IBM DB2
Toolbox
Tipps und Tricks

Ullrich Barmeyer
Dipl.Ing. (FH), Dipl.Inf. (TU)
BCS Beratung Computer Software
Kleinmachnow 2007

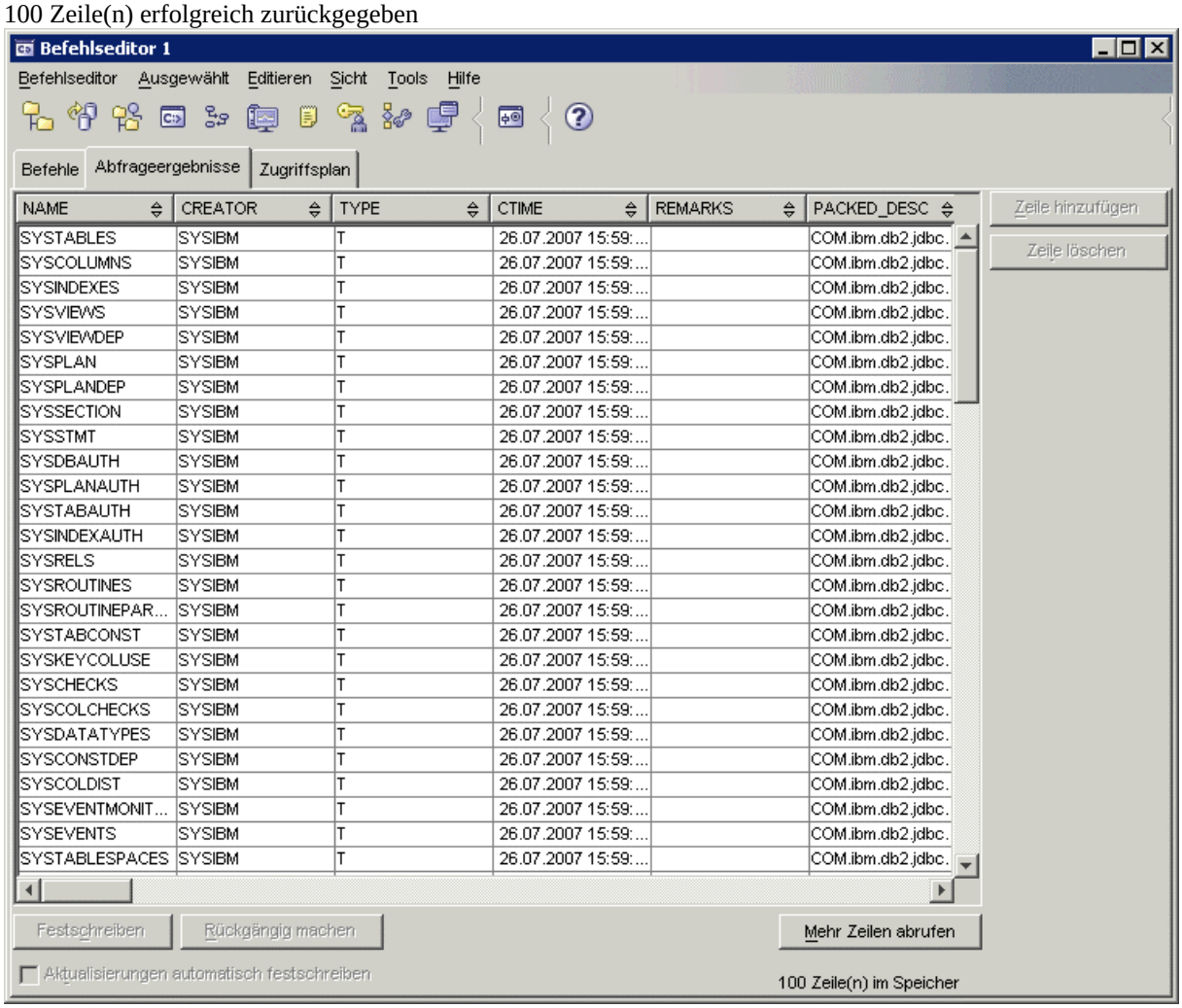
1. Systemtabellen

1.1. Abfrage der vorhandenen Datenbanktabellen

Die System-Datenbanktabelle SYSTABLES enthält die Namen aller DB2 Tabellen in dieser Datenbank.
Beispiel:

select * from sysibm.systables

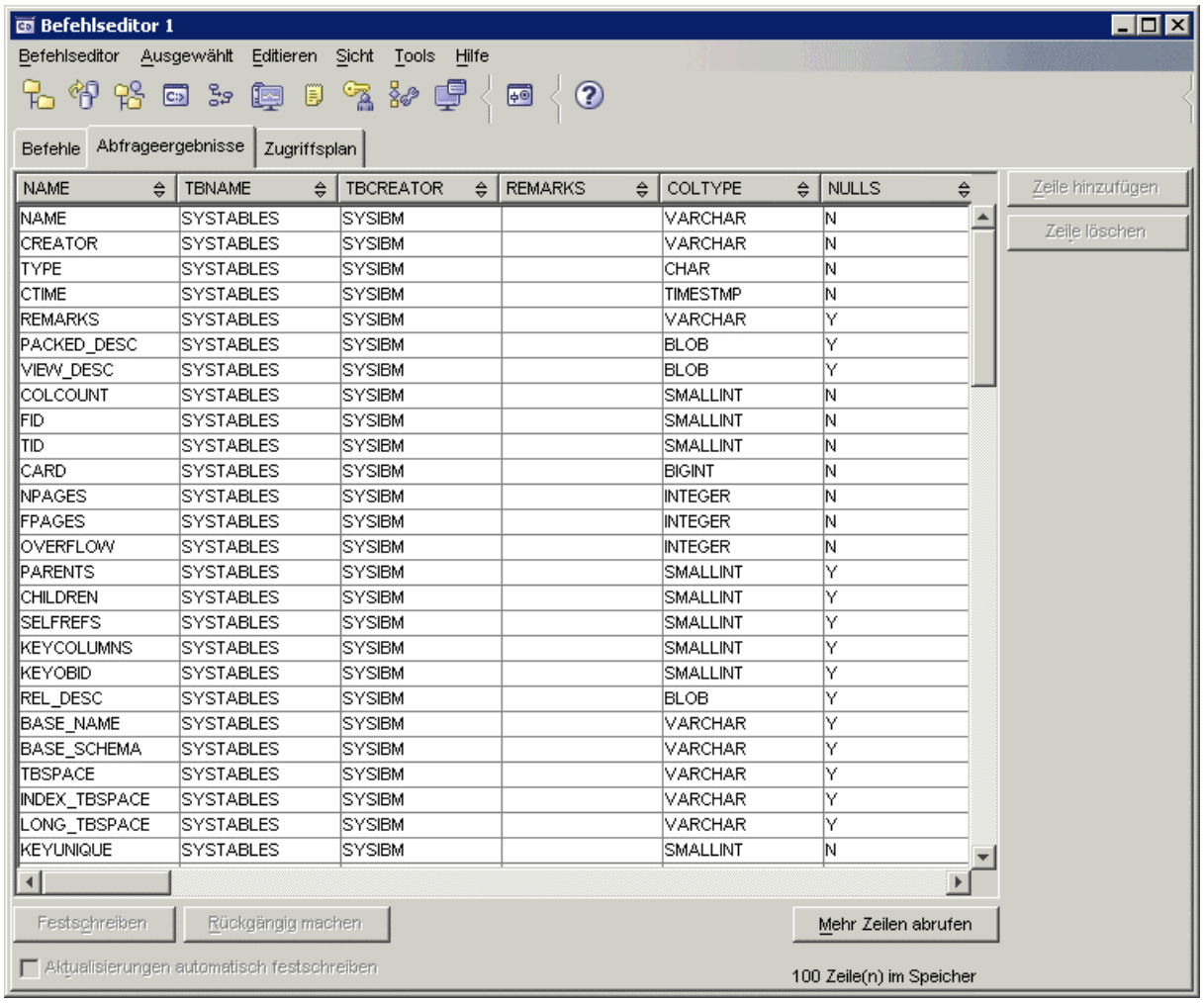
Ergebnisse für eine einzelne Abfrage werden auf der Indexzunge 'Abfrageergebnisse' angezeigt.
100 Zeile(n) erfolgreich zurückgegeben



