The SQL Guide to Ingres
Appendix A – Syntax of SQL

Rick F. van der Lans

Appendix A

Syntax of SQL

A.1 Introduction

In this appendix, we explain the notation method we have used to define the statements, the definitions of the SQL statements discussed in this book, and show the list of reserved words.

The definitions in this appendix can differ from those in the previous chapters. The main reason for this is that, in the chapters, we explained the statements and concepts step by step. To avoid too much detail, we sometimes used simple versions of the definitions. This appendix contains the complete definitions.

A.2 The BNF Notation

In this appendix and throughout the book, we have used a formal notation method to describe the syntax of all SQL statements and the common elements. This notation is a derivative of the so-called *Backus Naur Form* (BNF), which is named after John Backus and Peter Naur. The meaning of the metasymbols that we use is based on that of the metasymbols in the SQL standard.

BNF adopts a language of *substitution rules* or *production rules*, consisting of a series of symbols. Each production rule defines one *symbol*. A symbol could be, for example, an SQL statement, a table name or a colon. A *terminal symbol* is a special type of symbol. All symbols, apart from the terminal symbols, are defined in terms of other symbols in a production rule. Examples of terminal symbols are the term CREATE and the semicolon.

You can compare a production rule with the definition of an element, in which the definition of that element uses elements defined elsewhere. In this case, an element equates to a symbol.

The following *metasymbols* do not form part of the SQL language, but belong to the notation technique:

- < >
- ::=
- |
- [ ]
- ...
- { }
- ;
- "

We now explain each of these symbols.

The Symbols < and >

Nonterminal symbols are presented in angle brackets. A production rule exists for every non-terminal symbol. We show the names of the non-terminal symbols in lowercase letters. Two examples of non-terminal symbols are `<select statement>` and `<table reference>`.

The ::= Symbol

The `::=` symbol is used in a production rule to separate the non-terminal symbol that is defined (left) from its definition (right). The `::=` symbol should be read as "is defined as." See the following example of the production rule for the DROP INDEX statement:

```
<drop index statement> ::= DROP INDEX <index name>
```

Explanation: The DROP INDEX statement consists of the terminal symbols `DROP` and `INDEX` followed by the non-terminal symbol index name. A production rule also should exist for `<index name>`.

The | Symbol

Alternatives are represented by the | symbol. Here we give an example of the production rule for the element `<character>`:

```
<character> ::= <digit> | <letter> | <special character> | ''
```

Explanation: We can conclude from this that a character is a digit, a letter, a special character, or two quotation marks; it must be one of the four.

The Symbols [and]

Whatever is placed between square brackets *may* be used. This is the production rule for the ROLLBACK statement:

```
<rollback statement> ::= ROLLBACK [ WORK ]
```

Explanation: A ROLLBACK statement always consists of the term `ROLLBACK` and can optionally be followed by the term `WORK`.

The ... Symbol

The three periods indicate what may be repeated one or more times. Here our example is the production rule for an integer:

```
<whole number> ::= <digit>...
```

Explanation: An integer consists of a series of digits (with a minimum of one).

Combining the three points with square brackets enables us to indicate that a certain element can appear zero, one, or more times:

```
<from clause> ::=
   FROM <table reference> [ , <table reference> ]...
```

Explanation: A FROM clause begins with the terminal symbol `FROM` and is followed by at least one table reference. It is possible to follow this table reference with a list of elements, with each element consisting of a comma followed by a table reference. Do not forget that the comma is part of SQL and not part of the notation.

The Symbols { and }

All symbols between curly brackets form a group. For example, curly brackets used with the | symbol show precisely what the alternatives are. The following example is part of the production rule for the float literal:

```
<float literal> ::=
   <mantissa> { E | e } <exponent>
```

Explanation: A float literal start with a mantissa and ends with an exponent. In between, we can use the capital letter E or the small letter e; one must be used.

If we combine the curly brackets with three points, we can indicate that an element should appear one or more times. This means that in the production rule `A { B }...`, we first have to use the element A, and it should be followed by one or more B elements.

The ; Symbol

Some symbols have the same definition. Instead of repeating them, we can use the semicolon to shorten the definitions. The following definition

```
<character literal>    ;
<varchar literal>      ;
<long varchar literal> ::= <character string>
```

is equivalent to these three definitions:

```
<character literal>    ::= <character string>
<varchar literal>      ::= <character string>
<long varchar literal> ::= <character string>
```

The " Symbol

A small number of metasymbols, such as the " symbol, are part of particular SQL statements themselves. To avoid misunderstanding, these symbols are enclosed by double quotation marks. Among other things, this means that the symbol " that is used within SQL is represented in the production rules as """.

Additional Remarks

- Whatever is presented in capital letters, along as the symbols that are not part of the notation method, must be adopted unaltered.
- The sequence of the symbols in the right part of the production rule is fixed.
- Blanks in production rules have no significance. Generally, they have been added to make the rules more readable. Therefore, the two following production rules mean the same:

  ```
  <alphanumeric literal> ::= ' [ <character>... ] '
  ```

  and

  ```
  <alphanumeric literal> ::= '[<character>...]'
  ```

A.3 Reserved Words in SQL and Ingres

Ingres supports so-called *reserved words* or *keywords*, such as SELECT, and CREATE. In Ingres, these reserved words may not be used as names for database objects such as tables, columns, views, and users. The following list contains reserved words as defined in the SQL3 standard, followed by the list of reserved words of Ingres itself.

- ABSOLUTE, ACTION, ADD, ALL, ALLOCATE, ALTER, AND, ANY, ARE, AS, ASC, ASSERTION, AT, AUTHORIZATION, AVG
- BEGIN, BETWEEN, BIT, BIT_LENGTH, BOTH, BY
- CASCADE, CASCADED, CASE, CAST, CATALOG, CHAR, CHARACTER, CHAR_LENGTH, CHARACTER_LENGTH, CHECK, CLOSE, COALESCE, COLLATE, COLLATION, COLUMN, COMMIT, CONNECT, CONNECTION, CONSTRAINT, CONSTRAINTS, CONTINUE, CONVERT, CORRESPONDING, COUNT, CREATE, CROSS, CURRENT, CURRENT_DATE, CURRENT_TIME, CURRENT_TIMESTAMP, CURRENT_USER, CURSOR
- DATE, DAY, DEALLOCATE, DEC, DECIMAL, DECLARE, DEFAULT, DEFERRABLE, DEFERRED, DELETE, DESC, DESCRIBE, DESCRIPTOR, DIAGNOSTICS, DISCONNECT, DISTINCT, DOMAIN, DOUBLE, DROP
- ELSE, END, END-EXEC, ESCAPE, EXCEPT, EXCEPTION, EXEC, EXECUTE, EXISTS, EXTERNAL, EXTRACT
- FALSE, FETCH, FIRST, FLOAT, FOR, FOREIGN, FOUND, FROM, FULL
- GET, GLOBAL, GO, GOTO, GRANT, GROUP
- HAVING, HOUR
- IDENTITY, IMMEDIATE, IN, INDICATOR, INITIALLY, INNER, INPUT, INSENSITIVE, INSERT, INT, INTEGER, INTERSECT, INTERVAL, INTO, IS, ISOLATION
- JOIN
- KEY
- LANGUAGE, LAST, LEADING, LEFT, LEVEL, LIKE, LOCAL, LOWER
- MATCH, MAX, MIN, MINUTE, MODULE, MONTH
- NAMES, NATIONAL, NATURAL, NCHAR, NEXT, NO, NOT, NULL, NULLIF, NUMERIC
- OCTET_LENGTH, OF, ON, ONLY, OPEN, OPTION, OR, ORDER, OUTER, OUTPUT, OVERLAPS
- PARTIAL, POSITION, PRECISION, PREPARE, PRESERVE, PRIMARY, PRIOR, PRIVILEGES, PROCEDURE, PUBLIC
- READ, REAL, REFERENCES, RELATIVE, RESTRICT, REVOKE, RIGHT, ROLLBACK, ROWS
- SCHEMA, SCROLL, SECOND, SECTION, SELECT, SESSION, SESSION_USER, SET, SIZE, SMALLINT, SOME, SQL, SQLCODE, SQLERROR, SQLSTATE, SUBSTRING, SUM, SYSTEM_USER
- TABLE, TEMPORARY, THEN, TIME, TIMESTAMP, TIMEZONE_HOUR, TIMEZONE_MINUTE, TO, TRAILING, TRANSACTION, TRANSLATE, TRANSLATION, TRIM, TRUE
- UNION, UNIQUE, UNKNOWN, UPDATE, UPPER, USAGE, USER, USING
- VALUE, VALUES, VARCHAR, VARYING, VIEW
- WHEN, WHENEVER, WHERE, WITH, WORK, WRITE
- YEAR
- ZONE

This is the list of reserved words in Ingres:

