SQL-Syntax 3

```
query-expression ::=
    query-term
  | query-expression { UNION | EXEPT } query-term

query-term ::=
    query-primary
  | query-term  INTERSECT  query-primary

query-primary ::=
    query-specification
  | (query-expression)

query-specification ::=
    SELECT [ ALL | DISTINCT ] select-list
    FROM table-reference [{ , table-reference }...]
    [ WHERE search-condition ]
    [ order-by-clause ]
    [ group-by-clause ]
    [ having-clause ]

select-list ::=
    *
  | select-sublist [{ , select-sublist }...]

select-sublist ::=
    value-expression [[ AS ] column-name ]
  | table-name.*
  | range-variable.*

table-reference ::=
    table-name [ range-variable ]
  | ( query-expression )  range-variable
  | joined-table

value-expression ::=
    numeric-value-expression
  | string-value-expression
  | datetime-value-expression
  | interval-value-expression

numeric-value-expression ::=
    term
  | numeric-value-expression { + | - } term

term ::=
    factor
  | term { * | / } factor

factor ::=
    [ + | -] numeric-primary

numeric-primary ::=
    unsigned-numeric-value
  | column-name
  | set-function-specification
  | numeric-value-function
  | (numeric-value-expression)
set-function-specification ::=
    COUNT (*)
  | set-function-type ( [ ALL | DISTINCT ] numeric-value-expression )

set-function-type ::=
```

```
    AVG | MAX | MIN | SUM | COUNT

joined-table ::=
    table-reference [ join-type ] JOIN table-reference ON search-condition
  | ( joined-table )

join-type ::=
    INNER
  | { LEFT | FULL | RIGHT } [ OUTER]

search-condition ::=
    boolean-term
  | search-condition  OR  boolean-term

boolean-term ::=
    [ NOT ] boolean-primary
  | boolean-term  AND  [ NOT ] boolean-primary

boolean-primary ::=
    predicate
  | ( search-condition )

predicate ::=
    comparsion-predicate
  | null-predicate
  | between-predicate
  | like-predicate
  | in-predicate
  | quantified-comparsion-predicate
  | exists-predicate

comparsion-predicate ::=
    row-value-constructor  comp-op  row-value-constructor

comp-op ::=
     = | <> | < | > | <= | >=

row-value-constructor ::=
    value-expression
  | row-subquery

null-predicate ::=
    row-value-constructor IS [ NOT ] NULL

between-predicate ::=
    row-value-constructor [ NOT ] BETWEEN row-value-constructor AND row-value-constructor

like-predicate ::=
    character-value-expression [ NOT ] LIKE character-value-expression

in-predicate ::=
    row-value-constructor [ NOT ] IN in-predicate-value


in-predicate-value ::=
    ( value-expression [{ , value-expression }...] )
  | ( table-subquery )

quantified-comparsion-predicate ::=
    row-value-constructor  comp-op  { ALL |  { SOME | ANY } }  ( table-subquery )

exists-predicate ::=
    EXISTS (  table-subquery )
```

```
order-by-clause ::=
    ORDER BY column-name [ ASC | DSC ] [{ , column-name [ ASC | DSC ]}...]

group-by-clause ::=
    GROUP BY column-name [{ , column-name }...]

having-clause ::=
    HAVING search-condition

insert-statement ::=
    INSERT INTO table-name [( column-name [{ , column-name }...]  )] insert-source