NAME	CREATOR	TYPE	CTIME	REMARKS	PACKED_DESC
SYSTABLES	SYSIBM	T	26.07.2007 15:59:...		COM.ibm.db2.jdbc.
SYSCOLUMNS	SYSIBM	T	26.07.2007 15:59:...		COM.ibm.db2.jdbc.
SYSINDEXES	SYSIBM	T	26.07.2007 15:59:...		COM.ibm.db2.jdbc.
SYSVIEWS	SYSIBM	T	26.07.2007 15:59:...		COM.ibm.db2.jdbc.
SYSVIEWDEP	SYSIBM	T	26.07.2007 15:59:...		COM.ibm.db2.jdbc.
SYSPLAN	SYSIBM	T	26.07.2007 15:59:...		COM.ibm.db2.jdbc.
SYSPLANDEP	SYSIBM	T	26.07.2007 15:59:...		COM.ibm.db2.jdbc.
SYSSECTION	SYSIBM	T	26.07.2007 15:59:...		COM.ibm.db2.jdbc.
SYSSTMT	SYSIBM	T	26.07.2007 15:59:...		COM.ibm.db2.jdbc.
SYSDBAUTH	SYSIBM	T	26.07.2007 15:59:...		COM.ibm.db2.jdbc.
SYSPLANAUTH	SYSIBM	T	26.07.2007 15:59:...		COM.ibm.db2.jdbc.
SYSTABAUTH	SYSIBM	T	26.07.2007 15:59:...		COM.ibm.db2.jdbc.
SYSINDEXAUTH	SYSIBM	T	26.07.2007 15:59:...		COM.ibm.db2.jdbc.
SYSRELS	SYSIBM	T	26.07.2007 15:59:...		COM.ibm.db2.jdbc.
SYSROUTINES	SYSIBM	T	26.07.2007 15:59:...		COM.ibm.db2.jdbc.
SYSROUTINEPAR...	SYSIBM	T	26.07.2007 15:59:...		COM.ibm.db2.jdbc.
SYSTABCONST	SYSIBM	T	26.07.2007 15:59:...		COM.ibm.db2.jdbc.
SYSKEYCOLUSE	SYSIBM	T	26.07.2007 15:59:...		COM.ibm.db2.jdbc.
SYSCHECKS	SYSIBM	T	26.07.2007 15:59:...		COM.ibm.db2.jdbc.
SYSCOLCHECKS	SYSIBM	T	26.07.2007 15:59:...		COM.ibm.db2.jdbc.
SYSDATATYPES	SYSIBM	T	26.07.2007 15:59:...		COM.ibm.db2.jdbc.
SYSCONSTDEP	SYSIBM	T	26.07.2007 15:59:...		COM.ibm.db2.jdbc.
SYSCOLDIST	SYSIBM	T	26.07.2007 15:59:...		COM.ibm.db2.jdbc.
SYSEVENTMONIT...	SYSIBM	T	26.07.2007 15:59:...		COM.ibm.db2.jdbc.
SYSEVENTS	SYSIBM	T	26.07.2007 15:59:...		COM.ibm.db2.jdbc.
SYSTABLESPACES	SYSIBM	T	26.07.2007 15:59:...		COM.ibm.db2.jdbc.

1.2. Abfrage der Spaltennamen (Columns)

Die Tabelle SYSCOLUMNS enthält alle Spaltennamen aller Tabellen (gesammelt).
select * from sysibm.syscolumns;



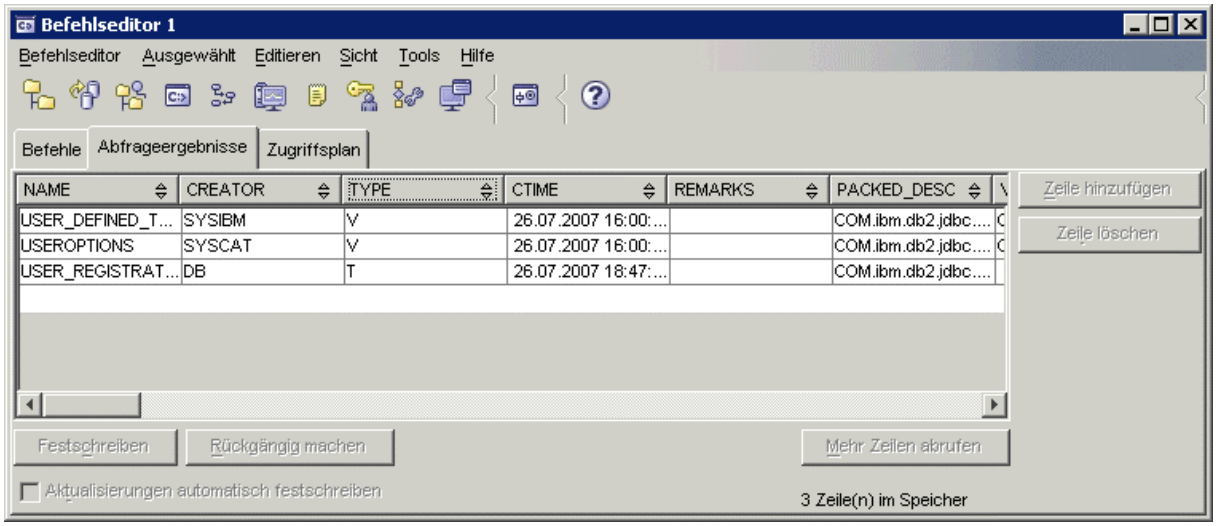
NAME	TBNAME	TBCREATOR	REMARKS	COLTYPE	NULLS
NAME	SYSTABLES	SYSIBM		VARCHAR	N
CREATOR	SYSTABLES	SYSIBM		VARCHAR	N
TYPE	SYSTABLES	SYSIBM		CHAR	N
CTIME	SYSTABLES	SYSIBM		TIMESTMP	N
REMARKS	SYSTABLES	SYSIBM		VARCHAR	Y
PACKED_DESC	SYSTABLES	SYSIBM		BLOB	Y
VIEW_DESC	SYSTABLES	SYSIBM		BLOB	Y
COLCOUNT	SYSTABLES	SYSIBM		SMALLINT	N
FID	SYSTABLES	SYSIBM		SMALLINT	N
TID	SYSTABLES	SYSIBM		SMALLINT	N
CARD	SYSTABLES	SYSIBM		BIGINT	N
NPAGES	SYSTABLES	SYSIBM		INTEGER	N
FPAGES	SYSTABLES	SYSIBM		INTEGER	N
OVERFLOW	SYSTABLES	SYSIBM		INTEGER	N
PARENTS	SYSTABLES	SYSIBM		SMALLINT	Y
CHILDREN	SYSTABLES	SYSIBM		SMALLINT	Y
SELFREFS	SYSTABLES	SYSIBM		SMALLINT	Y
KEYCOLUMNS	SYSTABLES	SYSIBM		SMALLINT	Y
KEYOBID	SYSTABLES	SYSIBM		SMALLINT	Y
REL_DESC	SYSTABLES	SYSIBM		BLOB	Y
BASE_NAME	SYSTABLES	SYSIBM		VARCHAR	Y
BASE_SCHEMA	SYSTABLES	SYSIBM		VARCHAR	Y
TBSPACE	SYSTABLES	SYSIBM		VARCHAR	Y
INDEX_TBSPACE	SYSTABLES	SYSIBM		VARCHAR	Y
LONG_TBSPACE	SYSTABLES	SYSIBM		VARCHAR	Y
KEYUNIQUE	SYSTABLES	SYSIBM		SMALLINT	N

Sehr gut geeignet auch für die Frage: Wo kommt überall die Spalte KUNDENR vor?