- ABORT, ACTIVATE, ADD, ADDFORM, AFTER, ALL, ALTER, AND, ANY, APPEND, ARRAY, AS, ASC, ASYMMETRIC, AT, AUTHORIZATION, AVG, AVGU
- BEFORE, BEGIN, BETWEEN, BREAKDISPLAY, BY, BYREF
- CALL, CALLFRAME, CALLPROC, CASCADE, CASE, CAST, CHECK, CLEAR, CLEARROW, CLOSE, COALESCE, COLLATE, COLUMN, COMMAND, COMMENT, COMMIT, COMMITTED, CONNECT, CONSTRAINT, CONTINUE, COPY, COPY_FROM, COPY_INTO, COUNT, COUNTU, CREATE, CURRENT, CURRENT_USER, CURRVAL, CURSOR, CYCLE
- DATAHANDLER, DBMS_PASSWORD, DECLARE, DEFAULT, DEFINE, DELETE, DELETEROW, DESC, DESCRIBE, DESCRIPTOR, DESTROY, DIRECT, DISCONNECT, DISPLAY, DISTINCT, DISTRIBUTE, DO, DOWN, DROP
- ELSE, ELSEIF, ENABLE, END, END-EXEC, ENDDATA, ENDDISPLAY, ENDFOR, ENDFORMS, ENDIF, ENDLOOP, ENDREPEAT, ENDRETRIEVE, ENDSELECT, ENDWHILE, ESCAPE, EXCEPT, EXCLUDE, EXCLUDING, EXECUTE, EXISTS, EXIT
- FETCH, FIELD, FINALIZE, FIRST, FOR, FOREIGN, FORMDATA, FORMINIT, FORMS, FROM, FULL
- GET, GETFORM, GETOPER, GETROW, GLOBAL, GOTO, GRANT, GRANTED, GROUP
- HAVING, HELP, HELP_FORMS, HELP_FRS, HELPFILE
- IDENTIFIED, IF, IIMESSAGE, IIPRINTF, IIPROMPT, IISTATEMENT, IMMEDIATE, IMPORT, IN, INCLUDE, INCREMENT, INDEX, INDICATOR, INGRES, INITIAL_USER, INITIALIZE, INITTABLE, INQUIRE_EQUEL, INQUIRE_FORMS, INQUIRE_FRS, INQUIRE_INGRES, INQUIRE_SQL, INSERT, INSERTROW, INTEGRITY, INTERSECT, INTO, IS, ISOLATION
- JOIN
- KEY
- LEAVE, LEFT, LEVEL, LIKE, LOADTABLE, LOCAL
- MAX, MAXVALUE, MENUITEM, MESSAGE, MIN, MINVALUE, MODE, MODIFY, MODULE, MOVE
- NATURAL, NEXT, NEXTVAL, NOCACHE, NOCYCLE, NOECHO, NOMAXVALUE, NOMINVALUE, NOORDER, NOT, NOTRIM, NULL, NULLIF
- OF, ON, ONLY, OPEN, OPTION, OR, ORDER, OUT, OUTER
- PARAM, PARTITION, PERMIT, PREPARE, PRESERVE, PRIMARY, PRINT, PRINTSCREEN, PRIVILEGES, PROCEDURE, PROMPT, PUBLIC, PURGETABLE, PUTFORM, PUTOPER, PUTROW
- QUALIFICATION
- RAISE, RANGE, RAWPCT, READ, REDISPLAY, REFERENCES, REFERENCING, REGISTER, RELOCATE, REMOVE, RENAME, REPEAT, REPEATABLE, REPEATED, REPLACE, REPLICATE, RESTART, RESTRICT, RESULT, RESUME, RETRIEVE, RETURN, REVOKE, RIGHT, ROLE, ROLLBACK, ROW, ROWS, RUN
- SAVE, SAVEPOINT, SCHEMA, SCREEN, SCROLL, SCROLLDOWN, SCROLLUP, SECTION, SELECT, SERIALIZABLE, SESSION, SESSION_USER, SET, SET_4GL, SET_EQUEL, SET_FORMS, SET_FRS, SET_INGRES, SET_SQL, SLEEP, SOME, SORT, SQL, START, STOP, SUBMENU, SUBSTRING, SUM, SUMU, SYMMETRIC, SYSTEM, SYSTEM_MAINTAINED, SYSTEM_USER
- TABLE, TABLEDATA, TEMPORARY, THEN, TO, TYPE

- UNCOMMITTED, UNION, UNIQUE, UNLOADTABLE, UNTIL, UP, UPDATE, USER, USING
- VALIDATE, VALIDROW, VALUES, VIEW
- WHEN, WHENEVER, WHERE, WHILE, WITH, WORK, WRITE

We strongly advise that you follow these recommendations when choosing the names of database objects:

- Avoid one-letter words, even if they do not occur in the list.
- Avoid words that could be seen as abbreviations of words in the list; for example, do not use DATA because the term DATABASE appears in the list.
- Avoid derivations of words in the list, such as plural and verbal forms. Therefore, do not use TABLES (plural of TABLE) or ORDERING (present participle of the verb ORDER).

A.4 Syntax Definitions of SQL Statements

This section contains the definitions of all the SQL statements as they are described in this book. Several statements use certain common elements, such as condition and column list. If an element belongs to only one statement, it is included in Section A.4.2 together with its statement. All others are explained in Section A.4.3. We begin with the different groups of SQL statements.

A.4.1 Groups of SQL Statements

The supported set of SQL statements can be divided into groups, such as DDL, DML, and DCL statements. In this section, we indicate which group each statement belongs to.

SQL Statement

```
<sql statement> ::=
   <executable statement>     |
   <non-executable statement>
```

Executable Statement

```
<executable statement> ::=
   <declarative statement> |
   <procedural statement>
```

Declarative Statement

```
<declarative statement> ::=
   <ddl statement> |
   <dml statement> |
   <dcl statement>
```

DDL statement

```
<ddl statement> ::=
   <alter location statement>                |
   <alter sequence statement>                |
   <alter table statement>                   |
   <comment statement>                       |
   <create dbevent statement>                |
   <create index statement>                  |
   <create integrity statement>              |
   <create location statement>               |
   <create procedure statement>              |
   <create schema statement>                 |
   <create sequence statement>               |
   <create synonym statement>                |
   <create table statement>                  |
   <create rule statement>                   |
   <create view statement>                   |
   <declare global temporary table statement> |
   <drop dbevent statement>                  |
   <drop index statement>                    |
   <drop integrity statement>                |
   <drop location statement>                 |
   <drop procedure statement>                |
   <drop sequence statement>                 |
   <drop synonym statement>                  |
   <drop table statement>                    |
   <drop rule statement>                     |
   <drop view statement>                     |
   <modify statement>                        |
   <register table statement>                |
   <save statement>
```

DML Statement

```
<dml statement> ::=
   <begin statement>              |
   <call statement>               |
   <close statement>              |
   <commit statement>             |
   <connect statement>            |
   <copy statement>               |
   <delete statement>             |
   <describe statement>           |
   <describe input statement>     |
   <disconnect statement>         |
   <end statement>                |
   <endexecute statement>         |
   <endselect statement>          |
   <execute statement>            |
   <execute immediate statement>  |
   <execute procedure statement>  |
   <fetch statement>              |
   <inquire_sql statement>        |
   <insert statement>             |
   <open statement>               |
   <prepare statement>            |
   <prepare to commit statement>  |
   <raise dbevent statement>      |
   <raise error statement>        |
   <register dbevent statement>   |
   <remove dbevent statement>     |
   <rollback statement>           |
   <savepoint statement>          |
   <select statement>             |
   <select into statement>        |
   <set statement>                |
   <set_sql statement>            |
   <update statement>
```

DCL Statement

```
<dcl statement> ::=
   <alter group statement>            |
   <alter profile statement>          |
   <alter role statement>             |
   <alter security_audit statement>   |
   <alter user statement>             |
   <create group statement>           |
   <create profile statement>         |
   <create role statement>            |
   <create security_alarm statement>  |
   <create user statement>            |
   <disable security_audit statement> |
   <drop group statement>             |
   <drop profile statement>           |
   <drop role statement>              |
   <drop security_alarm statement>    |
   <drop user statement>              |
   <enable security_audit statement>  |
   <grant statement>                  |
   <remove table statement>           |
   <revoke statement>
```

Non-executable Statement

```
<non-executable statement> ::=
   <begin declare statement>  |
   <declare cursor statement> |
   <declate statement>        |
   <declare table statement>  |
   <end declare statement>    |
   <include statement>        |
   <whenever statement>
```

Procedural Statement

```
<procedural statement> ::=
   <assignment statement>        |
   <commit statement>            |
   <delete statement>            |
   <endloop statement>           |
   <execute procedure statement> |
   <for statement>               |
   <if statement>                |
   <insert statement>            |
   <message statement>           |
   <raise dbevent statement>     |
   <raise error statement>       |
   <register dbevent statement>  |
   <remove dbevent statement>    |
   <return statement>            |
   <return row statement>        |
   <rollback statement>          |
   <select statement>            |
   <select into statement>       |
   <update statement>            |
   <while statement>
```

Flow-Control Statement

```
<flow control statement> ::=
   <endloop statement> |
   <for statement>     |
   <if statement>      |
   <while statement>
```

A.4.2 Definitions of SQL Statements

Alter Group Statement

```
<alter group statement> ::=
   ALTER GROUP <group name> [ , <group name> ]...
   { ADD USERS ( <user name> [ , <user name> ]... )  |
     DROP USERS ( <user name> [ , <user name> ]... ) |
     DROP ALL                                        }
```

Alter Location Statement

```
<alter location statement> ::=
   ALTER LOCATION <location name> USAGE = <usage specification>
```

Alter Profile Statement

```
<alter profile statement> ::=
   ALTER { DEFAULT PROFILE | PROFILE <profile name> }
      [ ADD PRIVILEGES ( <subject privileges> ) ]
      [ DROP PRIVILEGES (<subject privileges> ) ]
      [ WITH <user properties> ]
```

Alter Role Statement

```
<alter role statement> ::=
   ALTER ROLE <role name> [ , <role name> ]...
      [ ADD PRIVILEGES ( <subject privileges> ) ]
      [ DROP PRIVILEGES ( <subject privileges> ) ]
      [ WITH <role property> [ , <role property> ]...
```

Alter Security_audit Statement

```
<alter security_audit statement> ::=
   ALTER SECURITY_AUDIT
      [ SUSPEND | RESUME | RESTART | STOP ]
      [ WITH AUDIT_LOG = <file specification> ]
```

Alter Sequence Statement

```
<alter sequence statement> ::=
   ALTER SEQUENCE [ <user name> . ] <sequence name>
      [ <sequence option>... ]

<sequence option> ::=
   RESTART WITH <integer literal>                              |
   INCREMENT BY <integer literal>                              |
   { MAXVALUE < integer literal> | NOMAXVALUE | NO MAXVALUE } |
   { MINVALUE < integer literal> | NOMINVALUE | NO MINVALUE } |
   { CYCLE | NOCYCLE | NO CYCLE }                               |
   { ORDER | NOORDER | NO ORDER }                               |
   { CACHE < integer literal> | NOCACHE | NO CACHE }
```

Alter Table Statement

```
<alter table statement> ::=
   ALTER TABLE <table specification> <table structure change>

<table structure change> ::=
   <column add>                  |
   <column change>               |
   <column delete>               |
   <integrity constraint add>    |
   <integrity constraint delete>

<column add> ::=
   ADD [ COLUMN ] <column name> <data type>
      [ <null specification> ]
      [ <default specification> ]
      [ <column integrity constraint> ]
      [ <collation specification> ]

<column change> ::=
   ALTER [ COLUMN ] <column name> <data type>
      [ <null specification> ]
      [ <default specification> ]
      [ <column integrity constraint> ]
      [ <collation specification> ]

<column delete> ::=
   DROP [ COLUMN ] <column name> { RESTRICT | CASCADE }

<integrity constraint add> ::=
   ADD <table integrity constraint>

<integrity constraint delete> ::=
   DROP CONSTRAINT <constraint name> { RESTRICT | CASCADE }
```