insert-source ::=
    VALUES ( value-expression [{ , value-expression }...] )
  | TABLE table-name
  | query-expression
  | joined-table

delete-statement ::=
    DELETE FROM table-name [ range-variable ] [ WHERE search-condition ]

update-statement ::=
    UPDATE table-name [ range-variable ]
    SET coulmn-name = row-value-constructor [{ , column-name = row-value-constructor}...]
    [ WHERE search-condition ]

table-definition ::=
    CREATE TABLE table-name
    ( column-definition [{ , column-definition }...]
    [ table-constraint [{ , table-constraint }...] ] )

column-definition ::=
    column-name data-type [ column-constraint [{ , column-constraint }...] ]

column-constraint ::=
  | NOT NULL
  | UNIQUE
  | CHECK ( check-condition-1 )
  | DEFAULT { const | NULL }

table-constraint ::=
  | CHECK ( check-condition-2 )
  | UNIQUE ( column-name [{ , column-name }...]
  | PRIMARY KEY ( column-name [{ , column-name }...] )
  | FOREIGN KEY ( column-name [{ , column-name }...] ) reference-specification

reference-specification ::=
    REFERENCES table-name [ ( column-name [{ , column-name }...] ) ]
    [ON DELETE { CASCADE | SET NULL | RESTRICT | NO ACTION }]

alter-table-statement ::=
    ALTER TABLE table-name action

action ::=
    ADD column-definition
  | ADD table-constraint

rename-table-statement ::=
    RENAME TABLE table-name TO new-table-name

view-definition ::=
    CREATE VIEW table-name [ ( column-name [{ , column-name }...] )
    AS query-expression
    [ WITH CHECK OPTION ]
```

```
grant-statement ::=
    GRANT privileg [{ , privileg }...]
    ON table-name
    TO grantee [{ , grantee }...]
    [ WITH GRANT OPTION ]

privileg ::=
    ALL PRIVILEGES
  | SELECT
  | DELETE
  | INSERT [ ( column-name [{ , column-name }...] ) ]
  | UPDATE [ ( column-name [{ , column-name }...] ) ]

grantee ::=
    PUBLIC
  | authorization-identifier

revoke-statement ::=
    REVOKE privileg [{ , privileg }...]
    ON table-name
    FROM grantee [{ , grantee }...]

trigger-definition ::=
    CREATE TRIGGER trigger-name
    [ NO CASCADE BEFORE | AFTER ] trigger-event ON table-name
    [ REFERENCING referencing-specification  [{referencing-specification }...]  ]
    FOR EACH [ STATEMENT | ROW ]  MODE DB2SQL
    [ WHEN search-condition ]
    triggered-action

trigger-event ::=
    INSERT
  | DELETE
  | UPDATE [ OF column-name [{ , column-name }...] ]

referencing-specification ::=
    OLD [AS] range-variable
  | NEW [AS] range-variable
  | OLD_TABLE [AS] table-identifier
  | NEW_TABLE [AS] table-identifier



triggered-action ::=
    triggered-statement
  | BEGIN ATOMIC triggered-statement; [{ triggered-statement; }...] END

triggered-statement::=        (Before-Trigger)
    set-statement
  | signal-statement

set-statement ::=
    SET column-name = row-value-constructor [{ , column-name = row-value-constructor}...]

signal-statement ::=
    SIGNAL SQLSTATE state ( message )

triggered-statement::=        (After-Trigger)
    insert-statement
  | delete-statement
  | update-statement
  | signal-statement
```