1.3. Suchen nach einer bestimmten Tabelle

Durch verwendung des LIKE-Poperators lassen sich Tabellen mit ähnlichen Namen finden:

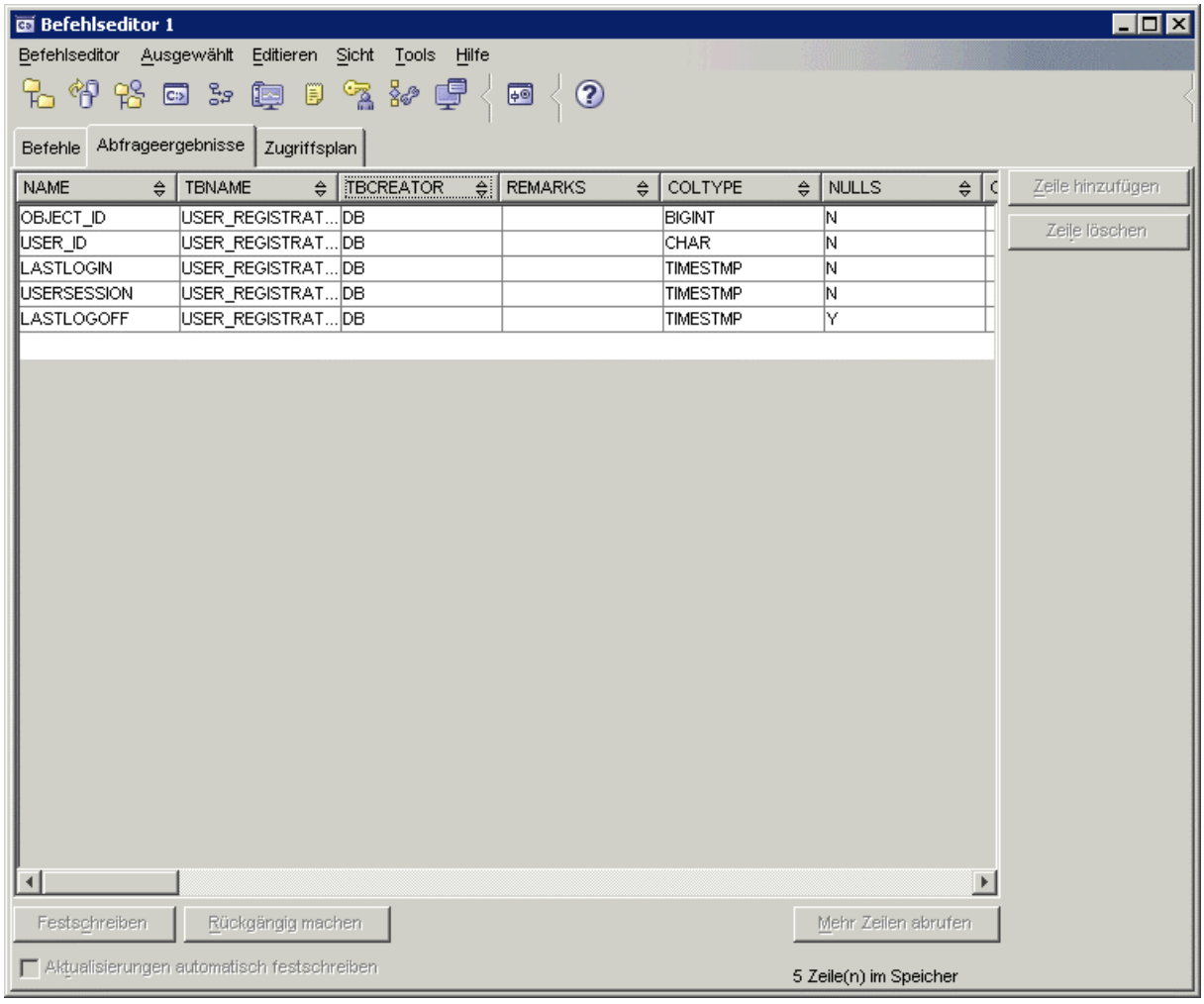
```
select * from sysibm.systables where name like 'USER%'
```



1.4. Anzeige der Tabelle

In obigem Ergebnis ist auch der Tabellen-CREATOR sichtbar. Dies entspricht dem Schema, mit dem die Abfrage direkt auf die gesuchte Tabelle durchgeführt werden muss:
Hier ist CREATOR = SCHEMA = „DB“

select * from **db**.user_registration



2. Liste der DB2 Funktionen

B.2 SCALAR FUNCTIONS

The following functions are called scalar functions because both their arguments and their result are scalar values. Some of these functions are built into the UDB implementation and reside in the SYSIBM schema. Others are external functions that are shipped with the system and reside in the SYSFUN schema. The schema in which each function resides is given in parentheses at the end of its description. The SYSIBM functions can in general be expected to have better performance than the SYSFUN functions. Since the default function path includes both the SYSIBM and SYSFUN schemas, you can make use of any of these functions without including a schema name in your function invocation.

abs (*<any numeric datatype>*) → *<same datatype>*

absval (*<any numeric datatype>*) → *<same datatype>*

Returns the absolute value of the argument. There is an exception to the rule that the result datatype is the same as the argument datatype: if the argument datatype is Decimal or Real, the result datatype is Double. (SYSFUN)

acos (Double) → Double

Returns the arccosine of the argument, as an angle expressed in radians. (SYSFUN)

ascii (*<any string datatype>*) → Integer

Returns the ASCII code of the first character of the argument string. (SYSFUN)

asin (Double) → Double

Returns the arcsine of the argument, as an angle expressed in radians. (SYSFUN)

atan (Double) → Double

Returns the arctangent of the argument, as an angle expressed in radians. (SYSFUN)

atan2 (Double *x*, Double *y*) → Double

Returns the arctangent of the point whose coordinates are *x* and *y*, as an angle expressed in radians. (SYSFUN)

blob (*<any string datatype>*) → Blob

blob (*<any string datatype>*, Integer *n*) → Blob(*n*)

blob (*<any DBCS datatype>*) → Blob

blob (*<any DBCS datatype>*, Integer *n*) → Blob(*n*)

blob (Blob) → Blob

blob (Blob, Integer *n*) → Blob(*n*)

Converts the first argument to a Blob datatype. The second argument, if specified, becomes the maximum length of the result (causing truncation if necessary). (SYSIBM)

ceil (*<any numeric datatype>*) → *<same datatype>*

ceiling (*<any numeric datatype>*) → *<same datatype>*

Returns the smallest integer greater than or equal to the argument. There is an exception to the rule that the return type is the same type as the argument type: if the argument type is Decimal or Real, the return type is Double. (SYSFUN)

char (Date) → Char(10)

char (Date, Keyword *k*) → Char(10)

char (Time) → Char(8)

char (Time, Keyword *k*) → Char(8)

char (Timestamp) → Char(26)

Returns a character-string representation of the argument. The keyword *k*, if present, governs the format of the string representation. Valid values for the keyword argument (which must not be enclosed in quotes) are ISO, USA, EUR, JIS, and LOCAL. (SYSIBM)

char (*<any string datatype>*) → Char( )

char (*<any string datatype>*, Integer *n*) → Char(*n*)

Converts the first argument to a fixed-length Char datatype. The second argument, if specified, becomes the length of the result (causing truncation or padding with blanks if necessary). The second argument, if present, must be between 0 and 254. (SYSIBM)

char (Smallint) → Char(6)

char (Integer) → Char(11)

char (Decimal) → Char( )

char (Decimal, Varchar) → Char( )

Returns a character-string representation of the first argument. If the first argument is Decimal and the second argument is Varchar, the second argument must be a single character and is used to represent the decimal point in the result string. (SYSIBM)

char (Double) → Char(24)

Returns a character-string representation of a double-precision floating-point value. (SYSFUN)

chr (Integer) → Char(1)

Returns the character whose ASCII code is equal to the argument. Returns null if the argument is not between 0 and 255. (SYSFUN)

clob (*<any string datatype>*) → Clob

clob (*<any string datatype>*, Integer *n*) → Clob(*n*)

Converts the first argument to a Clob datatype. The second argument, if specified, becomes the maximum length of the result (causing truncation if necessary). (SYSIBM)

coalesce (*<one or more arguments of compatible datatypes>*) → *<datatype of highest precedence>*