Alter User Statement

```
<alter user statement> ::=
   ALTER USER <user name>
      [ ADD PRIVILEGES ( <subject privileges> ) ]
      [ DROP PRIVILEGES ( <subject privileges> ) ]
      [ WITH <user properties> ]
```

Assignment Statement

```
<assignment statement> ::=
   { <local variable> | <parameter> } { = | := } <scalar expression>
```

Begin Statement

```
<begin statement> ::= BEGIN
```

Begin Declare Statement

```
<begin declare statement> ::=
   BEGIN DECLARE SECTION
```

Call Statement

```
<call statement> ::=
   { CALL SYSTEM ( COMMAND = <alphanumeric literal> ) |
     CALL <subsystem> ( DATABASE = <database name>
        [ , <parameter name> = <literal> } ]        }
```

Close Statement

```
<close statement> ::= CLOSE <cursor name>
```

Comment Statement

```
<comment statement> ::=
   COMMENT ON <documentation object> IS <alphanumeric literal>

<documentation object> ::=
   TABLE <table specification>                    |
   COLUMN [ <user name> . ] <column specification>
```

Commit Statement

```
<commit statement> ::=
   COMMIT [ WORK ]
```

Connect Statement

```
<connect statement> ::=
   CONNECT <database name>
      [ AS <connection name> ]
      [ SESSION <whole number> ]
      [ IDENTIFIED BY { <user name> | <alphanumeric literal> | <host variable> } ]
      [ DBMS_PASSWORD = <alphanumeric literal> ]
      [ OPTIONS = <flag> [ , <flag> ]... ]
      [ WITH HIGHDXID = <literal>, LOWDXID = <literal> ]

<connection> ::= <alphanumeric literal> | <host variable>

<flag> ::= -R <role name> [ / <password without quotation marks> ]
```

Copy Statement

```
<copy from statement> ::=
   COPY [ TABLE ] <table specification>
      ( [ <copy column> [ , <copy column> ]... ] )
      INTO ' <file specification> '
      [ WITH <copy option> [ , <copy option> ]... ]

<copy column> ::=
   <copy specification> [ WITH NULL [ ( <literal> ) ] ]

<copy specification> ::=
   <column name> = <copy data type> [ <delimiter> | '<special symbol>' | <letter> ] |
   <column name> = D0<delimiter>                                                     |
   <column name> = D0<letter>                                                        |
   <column name> = 'd0<special symbol>'                                              |
   <column name> = D0                                                                |
   <word>        = <D<whole number>                                                  |
   <delimiter>   = D<whole number>

<copy data type> ::=
   ANSIDATE                                          |
   BYTE ( <whole number> ) [ <delimiter> ]           |
   BYTE VARYING ( <whole number> ) [ <delimiter> ]   |
   CHAR ( <whole number> ) [ <delimiter> ]           |
   DATE                                              |
   DECIMAL                                           |
   FLOAT                                             |
   FLOAT4                                            |
   INGRESDATE                                        |
   INTEGER                                           |
   INTEGER1                                          |
   INTERVAL YEAR TO MONTH                            |
```

continues →

```
   INTERVAL DAY TO MONTH                          |
   LONG BYTE(0)                                   |
   LONG NVARCHAR(0)                               |
   MONEY                                          |
   NCHAR(0)                                       |
   NVARCHAR(0)                                    |
   SMALLINT                                       |
   TIME                                           |
   TIMESTAMP                                      |
   VARCHAR( <whole number> )

<delimiter> ::=
   NL | TAB | SP | NUL | NULL | COMMA | COLON | DASH |
   LPAREN | RPAREN | <special symbol>
```

Create Dbevent Statement

```
<create dbevent statement> ::=
   CREATE DBEVENT [ <user name> . ] <dbevent name>
```

Create Group Statement

```
<create group statement> ::=
   CREATE GROUP <group name> [ , <group name> ]...
      [ WITH USERS =  ( <user name> [ , <user name> ]... )
```

Create Index Statement

```
<create index statement> ::=
   CREATE [ UNIQUE ] INDEX <index specification> [ , <index specification> ]...

<index specification> ::=
   ( [ <user name> . ] <index name>
      ON <table specification>
      ( <column in index> [ , <column in index> ]... )
      [ UNIQUE ]
      [ WITH <index option> [ , <index option> ] ... ] )
```

Create Integrity Statement

```
<create integrity statement> ::=
   CREATE INTEGRITY ON <table specification> [ [ AS ] <pseudonym> ]
      IS <condition>
```

Create Location Statement

```
<create location statement> ::=
   CREATE LOCATION <location name>
      WITH AREA = <area specification>
      [ , <usage specification> ]
      [ , RAWPCT = <whole number> ]
```

Create Procedure Statement

```
<create procedure statement> ::=
   CREATE PROCEDURE <procedure name>
      [ <parameter specification list> ] AS
      [ RESULT ROW <data type list> ]
      [ <declare section> ]
      <procedure body>

<parameter specification list> ::=
   ( { <parameter specification> [ , <parameter specification> ]... |
     <parameter> = SET OF <column list>                            } )

<parameter specification> ::=
   [ IN | OUT | INOUT ] <parameter name> <data type>
      [ <null default specification> ]

<declare section> ::=
   DECLARE <variable declaration> [ <variable declaration> ]...

<procedure body> ::= <begin-end block>

<begin-end block> ::= BEGIN <statement list> END

<statement list> ::=
   <statement in procedure> { ; <statement in procedure> }... [ ; ]

<statement in procedure> ::=
   <dml statement> |
   <procedural statement>

<variable declaration> ::=
   <variable name> [ , <variable name> ]... [ = ] <data type>
      [ <null default specification> ]
```

Create Profile Statement

```
<create profile statement> ::=
   CREATE PROFILE <profile name> [ WITH <user properties> ]
```

Create Role Statement

```
<create role statement> ::=
   CREATE ROLE <role name> [ , <role name> ]...
     [ WITH <role property> [ , <role property> ]...
```

Create Schema Statement

```
<create schema statement> ::=
   CREATE SCHEMA AUTHORIZATION <schema name>
      <schema statement>...

<schema statement> ::=
   <create table statement> |
   <create view statement>  |
   <grant statement>
```

Create Security_alarm Statement

```
<create security_alarm statement> ::=
   CREATE SECURITY_ALARM [ <alarm name> ] ON
      <alarm object> [ , <alarm object> ]...
      [ IF { SUCCESS | FAILURE | SUCCESS , FAILURE } ]
      [ WHEN { SELECT | DELETE | INSERT | UPDATE | CONNECT | DISCONNECT } ]
      [ BY { { USER | GROUP | ROLE }
          <object name> [ , <object name> ]... | PUBLIC } ]
      [ RAISE DBEVENT [ <user name> . ] <dbevent name> [ <alphanumeric literal> ] ]
```

Create Sequence Statement

```
<create sequence statement> ::=
   CREATE SEQUENCE [ <user name> . ] <sequence name>
      [ <sequence option>... ]

<sequence option> ::=
   AS <data type>                                          |
   START WITH <integer literal>                            |
   INCREMENT BY <integer literal>                          |
   { MAXVALUE <integer literal> | NOMAXVALUE | NO MAXVALUE } |
   { MINVALUE <integer literal> | NOMINVALUE | NO MINVALUE } |
   { CYCLE | NOCYCLE | NO CYCLE }                          |
   { ORDER | NOORDER | NO ORDER }                          |
   { CACHE <integer literal> | NOCACHE | NO CACHE }
```

Create Synonym Statement

```
<create synonym statement> ::=
   CREATE SYNONYM <table name> FOR <table specification>
```

Create Table Statement

```
<create table statement> ::=
   CREATE TABLE <table specification> <table structure>
      [ WITH <table option> [ , <table option> ]... ]

<table structure> ::=
   <table schema>   |
   <table contents>

<table contents> ::= [ <column list> ] AS <table expression>
```

Create Rule Statement

```
<create rule statement> ::=
   CREATE { RULE | TRIGGER } [ <user name> . ] <rule name>
   <rule moment>
   <rule event>
   [ <rule condition> ]
   <rule action>

<rule moment> ::= BEFORE | AFTER

<rule event> ::=
   <rule operation> [ , <rule operation> ]...
   { ON | OF | FROM | INTO } <table specification>
   [ REFERENCING { OLD AS <table name> ] [ NEW AS <table name> ] ]

<rule condition> ::= WHERE <condition>

<rule action> ::=
   FOR EACH { ROW | STATEMENT } <execute procedure statement>

<rule operation> ::=
   INSERT | DELETE | UPDATE [ OF <column list> ]
```

Create User Statement

```
<create user statement> ::=
   CREATE USER <user name>
      [ WITH <user properties> ]
```

Create View Statement

```
<create view statement> ::=
   CREATE VIEW <view name> [ <column list> ]
      AS <table expression>
      [ WITH CHECK OPTION ]
```

Declare Cursor Statement

```
<declare cursor statement> ::=
   [ REPEATED ]
   DECLARE <cursor name> CURSOR FOR
   <table expression>
   [ <for clause> ]

<for clause> ::=
   FOR [ DEFERRED | DIRECT ] UPDATE [ OF <column name> [ , <column name> ]... ]
```

Declare Global Temporary Table Statement

```
<declare global temporary table statement> ::=
   DECLARE GLOBAL TEMPORARY TABLE
      [ SESSION . ] <table name> <table schema>
      ON COMMIT PRESERVE ROWS
      WITH NORECOVERY
```

Declare Statement

```
<declare statement> ::=
   DECLARE <statement name> [ , <statement name> ]... STATEMENT
```

Declare Table Statement

```
<declare table statement> ::=
   DECLARE <table specification> TABLE
      <declared column> [ , <declared column> ]...

<declared column> ::=
   <column name> <data type>
      [ <null specification> ] [ <default specification> ]
```

