

BSI

DD 151:1986

UDC 681.3.06 NDL

© British Standards Institution. No part of this publication may be photocopied or otherwise reproduced without the prior permission in writing of BSI

Draft for Development

Database language NDL (ISO/DIS 8907)

Langage de base de données réseau

Datenbanksprache NDL

British Standards Institution

AVA Documents
Document Sharing Hub

Contents

	BS page		BS page
National foreword	2	6.6 <component type>	36
Committees responsible	Back cover	6.7 <component name clause>	37
		6.8 <default clause>	38
Draft for Development		6.9 <record check clause>	40
1 Scope and field of application	3	6.10 <set type>	41
1.1 Scope	3	6.11 <set name clause>	42
1.2 Field of application	4	6.12 <owner clause>	43
2 References	5	6.13 <order clause>	44
3 Overview	6	6.14 <member clause>	47
3.1 Organization	6	6.15 <member record name clause>	48
3.2 Notation	6	6.16 <insertion clause>	49
3.3 Conventions	7	6.16 <retention clause>	51
3.4 Conformance	8	6.18 <member uniqueness clause>	52
4 Concepts	9	6.19 <key clause>	53
4.1 Data type	9	6.20 <member check clause>	56
4.1.1 Character strings	9	6.21 <component identifier>	58
4.1.2 Numbers	9	7 Subschema definition language	60
4.2 Components	10	7.1 <subschema>	60
4.3 Records	10	7.2 <subschema name clause>	61
4.3.1 Database keys	10	7.3 <record view>	62
4.3.2 System record	11	7.4 <component view>	63
4.4 Set types	11	7.5 <set view>	64
4.4.1 Singular sets	11	8 Module language	65
4.4.2 Recursive sets	11	8.1 <module>	65
4.5 Schema	11	8.2 <module name clause>	67
4.6 Subschemas	12	8.3 <temporary set specifications>	68
4.7 Database	12	8.4 <procedure>	69
4.8 Modules	12	9 Data manipulation language	74
4.9 Procedures	12	9.1 <commit statement>	74
4.10 Standard programming languages	13	9.2 <connect statement>	75
4.11 Temporary sets	13	9.3 <disconnect statement>	76
4.12 Parameters	13	9.4 <erase statement>	77
4.12.1 STATUS parameter	13	9.5 <find statement>	79
4.12.2 TEST parameter	13	9.6 <get statement>	84
4.12.3 RECORD parameter	13	9.7 <modify statement>	85
4.13 Session	13	9.8 <>nullify cursor statement>	87
4.14 Transactions	14	9.9 <ready statement>	88
4.15 Integrity constraints	14	9.10 <reconnect statement>	90
4.15.1 Check conditions	15	9.11 <rollback statement>	91
4.15.2 Default values	15	9.12 <store statement>	92
4.15.3 Uniqueness constraints	15	9.13 <test database key equal statement>	95
4.15.4 Set ordering criteria	15	9.14 <test database key null statement>	96
4.15.5 Set membership	15	9.15 <test set empty statement>	97
5 Common elements	17	9.16 <test set membership statement>	98
5.1 <condition>	17	9.17 <database key identifier>	99
5.2 <operand>	20	9.18 <component view identifier>	100
5.3 <identifier>	20	9.19 <parameter identifier>	102
5.4 <literal>	23	9.20 <to parameter move> and <to database move>	103
5.5 <data type>	25	10 Auxilliary operations	105
5.6 <occurs clause>	27	10.1 <insert operation>	105
5.7 <subscripts>	28	10.2 <remove operation>	108
5.8 Comments, spaces, and key words	29	11 Interpretive state	110
6 Schema definition language	31	11.1 <session state>	110
6.1 <schema>	31	11.2 <cursors>	111
6.2 <schema name clause>	32	11.3 <temporary sets>	113
6.3 <record type>	33	11.4 <ready list>	114
6.4 <record name clause>	34	12 Status codes	115
6.5 <record uniqueness clause>	35	13 Levels	117