Returns the value of its first non-null argument, or null if all arguments are null. (See Section 6.6.2 for rules governing compatibility of argument datatypes.) `coalesce` and `value` are different names for the same function. (SYSIBM)

concat (*<any string datatype>*, *<any string datatype>*) → *<datatype of highest precedence>*

concat (*<any DBCS datatype>*, *<any DBCS datatype>*) → *<datatype of highest precedence>*

concat (Blob, Blob) → Blob

Returns the concatenation of the two arguments. Equivalent to `concat` or `||` used as an infix operator. If the result cannot be represented in the datatype of highest precedence, a suitable datatype is chosen. For example, if a concatenation of two fixed-length strings results in a string with a length greater than 254, the result datatype will be Varchar rather than Char. (SYSIBM)

cos (Double) → Double

Returns the cosine of an angle expressed in radians. (SYSFUN)

cot (Double) → Double

Returns the cotangent of an angle expressed in radians. (SYSFUN)

date (*<date string>*) → Date

Converts the argument from a string representation of a Date to an actual Date. The argument may be in any of the following formats (examples represent the last day of the year 1999): (SYSIBM)
"1999-12-31"
"12/31/1999"
"31.12.1999"
"1999365"

days (Date) → Integer

days (*<date string>*) → Integer

days (Timestamp) → Integer

days (*<timestamp string>*) → Integer

Converts the argument to a date, then returns one more than the number of days between January 1, 0001 and this date. (SYSIBM)

dbclob (*<any DBCS datatype>*) → Dbclob

dbclob (*<any DBCS datatype>*, Integer *n*) → Dbclob(*n*)

Converts the first argument to a Dbclob datatype. The second argument, if specified, becomes the maximum length of the result (causing truncation if necessary). (SYSIBM)

decimal (*<any numeric datatype>*) → Decimal(15, 0)

decimal (*<any numeric datatype>*, Integer *p*) → Decimal(*p*, 0)

decimal (*<any numeric datatype>*, Integer *p*, Integer *s*) → Decimal(*p*, *s*)

Returns a decimal representation of the first argument, with the indicated precision and scale. The function name dec may be used as a synonym for `decimal`. (SYSIBM)

decimal (Varchar) → Decimal(15, 0)

decimal (Varchar, Integer *p*) → Decimal(*p*, 0)

decimal (Varchar, Integer *p*, Integer *s*) → Decimal(*p*, *s*)

decimal (Varchar, Integer *p*, Integer *s*, Varchar *d*) → Decimal(*p*, *s*)

The first argument is a character-string representation of a decimal number, which is converted to an actual decimal value with the indicated precision and scale. The fourth parameter, if present, is a single character that indicates how the decimal point is represented in the string input. The function name dec may be used as a synonym for `decimal`. (SYSIBM)

degrees (*<any numeric datatype>*) → Double

Converts the argument angle from radians to degrees. (SYSFUN)

difference (Varchar, Varchar) → Integer

Returns a measure of the difference between the sounds of two strings, as computed by the `soundex` function. The result is an integer between 0 and 4, where 4 represents the best sound match. (SYSFUN)

digits (Decimal) → Char( )

Returns a fixed-length character string representing the absolute value of the argument, not including its sign or decimal point. For example, digits(–123.45) is "12345." (SYSIBM)

double (*<any numeric datatype>*) → Double

double_precision (*<any numeric datatype>*) → Double

Converts the argument to a double-precision floating-point number. (SYSIBM)

double (Varchar) → Double

If the argument contains a valid representation of a floating-point value, this value is returned. (SYSFUN)

event_mon_state (Varchar) → Integer

The argument is the name of an Event Monitor (defined by a CREATE EVENT MONITOR statement). The function returns 1 if the named Event Monitor is active, 0 if it is inactive. (SYSIBM)

exp (Double *x*) → Double

Returns e^x, where x is the argument and e is the base of the natural logarithms. (SYSFUN)

float (*<any numeric datatype>*) → Double

Converts the argument to a double-precision floating-point number. (SYSIBM)

floor (*<any numeric datatype>*) → *<same datatype>*

Returns the largest integer less than or equal to the argument. There is an exception to the rule that the return type is the same type as the argument type: if the argument type is Decimal or Real, the return type is Double. (SYSFUN)

generate_unique ( ) → Char(13) FOR BIT DATA

Returns a value that is unique compared to any other execution of the same function. The value is based on the system clock and may be used as a unique key value when inserting rows into a table. The string returned by this function can be converted into a Universal Coordinated Time timestamp by passing it to the `timestamp` function. (SYSIBM)

graphic (*<any DBCS datatype>*) → Graphic

graphic (*<any DBCS datatype>*, Integer *n*) → Graphic(*n*)

Converts the argument to a fixed-length Graphic datatype, of the indicated length. If no length is specified, the length is derived from that of the argument. (SYSIBM)

hex (*<any built-in datatype>*) → Varchar

Returns a hexadecimal representation of the argument value. Each two bytes of the returned string represent one byte of the internal representation of the argument value. (SYSIBM)

hour (Time) → Integer

hour (*<time string>*) → Integer

hour (*<time duration>*) → Integer

hour (Timestamp) → Integer

hour (*<timestamp string>*) → Integer

hour (*<timestamp duration>*) → Integer

Returns the hour part of the argument. If the argument is a time, timestamp, or string, the result is an integer between 0 and 24. If the argument is a duration, the result is an integer between –99 and 99. (SYSIBM)

insert (Varchar *x*, Integer *m*, Integer *n*, Varchar *y*) → Varchar(4000)

insert (Clob(1M) *x*, Integer *m*, Integer *n*, Clob(1M) *y*) → Clob(1M)

insert (Blob(1M) *x*, Integer *m*, Integer *n*, Blob(1M) *y*) → Blob(1M)

The return string is computed by deleting from string *x* a substring of *n* characters beginning with character number *m*, and replacing the deleted substring by string *y*. (SYSFUN)

integer (*<any numeric datatype>*) → Integer

integer (Varchar) → Integer

Converts the argument to an integer, by truncation of the decimal part if necessary. If the argument is of datatype Varchar, it must be a character-string representation of an integer, such as "–123." The function name int may be used as a synonym for integer. (SYSIBM)

julian_day (Date) → Integer

julian_day (*<date string>*) → Integer

julian_day (Timestamp) → Integer

julian_day (*<timestamp string>*) → Integer

Returns the number of days between the beginning of the Julian calendar (January 1, 4712 B.C.) and the argument date. (SYSFUN)

lcase (Varchar) → Varchar(4000)

lcase (Clob(1M)) → Clob(1M)

Returns a copy of the argument string in which all the uppercase characters have been converted to lowercase. (SYSFUN)

left (Varchar *x*, Integer *n*) → Varchar(4000)

left (Clob(1M) *x*, Integer *n*) → Clob(1M)

left (Blob(1M) *x*, Integer *n*) → Blob(1M)

Returns the leftmost *n* characters of string *x*. (SYSFUN)

length (*<any built-in datatype>*) → Integer

Returns the length of the argument (not including a null indicator). For string-type arguments, returns the actual length (not the maximum length) in characters (for DBCS strings, each character is two bytes). For numeric or datetime arguments, returns the length in bytes of the internal representation. (SYSIBM)

ln (Double) → Double

Returns the natural logarithm of the argument. (SYSFUN)

locate (*<any string datatype> s1*, *<any string datatype> s2*) → Integer

locate (*<any string datatype> s1*, *<any string datatype> s2*, Integer *n*) → Integer

locate (Blob(1M) *s1*, Blob(1M) *s2*) → Integer

locate (Blob(1M) *s1*, Blob(1M) *s2*, Integer *n*) → Integer

Returns the starting position of the first occurrence of argument *s2* inside argument *s1*. The third argument, if any, indicates the position within *s1* where the search is to begin. If *s2* is not found within *s1*, 0 is returned. If any argument is a Clob, it is limited to a length of one megabyte. See `posstr` for a similar function. (SYSFUN)

log (Double) → Double

Returns the natural logarithm of the argument (same as `ln`). (SYSFUN)

log10 (Double) → Double

Returns the base-10 logarithm of the argument. (SYSFUN)

long_varchar (*<any string datatype>*) → Long Varchar

Converts the argument to the Long Varchar datatype, which has a maximum length of 32K characters. (SYSIBM)

long_vargraphic (*<any DBCS string datatype>*) → Long Vargraphic

Converts the argument to the Long Vargraphic datatype, which has a maximum length of 16K double-byte characters. (SYSIBM)

ltrim (Varchar) → Varchar(4000)

ltrim (Clob(1M)) → Clob(1M)