Embedded-SQL-Syntax

```
embedded-sql-include ::=
    EXEC SQL INCLUDE { header-file | SQLCA } ;

embedded-sql-statement ::=
    EXEC SQL statement-or-declaration ;

statement-or-declaration ::=
    SQL-procedure-statement
  | embedded-exeption-declaration
  | declare-cursor
  | dynamic-declare-cursor

SQL-procedure-statement ::=
    SQL-connection-statement
  | SQL-data-statement
  | SQL-transaction-statement
  | SQL-dynamic-statement
  | SQL-dynamic-data-statement
  | SQL-schema-statement

SQL-data-statement ::=
    insert-statement
  | delete-statement
  | update-statement
  | select-statement-single-row
  | open-statement
  | fetch-statement
  | close-statement
  | delete-statement-positioned
  | update-statement-positioned
```

SQL-connection-statement ::=
    CONNECT TO { db-name | host-variable }
    [IN SHARE MODE | IN EXCLUSIVE MODE]
    [USER { username | host-variable }]
    [USING { passwd | host-variable }]
  | CONNECT RESET

host-variable ::=
    : variable-name [[INDICATOR] : variable-name]

Indikatorwert	***Semantik***
-1	Die Hostvariable hat den Wert NULL
0	Die Hostvariable hat einen realen Wert (nicht NULL)

embedded-sql-declare-section ::=
    EXEC SQL BEGIN DECLARE SECTION ;
    host-variable-definition [{ , host-variable-definition }...]
    EXEC SQL END DECLARE SECTION ;

select-statement-single-row ::=
    SELECT [ALL | DISTINCT] select-list
    INTO host-variable [{ , host-variable }...]
    FROM table-reference [{ , table-reference }...]
    [WHERE search-condition]

declare-cursor ::=
    DECLARE cursor-name CURSOR [WITH HOLD] FOR cursor-specification

cursor-specification ::=
    query-expression [updatability-clause]

updatability-clause ::=
    FOR READ ONLY
  | FOR UPDATE [OF column-name [{ , colunm-name }...]]

open-statement ::=
    OPEN cursor-name

fetch-statement ::=
    FETCH cursor-name INTO host-variable [{ , host-variable }...]

SQLCODE-Wert	Semantik
0	SQL-Anweisung erfolgreich
100	SQL-Anweisung erfolgreich, aber keine weiteren Daten gefunden
<0	Fehler, SQL-Anweisung nicht ausgeführt

close-statement ::=
    CLOSE cursor-name

update-statement-positioned ::=
    update-statement WHERE CURRENT OF cursor-name

delete-statement-positioned ::=
    delete-statement WHERE CURRENT OF cursor-name

```
SQL-transaction-statement ::=
    COMMIT [ WORK ]
  | ROLLBACK [ WORK ]
  | compound-statement

compound-statement ::=
    BEGIN COMPOUND { ATOMIC | NOT ATOMIC } STATIC
            SQL-statement ; [{ SQL-statement ; }...]
    END COMPOUND

embedded-exeption-declaration ::=
    WHENEVER { NOT FOUND | SQLERROR | SQLWARNING } action

action ::=
    { CONTINUE | GO TO label }

SQL-dynamic-statement ::=
    prepare-statement
  | execute-statement
  | execute-immediate-statement
  | describe-statement

execute-immediate-statement ::=
    EXECUTE IMMEDIATE statement-variable

prepare-statement ::=
    PREPARE statement-name [ INTO descriptor ] FROM statement-variable

describe-statement ::=
    DESCRIBE statement-name INTO descriptor

execute-statement ::=
    EXECUTE statement-name [ using-clause ]

using-clause ::=
    USING host-variable [{ , host-variable }...]
  | USING DESCRIPTOR descriptor

dynamic-declare-cursor ::=
    DECLARE cursor-name CURSOR [ WITH HOLD ] FOR statement-name

SQL-dynamic-data-statement ::=
    dynamic-open-statement
  | dynamic-fetch-statement
  | dynamic-close-statement

dynamic-open-statement ::=
    OPEN cursor-name [ using-clause ]

using-clause ::=
    USING host-variable [{ , host-variable }...]
  | USING DESCRIPTOR descriptor

dynamic-fetch-statement ::=
    FETCH cursor-name  into-using-clause

into-using-clause ::=
    INTO host-variable [{ , host-variable }...]
  | USING DESCRIPTOR descriptor

dynamic-close-statement ::=
    CLOSE cursor-name
```

SQLDA-Struktur

```
struct  sqlda
{
    char            sqldaid[8];
    long                sqldabc;
    short               sqln;
    short           sqld;
    struct sqlvar       sqlvar[1];
};

struct  sqlvar
{
   short                sqltype;
   short                sqllen;
   char *               sqldata;
   short *              sqlind;
   struct sqlname   sqlname;
};

struct sqlname
{
   short            length;
   char                 data[30];
};
```