Delete Statement

The version for interactive SQL:

```
<delete statement> ::=
   DELETE
   FROM    <table reference>
   [ <where clause> ]
```

The version for embedded SQL:

```
<delete statement> ::=
   [ REPEATED ]
   DELETE
   FROM    <table specification>
   [ WHERE { <condition> | CURRENT OF <cursor name> } ]
```

Describe Statement

```
<describe statement> ::=
   DESCRIBE <statement name>
      [ INTO | USING ] [ :] <descriptor name> [ USING NAMES ]
```

Describe Input Statement

```
<describe input statement> ::=
   DESCRIBE INPUT <statement name>
      USING [ SQL ] DESCRIPTOR <descriptor name> [ WITHOUT NESTING ]
```

Disable Security_audit Statement

```
<disable security_audit statement> ::=
   DISABLE SECURITY_AUDIT { <audit type> | ALL }
```

Disconnect Statement

```
<disconnect statement> ::=
   DISCONNECT [ CURRENT | <connection name> | SESSION <whole number> ] | ALL ]
```

Drop Dbevent Statement

```
<drop dbevent statement> ::=
   DROP DBEVENT [ <user name> . ] <dbevent name>
```

Drop Group Statement

```
<drop group statement> ::=
   DROP GROUP <group name> [ , <group name> ]...
```

Drop Index Statement

```
<drop index statement> ::=
   DROP INDEX [ <user name> . ] <index name>
      [ , [ <user name> . ] <index name> ]...
```

Drop Integrity Statement

```
<drop integrity statement> ::=
   DROP INTEGRITY ON <table specification> { ALL | <integer> [ . <integer> ]... }
```

Drop Location Statement

```
<drop location statement> ::=
   DROP LOCATION <location name>
```

Drop Procedure Statement

```
<drop procedure statement> ::=
   DROP PROCEDURE [ <user name> . ] <procedure name>
```

Drop Profile Statement

```
<drop profile statement> ::=
   DROP PROFILE <profile name> [ CASCADE | RESTRICT ]
```

Drop Role Statement

```
<drop role statement> ::=
   DROP ROLE <role name> [ , <role name> ]...
```

Drop Security_alarm Statement

```
<drop security_alarm statement> ::=
   DROP SECURITY_ALARM ON <alarm object>
      { <alarm specification> [ , <alarm specification> ]... | ALL }

<alarm specification> ::=
   <alarm name> | <integer>
```

Drop Sequence Statement

```
<drop sequence statement> ::=
   DROP SEQUENCE [ <user name> . ] <sequence name>
```

Drop Synonym Statement

```
<drop synonym statement> ::=
   DROP SYNONYM <table specification> [ , <table specification> ]...
```

Drop Table Statement

```
<drop table statement> ::=
   DROP [ TABLE ] <table specification> [ , <table specification> ]...
```

Drop Rule Statement

```
<drop rule statement> ::=
   DROP { RULE | TRIGGER } [ <user name> . ] <rule name>
```

Drop User Statement

```
<drop user statement> ::=
   DROP USER <user name>
```

Drop View Statement

```
<drop view statement> ::=
   DROP [ VIEW ] <table specification> [ , <table specification> ]...
```

Enable Security_audit Statement

```
<enable security_audit statement> ::=
   ENABLE SECURITY_AUDIT { <audit type> | ALL }
```

End Statement

```
<end statement> ::= END
```

End Declare Statement

```
<end declare statement> ::=
   END DECLARE SECTION
```

Endexecute Statement

```
<endexecute statement> ::= ENDEXECUTE
```

Endloop Statement

```
<endloop statement> ::=
   ENDLOOP [ <label> ]
```

Endselect Statement

```
<endselect statement> ::=
   ENDSELECT
```

Execute Statement

```
<execute statement> ::=
   EXECUTE <statement name>
      [ USING <host variable list>         |
        USING DESCRIPTOR <descriptor name> ]
```

Execute Immediate Statement

```
<execute immediate statement> ::=
   EXECUTE IMMEDIATE <host variable>
      [ INTO <host variable list>              |
        USING [ DESCRIPTOR ] <descriptor name> ]
```

Execute Procedure Statement

The version for interactive SQL:

```
<execute procedure statement> ::=
   { EXECUTE PROCEDURE | CALLPROC }
      [ <user name> . ] <procedure name>
      [ <parameter assignment list> ]

<parameter assignment list> ::=
   ( { <parameter assignment> [ , <parameter assignment> ]... } |
     { <parameter name> = SESSION . <temporary table name>    } )

<parameter assignment> ::=
   <parameter name> = <scalar expression>
```

The version for embedded SQL:

```
<execute procedure statement> ::=
   { EXECUTE PROCEDURE | CALLPROC }
      [ <user name> . ] <procedure name>
      [ ( <parameter assignment list> ) |
        RESULT ROW ( <host variable list> ) ]
      [ INTO <host variable name> ]

<parameter assignment list> ::=
   ( { <parameter assignment> [ , <parameter assignment> ]... } |
     { <parameter name> = SESSION . <temporary table name>    } )

<parameter assignment> ::= <parameter name> = <scalar expression>
```

Fetch Statement

```
<fetch statement> ::=
   FETCH [ FROM ] <cursor name>
   INTO  <host variable list>
```

For Statement

```
<for statement> ::=
   [ <label> : ] FOR <select statement> DO <statement list> ENDFOR
```

Grant Statement

```
<grant statement> ::=
   <grant table privilege statement>    |
   <grant database privilege statement> |
   <grant role statement>               |
   <grant sequence privilege statement> |
   <grant execute privilege statement>  |
   <grant dbevent privilege statement>

<grant table privilege statement> ::=
   GRANT <table privileges>
   ON    [ TABLE ] <table specification> [ , <table specification> ]...
   TO    <grantees>
   [ WITH GRANT OPTION ]

<grant database privilege statement> ::=
   GRANT <database privileges>
   ON    { DATABASE <database name> [ , <database name> ]... | CURRENT INSTALLATION }
   TO    <grantees>

<grant role statement> ::=
   GRANT <role name> [ , <role name> ]...
   TO    <grantees>

<grant sequence privilege statement> ::=
   GRANT { ALL [ PRIVILEGES ] | NEXT }
   ON    SEQUENCE <sequence name> [ , <sequence name> ]...
   TO    <grantees>
   [ WITH GRANT OPTION ]

<grant execute privilege statement> ::=
   GRANT { ALL [ PRIVILEGES ] | EXECUTE }
   ON    PROCEDURE <procedure name> [ , <procedure name> ]...
   TO    <grantees>
   [ WITH GRANT OPTION ]

<grant dbevent privilege statement> ::=
   GRANT <dbevent privileges>
   ON    DBEVENT <dbevent name> [ , <dbevent name> ]...
   TO    <grantees>
   [ WITH GRANT OPTION ]
```

If Statement

```
<if statement> ::=
   IF <procedure condition> THEN <statement list>
      [ ELSEIF <procedure condition> THEN <statement list> ]...
      [ ELSE <statement list> ]
   ENDIF
```

Include Statement

```
<include statement> ::=
   INCLUDE <file>
```

Inquire_sql Statement

```
<inquire_sql statement> ::=
   { INQUIRE_SQL | INQUIRE_INGRES }
      ( <inquire specification> [ , <inquire specification> ]...

<inquire specification> ::=
   <host variable> = <inquire object>

<inquire object> ::=
   dbeventname | dbeventowner | dbeventdatabase |
   dbeventtime | dbeventtext
```

Insert Statement

```
<insert statement> ::=
   INSERT INTO <table specification> <insert specification>

<insert specification> ::=
   [ <column list> ] <values clause>    |
   [ <column list> ] <table expression>
```

Message Statement

```
<message statement> ::=
   MESSAGE { <message number>                |
             <message text>                  |
             <message number> <message text> }
      [ WITH DESTINATION =
         ( <message destination> [ , <message destination> ]... ) ]

<message text> ::=
   <alphanumeric literal> |
   <local variable>       |
   <parameter name>

<message number> ::=
   <integer literal> |
   <local variable>  |
   <parameter name>

<message destination> ::= SESSION | ERROR_LOG | AUDIT_LOG
```

Modify Statement

```
<modify statement> ::=
   MODIFY <modify object>
      TO <modify action> [ UNIQUE ]
      [ ON <column in index> [ , <column in index> ]... ]
      [ WITH <modify option> [ , <modify option> ]... ]

<modify object> ::=
   <table specification>                                                    |
   [ <user name> . ] <index name>                                           |
   <table specification> PARTITION <partition name> [ , <partition name> ]...

<modify action> ::=
   ISAM | HASH | HEAP | HEAPSORT | BTREE | RECONSTRUCT |
   TRUNCATED | REORGANIZE | RELOCATE | MERGE | ADD_EXTEND |
   READONLY | NOREADONLY | PHYS_CONSISTENT | PHYS_INCONSISTENT |
   LOG_CONSISTENT | LOG_INCONSISTENT | TABLE_RECOVERY_ALLOWED |
   TABLE_RECOVERY_DISALLOWED | PERSISTENCE | NOPERSISTENCE |
   UNIQUE_SCOPE = { ROW | STATEMENT } | TABLE_DEBUG |
   PRIORITY = <digit>
```

Open Statement

```
<open statement> ::=
   OPEN <cursor name>
```

Prepare Statement

```
<prepare statement> ::=
   PREPARE <statement name>
      [ INTO <descriptor name> [ USING NAMES ] ]
      FROM { <alphanumeric literal> | <host variable> }
```

Prepare to Commit Statement

```
<prepare to commit statement> ::=
   PREPARE TO COMMIT WITH
      HIGHXID = <scalaire integer expressie> ,
      LOWXID  = <scalaire integer expressie>
```

Raise Dbevent Statement

```
<raise dbevent statement> ::=
   RAISE DBEVENT [ <user name> . ] <dbevent name>
      <dbevent text> [ WITH { SHARE | NOSHARE } ]

<dbevent text> ::= <alphanumeric literal>
```

Raise Error Statement

```
<raise error statement> ::=
   RAISE ERROR <error number> [ <error text> ]
      [ WITH DESTINATION =
         ( <error destination> [ , <error destination> ]... )

<error number> ::=
   <integer literal> |
   <local variable>  |
   <parameter name>

<error text> ::=
   <alphanumeric literal> |
   <local variable>       |
   <parameter name>

<error destination> ::= SESSION | ERROR_LOG | AUDIT_LOG
```