Returns a copy of the argument with leading blanks removed. (SYSFUN)

microsecond (Timestamp) → Integer

microsecond (*<timestamp string>*) → Integer

microsecond (*<timestamp duration>*) → Integer

Returns the microsecond part of the argument. (SYSIBM)

midnight_seconds (Time) → Integer

midnight_seconds (*<time string>*) → Integer

midnight_seconds (Timestamp) → Integer

midnight_seconds (*<timestamp string>*) → Integer

Returns the number of seconds between the argument time and the previous midnight. (SYSFUN)

minute (Time) → Integer

minute (*<time string>*) → Integer

minute (*<time duration>*) → Integer

minute (Timestamp) → Integer

minute (*<timestamp string>*) → Integer

minute (*<timestamp duration>*) → Integer

Returns the minute part of the argument. If the argument is a time, timestamp, or string, the result is an integer between 0 and 59. If the argument is a duration, the result is an integer between –99 and 99. (SYSIBM)

mod (Smallint *m*, Smallint *n*) → Smallint

mod (Integer *m*, Integer *n*) → Integer

Returns the remainder (modulus) of argument *m* divided by argument *n*. (SYSFUN)

month (Date) → Integer

month (*<date string>*) → Integer

month (*<date duration>*) → Integer

month (Timestamp) → Integer

month (*<timestamp string>*) → Integer

month (*<timestamp duration>*) → Integer

Returns the month part of the argument. If the argument is a date, timestamp, or string, the result is an integer between 1 and 12. If the argument is a duration, the result is an integer between –99 and 99. (SYSIBM)

monthname (Date) → Varchar(100)

monthname (*<date string>*) → Varchar(100)

monthname (Timestamp) → Varchar(100)

monthname (*<timestamp string>*) → Varchar(100)

Returns the name of the month part of the argument, as a mixed-case character string. (SYSFUN)

nodenumber (*<any datatype>*) → Integer

The argument must be the name of a column in some table that is referenced in the current SQL statement. For each row of the referenced table, `nodenumber` returns the number of the partition on which the row is stored, or zero if the database is not partitioned. (SYSIBM)

nullif (*<any datatype>*, *<any compatible datatype>*) → *<datatype of highest precedence>*

Returns null if the arguments are equal; otherwise, returns the first argument. (SYSIBM)

partition (*<any datatype>*) → Integer

The argument must be the name of a column in some table that is referenced in the current SQL statement. For each row of the referenced table, `partition` returns the partitioning map index that applies to the row (an integer between 0 and 4,095), or zero if the table has no partitioning key. (SYSIBM)

posstr (*<any string datatype>* *s1*, Varchar(4000) *s2*) → Integer

posstr (*<any DBCS datatype>* *s1*, Vargraphic(2000) *s2*) → Integer

posstr (Blob *s1*, Blob(4000) *s2*) → Integer

Returns the starting position of the first occurrence of argument *s2* inside argument *s1*. If *s2* is not found within *s1*, 0 is returned. See `locate` for a similar function. (SYSFUN)

power (Integer *x*, Integer *n*) → Integer

power (Double *x*, Double *n*) → Double

Returns the first argument raised to the power of the second argument, x^n. (SYSFUN)

quarter (Date) → Integer

quarter (*<date string>*) → Integer

quarter (Timestamp) → Integer

quarter (*<timestamp string>*) → Integer

Returns an integer between 1 and 4 representing the quarter of the year in which the argument occurs. (SYSFUN)

radians (Double) → Double

Converts the argument angle from degrees to radians. (SYSFUN)

raise_error (Varchar, Varchar) → *<void>*

Causes an error condition to be returned in the SQLCA. The first argument must be exactly 5 characters in length and becomes the SQLSTATE. The second argument is a message of up to 70 characters that is placed in the SQLERRMC field of the SQLCA. The SQLCODE is set to –438. The `raise_error` function returns no value but is considered to be compatible with the context in which it is used (often inside a CASE expression). (For a more complete discussion of `raise_error`, see Section 5.1.3.) (SYSIBM)

rand ( ) → Double

rand (Integer) → Double

Returns a random double-precision floating-point number between 0 and 1. The optional argument is used as a "seed" to begin a new random number sequence. (SYSFUN)

real (*<any numeric datatype>*) → Real

Converts the argument to a real (single-precision) number. (SYSIBM)

repeat (Varchar *x*, Integer *n*) → Varchar(4000)

repeat (Clob(1M) *x*, Integer *n*) → Clob(1M)

repeat (Blob(1M) *x*, Integer *n*) → Blob(1M)

Returns a string consisting of *n* repetitions of the string *x*. (SYSFUN)

replace (Varchar *x*, Varchar *y*, Varchar *z*) → Varchar(4000)

replace (Clob(1M) *x*, Clob(1M) *y*, Clob(1M) *z*) → Clob(1M)

replace (Blob(1M) *x*, Blob(1M) *y*, Blob(1M) *z*) → Blob(1M)

Returns a copy of the string *x*, with all occurrences of the string *y* replaced by string *z*. (SYSFUN)

right (Varchar *x*, Integer *n*) → Varchar(4000)

right (Clob(1M) *x*, Integer *n*) → Clob(1M)

right (Blob(1M) *x*, Integer *n*) → Blob(1M)

Returns the rightmost *n* characters of string *x*. (SYSFUN)

round (Integer *x*, Integer *n*) → Integer

round (Double *x*, Integer *n*) → Double

Rounds the argument *x* in such a way that its least significant digit is *n* digits to the right of the decimal point (if *n* is negative, the least significant digit is to the left of the decimal point). For example, `round(12349, -2)` is 12,350. For a related function, see `truncate`. (SYSFUN)

rtrim (Varchar) → Varchar(4000)

rtrim (Clob(1M)) → Clob(1M)

Returns a copy of the argument with trailing blanks removed. (SYSFUN)

second (Time) → Integer

second (*<time string>*) → Integer

second (*<time duration>*) → Integer

second (Timestamp) → Integer

second (*<timestamp string>*) → Integer

second (*<timestamp duration>*) → Integer

Returns the seconds part of the argument. If the argument is a time, timestamp, or string, the result is an integer between 0 and 59. If the argument is a duration, the result is an integer between –99 and 99. (SYSIBM)

sign (*<any numeric datatype>*) → *<same datatype>*

Returns +1 if the argument is positive, –1 if the argument is negative, 0 if the argument is zero. Exception to return type rule: if argument type is Decimal or Real, return type is Double. (SYSFUN)

sin (Double) → Double

Returns the sine of an angle expressed in radians. (SYSFUN)

smallint (*<any numeric datatype>*) → Integer

smallint (Varchar) → Integer

Converts the argument to a small integer, by truncation of the decimal part if necessary. If the argument is of datatype Varchar, it must be a character-string representation of an integer between –32,768 and +32,767. (SYSIBM)

soundex (Varchar) → Char(4)

Returns a four-character code that represents the sound of the words in the argument. For a related function, see `difference`. (SYSFUN)

space (Integer *n*) → Varchar(4000)