	BS page		BS page
Annexes			
A Example suppliers and parts problem	118	B.3 Example recursive set program	122
A.1 Example suppliers and parts <schema>	118	B.4 Example recursive set <module>	123
A.2 Example suppliers and parts <subschema>	119	C Example bill of materials problem	125
A.3 Example suppliers and parts program	119	C.1 Example bill of materials <schema>	125
A.4 Example suppliers and parts <module>	120	C.2 Example bill of materials <subschema>	125
B Example recursive set problem	122	C.3 Example bill of materials program	126
B.1 Example recursive set <schema>	122	C.4 Example bill of materials <module>	127
B.2 Example recursive set <subschema>	122	Index	129

This publication is not to be regarded as a British Standard.

It is being issued in the Draft for Development series of publications and is of a provisional nature because the subject matter is in the process of ongoing development within the International Standards Committee (ISO/TC 97/SC 21 Information Retrieval, Transfer and Management for Open Systems Interconnection).

It should be applied on this provisional basis, so that information and experience of its practical application may be obtained.

A review of this Draft for Development will be carried out not later than two years after its publication. Notification of

the start of the review period, with a request for the submission of comments from users of this Draft for Development will be made in an announcement in the appropriate issue of *BSI News*. Observations which it is felt should receive attention before the official call for comments will be welcomed.

According to the replies received, the responsible BSI Committee will judge whether the Draft for Development can be converted into a British Standard or what other action should be taken.

National foreword

This Draft for Development has been prepared under the direction of the Office and Information Standards Committee.

The primary objective of this Draft for Development is to give advance guidance to product and systems designers concerned with database languages.

The text of the Draft for Development is identical with Draft International Standard ISO/DIS 8907 issued May 1986 by the International Organization for Standardization (ISO).

For ease of reproduction the ISO text has been reproduced unchanged.

The UK has actively participated through BSI Technical Committee IST/21 in the development of the Draft International Standard. The Technical Committee will review any agreed amendments or addenda to the ISO text and will, where appropriate, up-date the text of this Draft for Development via an amendment or revision. When the ISO Draft reaches an appropriate level of agreement and stability, this Draft for Development may then be converted to a British Standard.

A brochure giving details of existing and projected British Standards publications for Open Systems Interconnection (OSI), data communications, database and computer graphics may be obtained from the Sales Department, BSI, Linford Wood, Milton Keynes, MK14 6LE, at no charge. Press releases for new British Standard publications in these areas will be issued to the computing press, and details of all new publications will be recorded in full in *BSI News*.

Terminology and conventions. The text of the Draft International Standard has been approved as suitable for publication as a Draft for Development without deviation. Some terminology and certain conventions are not identical with those used in Drafts for Development; attention is drawn especially to the following.

Wherever the word 'standard' appears, referring to this Draft for Development, it should be read as 'Draft for Development'.

Wherever page numbers are quoted, they are ISO page numbers. Readers should note that ISO page numbers 4, 8, 60, 66, 76, 108, 114, 120, 124 and 132 are blank pages and have not been reproduced in this Draft for Development.

Cross-references

International standard	Corresponding British Standard
ISO 7185-1983	BS 6192 : 1982 Specification for computer programming language Pascal (Identical)

The Technical Committee has reviewed the provisions of ISO 1539, ISO 1989 and ISO 6160, to which reference is made in the text, and has decided that they are acceptable for use in conjunction with this Draft for Development.

1. Scope and Field of Application

1.1 Scope

1. This standard specifies the syntax and semantics of three database languages:
 - a. A schema definition language, for declaring the structures and integrity constraints of an NDL database.
 - b. A subschema definition language, for declaring a user view of that database.
 - c. A module language and data manipulation language, for declaring the database procedures and executable statements of a specific database application.
2. This standard defines the logical data structures and basic operations for an NDL database. It provides functional capabilities for designing, accessing, maintaining, controlling, and protecting the database.
3. This standard provides a vehicle for portability of database definition and application modules between conforming systems.
4. This standard specifies two levels. Level 2 is the complete NDL database language. Level 1 is the subset of Level 2 defined in Clause 13, "Levels" on page 123.
5. This standard does not specify some of the facilities that might be provided in operational database environments, such as the following:
 - a. An access control facility for granting access and operational privileges to specific users.
 - b. Additional integrity control capabilities for specifying more complex integrity constraints on the database.
 - c. A facility to import and export schema definitions.
 - d. A database unload facility for copying record and set populations to standard files for information interchange.
 - e. A schema database for making schema and subschema information available to accessing applications.
 - f. A schema manipulation language for creating, modifying, or deleting portions of the schema or subschemas.
 - g. Interfaces to a data dictionary.
 - h. Application program pre-processing facilities for producing separate standard database modules and standard language programs.