Externe skalare Funktionen

```
extern-scalar-function ::=
    void SQL_API_FN function-name ( parameter )
    {
        function-body
    }

parameter ::=
    [ { input-parameter , }... ]
    output-parameter ,
    [ { input-parameter-indicator , }... ]
    output-parameter-indicator ,
    sql-state,
    function-name,
    specific-name,
    error-message
    [ , scratchpad  [ , call-type  [ , dbinfo  ] ] ]

input-parameter ::=
    c-data-type  *  variable-pointer

output-parameter ::=
        c-data-type  *  variable-pointer

input-parameter-indicator ::=
    short * variable-pointer
  | SQLUDF_NULLIND * variable-pointer
```

```
output-parameter-indicator ::=
    short * variable-pointer
  | SQLUDF_NULLIND * variable-pointer

sql-state ::=
    char * sql-state-pointer

function-name ::=
    char * function-name-pointer

specific-name ::=
    char * specific-name-pointer

error-message ::=
    char * error-message-pointer

scratchpad ::=
    struct sqludf_scratchpad * scratchpad-pointer

call-type  ::=
    enum sqludf_call_type * call-type-pointer

dbinfo  ::=
    struct  sqludf_dbinfo * dbinfo-pointer

create-extern-scalar-function ::=
    CREATE FUNCTION  function-name
    ( [ parameter-declaration [ { , parameter-declaration  }... ] ]  )
    RETURNS return-type
    EXTERNAL [ NAME external-name ]
    LANGUAGE { C | JAVA | CLR | OLE }
    PARAMETER STYLE { DB2GENERAL | JAVA | SQL  }
    [ { option } ... ]

option ::=
    specific-clause
  | fenced-clause
  | null-call-clause
  | dbinfo-clause
  | deterministic-clause
  | external-action-clause
  | scratchpad-clause
  | final-call-clause
  | sql-clause

function-name ::=
    [ schema-name. ] name

parameter-declaration ::=
    [ parameter-name ] sql-data-type [ AS LOCATOR ]

return-type ::=
```

```
    sql-data-type [ AS LOCATOR ]
  | sql-data-type1  CAST FROM sql-data-type2 [ AS LOCATOR ]

external-name ::=
    [ path ] library [ ! funct-id ]

specific-clause  ::=
    SPECIFIC specific-function-name

fenced-clause ::=
    [ NOT ] FENCED

null-call-clause ::=
    RETURNS NULL ON NULL INPUT
  | CALLED ON NULL INPUT

dbinfo-clause ::=
    [ NO ]  DBINFO

deterministic-clause ::=
    [ NOT ] DETERMINISTIC

external-action-clause ::=
    [ NO ] EXTERNAL ACTION

scratchpad-clause ::=
    NO SCRATCHPAD
  | SCRATCHPAD [ length ]

final-call-clause ::=
    NO FINAL CALL
  | FINAL CALL

sql-clause ::=
    NO SQL
  | CONTAINS SQL
  | READS SQL DATA
```

Externe Tabellenfunktionen

```
extern-table-function ::=
    void SQL_API_FN function-name ( parameter )
    {
            function-body
    }

parameter ::=
    [ { input-parameter, }... ]
    output-parameter, [ { output-parameter,  }... ]
    [ { input-parameter-indicator, }... ]
    output-parameter-indicator, [ { output-parameter-indicator, }... ]
```

```
    sql-state,
    function-name,
   specific-name,
   error-message,
   scratchpad,
   call-type,
   [ dbinfo  ]


create-extern-table-function ::=
   CREATE FUNCTION  function-name
   ( [ parameter-declaration  [ { ,  parameter-declaration  }... ] ]  )
   RETURNS TABLE ( column-declaration  [ { , column-declaration }... ] )
   EXTERNAL [ NAME external-name ]
   LANGUAGE { C | JAVA | CLR | OLE }
   PARAMETER STYLE { DB2GENERAL | SQL  }
   DISALLOW PARALLEL
   [ { option }... ]

option ::=
   specific-clause
  | fenced-clause
  | null-call-clause
  | dbinfo-clause
  | deterministic-clause
  | external-action-clause
  | scratchpad-clause
  | final-call-clause
  | sql-clause


column-declaration ::=
   column-name  sql-data-type  [ AS LOCATOR ]

final-call-clause::=
   NO FINAL CALL
  | FINAL CALL

sql-clause ::=
   NO SQL
  | CONTAINS SQL
  | READS SQL DATA
```