Returns a string consisting of *n* spaces. (SYSFUN)

sqrt (Double) → Double

Returns the square root of the argument. (SYSFUN)

substr (*<any string datatype>* s, Integer *m*, Integer *n*) → *<string datatype>*

substr (*<any string datatype>* s, Integer *m*) → *<string datatype>*

substr (*<any DBCS datatype>* s, Integer *m*, Integer *n*) → *<DBCS datatype>*

substr (*<any DBCS datatype>* s, Integer *m*) → *<DBCS datatype>*

substr (Blob s, Integer *m*, Integer *n*) → Blob

substr (Blob s, Integer *m*) → Blob

Returns a substring of string *s*, beginning at character *m* and containing *n* characters. If the third argument is omitted, the substring begins at character *m* and contains the remainder of string *s*. The argument *s* is padded with blanks if necessary to make a result string of length *n*. The result string is the same datatype as string *s*, with certain exceptions: if string *s* is a Varchar or Long Varchar and *n* is a constant less than 255, the result datatype is Char(*n*). Similarly, if string *s* is a Vargraphic or Long Vargraphic and *n* is a constant less than 128, the result datatype is Graphic(*n*). (SYSIBM)

table_name (Varchar *t*, Varchar *s*) → Varchar(18)

table_name (Varchar *t*) → Varchar(18)

This function is used to resolve aliases. If the name *t* (or the qualified name *s.t*) is an alias, it is resolved to a table or view. The unqualified name of the resolved table or view is returned. If *t* (or *s.t*) is not an alias, *t* is returned. (SYSIBM)

table_schema (Varchar *t*, Varchar *s*) → Char(8)

table_schema (Varchar *t*) → Char(8)

This function is used to resolve aliases. If the name *t* (or the qualified name *s.t*) is an alias, it is resolved to a table or view. The schema name of the resolved table or view is returned. If *t* (or *s.t*) is not an alias, the schema portion of the input name is returned (*s*, or if argument *s* is not present, the current authid). (SYSIBM)

tan (Double) → Double

Returns the tangent of an angle expressed in radians. (SYSFUN)

time (Time) → Time

time (*<time string>*) → Time

time (Timestamp) → Time

time (*<timestamp string>*) → Time

Returns the time portion of the argument. If the argument is a string representation of a Time or a Timestamp, it is converted to an actual Time. (SYSIBM)

timestamp (Timestamp) → Timestamp

timestamp (*<timestamp string>*) → Timestamp

Returns the Timestamp represented by the argument. If the argument is a string representing a Timestamp, it is converted to a real Timestamp. (SYSIBM)

timestamp (Date, Time) → Timestamp

timestamp (Date, *<time string>*) → Timestamp

timestamp (*<date string>*, Time) → Timestamp

timestamp (*<date string>*, *<time string>*) → Timestamp

Returns a Timestamp whose date part is taken from the first argument and whose time part is taken from the second argument. The microsecond part of the Timestamp is set to 0. (SYSIBM)

timestampdiff (Integer *u*, Char(22) *d*) → Integer

The first argument, *u*, indicates a time unit, as follows: 256 = years, 128 = quarters, 64 = months, 32 = weeks, 16 = days, 8 = hours, 4 = minutes, 2 = seconds, 1 = microseconds (symbolic constants for these units can be found in `sqllib/include/sqlcli1.h`). The second argument, *d*, is the result of subtracting two Timestamps and converting the result to character form. The function returns an integer representing an estimation of the interval *d* expressed in units *u*. For example, the number of days between Timestamps `t1` and `t2` can be found by `timestampdiff(16, char(t2-t1))`. (SYSFUN)

timestamp_iso (Date) → Timestamp

timestamp_iso (*<date string>*) → Timestamp

timestamp_iso (Time) → Timestamp

timestamp_iso (*<time string>*) → Timestamp

timestamp_iso (Timestamp) → Timestamp

timestamp_iso (*<timestamp string>*) → Timestamp

Converts the argument to a Timestamp. If the argument is a Date, inserts zeros into the time fields of the Timestamp. If the argument is a Time, inserts the current date into the date fields of the Timestamp. (SYSFUN)

translate (Char) → Char( )

translate (Varchar) → Varchar( )

Returns a copy of the argument string in which all lowercase characters have been translated to uppercase. For a similar function, see ucase. (SYSIBM)

translate (Char *s*, Varchar *t*, Varchar *f*) → Char

translate (Varchar *s*, Varchar *t*, Varchar *f*) → Varchar

translate (Graphic *s*, Vargraphic *t*, Vargraphic *f*) → Graphic

translate (Vargraphic *s*, Vargraphic *t*, Vargraphic *f*) → Vargraphic

translate (Char *s*, Varchar *t*, Varchar *f*, Varchar *p*) → Char

translate (Varchar *s*, Varchar *t*, Varchar *f*, Varchar *p*) → Varchar

translate (Graphic *s*, Vargraphic *t*, Vargraphic *f*, Vargraphic *p*) → Graphic

translate (Vargraphic *s*, Vargraphic *t*, Vargraphic *f*, Vargraphic *p*) → Vargraphic

Returns a copy of string *s*, in which some of the characters are translated to different characters. String *f* (the "from string") specifies the characters to be translated, and string *t* (the "to string") specifies the characters to which they are to be translated. Any character in *s* that is also found in *f* is replaced by the corresponding character in *t*. If string *t* is shorter than string *f*, it is padded to the same length by the "pad character," *p*, which must be a single character. If no pad character is specified, the pad character is assumed to be a single-byte or double-byte blank. (SYSIBM)

truncate (Integer *x*, Integer *n*) → Integer

truncate (Double *x*, Integer *n*) → Double

Truncates the argument *x* in such a way that its least significant digit is *n* digits to the right of the decimal point (if *n* is negative, the least significant digit is to the left of the decimal point). For example, `truncate(12349, -2)` is 12,340. `truncate` can be abbreviated as `trunc`. For a related function, see `round`. (SYSFUN)

ucase (Varchar) → Varchar(4000)

Returns a copy of the argument string in which all lowercase characters have been converted to uppercase. (SYSFUN)

value (*<one or more arguments of compatible datatypes>*) → *<datatype of highest precedence>*

Returns the value of its first non-null argument, or null if all arguments are null. (See Section 6.6.2 for rules governing the compatibility of argument datatypes.) value and coalesce are different names for the same function. (SYSIBM)

varchar (*<any string datatype>*) → Varchar( )

varchar (*<any string datatype>*, Integer *n*) → Varchar(*n*)

varchar (*<any datetime datatype>*) → Varchar( )

Returns a copy of the first argument converted to datatype Varchar. The second argument, if present, becomes the maximum length of the result string, causing truncation if necessary. (SYSIBM)

vargraphic (*<any DBCS datatype>*) → Vargraphic( )

vargraphic (*<any DBCS datatype>*, Integer *n*) → Vargraphic(*n*)

vargraphic (Varchar) → Vargraphic( )

Returns a copy of the first argument converted to datatype Vargraphic. The second argument, if present, becomes the maximum length of the result string, causing truncation if necessary. (SYSIBM)

week (Date) → Integer

week (*<date string>*) → Integer

week (Timestamp) → Integer

week (*<timestamp string>*) → Integer

Returns the week of the year in which the argument occurs, expressed as an integer between 1 and 54. Each week is considered to start on Sunday. (SYSFUN)

year (Date) → Integer

year (*<date string>*) → Integer

year (*<date duration>*) → Integer

year (Timestamp) → Integer

year (*<timestamp string>*) → Integer

year (*<timestamp duration>*) → Integer

Returns the year part of the argument. If the argument is a date, timestamp, or string, the result is an integer between 1 and 9999. If the argument is a duration, the result is an integer between –9999 and 9999. (SYSIBM)