Register Dbevent Statement

```
<register dbevent statement> ::=
   REGISTER DBEVENT [ <user name> . ] <dbevent name>
```

Register Table Statement

```
<register table statement> ::=
   <register remote table>   |
   <register auditing table>

<register remote table> ::=
   REGISTER [ TABLE ] <table name>
      [ <column list> ]
      AS LINK { <remote database> | WITH REFRESH }

<remote database> ::=
   FROM <table specification>
      [ WITH [ NODE = <node name> , DATABASE = <database name> ]
          [ , DBMS = <server class> ]

<register auditing table> ::=
   REGISTER TABLE <table specification>
      ( <registered column> [ , <registered column> ]... )
      AS IMPORT FROM ' <file specification> '
      WITH DBMS = SXA [ , ROWS = <whole number> ]

<registered column> ::=
   <column name> <data type>
      [ <null specification> ]
      [ IS ' <column name> ' ]
```

Remove Dbevent Statement

```
<remove dbevent statement> ::=
   REMOVE DBEVENT [ <user name> . ] <dbevent name>
```

Remove Table Statement

```
<remove table statement> ::=
   REMOVE [ TABLE ]
      <table specification> [ , <table specification> ]...
```

Return Statement

```
<return statement> ::=
   RETURN <scalar integer expression>
```

Return Row Statement

```
<return row statement> ::=
   RETURN ROW <scalar expression list>
```

Revoke Statement

```
<revoke statement> ::=
   <revoke table privilege statement>    |
   <revoke database privilege statement> |
   <revoke role statement>               |
   <revoke execute privilege statement>  |
   <revoke dbevent privilege statement>

<revoke table privilege statement> ::=
   REVOKE  [ GRANT OPTION FOR ] <table privileges>
   ON      [ TABLE ] <table specification> [ , <table specification> ]...
   FROM    <grantees> { CASCADE | RESTRICT }

<revoke database privilege statement> ::=
   REVOKE  <database privileges>
   ON      { DATABASE <database name> | CURRENT INSTALLATION }
   FROM    <grantees> [ CASCADE | RESTRICT ]

<revoke role statement> ::=
   REVOKE <role name> [ , <role name> ]...
   FROM   <grantees> [ CASCADE | RESTRICT ]

<revoke execute privilege statement> ::=
   REVOKE [ GRANT OPTION FOR ] { ALL [ PRIVILEGES ] | EXECUTE }
   ON     PROCEDURE <procedure name> [ , <procedure name> ]...
   FROM   <grantees> [ CASCADE | RESTRICT ]

<revoke dbevent privilege statement> ::=
   REVOKE [ GRANT OPTION FOR ]
          { ALL [ PRIVILEGES ] | <dbevent privileges> }
   ON     DBEVENT <dbevent name> [ , <dbevent name> ]...
   FROM   <grantees> [ CASCADE | RESTRICT ]
```

Rollback Statement

```
<rollback statement> ::=
   ROLLBACK [ WORK ] [ TO <savepoint name> ]
```

Save Statement

```
<save statement> ::=
   SAVE <table specification> [ UNTIL <expiration date> ]

<expiration date> ::=
   <expiration date month> <whole number> <whole number>

<expiration date month> ::=
   <whole number> | JANUARY | JAN | FEBRUARY | FEB | MARCH | MAR |
   APRIL | APR | MAY | JUNE | JUN | JULY |JUL | AUGUST | AUG |
   SEPTEMBER | SEP | OCTOBER | OCT | NOVEMBER | NOV |
   DECEMBER | DEC
```

Savepoint Statement

```
<savepoint statement> ::=
   SAVEPOINT <savepoint name>
```

Select Statement

```
<select statement> ::=
   <table expression>
```

Select Into Statement

The cursor version for embedded SQL

```
<select into statement> ::=
   <select clause>
   <into clause>
   [ <from clause> ]
   [ <where clause> ]
   [ <group-by clause>
   [ <having clause> ] ]
   [ <set operator> <table expression> ]
   [ <select block tail> ]
   [ <for clause> ]
```

The non-cursor version for embedded SQL

```
<select into statement> ::=
   [ REPEATED ] <select clause>
   <into clause>
   [ <from clause> ]
   [ <where clause> ]
   [ <group-by clause>
   [ <having clause> ] ]
   [ <set operator> <table expression> ]
   [ <select block tail> ]
```

The version for use within stored procedures:

```
<select into statement> ::=
   <select clause>
   <into clause>
   [ <from clause> ]
   [ <where clause> ]
   [ <group-by clause>
   [ <having clause> ] ]
   [ <set operator> <table expression> ]
   [ <select block tail> ]
```

Set Statement

Below we show the features of the SET statement as described in this book. For more features we refer to the manuals.

```
<set statement> ::=
   <set connection statement>        |
   <set rules statement>             |
   <set lockmode statement>          |
   <set session option statement>    |
   <set session isolation statement>

<set connection statement> ::=
   SET CONNECTION { NONE | <connection name> }

<set rules statement> ::= SET { RULES | NORULES }

<set lockmode statement> ::=
   SET LOCKMODE { SESSION | TABLE <table specification> }
      WHERE TIMEOUT = { <whole number> | SESSION | SYSTEM | NOWAIT }
```

continues →

```
<set session option statement> ::=
   SET SESSION <session option>

<session option> ::=
   ADD PRIVILEGES ( <subject privileges> )                          |
   DROP PRIVILEGES ( <subject privileges> )                         |
   WITH PRIVILEGES = { ( <subject privileges> ) | ALL | DEFAULT } |
   WITH NOPRIVILEGES

<set session isolation statement> ::=
   SET { SESSION | TRANSACTION } { READ ONLY | READ WRITE }
      [ , ISOLATION LEVEL <isolation level> ]

<isolation level> ::=
   READ UNCOMMITTED | READ COMMITTED | REPEATABLE READ | SERIALIZABLE
```

Set_sql Statement

```
<set_sql statement> ::=
   SET_SQL ( SESSION = <whole number> )
```

Update Statement

The version for interactive SQL

```
<update statement> ::=
   UPDATE <table specification>
   SET    <column assignment> [ , <column assignment> ]...
   [ WHERE  <condition> ]

<column assignment> ::=
   <column name> = <scalar expression>
```

The version for embedded SQL:

```
<update statement> ::=
   UPDATE <table specification>
   SET    <column assignment> [ , <column assignment> ]...
   [ WHERE  { <condition> | CURRENT OF <cursor name> } ]

<column assignment> ::=
   <column name> = <scalar expression>
```

Whenever Statement

```
<whenever statement> ::=
   WHENEVER <whenever condition> <whenever action>

<whenever condition> ::=
   SQLWARNING | SQLERROR | SQLMESSAGE | NOT FOUND | DBEVENT

<whenever action> ::=
   CONTINUE | GOTO <label> | STOP | CALL <procedure name>
```

While Statement

```
<while statement> ::=
   [ <label> : ] WHILE <procedurecondition> DO <statement list>
   ENDWHILE
```

A.4.3 Common Elements

This section contains the general common elements used in various SQL statements. The elements that are defined as a name are all grouped at the end of this section.

