT ...` since MS SQL Server 2005. Can return both old and new values using `inserted.` or `deleted.` prefixes, see the documentation (https://msdn.microsoft.com/en-us/library/ms177564.aspx).
MySQL	No support in 5.7.
Oracle	`UPDATE ... RETURNING ... INTO ...` Supports retrieving values of multiple rows with `BULK COLLECT INTO`, see the documentation (http://docs.oracle.com/cd/B28359_01/appdev.111/b28370/returninginto_clause.htm#LNPLS01354).
PostgreSQL	`UPDATE ... RETURNING ...` since 8.2. Only returns the updated values, see the documentation (http://www.postgresql.org/docs/9.4/static/sql-update.html).
SQLite	No support in 3.8.
Virtuoso	?

Transactions

Transactions

SQL version	Feature	Standard SQL:2011	DB2	Firebird	
?	Start	START TRANSACTION [transaction characteristics]	Implicit	SET TRANSACTION	?
?	Commit	COMMIT [WORK]	COMMIT [WORK] (http://publib.boulder.ibm.com/infocenter/db2luw/v9/index.jsp?topic=/com.ibm.db2.udb.admin.doc/doc/r0000903.htm)	COMMIT [WORK]	[E (h /9 /c
?	Rollback whole transaction	ROLLBACK [WORK]	ROLLBACK [WORK] (http://publib.boulder.ibm.com/infocenter/db2luw/v9/index.jsp?topic=/com.ibm.db2.udb.admin.doc/doc/r0000992.htm)	ROLLBACK [WORK]	[E (h /9 /r
?	Define a savepoint *x*, while inside a transaction	SAVEPOINT *x*	SAVEPOINT *x* ON ROLLBACK RETAIN CURSORS (http://publib.boulder.ibm.com/infocenter/db2luw/v9/index.jsp?topic=/com.ibm.db2.udb.admin.doc/doc/r0003271.htm)	SAVEPOINT *x*	SA (h /9 /s
?	Rollback to given savepoint *x*	ROLLBACK [WORK] TO SAVEPOINT *x*	ROLLBACK [WORK] TO SAVEPOINT *x* (http://publib.boulder.ibm.com/infocenter/db2luw/v9/index.jsp?topic=/com.ibm.db2.udb.admin.doc/doc/r0000992.htm)	ROLLBACK [WORK] TO [SAVEPOINT] *x*	[E TO /Ir /9 /r
?	Release (forget) savepoint *x*	RELEASE SAVEPOINT *x*	RELEASE [TO] SAVEPOINT *x* (http://publib.boulder.ibm.com/infocenter/db2luw/v9/index.jsp?topic=/com.ibm.db2.udb.admin.doc/doc/r0003269.htm)	RELEASE SAVEPOINT *x*	
?	Prepare transaction named *id*	?	?	N/A	?

?	Commit prepared transaction named *id*	?	?	N/A	?
?	Rollback prepared transaction named *id*	?	?	N/A	?

	Start	Commit	Rollback	Prepare	Execute prepared
Linter	`Implicit`	▪ a `COMMIT` statement is executed ▪ any DDL statement is executed ▪ any statement is executed in AUTOCOMMIT mode	▪ a `ROLLBACK` statement is executed ▪ a user process is terminated abnormally or disconnects without `COMMIT/ROLLBACK`	?	?

Procedural language/Stored procedures

Stored procedures

Database	Create syntax

DB2	`CREATE PROCEDURE procedure_name(...)`
	`  BEGIN`
	`  /* SQL code */`
	`  END`

Firebird	`SET TERM $$ ;`
	
	`CREATE PROCEDURE nameprocedure`
	`  (input_parameter_name datatype, ... )`
	`RETURNS`
	`  (output_parameter_name datatype, ... )`
	`AS`
	`DECLARE VARIABLE variable_name datatype;`
	`BEGIN`
	`  /* SQL code */`
	`END$$`
	
	`SET TERM ; $$`

MonetDB	`CREATE PROCEDURE procedure_name(...)`
	`  BEGIN`
	`  /* SQL code */`
	`  END`

MySQL	`DELIMITER $$`
	
	`CREATE PROCEDURE nameprocedure`
	`  (input_parameter_name datatype, ... )`
	`BEGIN`
	`  /* SQL code */`
	`END$$`
	
	`DELIMITER ;`

Linter	`CREATE [OR REPLACE] PROCEDURE procedure_name([IN/OUT/INOUT] parameter_name datatype,`
	`  DECLARE`
	`  /* variables declaration */`
	`  CODE`
	`  /* stored procedure code`
	`  (including SQL code)*/`
	`  EXCEPTIONS`
	`  /* exceptions declarations */`
	`  END`

OpenLink Virtuoso	
Oracle	`CREATE [OR REPLACE] PROCEDURE procedure_name [ (parameter [,parameter]) ]` `IS`