- i. A data storage definition language for defining physical storage structures and physical access methods.
- j. Database procedures for user specified assertions and triggers.
- k. A natural language query facility for ad hoc access to the database.
- l. Report generator facilities for producing output tables and charts.
- m. Graphics capabilities for direct database interface with standard graphics systems.
- n. A distributed database facility for defining and accessing data at different nodes in a communications network.

Standards for such additional facilities could be specified in a manner that is upward compatible with this standard. Some of the additional capabilities might be specified by subsequent versions of this standard, others could be specified by separate standards, and some may always be implementor-defined.

1.2 Field of Application

- 1. This standard applies to implementations that exist in an environment that may include application programming languages, end-user query languages, report generator systems, data dictionary systems, program library systems, and distributed communication systems, as well as various tools for database design, data administration, and performance optimization.

[ISO page 2]

This publication is a Draft for Development and is therefore not to be regarded as a British Standard. See page 2.

2. References

1. ISO 1539, *Programming Languages - FORTRAN*, (endorsement by ISO of ANSI X3.9).
2. ISO 1989, *Programming Languages - COBOL*, (endorsement by ISO of ANSI X3.23).
3. ISO 6160, *Programming Languages - PL/I*, (endorsement by ISO of ANSI X3.53).
4. ISO 7185, *Programming Languages - Pascal*, (endorsement by ISO of British Standard BS6192 for the English text).

[ISO page 3]

Documents

This publication is a Draft for Development and is therefore not to be regarded as a British Standard. See page 2.

3. Overview

3.1 Organization

1. Clause 3.2, "Notation" on page 5 and Clause 3.3, "Conventions" on page 6 define the notations and conventions used in this standard.
2. Clause 3.4, "Conformance" on page 7 defines conformance criteria.
3. Clause 4, "Concepts" on page 9 defines terms and presents concepts used in the definition of NDL.
4. Clause 5, "Common Elements" on page 17 defines language elements that occur in several parts of NDL language.
5. Clause 6, "Schema Definition Language" on page 31 defines the NDL facilities for specifying a database.
6. Clause 7, "Subschema Definition Language" on page 61 defines the NDL facilities for specifying portions of a database that can be referenced by an application program.
7. Clause 8, "Module Language" on page 67 defines NDL modules and procedures.
8. Clause 9, "Data Manipulation Language" on page 77 defines the data manipulation statements of NDL.
9. Clause 10, "Auxiliary Operations" on page 109 defines certain operations that are used in the definition of NDL data manipulation statements.
10. Clause 11, "Interpretive State" on page 115 defines the "session state", which is used in the definition of NDL data manipulation statements.
11. Clause 12, "Status Codes" on page 121 defines the values of the STATUS parameter after the execution of NDL data manipulation statements.
12. Clause 13, "Levels" on page 123 defines the two levels of NDL.

3.2 Notation

1. The syntactic notation used in this document is BNF ("Backus Normal Form" or "Backus-Naur Form"), with the following extensions:
 - a. Square brackets ([]) indicate optional elements.
 - b. Ellipses (...) indicate elements that may be repeated one or more times.
 - c. Braces ({}) group sequences of elements.