Note: All the reserved words, such as ALL, TABLE, and DATABASE can be specified in small letters and in capitals. In fact, they may be mixed. For example, the following three terms are all allowed TABLE, table, and TABle. In the definitions below, however, all the reserved words are specified in capitals.

```
<aggregation function> ::=
   COUNT       ( [ DISTINCT | ALL ] { * | <scalar expression> } ) |
   MIN         ( [ DISTINCT | ALL ] <scalar expression> )         |
   MAX         ( [ DISTINCT | ALL ] <scalar expression> )         |
   SUM         ( [ DISTINCT | ALL ] <scalar expression> )         |
   AVG         ( [ DISTINCT | ALL ] <scalar expression> )         |
   ANY         ( [ DISTINCT | ALL ] <scalar expression> )         |
   STDDEV_POP  ( [ DISTINCT | ALL ] <scalar expression> )         |
   STDDEV_SAMP ( [ DISTINCT | ALL ] <scalar expression> )         |
   VAR_POP     ( [ DISTINCT | ALL ] <scalar expression> )         |
   VAR_SAMP    ( [ DISTINCT | ALL ] <scalar expression> )
```

```
<alarm object> ::=
   [ TABLE ] [ <user name> . ] <table name> |
   DATABASE <database name>                 |
   CURRENT INSTALLATION
```

```
<alphanumeric data type> ::=
   CHAR [ ( <length> ) ]           |
   CHARACTER [ ( <length> ) ]      |
   VARCHAR ( <length> )            |
   CHARACTER VARYING ( <length> )  |
   LONG VARCHAR                    |
   CLOB                            |
   CHAR LARGE OBJECT               |
   CHARACTER LARGE OBJECT          |
   TEXT [ ( <length> ) ]           |
   VCHAR [ ( <length> ) ]          |
   C [ ( <length> ) ]              |
   NCHAR [ ( <length> ) ]          |
   NVARCHAR [ ( <length> ) ]       |
   LONG NVARCHAR                   |
   NCLOB                           |
   NCHAR LARGE OBJECT              |
   NATIONAL CHARACTER LARGE OBJECT
```

```
<alphanumeric literal> ::= [ U& ] <character list>
```

```
<alternate key> ::= UNIQUE <column list>
```

```
<ansidate literal> ::= DATE ' <years> - <months> - <days> '
```

```
<ansi-interval literal> ::=
   { INTERVAL | interval } [ + | - ] ' <interval length> ' <interval unit>
```

```
<any all operator> ::= <comparison operator> { ALL | ANY | SOME }
```

```
<area specification> ::= <file specification>
```

```
<audit option> ::= ALL_EVENTS | DEFAULT_EVENTS | QUERY_TEXT
```

```
<audit type> ::=
   ALARM | DATABASE | DBEVENT | LOCATION | PROCEDURE |
   ROLE | RULE | SECURITY | TABLE | USER | VIEW |
   LEVEL ( ' <security_label> ' ) | ROW | QUERY_TEXT | RESOURCE
```

```
<automatic partitioning> ::=
   AUTOMATIC <automatic partitioning specification>
               [ , <automatic partitioning specification> ]...
```

```
<automatic partitioning specification> ::=
   PARTITION <partition name> [ WITH LOCATION = <location list> ] |
   <whole number> PARTITIONS [ WITH LOCATION = <location list> ]
```

```
<begin-end block> ::= BEGIN <statement list> END
```

```
<case expression> ::=
   CASE <when definition> [ ELSE <scalar expression> ] END
```

```
<cast expression> ::= CAST ( <scalar expression> AS <data type> )
```

```
<character> ::= <digit> | <letter> | <special character> | ''
```

```
<character list> ::= ' [ <character>... ] '
```

```
<check integrity constraint> ::= CHECK ( <condition> )
```

```
<collation specification> ::=
   COLLATE { UNICODE | UNICODE_CASE_INSENSITIVE | SQL_CHARACTER }
```

```
<column definition> ::=
   <column name> <data type>
      [ <null specification> ]
      [ <default specification> ]
      [ <column integrity constraint> ]
      [ <collation specification> ]
```

```
<column in index> ::= <column name> [ ASC | DESC ]
```

```
<column integrity constraint> ::=
   [ CONSTRAINT <constraint name> ]
   { PRIMARY KEY                  |
     UNIQUE                       |
     <check integrity constraint> |
     <referencing specification>  }
   [ WITH <key option> [ , <key option> ]... ]
```

```
<column list> ::= ( <column name> [ , <column name> ]... )
```

```
<column specification> ::= [ <table specification> . ] <column name>
```

```
<column subquery> ::= ( <table expression> )
```

```
<comparison operator> ::= = | < | > | <= | >= | <> | !=
```

```
<compound alphanumeric expression> ::=
   { <scalar alphanumeric expression> | <scalar numeric expression> } { "||" | + }
        { <scalar alphanumeric expression> | <scalar numeric expression> }
```

```
<compound ansidate expression> ::=
   <scalar ansidate expression> { + | - } <scalar ansi-interval expression> |
   <scalar ansi-interval expression> + <scalar ansidate expression>         |
   ( <scalar ansidate expression> )
```

```
<compound ansi-interval expression> ::=
   <scalar ansi-interval expression> { + | - } <scalar ansi-interval expression> |
   <scalar ansi-interval expression> { + | - } <alphanumeric expression>         |
   <scalar ansidate expression> - <scalar ansidate expression>                   |
   <scalar time expression> - <scalar time expression>                           |
   <scalar timestamp expression> - <scalar timestamp expression>                 |
   ( <scalar ansi-interval expression> )
```

```
<compound date expression> ::=
   <compound ansidate expression>   |
   <compound ingresdate expression>
```

```
<compound hexadecimal expression> ::=
   <scalar hexadecimal expression> { "||" | + } <scalar hexadecimal expression>
```

```
<compound ingresdate expression> ::=
   <scalar ingresdate expression> { + | - } <scalar ingres-interval expression> |
   <scalar ingresdate expression> + <scalar ansi-interval expression>           |
   <scalar ansi-interval expression> + <scalar ingresdate expression>           |
   ( <scalar ingresdate expression> )
```

```
<compound ingres-interval expression> ::=
   <scalar ingresdate expression> - <scalar ingresdate expression> |
   <scalar ingresdate expression> - <scalar ansidate expression>   |
   ( <scalar ingres-interval expression> )
```

```
<compound interval expression> ::=
   <compound ansi-interval expression>   |
   <compound ingres-interval expression>
```

```
<compound numeric expression> ::=
   [ + | - ] <scalar numeric expression>                                  |
   ( <scalar numeric expression> )                                        |
   { <scalar numeric expression> | <scalar alphanumeric expression > }
        <mathematical operator>
        { <scalar numeric expression> | <scalar alphanumeric expression> } |
   [ + | - ] <scalar alphanumeric expression>
```

```
<compound scalar expression> ::=
   <compound numeric expression>      |
   <compound alphanumeric expression> |
   <compound interval expression>     |
   <compound date expression>         |
   <compound time expression>         |
   <compound timestamp expression>    |
   <compound hexadecimal expression>
```

```
<compound table expression> ::=
   <table expression> <set operator> <table expression>
```

```
<compound time expression> ::=
   <scalar time expression> { + | - } <scalar ansi-interval expression> |
   ( <scalar time expression> )
```

```
<compound timestamp expression> ::=
   <scalar timestamp expression> { + | - } <scalar ansi-interval expression> |
   <scalar ansi-interval expression> + <scalar timestamp expression>         |
   ( <scalar timestamp expression> )
```

```
<condition> ::=
   <predicate>                 |
   <predicate> OR <predicate>  |
   <predicate> AND <predicate> |
   ( <condition> )             |
   NOT <condition>
```

```
<copy option> ::=
   ON_ERROR = { TERMINATE | CONTINUE } |
   ERROR_COUNT = <whole number>        |
   ROLLBACK = { ENABLED | DISABLED }   |
   LOG = <file specification>          |
   ALLOCATION = <whole number>         |
   EXTEND = <whole number>             |
   FILLFACTOR = <whole number>         |
   MINPAGES = <whole number>           |
   MAXPAGES = <whole number>           |
   LEAFFILL = <whole number>           |
   NONLEAFFILL = <whole number>        |
   ROW_ESTIMATE = <whole number>
```

```
<database privileges> ::=
   ALL [ PRIVILEGES ] |
   <database privilege> [ , <database privilege> ]...
```

```
<database privilege> ::=
   ACCESS                              |
   NOACCESS                            |
   CREATE_PROCEDURE                    |
   NOCREATE_SEQUENCE                   |
   CREATE_SEQUENCE                     |
   NOCREATE_SEQUENCE                   |
   CREATE_TABLE                        |
   NOCREATE_TABLE                      |
   DB_ADMIN                            |
   NODB_ADMIN                          |
   LOCKMODE                            |
   NOLOCKMODE                          |
   QUERY_COST_LIMIT <whole number>     |
   NOQUERY_COST_LIMIT                  |
   QUERY_CPU_LIMIT <whole number>      |
   NOQUERY_CPU_LIMIT                   |
   QUERY_IO_LIMIT <whole number>       |
   NOQUERY_IO_LIMIT                    |
   QUERY_PAGE_LIMIT <whole number>     |
   NOQUERY_PAGE_LIMIT                  |
   QUERY_ROW_LIMIT <whole number>      |
   NOQUERY_ROW_LIMIT                   |
   UPDATE_SYSCAT                       |
   NOUPDATE_SYSCAT                     |
   SELECT_SYSCAT                       |
   NOSELECT_SYSCAT                     |
   CONNECT_TIME_LIMIT <whole number>   |
   NOCONNECT_TIME_LIMIT                |
   IDLE_TIME_LIMIT <whole number>      |
   NOIDLE_TIME_LIMIT                   |
   SESSION_PRIORITY <whole number>     |
   NOSESSION_PRIORITY                  |
   TABLE_STATISTICS                    |
   NOTABLE_STATISTICS
```

```
<data type> ::=
   <numeric data type>      |
   <alphanumeric data type> |
   <temporal data type>     |
   <hexadecimal data type>  |
   <money data type>        |
   <logical key data type>
```

```
<data type list> ::= ( <data type> [ , <data type> ]... )
```

```
<date literal> ::=
   <ansidate literal>   |
   <ingresdate literal>
```

```
<date part> ::= <character list>
```

```
<days> ::= <digit> [ <digit> ]
```

```
<dbevent privilege> ::= RAISE | REGISTER
```

```
<dbevent privileges> := <dbevent privilege> [ , <dbevent privilege> ]...
```

```
<decimal data type> ::=
   DECIMAL [ ( <precision> [ , <scale> ] ) ] |
   DEC     [ ( <precision> [ , <scale> ] ) ] |
   NUMERIC [ ( <precision> [ , <scale> ] ) ]
```

```
<decimal literal> ::=
   [ + | - ] <whole number> [ . <whole number> ] |
   [ + | - ] <whole number> .                    |
   [ + | - ] . <whole number>
```

```
<default specification> ::=
   WITH DEFAULT                                    |
   [ WITH ] DEFAULT { <literal> | <system variable> } |
   NOT DEFAULT
```

```
<digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

```
<exponent> ::= <integer literal>
```

```
<expression> ::=
   <scalar expression> |
   <row expression>    |
   <table expression>
```

```
<file specification> ::= Specification of a file according to the
                         rules of the operating system.
```

```
<first specification> ::= FIRST <whole number>
```

```
<float data type> ::=
   FLOAT [ ( <length> ) ] |
   FLOAT4 | REAL |
   FLOAT8 | F | DOUBLE PRECISION
```

```
<float literal> ::= <mantissa> { E | e } <exponent>
```

```
<flow control statement> ::=
   <if statement>      |
   <while statement>   |
   <for statement>     |
   <endloop statement>
```

```
<for clause> ::=
   FOR [ DEFERRED | DIRECT ] UPDATE [ OF <column name> [ , <column name> ]... ]
```

```
<foreign key> ::=
   FOREIGN KEY <column list> <referencing specification>
```

```
<from clause> ::= FROM <table reference> [ , <table reference> ]...
```

```
<grantees> ::=
   PUBLIC                                        |
   [ USER ]  <user name>  [ , <user name> ]...   |
   [ GROUP ] <group name> [ , <group name> ]... |
   [ ROLE ]  <role name>  [ , <role name> ]...
```

```
<group by clause> ::= GROUP BY <group by specification list>
```

```
<group by specification> ::= <scalar expression>
```

```
<group by specification list> ::=
   <group by specification> [ , <group by specification> ]...
```

```
<hash partitioning> ::=
   HASH ON <column list>
   <hash partitioning specification> [ , <hash partitioning specification> ]...
```

```
<hash partitioning specification> ::=
   PARTITION [ <partition name> ] [ WITH LOCATION = <location list> ] |
   <whole number> PARTITIONS [ WITH LOCATION = <location list> ]
```

```
<having clause> ::= HAVING <condition>
```

```
<hexadecimal character> ::=
   <digit> | A | B | C | D | E | F | a | b | c | d | e | f
```