Externe Prozeduren

```
extern-procedure ::=
   SQL_API_RC SQL_API_FN procedure-name (  parameter-list  )
   {
         procedure-body
   }

create-extern-procedure ::=
   CREATE PROCEDURE  procedure-name
```

```
    ( [ parameter-declaration  [ { ,  parameter-declaration  }... ] ]  )
    EXTERNAL [ NAME external-name ]
    LANGUAGE { C | JAVA | COBOL | CLR | OLE }
    PARAMETER STYLE { parameter-style }
    [ { option }... ]

option ::=
    specific-clause
  | fenced-clause
  | null-call-clause
  | dbinfo-clause
  | deterministic-clause
  | external-action-clause
  | sql-clause
  | program-type-clause
  | result-set-clause

parameter-declaration ::=
    [ IN  | OUT | INOUT ] [ parameter-name ] sql-data-type

parameter-style ::=
    SQL
  | DB2SQL
  | GENERAL
  | GENERAL WITH NULLS
  | DB2GENERAL
  | JAVA
```

Prozedurdefinition für PARAMETER STYLE DB2SQL

```
parameter-list ::=
    [ { parameter , }... { indicator , }... ]
    sql-state,
    function-name,
    specific-name,
    error-message
    [ , dbinfo  ]
```

Prozedurdefinition für PARAMETER STYLE GENERAL

```
parameter-list ::=
    [ { parameter , }... ] [  dbinfo  ]
```

Prozedurdefinition für PARAMETER STYLE GENERAL WITH NULLS

```
parameter-list ::=
    [ { parameter , }...  parameter-indicator-array ] [  dbinfo  ]

sql-clause ::=
    NO SQL
  | CONTAINS SQL
  | READS SQL DATA
  | MODIFIES SQL DATA
```

```
program-type-clause ::=
    PROGRAM TYPE { SUB  |  MAIN }

result-set-clause ::=
    DYNAMIC RESULT SET {  0  |  int-value }
call-statment ::=
    CALL procedure-name (  [ expression [ { , expressiont ] } ] )
```

SQL-Routinen

```
SQL-routine-statement ::=
    SQL-procedure-statement
  | SQL-function-statement

set-variable-statement ::=
    SET sql-variable-name = { expression | NULL }
  | SET ( sql-variable-name  [ { , sql-variable-name }... ] )
                    = ( { expression | NULL } [ { , { expression | NULL }  }... ] )
  | SET ( sql-variable-name  [ { , sql-variable-name }... ] ) = ( row-select )
  | SET parameter-name = { expression | NULL }

if-statement ::=
    IF search-condition THEN { SQL-routine-statement ; }...
    [ { ELSEIF search-condition THEN { SQL-routine-statement ; }...  }... ]
    [ ELSE { SQL-routine-statement ; }...  ]
    END IF

case-statement ::=
    CASE { searched-when-clause | simple-when-clause } END CASE

simple-when-clause ::=
    expression1 { WHEN expression2  THEN  { SQL-procedure-statement ; }... }...
    [ ELSE { SQL-procedure-statement ; }... ]

searched-when-clause ::=
    { WHEN search-condition  THEN  { SQL-procedure-statement ; }... }...
    [ ELSE { SQL-procedure-statement ; }... ]

while-statement ::=
    [ label : ]
    WHILE search-condition  DO  { SQL-routine-statement ; }... END WHILE
    [ label ]

repeat-statement ::=
    [ label : ]
    REPEAT { SQL-routine-statement ; }... UNTIL search-condition END REPEAT
    [ label ]

loop-statement ::=
    [ label : ]
    LOOP { SQL-routine-statement ; }...  END LOOP
```

```
    [ label ]

leave-statement ::=
    LEAVE  label

iterate-statement ::=
    ITERATE  label

for-statement ::=
    [ label : ]
    FOR  for-loop-name AS  select-statement DO { SQL-routine-statement ; }...  END FOR
    [ label ]

SQL-variable-declaration ::=
    DECLARE SQL-variable-name
                [ { , SQL-variable-name } ] SQL-data-type [ DEFAULT { NULL | constant } ]

condition-declaration ::=
    DECLARE condition-name CONDITION FOR SQLSTATE string-constant

handler-declaration ::=
    DECLARE handler-type HANDLER FOR  condition  SQL-procedure-statement

handler-type ::=
    CONTINUE
  | EXIT
  | UNDO

condition ::=
    general-condition [ { , general-condition } ]
  | specific-condition [ { , specific-condition } ]

general-condition
    SQLEXCEPTION
  | SQLWARNING
  | NOT FOUND

specific-condition ::=
    condition-name
  | SQLSTATE string-constant

signal-statement ::=
    SIGNAL SQLSTATE state [ message ]

message ::=
    SET MESSAGE_TEXT = string-expression
  | (string-expression)

resignal-statement ::=
    RESIGNAL  [ SQLSTATE state ] [ message ]

message ::=
    SET MESSAGE_TEXT = { string-expression | variable-name }
```