[ISO page 5]

This publication is a Draft for Development and is therefore not to be regarded as a British Standard. See page 2

Documents

Document Sharing Hub

2. In the BNF syntax, a production symbol $\langle A \rangle$ is defined to "contain" a production symbol $\langle B \rangle$ if $\langle B \rangle$ occurs someplace in the expansion of $\langle A \rangle$. If $\langle A \rangle$ contains $\langle B \rangle$, then $\langle B \rangle$ is "contained in" $\langle A \rangle$. If $\langle A \rangle$ contains $\langle B \rangle$, then $\langle A \rangle$ is the "containing" $\langle A \rangle$ production symbol for $\langle B \rangle$.
3. A production symbol $\langle A \rangle$ "immediately contains" a production symbol $\langle B \rangle$ if $\langle A \rangle$ contains $\langle B \rangle$ and if $\langle B \rangle$ occurs in the first expansion of $\langle A \rangle$.
4. The lexical units of this standard (e.g. $\langle \text{separator} \rangle$, $\langle \text{key word} \rangle$, $\langle \text{identifier} \rangle$, and $\langle \text{literal} \rangle$) are defined elsewhere in this document. The following characters have special significance in some production rules.

'	In $\langle \text{escape identifier} \rangle$
"	In $\langle \text{character string literal} \rangle$
(	In $\langle \text{comment} \rangle$, $\langle \text{condition} \rangle$, and $\langle \text{subscripts} \rangle$
)	In $\langle \text{comment} \rangle$, $\langle \text{condition} \rangle$, and $\langle \text{subscripts} \rangle$
*	In $\langle \text{comment} \rangle$
+	In $\langle \text{numeric literal} \rangle$
-	In $\langle \text{numeric literal} \rangle$
<	In $\langle \text{relation} \rangle$
>	In $\langle \text{relation} \rangle$
=	In $\langle \text{component identifier match} \rangle$, $\langle \text{relation} \rangle$, and $\langle \text{test database key equal statement} \rangle$
E	In $\langle \text{approximate numeric literal} \rangle$
—	In $\langle \text{identifier} \rangle$
	In $\langle \text{component identifier} \rangle$, $\langle \text{component view identifier} \rangle$, $\langle \text{numeric literal} \rangle$, $\langle \text{subschema name clause} \rangle$, and $\langle \text{subschema specification} \rangle$

3.3 Conventions

1. Syntactic elements of this standard are specified in terms of:
 - a. *Function*: A short statement of the purpose of the element.
 - b. *Format*: A BNF definition of the syntax of the element.
 - c. *Syntax Rules*: Additional syntactic constraints not expressed in BNF that the element shall satisfy.
 - d. *General Rules*: The run-time effect of the element on the database and the session state.

2. In the Syntax Rules, the term "shall" defines conditions that are required to be true of syntactically conforming NDL language. The treatment of NDL language that does not conform to the Formats or the Syntax Rules is implementor-defined.
3. A conforming implementation is not required to perform the exact sequence of actions defined in the General Rules, but shall achieve the same effect on the database as that sequence.
4. The term "persistent object" is used to characterize objects such as <module>s and <schema>s that are created and destroyed by implementor-defined mechanisms.
5. In this standard, clauses begin a new odd-numbered page, and in clause 5, "Common Elements" on page 17 through clause 11, "Interpretive State" on page 115 sub-clauses begin a new page. The resulting blank space is not significant.

3.4 Conformance

1. This standard specifies conforming NDL language and conforming NDL implementations. Conforming NDL language shall abide by the BNF Format and associated Syntax Rules. A conforming NDL implementation shall process standard conforming NDL language according to the General Rules.
2. A conforming implementation may provide additional facilities or options not specified by this standard. An implementation remains conforming even if it provides user options to process non-conforming NDL language or to process conforming NDL language in a non-conforming manner.
3. This standard does not define the method or the time of binding between application subsystems and database management system components.
4. An implementation that conforms to Level 1 of this standard shall meet the requirements for Level 2 conformance except that it may omit certain facilities as specified in Clause 13, "Levels" on page 123.
5. This standard specifies three languages and defines the syntax and semantics of each of them. Implementors of database management systems may claim conformance for their products to this standard in two ways. An implementation may support the identical syntax of the languages and claim both syntactic and functional conformance, or it may use different syntax to express the same concepts, semantics, and functionality, and claim functional conformance only.
6. Implementations claiming functional conformance to any of the three standard languages shall specify equivalence with the semantics and functionality of those standard languages. For example, it should be possible to show that a schema displayed by a database management system display utility could be recoded using the standard schema definition language syntax and have equivalent functionality.