```
<hexadecimal data type> ::=
   BYTE [ ( <length> ) ]         |
   BINARY [ ( <length> ) ]       |
   BYTE VARYING [ ( <length> ) ] |
   BYTE VAR [ ( <length> ) ]     |
   VARBINARY [ ( <length> ) ]    |
   LONG BYTE                     |
   BLOB                          |
   BINARY LARGE OBJECT
```

```
<hexadecimal literal> ::=
   { X | x } ' <hexadecimal character>... ' |
   { 0x | 0X } <hexadecimal character>...
```

```
<host variable> ::= ":" <host variable name>
```

```
<host variable element> ::= <host variable> [ <null indicator> ]
```

```
<host variable list> ::= <host variable element> [ , <host variable element> ]...
```

```
<hours> ::= <digit> [ <digit> ]
```

```
<index option> ::=
   LOCATION = ( <location name> [ , <location name> ]... )   |
   DUPLICATES | NODUPLICATES                                 |
   PAGE_SIZE = <whole number>                                |
   STRUCTURE = { BTREE | ISAM | HASH | RTREE }               |
   KEY = <column list>                                       |
   FILLFACTOR = <whole number>                               |
   MINPAGES = <whole number>                                 |
   MAXPAGES = <whole number>                                 |
   LEAFFILL = <whole number>                                 |
   NONLEAFFILL = <whole number>                              |
   COMPRESSION [ = ( [ KEY | NOKEY ] [ , DATA | NODATA ] ) ] |
   NOCOMPRESSION                                             |
   ALLOCATION = <whole number>                               |
   EXTEND = <whole number>                                   |
   UNIQUE_SCOPE = { STATEMENT | ROW }                        |
   RANGE = ( ( <whole number> , <whole number> ) ,
             ( <whole number> , <whole number> ) )           |
   PRIORITY = <digit>
```

```
<ingresdate literal> ::=
   { DATE | TIME | TIMESTAMP }
   { ' <date part> <space> <time part> ' |
     ' <date part> '                     |
     ' <time part> '                     }
```

```
<ingres-interval literal> ::= ' <interval specification>... '
```

```
<integer data type> ::=
   TINYINT  | INT1 |
   SMALLINT | INT2 |
   INTEGER  | INT4 | INT | I |
   BIGINT   | INT8
```

```
<integer literal> ::= [ + | - ] <whole number>
```

```
<interval length> ::=
   <whole number>                  |
   <whole number> - <whole number> |
   <decimal literal>               |
   <whole number> <space> <whole number> : <whole number> :
      <whole number> [ . <whole number> ]
```

```
<interval literal> ::=
   <ansi-interval literal>   |
   <ingres-interval literal>
```

```
<interval precision> ::=
   YEAR TO MONTH                          |
   DAY TO SECOND [ ( <time precision> ) ]
```

```
<interval specification> ::=
   <whole number> { YRS | MOS | DAYS | HRS | MINS | SECS }
```

```
<interval unit> ::=
   YEAR TO MONTH | DAY TO SECOND | YEAR | MONTH | DAY |
   HOUR | MINUTE | SECOND
```

Two versions of the into clause exist. The first one is for use within embedded SQL, and the second one for use within stored procedures.

```
<into clause> ::= INTO <host variable> [ , { <host variable> ]...

<into clause> ::=
   INTO { <local variable> | <parameter name> }
      [ , { <local variable> | <parameter name> } ]...
```

```
<join condition> ::= ON <condition> | USING <column list>
```

```
<join specification> ::=
   <table reference> <join type> <table reference> <join condition>
```

```
<join type> ::=
   [ INNER ] JOIN       |
   LEFT  [ OUTER ] JOIN |
   RIGHT [ OUTER ] JOIN |
   FULL  [ OUTER ] JOIN |
   CROSS JOIN
```

```
<key option> ::=
   LOCATION = <location list>                                    |
   PAGE_SIZE = { 2048 | 4096 | 8192 | 16384 | 32768 | 65536 } |
   STRUCTURE = { HASH | ISAM | BTREE }                           |
   FILLFACTOR = <whole number>                                   |
   MINPAGES = <whole number>                                     |
   MAXPAGES = <whole number>                                     |
   LEAFFILL = <whole number>                                     |
   NONLEAFFILL = <whole number>                                  |
   ALLOCATION = <whole number>                                   |
   EXTEND = <whole number>                                       |
   INDEX = { <index name> | BASE_TABLE_STRUCTURE }               |
   NOINDEX
```

```
<label> ::= <object name>
```

```
<length> ::= <whole number>
```

```
<letter> ::=
   a | b | c | d | e | f | g | h | i | j | k | l | m |
   n | o | p | q | r | s | t | u | v | w | x | y | z |
   A | B | C | D | E | F | G | H | I | J | K | L | M |
   M | O | P | Q | R | S | T | U | V | W | X | Y | Z
```

```
<like pattern> ::= <scalar alphanumeric expression>
```

```
<list partitioning> ::=
   LIST ON <column list>
   <list partitioning specification> [ , <list partitioning specification> ]...
```

```
<list partitioning specification> ::=
   PARTITION [ <partition name> ]
      VALUES <list values> [ WITH LOCATION = <location list> ]
```

```
<list values> ::=
   <literal list>                          |
   ( <literal list> [ , <literal list> ]... ) |
   ( DEFAULT )
```

```
<literal> ::=
   <numeric literal>      |
   <alphanumeric literal> |
   <temporal literal>     |
   <hexadecimal literal>
```

```
<literal list> ::= ( <literal> [ , <literal> ]... )
```

```
<local variable> ::= <object name>
```

```
<location list> ::= ( <location name> [ , <location name> ]... )
```

```
<logical key data type> ::=
   TABLE_KEY [ WITH SYSTEM_MAINTAINED | NOT SYSTEM_MAINTAINED ]  |
   OBJECT_KEY [ WITH SYSTEM_MAINTAINED | NOT SYSTEM_MAINTAINED ]
```

```
<mantissa> ::= <decimal literal>
```

```
<mathematical operator> ::= * | / | + | - | **
```

```
<minutes> ::= <digit> [ <digit> ]
```

```
<modify option> ::=
   NODEPENDENCY_CHECK                                                    |
   ALLOCATION = <whole number>                                           |
   EXTEND = <whole number>                                               |
   FILLFACTOR = <whole number>                                           |
   MINPAGES = <whole number>                                             |
   MAXPAGES = <whole number>                                             |
   LEAFFILL = <whole number>                                             |
   NONLEAFFILL = <whole number>                                          |
   BLOB_EXTEND = <whole number>                                          |
   NEWLOCATION = <location list>                                         |
   OLDLOCATION = <location list>                                         |
   LOCATION = <location list>                                            |
   COMPRESSION [ = ( [ KEY | NOKEY ] [ , DATA | NODATA | HIDATA ] ) ] |
   PERSISTENCE                                                           |
   NOPERSISTENCE                                                         |
   UNIQUE_SCOPE = { ROW | STATEMENT }                                    |
```

continues →

```
   PAGE_SIZE = { 2048 | 4096 | 8192 | 16384 | 32768 | 65536 }         |
   PRIORITY = <number>                                                |
   NOPARTITION                                                        |
   PARTITION = ( <partitioning scheme> )                              |
   CONCURRENT_UPDATES
```

```
<money data type> ::= MONEY
```

```
<months> ::= <digit> [ <digit> ]
```

```
<null default specification> ::=
   NOT NULL               |
   WITH NULL              |
   NOT NULL WITH DEFAULT  |
   NOT NULL NOT DEFAULT
```

```
<null indicator> ::= <host variable>
```

```
<null specification> ::= WITH NULL | NOT NULL
```

```
<numeric data type> ::=
   <integer data type> |
   <decimal data type> |
   <float data type>
```

```
<numeric literal> ::=
   <integer literal> |
   <decimal literal> |
   <float literal>
```

```
<order by clause> ::=
   ORDER BY <sort specification> [ , <sort specification> ]...
```

```
<partitioning option> ::= PARTITION = <partitioning scheme>
```

```
<partitioning rule> ::=
   <automatic partitioning> |
   <hash partitioning>      |
   <list partitioning>      |
   <range partitioning>
```

```
<partitioning scheme> ::=
   { ( <partitioning rule> )                    |
     ( ( <partitioning rule> )
        SUBPARTITION ( <partitioning rule> ) ) }
```

```
<password> ::= <alphanumeric literal>
```

```
<password without quotation marks> ::= <character list>
```

```
<precision> ::= <whole number>
```

```
<predicate> ::=
   <predicate with comparison> |
   <predicate with in>         |
   <predicate with between>    |
   <predicate with like>       |
   <predicate with null>       |
   <predicate with data type>  |
   <predicate with exists>     |
   <predicate with any all>
```

```
<predicate with any all> ::=
   <scalar expression> <any all operator> <column subquery>
```

```
<predicate with between> ::=
   <scalar expression> [ NOT ] BETWEEN <scalar expression>
      AND <scalar expression>
```

```
<predicate with comparison> ::=
   <scalar expression> <comparison operator> <scalar expression>
```

```
<predicate with data type> ::=
   <scalar expression> IS { INTEGER | DECIMAL | FLOAT }
```

```
<predicate with exists> ::= EXISTS <table subquery>
```

```
<predicate with in> ::=
   <scalar expression> [ NOT ] IN <scalar expression list> |
   <scalar expression> [ NOT ] IN <column subquery>
```

```
<predicate with like> ::=
   <scalar expression> [ NOT ] LIKE <like pattern> [ ESCAPE <special character> ]
```

```
<predicate with null> ::= <scalar expression> IS [ NOT ] NULL
```

```
<primary key> ::= PRIMARY KEY <column list>
```

```
<procedure condition> ::=
   <procedure predicate>                          |
   <procedure predicate> OR <procedure predicate>  |
   <procedure predicate> AND <procedure predicate> |
   ( <procedure condition> )                       |
   NOT <procedure condition>
```

```
<procedure predicate> ::=
   <predicate with comparison> |
   <predicate with between>    |
   <predicate with like>       |
   <predicate with null>       |
   <predicate with data type>  |
   <predicate with in>
```

```
<pseudonym> ::= <object name>
```

```
<range operator> ::= < | <= | > | >=
```

```
<range partitioning> ::=
   RANGE ON <column list>
   <range partitioning specification> [ , <range partitioning specification> ]...
```

```
<range partitioning specification> ::=
   PARTITION [ <partition name> ]
      VALUES <range operator> <range values> [ WITH LOCATION = <location list> ]
```

```
<range values> ::= <literal> | <literal list>
```

```
<referencing action> ::=
   ON UPDATE { CASCADE | RESTRICT | SET NULL | NO ACTION } |
   ON DELETE { CASCADE | RESTRICT | SET NULL | NO ACTION }
```

```
<referencing specification> ::=
   REFERENCES <table specification> [ <column list> ] [ <referencing action>... ]
```

```
<role property> ::=
   PASSWORD = <password>                |
   EXTERNAL_PASSWORD                    |
   NOPASSWORD                           |
   NO PRIVILEGES                        |
   PRIVILEGES = ( <subject privileges> ) |
   NOSECURITY_AUDIT                     |
   SECURITY_AUDIT
```

```
<row expression> ::= <singular row expression>
```

For each alphanumeric, hexadecimal, numeric, and temporal data type a version of the scalar expression exists.