BEISPIELE

```
*************************************************************************
**
*
* README for IBM COBOL Samples on Windows
*
* Last updated: October 2002
*
* These programming examples are for the COBOL programming language, and
can
* be found in the "sqllib\samples\cobol" directory. Copy these files to
your
* working directory prior to building the sample programs.
*
* WARNING: Some of these samples may change your database or database
manager
*          configuration. Execute the samples against a 'test' database
only,
*          such as the DB2 SAMPLE database.
*
*************************************************************************
**
*     QUICKSTART
*
*  1) Copy \sqllib\samples\cobol\* to a working directory
*  2) Modify the makefile to reflect your environment:
*       o set DB (database name as cataloged: default is "sample")
*       o set UID (user ID to access the database)
*       o set PWD (password to access the database)
*
*  3) Excute 'nmake all' from the working directory
*
*************************************************************************
**
* For information on developing COBOL applications, see the
* Application Development Guide.
*
* For information on DB2 APIs, see the Administrative API Reference.
*
* For information on using SQL statements, see the SQL Reference.
*
* For the latest information on programming, compiling, and running
* DB2 applications, visit the DB2 application development website:
*     http://www.software.ibm.com/data/db2/udb/ad
*************************************************************************
**
*
*  +-----------------------------------------+
+--+How to build the SAMPLE programs using nmake+-------------------------
---+
|  +-----------------------------------------+
|
|  Note: The supplied makefile runs the IBM VisualAge COBOL compiler.
|
```

```
|
| |
|
|    Before building the sample programs, you require:
|
|    - The sample database (created with the db2sampl commmand).
|
|    - The database manager must be started (with the db2start command).
|
|    - Write permission to the the "sqllib\function" directory of
|      the database instance prior to building the following programs:
|
|
|         outsrv
|         inpsrv
|
|
|  Using the makefile in this directory, you can build any of the supplied
|  sample programs in one step by entering the following command:
|
|
|     nmake <prog_name>  (eg. nmake updat)
|
|
|  You can also use the makefile to build all the supplied sample programs
|  in one step by entering the following command:
|
|
|     nmake all
|
|
|  To remove the intermediate files, enter:
|
|
|     nmake clean
|
|
|  To remove all intermediate and the resulting executables, enter:
|
|
|     nmake cleanall
|
|
|
```

```
|  Although the makefile is easier to use, it may be more difficult to
|
|  understand, so batch files are also included, as described below.
|
+------------------------------------------------------------------------
---+


   +-----------------------------------------------------+
+--+ How to build the SAMPLE programs using batch files +-----------------
---+
|  +-----------------------------------------------------+
|
| To compile the following programs, you need to invoke the proper build
|
| commands. The supplied batch files are listed near the end of the README.
|
+------------------------------------------------------------------------
---+

Table Legend:
=============
  NOTES:  CLI    A client application, which calls a stored procedure.
          SRV    You must run this sample locally on the server.
          STP    A stored procedure, which must be stored on the server,
                 and must be called by a corresponding client application.

-------------+------+-----------------------------------------------------
 SAMPLE      |NOTES | PROGRAM DESCRIPTION
 PROGRAM     |      |
 NAME        |      |
-------------+------+-----------------------------------------------------
 advsql.sqb  |      |An example using advanced SQL expressions like
             |      |CASE, CAST, and scalar fullselects.
-------------+------+-----------------------------------------------------
 checkerr.cbl|      |Demonstrates the use of the following APIs:
             |      |     GET ERROR MESSAGE
             |      |     INSTALL SIGNAL HANDLER
             |      |     INTERRUPT
             |      |This program also contains code to output
             |      |information from an SQLDA.
-------------+------+-----------------------------------------------------
 client.cbl  |      |Demonstrates the use of the following APIs:
             |      |     SET CLIENT
             |      |     QUERY CLIENT
-------------+------+-----------------------------------------------------
 cursor.sqb  |      |Demonstrates the use of a "CURSOR" using static
             |      |SQL.
-------------+------+-----------------------------------------------------
 d_dbconf.cbl|      |Demonstrates the use of the following API:
             |      |     GET DATABASE CONFIGURATION DEFAULTS
-------------+------+-----------------------------------------------------
 d_dbmcon.cbl|      |Demonstrates the use of the following API:
             |      |     GET DATABASE MANAGER CONFIGURATION DEFAULTS
-------------+------+-----------------------------------------------------
 db_udcs.cbl |      |Demonstrates the use of the following APIs to
             |      |simulate the collating behavior of a DB2 for
```

		MVS/ESA CCSID 500 (EBCDIC International) collating sequence: CREATE DATABASE DROP DATABASE
dbauth.sqb		Demonstrates the use of the GRANT (Database Authorities) SQL statement
dbcat.cbl		Demonstrates the use of the following APIs: CATALOG DATABASE CLOSE DATABASE DIRECTORY SCAN GET NEXT DATABASE DIRECTORY ENTRY OPEN DATABASE DIRECTORY SCAN UNCATALOG DATABASE
dbcmt.cbl		Demonstrates the use of the following API: CHANGE DATABASE COMMENT
dbconf.cbl		Demonstrates the use of the following APIs: CREATE DATABASE DROP DATABASE GET DATABASE CONFIGURATION RESET DATABASE CONFIGURATION UPDATE DATABASE CONFIGURATION
dbinst.cbl		Demonstrates the use of the following APIs: ATTACH TO INSTANCE DETACH FROM INSTANCE GET INSTANCE
dbmconf.cbl		Demonstrates the use of the following APIs: GET DATABASE MANAGER CONFIGURATION RESET DATABASE MANAGER CONFIGURATION UPDATE DATABASE MANAGER CONFIGURATION
dbsnap.cbl		Demonstrates the use of the following API: DATABASE MONITOR SNAPSHOT
dbstart.cbl	SRV	Demonstrates the use of the following API: START DATABASE MANAGER
dbstat.sqb		Demonstrates the use of the following APIs: REORGANIZE TABLE RUN STATISTICS
dbstop.cbl	SRV	Demonstrates the use of the following APIs: FORCE USERS STOP DATABASE MANAGER
dcscat.cbl		Demonstrates the use of the following APIs: ADD DCS DIRECTORY ENTRY CLOSE DCS DIRECTORY SCAN GET DCS DIRECTORY ENTRY FOR DATABASE GET DCS DIRECTORY ENTRIES OPEN DCS DIRECTORY SCAN UNCATALOG DCS DIRECTORY ENTRY

delet.sqb		Uses static SQL to delete items from a database.
dynamic.sqb		Demonstrates the use of a "CURSOR" using dynamic SQL.
ebcdicdb.cbl		Demonstrates the use of the following APIs to simulate the collating behavior of a DB2 for MVS/ESA CCSID 037 (EBCDIC US English) collating sequence: CREATE DATABASE DROP DATABASE
expsamp.sqb		Demonstrates the use of the following APIs: EXPORT IMPORT in conjunction with a DRDA database.
impexp.sqb		Demonstrates the use of the following APIs: EXPORT IMPORT
inpcli.sqb	CLI	Demonstrates how to call a parameter style GENERAL stored procedure. This is the client program of a client/server example. (The server program is called "inpsrv".) The SQLCA status is returned to the client program. This program uses an embedded SQL CALL statement to call the stored procedure.
inpsrv.sqb	STP	Demonstrates parameter style GENERAL stored procedures. This is the server program of a client/server example. (The client program is called "inpcli".) The procedure in this file inserts a row into the ORG table in the SAMPLE database. The server program does all the database processing and returns the SQLCA status to the client program.
joinsql.sqb		An example using advanced SQL join expressions.
loadqry.sqb		Demonstrates how APIs are implemented in order to query the current status of a LOAD being invoked on the table "TABLE" in the database "SAMPLE".
lobeval.sqb		Demonstrates the use of deferring the evaluation of a LOB within a database.
lobfile.sqb		Demonstrates the use of LOB file handles.
lobloc.sqb		Demonstrates the use of LOB locators.
migrate.cbl		Demonstrates the use of the following API: MIGRATE DATABASE
monreset.cbl		Demonstrates the use of the follwing API: RESET DATABASE SYSTEM MONITOR DATA AREAS