4. Concepts

4.1 Data Type

1. A <data type> is a set of representable values. The logical representation of a value is a <literal>. The physical representation of a value is implementor-defined.
2. A value is primitive, in that it has no logical subdivision within this standard. Values are the basis of definition for data items and arrays.
3. A value is either a character string or a number. A character string and a number are not comparable values.

4.1.1 Character Strings

1. A character string consists of a sequence of characters of the implementor-defined character set. A character string has a length, which is a positive integer that specifies the number of characters in the sequence.

4.1.2 Numbers

1. A number is either an exact numeric value or an approximate numeric value. All numbers are comparable values.
2. An exact numeric value has a precision and a scale. The precision is a positive integer that determines the number of significant decimal digits. The scale is a signed integer. A scale of 0 indicates that the number is an integer. For a scale of N, the exact numeric value is the integer value of the significant digits multiplied by 10 to the power -N.
3. An approximate numeric value consists of a mantissa and an exponent. The mantissa is a signed numeric value, and the exponent is a signed integer that specifies the magnitude of the mantissa. An approximate numeric value has a precision. The precision is a positive integer that specifies the number of significant decimal digits in the mantissa.
4. Whenever an exact numeric value is assigned to a data item or parameter representing an exact numeric value, its value shall be exactly representable in the data type of the target. The value is converted to have the precision and scale of the target.
5. Whenever an exact or approximate numeric value is assigned to a data item or parameter representing an approximate numeric value, an approximation of its value is represented in the data type of the target. The value is converted to have the precision of the target.

4.2 Components

1. A component is either a data item or an array of data items. A **<component type>** defines a collection of component occurrences all having the same **<data type>**.
2. A data item consists of a single value. An array consists of a sequence of data items.
3. An array is specified by **<extents>** that are a list of **<unsigned integers>**. The number of data items that occur in an array is equal to the product of the **<unsigned integer>**s.
4. A data item not contained in an array is referenced by a **<component name>** and a **<database key>**. A data item contained in an array is referenced by a **<component name>**, a **<database key>**, and **<subscripts>**.
5. All values of the same **<component type>** have the same **<data type>**. If the **<component type>** defines an array, then each component occurrence is an array with the same dimension and extent integers. All character strings of the same character string component have the same **<length>**. All numbers of the same exact numeric component have the same **<precision>** and **<scale>**. All numbers of the same approximate numeric component have the same **<precision>**.
6. Every **<component type>** is defined as part of a **<record type>**. The **<component type>** specifies the **<data type>**. If a **<component type>** defines an array, then it specifies the **<extents>**. A character string **<component type>** specifies the **<length>** of its character strings. An exact numeric **<component type>** specifies the **<precision>** and **<scale>** of its numbers. An approximate numeric **<component type>** specifies the **<precision>** of its numbers.

4.3 Records

1. The record is the basic unit of manipulation in this standard. Records are stored, erased, found, modified, and connected, disconnected, and reconnected in sets.
2. A record is a collection of components. A **<record type>** defines a collection of record occurrences all having the same **<component type>**s. A **<record type>** defines its **<component type>**s, and specifies integrity constraints that its record occurrences shall satisfy.
3. Each record is an occurrence of exactly one **<record type>**, and consists of exactly the components defined by that **<record type>**.
4. The records of each **<record type>** are maintained in an implementor-defined order.

4.3.1 Database Keys

1. All records are distinguishable. Each record is identified uniquely by a **<database key>**. A **<database key>** is either *null*, in which case it identifies no record, or it is non-null and identifies exactly one record in the database. A **<database key>** has no logical representation, and is not available through any interface defined by this standard. The physical representation of a **<database key>** is implementor-defined.
2. **<database key>**s are used to model cursors in the session state and to define the semantics of statements in the data manipulation language. In the session state, a **<database key>** is referenced by **<database key>**. In data manipulation language statements, a **<database key>** is referenced by a **<database key identifier>**.