```
<scalar expression> ::=
   <singular scalar expression> |
   <compound scalar expression>
```

```
<scalar expression list> ::= ( <scalar expression> [ , <scalar expression> ]... )
```

```
<scale> ::= <whole number>
```

```
<seconds> ::= <digit> [ <digit> ]
```

```
<seconds fraction> ::= <whole number>
```

```
<security label> ::= C2
```

```
<select block head> ::=
   <select clause>
   [ <from clause>
   [ <where clause> ]
   [ <group by clause> ]
   [ <having clause> ] ]
```

```
<select block tail> ::= <order by clause>
```

```
<select clause> ::=
   SELECT [ <first specification> ] [ DISTINCT | ALL ] <select element list>
```

```
<select element> ::=
   <scalar expression> [ [ AS ] <column name> ] |
   <table specification>.*                      |
   <pseudonym>.*                                |
   *
```

```
<select element list> ::= <select element> [ , <select element> ]...
```

```
<sequence expression> ::=
   NEXT VALUE FOR <sequence name>    |
   <sequence name>.NEXTVAL           |
   CURRENT VALUE FOR <sequence name> |
   <sequence name>.CURRVAL
```

```
<sequence number> ::= <whole number>
```

```
<set operator> ::= UNION | UNION ALL
```

```
<singular row expression> ::=
   ( <scalar expression> [ , <scalar expression> ]... ) |
   <row subquery>
```

For each alphanumeric, hexadecimal, numeric, and temporal data type a version of the singular scalar expression exists. Host variables are only allowed when the singular scalar expression is used in embedded SQL. The parameter name and the local variable are only allowed inside stored procedures.

```
<singular scalar expression> ::=
   <literal>               |
   <column specification>  |
   <system variable>       |
   <cast expression>       |
   <case expression>       |
   NULL                    |
   ( <scalar expression> ) |
   <scalar function>       |
   <aggregation function>  |
   <scalar subquery>       |
   TID                     |
   TIDP                    |
   <sequence expression>   |
   <host variable>         |
   <parameter name>        |
   <local variable>
```

```
<sort direction> ::= ASC | DESC
```

```
<sort specification> ::=
   <column name> [ <sort direction> ]       |
   <scalar expression> [ <sort direction> ] |
   <sequence number> [ <sort direction> ]
```

```
<special symbol> ::= all unusual characters, such as !, # and *
```

```
<subject privilege> ::=
   CREATEDB | TRACE | SECURITY | OPERATOR | MAINTAIN_LOCATIONS | AUDITOR |
   MAINTAIN_AUDIT | MAINTAIN_USERS
```

```
<subject privileges> ::= <subject privilege> [ , <subject privilege> ]...
```

```
<subquery> ::= ( <table expression> )
```

```
<system variable> ::=
   CURRENT_DATE | CURRENT_TIME | CURRENT_TIMESTAMP |
   CURRENT_USER | INITIAL_USER | LOCAL_TIME | LOCAL_TIMESTAMP |
   SESSION_USER | SYSTEM_USER | USER
```

```
<table element> ::=
   <column definition>          |
   <table integrity constraint>
```

```
<table expression> ::=
   { <select block head>         |
     ( <table expression> )      |
     <compound table expression> }
   [ <select block tail> ]
```

```
<table integrity constraint> ::=
   [ CONSTRAINT <constraint name> ]
   { <primary key>               |
     <alternate key>             |
     <foreign key>               |
     <check integrity constraint> } [ WITH <key option> [ , <key option> ]... ]
```

```
<table option> ::=
   LOCATION = <location list>                                  |
   JOURNALING | NOJOURNALING                                   |
   DUPLICATES | NODUPLICATES                                   |
   PAGE_SIZE = { 2048 | 4096 | 8192 | 16384 | 32768 | 65536 } |
   SECURITY_AUDIT = ( { TABLE | ROW | NOROW }... )             |
   SECURITY_AUDIT_KEY = ( <column name> )                      |
   NOPARTITION | PARTITION = <partitioning scheme>             |
   STRUCTURE = { HASH | HEAP | ISAM | BTREE }                  |
   KEY = <column list>                                         |
   FILLFACTOR = <whole number>                                 |
   MINPAGES = <whole number>                                   |
   MAXPAGES = <whole number>                                   |
   LEAFFILL = <whole number>                                   |
   NONLEAFFILL = <whole number>                                |
   COMPRESSION [ = ( [ KEY | NOKEY ] [ , DATA | NODATA ] ) ]   |
```

continues →

```
   NOCOMPRESSION                                          |
   ALLOCATION = <whole number>                            |
   EXTEND =  whole number>                                |
   PRIORITY = <digit>
```

```
<table privilege> ::=
   SELECT                                  |
   INSERT                                  |
   DELETE                                  |
   UPDATE [ EXCLUDING ] [ <column list> ]  |
   REFERENCES [ EXCLUDING ] [ <column list> ] |
   COPY_INTO                               |
   COPY_FROM
```

```
<table privileges> ::=
   ALL [ PRIVILEGES ] |
   <table privilege> [ , <table privilege> ]...
```

In the table reference below, use of the index specification is allowed only in queries, not in updates, deletes, or inserts.

```
<table reference> ::=
   <table specification> [ [ AS ] <pseudonym> ] |
   <join specification>                         |
   ( <join specification> )                     |
   <table subquery> [ AS ] <pseudonym>          |
   <index specification> [ [ AS ] <pseudonym> ]
```

```
<table schema> ::= ( <table element> [ , <table element> ]... )
```

```
<table specification> ::= [ <user name> . ] <table name>
```

```
<table subquery> ::= ( <table expression> )
```

```
<temporal data type> ::=
   DATE                                                              |
   ANSIDATE                                                          |
   INGRESDATE                                                        |
   TIME [ ( <time precision> ) ] [ <time zone specification> ]       |
   TIMESTAMP [ ( <time precision> ) ] [ <timezone specification> ] |
   INTERVAL <interval precision>
```

```
<temporal literal> ::=
   <date literal>      |
   <time literal>      |
   <timestamp literal> |
   <interval literal>
```

```
<time literal> ::=
   TIME ' <hours> : <minutes> : <seconds> [ . <seconds fraction> ] '
```

```
<time part> ::=
   <hours> : <minutes> [ : <seconds> ] [ AM | PM ] [ <time zone> ]
```

```
<time precision> ::= <whole number>
```

```
<timestamp literal> ::=
   { TIMESTAMP | timestamp }
   ' <years> - <months> - <days> <space>
     <hours> : <minutes> : <seconds> [ . < seconds fraction> ] [ <time zone> ] '
```

```
<time zone> ::= { + | - } <hours> : <minutes>
```

```
<time zone specification> ::=
   WITH TIME ZONE       |
   WITHOUT TIME ZONE    |
   WITH LOCAL TIME ZONE
```

```
<usage specification> ::=
   USAGE = ( <usage type> [ , <usage type> ]... ) |
   NOUSAGE
```

```
<usage type> ::=
   DATABASE | WORK | JOURNAL | CHECKPOINT | DUMP | ALL
```

```
<user properties> ::= <user property> [ , <user property> ]...
```

```
<user property> ::=
   PASSWORD = <password>                                                    |
   NOPASSWORD                                                               |
   EXTERNAL_PASSWORD                                                        |
   PRIVILEGES = ( <subject privileges> )                                    |
   NOPRIVILEGES                                                             |
   DEFAULT_PRIVILEGES = ( <subject privileges>                              |
   DEFAULT_PRIVILEGES = ALL                                                 |
   NODEFAULT_PRIVILEGES                                                     |
   GROUP = <group name>                                                     |
   NOGROUP                                                                  |
   EXPIRE_DATE = { <date literal> | <time literal> | <timestamp literal> } |
   NOEXPIREDATE                                                             |
   PROFILE = <profile name>                                                 |
   NOPROFILE                                                                |
   SECURITY_AUDIT = ( <audit option> [ , <audit option> ]... )              |
   NOSECURITY_AUDIT
```

```
<values clause> ::=
   VALUES ( <scalar expression> [ , <scalar expression> ]... )
```

```
<when definition> ::= <when definition-1> | <when definition-2>
```

```
<when definition-1> ::=
   <scalar expression>
   WHEN <scalar expression> THEN <scalar expression>
   [ WHEN <scalar expression> THEN <scalar expression> ]...
```

```
<when definition-2> ::=
   WHEN <condition> THEN <scalar expression>
   [ WHEN <condition> THEN <scalar expression> ]...
```

```
<where clause> ::= WHERE <condition>
```

```
<whole number> ::= <digit>...
```

```
<years> ::= <digit> <digit> <digit> <digit>
```

```
<alarm name>       ;
<collation name>   ;
<column name>      ;
<connection name>  ;
<constraint name>  ;
<cursor name>      ;
<database name>    ;
<dbevent name>     ;
<group name>       ;
<index name>       ;
<location name>    ;
<partition name>   ;
<profile name>     ;
<procedure name>   ;
<role name>        ;
<savepoint name>   ;
<schema name>      ;
<sequence name>    ;
<table name>       ;
<transaction name> ;
<rule name>        ;
<user name>        ;
<view name>        ::= <object name>
```

```
<object name> ::=
   <letter> [ <letter> | <digit> | _ | # | @ | $ ]...
```