monsz.cbl		Demonstrates the use of the following APIs: ESTIMATE DATABASE SYSTEM MONITOR BUFFER SIZE DATABASE SYSTEM MONITOR SNAPSHOT
nodecat.cbl		Demonstrates the use of the following APIs: CATALOG NODE CLOSE NODE DIRECTORY SCAN GET NEXT NODE DIRECTORY ENTRY OPEN NODE DIRECTORY SCAN UNCATALOG NODE
openftch.sqb		Uses static SQL to fetch rows and then UPDATE or DELETE them.
outcli.sqb	CLI	Demonstrates stored procedures using the SQLDA structure. This is the client program of a client/server example. (The server program is called outsrv.sqb) This program allocates and initializes a one variable SQLDA, and passes it to the server program for further processing. The filled SQLDA is returned to the client program along with the SQLCA status. This uses an embedded SQL CALL statement to call a stored procedure.
outsrv.sqb	STP	Demonstrates stored procedures using the SQLDA structure. This is the server program of a client/server example. (The client program is called outcli.sqb) The program fills the SQLDA with the median "SALARY" of the employees in the "STAFF" table of the "SAMPLE" database. The server program does all the database processing (finding the median). The server program returns the filled SQLDA and the SQLCA status to the client program.
prepbind.sqb	SRV	Demonstrates the use of the following APIs: BIND PRECOMPILE PROGRAM
qload.sqb		Demonstrates the use of the LOAD QUERY API
rebind.sqb		Demonstrates the use of the REBIND API
restart.cbl	SRV	Demonstrates the use of the following API: RESTART DATABASE MANAGER
setact.cbl		Demonstrates the use of the following API: SET ACCOUNTING STRING
spcat		Script to uncatalog/catalog stored procedures by calling spdrop.db2 and spcreate.db2. This script must be used for inpsrv, but is not required for outsrv.
spcreate.db2		CLP script to catalog the stored procedure called by

		inpcli. This is not required for outsrv.
spdrop.db2		CLP script to uncatalog the stored procedure called by inpcli. This is not required for outsrv.
static.sqb		Uses static SQL to retrieve information.
sws.cbl		Demonstrates the use of the following API: DATABASE MONITOR SWITCH
tabscont.sqb		Demonstrates the use of the following APIs: TABLESPACE CONTAINER QUERY OPEN TABLESPACE CONTAINER QUERY FETCH TABLESPACE CONTAINER QUERY CLOSE TABLESPACE CONTAINER QUERY SET TABLESPACE CONTAINER QUERY
tabspace.sqb		Demonstrates the use of the following APIs: TABLESPACE QUERY SINGLE TABLESPACE QUERY OPEN TABLESPACE QUERY FETCH TABLESPACE QUERY GET TABLESPACE STATISTICS CLOSE TABLESPACE QUERY
tabsql.sqb		An example using advanced SQL table expressions.
tload.sqb	SRV	Demonstrates the use of the following APIs: EXPORT QUIESCE TABLESPACES FOR TABLE LOAD
trigsql.sqb		An example using advanced SQL triggers and contraints.
tspace.sqb		Demonstrates the use of the following APIs: COPY MEMORY FREE MEMORY TABLESPACE QUERY TABLESPACE CONTAINER QUERY SINGLE TABLESPACE QUERY OPEN TABLESPACE QUERY FETCH TABLESPACE QUERY CLOSE TABLESPACE QUERY OPEN TABLESPACE CONTAINER QUERY FETCH TABLESPACE CONTAINER QUERY CLOSE TABLESPACE CONTAINER QUERY SET TABLESPACE CONTAINERS GET TABLESPACE STATISTICS
updat.sqb		Uses static SQL to update a database.
varinp.sqb		An example of variable input to Embedded Dynamic SQL calls using parameter markers.

BATCH FILE NAME	NOTES	DESCRIPTION
bldapp.bat		Builds an IBM COBOL application program.
bldrtn.bat		Builds an IBM COBOL routine (stored procedure) program.
embprep.bat		Batch file to prep and bind a COBOL embedded SQL program.

OTHER	NOTES	DESCRIPTION
makefile		Builds most of the supplied programming examples in the "sqllib\samples\cobol" subdirectory.
README		Lists and describes at a high-level, all files in the "sqllib\samples\cobol" sub-directory (this file).

```
*************************************************************************
** Licensed Materials - Property of IBM
**
** Governed under the terms of the International
** License Agreement for Non-Warranted Sample Code.
**
** (C) COPYRIGHT International Business Machines Corp. 1995 - 2002
** All Rights Reserved.
**
** US Government Users Restricted Rights - Use, duplication or
** disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

*************************************************************************
**
** SOURCE FILE NAME: updat.sqb
**
** SAMPLE: How to update, delete and insert table data
**
**         This sample program demonstrates the use of static SQL.
**         It will obtain all managers in the STAFF table of the
**         SAMPLE database and change their job from 'Mgr' to
**         'Clerk', deletes all who are 'Sales', and inserts a new
**         row.  In all three SQL statements (UPDATE, DELETE,
**         INSERT) a host variable is implemented. A ROLLBACK will
**         be done so that the SAMPLE database remains unchange.
**
** SQL STATEMENTS USED:
**         BEGIN DECLARE SECTION
**         END DECLARE SECTION
**         ROLLBACK
**         CONNECT
**         UPDATE
**         DELETE
**         INSERT
**
** OUTPUT FILE: updat.out (available in the online documentation)

*************************************************************************
**
** For more information on the sample programs, see the README file.
**
** For information on developing COBOL applications, see the
** Application Development Guide.
**
** For information on using SQL statements, see the SQL Reference.
**
** For the latest information on programming, compiling, and running
** DB2 applications, visit the DB2 application development website:
**     http://www.software.ibm.com/data/db2/udb/ad

*************************************************************************

       Identification Division.
       Program-ID. "updat".

       Data Division.
       Working-Storage Section.
```

```
           copy "sql.cbl".
           copy "sqlenv.cbl".
           copy "sqlca.cbl".                                          1

           EXEC SQL BEGIN DECLARE SECTION END-EXEC.                   2
       01 statement          pic x(80).
       01 userid             pic x(8).
       01 passwd.
         49 passwd-length    pic s9(4) comp-5 value 0.
         49 passwd-name      pic x(18).
       01 job-update         pic x(5).
           EXEC SQL END DECLARE SECTION END-EXEC.

      * Local variables
       77 errloc           pic x(80).
       77 error-rc         pic s9(9) comp-5.
       77 state-rc         pic s9(9) comp-5.

      * Variables for the GET ERROR MESSAGE API
      * Use application specific bound instead of BUFFER-SZ
       77 buffer-size      pic s9(4) comp-5 value 1024.
       77 line-width       pic s9(4) comp-5 value 80.
       77 error-buffer     pic x(1024).
       77 state-buffer     pic x(1024).

       Procedure Division.
       Main Section.
           display "Sample COBOL program:  UPDAT".

           display "Enter your user id (default none): "
                with no advancing.
           accept userid.

           if userid = spaces
             EXEC SQL CONNECT TO sample END-EXEC
           else
             display "Enter your password : " with no advancing
             accept passwd-name.

      * Passwords in a CONNECT statement must be entered in a VARCHAR
format
      * with the length of the input string.
           inspect passwd-name tallying passwd-length for characters
              before initial " ".

           EXEC SQL CONNECT TO sample USER :userid USING :passwd      3
               END-EXEC.
           move "CONNECT TO" to errloc.
           call "checkerr" using SQLCA errloc.

           move "Clerk" to job-update.
           EXEC SQL UPDATE staff SET job=:job-update                  4
                    WHERE job='Mgr' END-EXEC.
           move "UPDATE STAFF" to errloc.
           call "checkerr" using SQLCA errloc.
```

```
        display "All 'Mgr' have been demoted to 'Clerk'!".

        move "Sales" to job-update.
        EXEC SQL DELETE FROM staff WHERE job=:job-update END-EXEC.   5
        move "DELETE FROM STAFF" to errloc.
        call "checkerr" using SQLCA errloc.

        display "All 'Sales' people have been deleted!".

        EXEC SQL INSERT INTO staff VALUES (999, 'Testing', 99,       6
                 :job-update, 0, 0, 0) END-EXEC.
        move "INSERT INTO STAFF" to errloc.
        call "checkerr" using SQLCA errloc.

        display "New data has been inserted".

        EXEC SQL ROLLBACK END-EXEC.                                  7
        move "ROLLBACK" to errloc.
        call "checkerr" using SQLCA errloc.

        DISPLAY "On second thought -- changes rolled back."

        EXEC SQL CONNECT RESET END-EXEC.
        move "CONNECT RESET" to errloc.
        call "checkerr" using SQLCA errloc.

    End-Prog.
        stop run.
```

QUELLEN

Kapitel 2 - DB2 Funktionen

Chamberlin,D.:
Comp.Guide.DB2 Univ.Data.
KNV **9 78 30 72** ISBN 1-55860-482-0 WG **16355**
62,30 EUR