[ISO page 10]

4.3.2 System Record

1. This standard assumes the existence of a special SYSTEM <record type>. The SYSTEM <record type> has no <component type>s, no integrity constraints, and exactly one imaginary record occurrence, called the system record.
2. This standard does not specify a <database key> for the system record. The system record and SYSTEM <record type> exist only for the definition of singular sets.

4.4 Set Types

1. A <set type> is a defined relationship between two or more <record type>s.
2. In a <set type>, one <record type> is designated as the *owner* <record type>, and each of the other <record type>s is referred to as a *member* <record type> of the <set type>.
3. For each <set type>, there may be zero, one, or more *sets* in a database.
4. A set consists of an *owner record*, whose <record type> is the owner <record type>, and zero, one, or more *member records*, each of whose <record type> is a member <record type>.
5. A set is identified uniquely in a database by its <set name> and its owner record occurrence. When an occurrence of the owner <record type> of a <set type> is stored in the database, a new set of that <set type> is created.
6. When an occurrence of <record type> is stored in the database, it belongs to at most one set of each <set type> in which the <record type> is a member <record type>.
7. The member records of each set of a <set type> are maintained in a sequential order determined by the ordering criteria for that <set type>.

4.4.1 Singular Sets

1. A singular <set type> is a <set type> having SYSTEM as its owner <record type>. A singular set is the unique occurrence of a singular <set type>.

4.4.2 Recursive Sets

1. A recursive <set type> is a set type having the same <record type> as both the owner <record type> and as a member <record type>. A recursive set is an occurrence of a recursive <set type>.

4.5 Schema

1. A <schema> is a persistent object specified by the schema definition language. The <schema> is a logical description of the database. A <schema> consists of a <schema name>, a collection of <record type> descriptions, and a collection of <set type> descriptions.

4.6 Subschemas

1. A <subschema> is a persistent object specified by the subschema definition language. The <subschema> is a logical description of that portion of a database available to an accessing <module>. A <subschema> consists of a <subschema name>, a collection of <record view>s, and a collection of <set view>s.
2. A <record view> specifies the available <component type>s of a given <record type>. A <set view> specifies an available <set type>. <record view>s and <set view>s taken together determine the available member <record type>s of each <set type>.
3. <Record view>s and <set view>s may define alias <component view name>s, <record view name>s and <set view name>s for <component type>s, <record type>s, and <set type>s. The alias names become the only names available for those <component type>s, <record type>s, and <set type>s to an accessing <module>.
4. The records available to an accessing <module> are maintained in an implementor-defined order. This order is not necessarily the same as the implementor-defined order of the records of each <record type>.

4.7 Database

1. A database is the collection of all data defined by a <schema>. It consists of the occurrences of each <record type> and each <set type>. There is exactly one database for each <schema> definition.

4.8 Modules

1. A <module> is a persistent object specified in the module language. A <module> consists of an optional <module name>, a <subschema specification>, an optional <temporary set specifications>, and one or more <procedure>s.
2. An application program is a segment of executable code, possibly consisting of multiple sub-programs. A single <module> is associated with an application program during its execution. An application program shall be associated with at most one <module>. The manner in which this association is specified, including the possible requirement for execution of some implementor-defined statement, is implementor-defined.

4.9 Procedures

1. A <procedure> consists of a <procedure name>, a sequence of <parameter declaration>s, and a sequence of <NDL statement>s.
2. An application program associated with a <module> may reference the <procedure>s of that <module> by a "call" statement that specifies the <procedure name> of the <procedure> and supplies a sequence of parameter values corresponding in number and in <data type> to the <parameter declaration>s of the procedure. A call of a <procedure> causes the sequence of <NDL statement>s that it contains to be executed.
3. The <subschema specification> of a <module> specifies the <subschema> that defines the records and sets that can be referenced by the <module>.
4. The <temporary set specifications> of a <module> defines temporary <set type>s that may be referenced by the <module>.