Database	Create syntax

```
    [declaration_section]
BEGIN
    executable_section
    [EXCEPTION
        exception_section]
END [procedure_name];
```

PostgreSQL

```
CREATE FUNCTION function_name
  (input_parameter_name datatype, ...)
RETURNS return_type
AS $$
DECLARE
  variable_name datatype;
BEGIN
  /* SQL code */
END;
$$ LANGUAGE plpgsql;
```

SQL Server

```
CREATE PROCEDURE nameprocedure
  (input_parameter_name datatype, ... )
AS
  /* SQL code */
GO
```

SQLite

N/A

Procedural language/User-defined functions

User-defined functions (UDF)

Database	Create syntax	

DB2	`CREATE FUNCTION function_name`
	`  (input_parameter_name datatype, ...)`
	`RETURNS return_type`
	`BEGIN`
	`  /* SQL code */`
	`END`
		`VALUES function_name(...)`
		
		`or`
		
		`SELECT function_name(...) FRO`

SQLite	N/A	N/A

MonetDB	`CREATE FUNCTION function_name`
	`  (input_parameter_name datatype, ...)`
	`RETURNS return_type`
	`BEGIN`
	`  /* SQL code */`
	`END`
		`SELECT function_name(...) FRO`

MySQL	`DELIMITER $$`
	
	`CREATE FUNCTION function_name`
	`  (input_parameter_name datatype, ... )`
	`RETURNS datatype`
	`BEGIN`
	`RETURN`
	`  /* SQL code */`
	`END$$`
	
	`DELIMITER ;`
		`SELECT function_name(...)`

PostgreSQL	`CREATE FUNCTION function_name`
	`  (input_parameter_name datatype, ...)`
	`RETURNS datatype`
	`AS $$`
	`DECLARE`
	`  variable_name datatype;`
	`BEGIN`
	`  /* SQL code */`
	`END;`
	`$$ LANGUAGE plpgsql;`
		`SELECT function_name(...)`

Firebird	UDFs are written in external tools and compiled into executable form.
	They are dynamically loaded at runtime.
	In order for server to pick them up, they need to be registered, like this:
	`DECLARE EXTERNAL FUNCTION function_name [datatype, ...]`
	`RETURNS datatype`
	`ENTRY_POINT 'entryname'`
	`MODULE_NAME 'modulename';`
		`SELECT function_name(...)`

SQL XML

The ISO publication (ISO/IEC 9075-14 (http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=53686)) is part of the SQL standard. Formerly it was named SQLX or SQL/XML. It defines the data type `XML` and functions acting on this data type as well as functions creating and handling XML objects (elements, attributes, ...) within this data type. Oracle denotes the data type XMLType.

Function	Description	DB2	SQLite	MonetDB	MySQL	Po
Version	Software Version(s) Supported	V9	N/A	V11+	Planned	8.:
XMLElement()	Create an XML Element	XMLElement()	N/A	xmlelement()	Planned	xn
XMLForest()	Create an XML Fragment from passed-in components.	XMLForest()	N/A	xmlforest()	xmlforest()	Pla (p ex
XMLColAttVal()	Create an XML fragment and then expands the resulting XML so that each XML fragment has the name "column" with the attribute "name"		N/A	N/A	Planned	Pla
ExtractValue()	Takes as arguments an XML instance and an XPath expression and returns a scalar value of the resultant node.		N/A	N/A	Planned	Pla
XMLTransform()	Takes as arguments an XML instance and an XSL style sheet, which is itself a form of XML instance. It applies the style sheet to the instance and returns an XML.		N/A	N/A	Planned	Pla
XMLSequence()	Takes input and returns either a varray of the		N/A	xmlsequence()	Planned	Pla

	top-level nodes in the XML, or an XMLSequence type an XML document for each row of the cursor.					
XMLConcat()	Takes as input a series of XML instances, concatenates the series of elements for each row, and returns the concatenated series.	XMLConcat	N/A	N/A	Planned	Pl (p ex
UpdateXML()	Takes as arguments an XML instance and an XPath-value pair, and returns an XML instance with the updated value.		N/A	N/A	Planned	Pl

Notes

1. The MSSQL 2005 and higher `xml` data type `.query()` method performs this function.
2. The MSSQL 2000 and higher `FOR XML` clause of the SQL `SELECT` statement performs a similar function.
3. The MSSQL 2005 and higher `xml` data type `.value()` method performs this function.
4. In MSSQL 2005 and higher the `xml` data type and SQLCLR can be used to create functions/procedures that simulate this functionality.
5. The MSSQL 2005 and higher `xml` data type `.nodes()` method performs this function.
6. In MSSQL 2005 and higher the + string concatenation operator can be used in conjunction with explicit conversions of `xml` data type instances to character data types to simulate this function.
7. The MSSQL 2005 and higher `xml` data type `.modify()` method performs this function using XML DML.

Administration

While this topic is not directly about SQL language, it is sufficiently close to gather information about frequently undertaken administrative tasks together.