return-statement ::=
    RETURN [return-value]

return-value ::=
    expression
  | *NULL*
  | *query-expression*

SQL-Funktionen

create-SQL-function ::=
    CREATE FUNCTION function-name
    ([parameter-declaration [{ , parameter-declaration }...]])
    RETURNS return-types
    [{ option }...]
    SQL-function-body

option ::=
    specific-clause
  | language-clause
  | null-call-clause
  | deterministic-clause
  | external-action-clause
  | sql-clause
parameter-declaration ::=
    parameter-name sql-data-type

return-types ::=
    sql-data-type
  | TABLE (column-name sql-data-type [{ , column-name sql-data-type }...])
  | ROW (column-name sql-data-type [{ , column-name sql-data-type }...])

language-clause ::=
    LANGUAGE SQL

sql-clause ::=
    CONTAINS SQL
  | READS SQL DATA
  | MODIFIES SQL DATA

SQL-function-body ::=
    RETURN SQL-statement
  | *SQL-function-compound-statement*

SQL-function-compound-statement ::=
    [label:]
    BEGIN ATOMIC
    [{ SQL-variable-declaration ; }...]
    [{ condition-declaration ; }...]
    { SQL-function-statement ; }...
    END

[label]

SQL-function-statement ::=
query-expression
| *insert-statement*
| *delete-statement*
| *update-statement*
| *call-statement*
| *set-variable-statement*
| *if-statement*
| *for-statement*
| *while-statement*
| *iterate-statement*
| *leave-statement*
| *signal-statement*

SQL-Prozeduren

create-SQL-procedure ::=
CREATE PROCEDURE procedure-name
([parameter-declaration [{ , parameter-declaration }...]])
[{ option }...]
SQL-procedure-body

option ::=
specific-clause
| language-clause
| null-call-clause
| deterministic-clause
| external-action-clause
| sql-clause
| result-set-clause

parameter-declaration ::=
[IN | OUT | INOUT] parameter-name sql-data-type

language-clause ::=
LANGUAGE SQL

sql-clause ::=
CONTAINS SQL
| READS SQL DATA
| MODIFIES SQL DATA

SQL-procedure-body ::=
SQL-procedure-statement
| SQL-procedure-compound-statement

SQL-procedure-compound-statement ::=
[label:]
BEGIN [NOT ATOMIC | ATOMIC]

```
    [ {  SQL-variable-declaration ; }... ]
    [ {  condition-declaration ; }... ]
    [ {  return-codes-declaration ; }... ]
    [ {  statement-declaration ; }... ]
    [ {  declare-cursor-statement ; }... ]
    [ {  handler-declaration ; }... ]
    { SQL-procedure-statement ; }...
    END
    [ label ]

return-codes-declaration ::=
    DECLARE  return-code-variable

return-code-variable ::=
    SQLSTATE  CHAR(5) [ DEFAULT  { '00000' | string-const } ]
     |  SQLCODE  INTEGER [ DEFAULT  { 0 | integer-const } ]

statement-declaration ::=
    DECLARE statement-name STATEMENT

declare-cursor-statement ::=
    DECLARE  cursor-name  CURSOR  [ WITH HOLD ]
    [ WITH RETURN [ TO CALLER | TO CLIENT ] ]
    FOR  { select-statement | statement-name }

call-statement ::=
    CALL procedure-name (  [ argument [ { , argument ] } ] )

argument ::=
    expression
  | NULL
  | ?
```