[ISO page 12]

This publication is a Draft for Development and is therefore not to be regarded as a British Standard. See page 2.

Documents

Document Sharing Hub

4.10 Standard Programming Languages

1. This standard specifies the actions of <procedure>s in <module>s when those <procedure>s are called by programs that conform to specified standard programming languages. The terms "standard COBOL program", "standard FORTRAN program", "standard Pascal program", and "standard PL/I program" refer to programs that meet the conformance criteria of ISO 1989, ISO 1539, ISO 7185, and ISO 6160, respectively. Note: See Clause 2, "References" on page 3.

4.11 Temporary Sets

1. A <temporary set> is an occurrence of a temporary <set type>. A temporary <set type> is declared by a <temporary set specification> in a <module>.
2. For each <temporary set specification> in a <module>, an occurrence of the corresponding temporary <set type> is created when the <module> is initiated and destroyed when the <module> is terminated.

4.12 Parameters

1. A parameter is declared in a <procedure> by a <parameter declaration>. The <parameter declaration> specifies whether the parameter is an elementary data item or an array of data items, and specifies the <data type> of its values. A parameter either assumes or supplies the value or values of the corresponding argument in the call of that procedure.

4.12.1 STATUS Parameter

1. The STATUS parameter is a special parameter of <length> 5 characters. If such a parameter is declared in a <procedure>, then its value is set to a status code that either indicates that a call of the <procedure> was successfully completed or that identifies an exception condition that occurred during execution of the <procedure>.

4.12.2 TEST Parameter

1. The TEST parameter is a special parameter of <length> 1 character. If a test statement is executed in a <procedure> containing this parameter, then the value of the parameter is set to "1" if the test is true, and to "0" if the test is false.

4.12.3 RECORD Parameter

1. The RECORD parameter is a special parameter of <length> 30 characters. If such a parameter is declared in a <procedure>, then it is set to the <record name> of the record identified by the <session cursor> of the <session state>.

4.13 Session

1. A database operation is the execution of an <NDL statement>.
2. A session is the sequence of database operations performed during the execution of an application program associated with a <module>.

3. A <session state> is an ephemeral object associated with a session. A <session state> is created prior to the first database operation in a session and is destroyed after the last database operation in the session.
4. A <session state> consists of <cursors>, <temporary sets>, and a <ready list>. The <cursors> include: A <session cursor> to identify the current session record, a <record cursor> to identify the current record of each <subschema> <record type>, and a <set cursor> to identify the owner record and the current member record in the current set of each <subschema> <set type>. The <temporary sets> maintain the contents of each <temporary set> defined by the associated <module>. The <ready list> maintains the <ready specification>s for each <record type> that has been activated by a <ready statement>.
5. The <session state> is available to an application only through <NDL statement>s. The contents of the <session state> are modified by the database management system upon execution of <NDL statement>s in a <procedure> of its associated <module>. The data manipulation language includes <NDL statement>s to find and compare <database key>s and to modify the contents of the <ready list> and each <temporary set>.

4.14 Transactions

1. A transaction is a sequence of operations, including database operations, that is atomic with respect to recovery and concurrency. Transactions terminate with a <commit statement> or a <rollback statement>. If a transaction terminates with a <commit statement>, then all changes made to the database by that transaction are made accessible to all concurrent transactions. If a transaction terminates with a <rollback statement>, then all changes made to the database by that transaction are canceled. Committed changes cannot be canceled. Changes made to the database by a transaction can be perceived by that transaction, but cannot be perceived by other transactions until that transaction terminates with a <commit statement>.
2. The execution of concurrent transactions is guaranteed to be serializable. A serializable execution is defined to be an execution of the operations of concurrently executing transactions that produces the same effect as some serial execution of those same transactions. A serial execution is one in which each transaction executes to completion before the next transaction begins.
3. The execution of an <NDL statement> within a transaction has no effect on the database or <session state> other than the effect stated in the General Rules for that <NDL statement>. Together with serializable execution, this implies that all read operations are reproducible within a transaction, except for changes explicitly made by the transaction itself.