Dumping database

Dumping the database involves creation of a file that stores all the data (rows) and the schema for this data from the desired databases/tables. Most usually it's done as a series of SQL statements, combining CREATE DATABASE / CREATE TABLE statements to recreate the schema and INSERT statements to refill the rows — thus, usually the resulting file can be directly imported using command-line SQL client by simple execution of queries. However, sometimes other output formats are desired and some databases (at least MSSQL) lack the ability to serialize the database as SQL statements from command-line dumper client.

SQL version	Feature	Standard SQL:2011	DB2	Firebird	Ingres (http://docs.ingres.com/9.3/Command%20Reference%20/copydbcmd.htm)
?	Basic invocation	?	?	?	copydb *dbname options*
?	Authentication to target server	?	?	?	-u*username* -P
?	Save dump into file	?	?	?	?
?	Dump only schema	?	?	?	?
?	Add DROP schema at start	?	?	?	-add_drop
?	Prevent dumping of access privileges	?	?	?	?
?	Dump only data	?	?	?	?
?	Dump data as tab-separated	?	?	?	?
?	Dump data as INSERT statements	?	?	?	?
?	Dump everything as XML file	?	?	?	?
?	Ordering of data	?	?	?	?

Restoring database

Command-line SQL query

SQL version	Feature	Standard SQL:2011	DB2	Firebird	Ingres (http://docs.ingres.com/9.3/Command%20Reference%20/sqlcmd.htm)
?	Basic invocation	?	?	?	sql *options database*
?	Authentication to target server	?	?	?	-u*username* -P*password* -G*group* -
?	Execute query from command line and exit	?	?	?	N/A
?	Execute query from command line and continue with interactive prompt	?	?	?	N/A
?	Input file[1]	?	?	?	N/A
?	Output to file instead of stdout[2]	?	?	?	N/A
?	Copy output also to file	?	?	?	N/A

References

1. Can be also done using shell **<** redirection
2. Can be also done using shell **>** redirection

References

Standards

- SQL-86
- SQL-89
- SQL-92 (http://www.contrib.andrew.cmu.edu/~shadow/sql/sql1992.txt) — full text is available
- SQL:1999 — can be purchased from ISO
- SQL:2003 — can be purchased from ISO

- working draft of SQL:2008, downloadable (http://www.wiscorp.com/sql200n.zip) from Whitemarsh Information Systems Corporation.
- SQL:2008 — can be purchased from ISO
- SQL:2011 — can be purchased from ISO

Official product documentations

- DB2 Version 9 Information Center (http://publib.boulder.ibm.com/infocenter/db2luw/v9/index.jsp)
- MonetDB documentation (http://www.monetdb.org/)
- MySQL 5.0 reference manual (http://dev.mysql.com/doc/refman/5.0/en/)
- PostgreSQL 8.2 reference manual (http://www.postgresql.org/docs/8.2/interactive/)
- SQLite documentation (http://www.sqlite.org/docs.html)
- Firebird / Interbase 6 language reference (http://www.ibphoenix.com/downloads/60All.zip) (ZIP archive, 8.9MB)
- OpenLink Virtuoso Universal Server Documentation (http://docs.openlinksw.com/virtuoso/)
- Oracle Database 10g2 SQL Reference (http://docs.oracle.com/cd/B28359_01/server.111/b28286/toc.htm)
- Oracle Database 12g1 SQL Reference (http://docs.oracle.com/database/121/SQLRF/toc.htm)
- Microsoft SQL Server Documentation (http://msdn2.microsoft.com/en-us/library/ms130214.aspx)
- Linter SQL Server Documentation (English, old version) (http://www.linter.ru/en/documentation/);Linter SQL Server Documentation (Russian) (http://www.linter.ru/ru/documentation/)

Single implementation references and cheat sheets

- Oracle PL/SQL Cheatsheet
- Oracle XML Cheatsheet

Comparisons

- Comparison of different SQL implementations (http://troels.arvin.dk/db/rdbms/)
- Comparison of Oracle, MySQL and PostgreSQL DBMS (http://www-css.fnal.gov/dsg/external/freeware/mysql-vs-pgsql.html) by Computing Division of Fermi National Accelerator Laboratory
- SQL Function Reference: Oracle vs. SQL Server (http://www.webucator.com/resources/sql/reference.html)
- Kline, Kevin E.; Kline, Daniel (January 2001). "SQL Functions". *SQL in a Nutshell*. Cambridge, MS: O'Reilly. ISBN 1-56592-744-3. http://www.oreilly.com/catalog/sqlnut/chapter/ch04.html.
- SQL Functions (http://ericbrian.com/2005/11/29/sql-functions/) — MySQL vs Oracle comparison by Eric Brian
- Standard SQL (http://www.dbazine.com/db2/db2-disarticles/gulutzan3) - an article by Peter Gulutzan that compares SQL standards support in DB2, MSSQL and Oracle

Migration

- Converting MySQL to PostgreSQL