4.15 Integrity Constraints

1. Integrity constraints restrict the valid states of the database. Integrity constraints may be defined for <record type>s or <set type>s, and for component constraints between owner and member <record type>s.
2. Integrity constraints are checked after execution of each <NDL statement>. If the objects associated with the integrity constraint do not satisfy the constraint, then the <NDL statement> has no effect on the database, and the STATUS parameter is set to indicate the specified exception.

[ISO page 14]

4.15.1 Check Conditions

1. A check condition is an expression that shall be satisfied by the values of a record when its status in the database is altered. The expression is specified by a <condition> of a <record check clause> of a <record type> or a <member check clause> of a <set type>.

4.15.2 Default Values

1. A default value is a value to be assumed by all component occurrences in the absence of an explicit value supplied by a <procedure>. A default value is specified by a <literal> in a <default clause> of a <component type>.

4.15.3 Uniqueness Constraints

1. A uniqueness constraint is a specification that no two records of a given <record type>, or no two member records of a set of a given <set type>, may occur in the database with identical values for specified components. Uniqueness constraints are specified by <record uniqueness clause>s in a <record type>, or by <member uniqueness clause>s in a <member clause> of a <set type>.

4.15.4 Set Ordering Criteria

1. A set ordering specifies the sequential ordering of member records in a set. The ordering is one of the following:
 - a. Sorted by component values.
 - b. Chronological or reversed chronological.
 - c. Relative positioning as next or prior to a given member record.
 - d. Implementor-defined.
2. Set ordering criteria are specified by the <order clause> of a <set type>, and if the order is sorted, by the <key clause> of each <member clause> contained in that <set type>.

4.15.5 Set Membership

1. Set membership specifies the insertion and retention modes of a record for each set in which it occurs as a member record. Each member <record type> has an insertion mode and a retention mode.
2. Insertion is either automatic, structural, or manual. If insertion is manual, then records shall be inserted as member records in a set only by an explicit <connect statement>. If insertion is automatic or structural, then each record of a member <record type> becomes a member of some set of that <set type> when it is initially stored in the database. If insertion is automatic or manual, then the owner record is identified by the application. If insertion is structural, then the owner record is selected by the database management system to have values of specified data items equal to those of the record to be inserted. The insertion mode is specified by an <insertion clause> in each <member clause> of the <schema>.
3. Retention is either fixed, mandatory, or optional. If retention is fixed, then a record, having once become a member record of some set, remains a member of that set until it is erased from the database. If retention is mandatory, then a record, having once become a member record of a set, remains a member of some set of the same <set type> until it is erased from the database. If retention is optional, then a record, having

once become a member record of some set, need not remain a member of that set or of any set of that <set type>. The retention mode is specified by a <retention clause> in each <member clause> of the <schema>.

5. Common Elements

5.1 <condition>

Function

Specify an expression that shall be evaluated as either *true* or *false*.

Format

<condition> ::=
 <alternative> [{ OR <alternative>}...]

<alternative> ::=
 <simple condition> [{AND <simple condition>}...]

<simple condition> ::=
 <subcondition>
 | <negated subcondition>
 | <relation condition>

<subcondition> ::= (<condition>)

<negated subcondition> ::=
 NOT (<condition>)

<relation condition> ::=
 <operand> <relation> <operand>

<relation> ::=
 < | <= | = | >= | > | <>

Syntax Rules

1. If the first <operand> in a <relation condition> specifies an array, then the second <operand> in that <relation condition> shall not specify an array.
2. If the type of the first operand is character string, then the type of the second operand shall be character string. If the type of the first operand is exact numeric, then the type of the second operand shall be exact numeric or approximate numeric. If the type of the first operand is approximate numeric, then the type of the second operand shall be exact numeric or approximate numeric.

General Rules

1. A <condition> is *true* if any <alternative> contained in it is *true*; otherwise, it is *false*.
2. An <alternative> is *true* if every <simple condition> contained in it is *true*; otherwise, it is *false*.
3. A <simple condition> that immediately contains a <subcondition> is *true* if the <subcondition> is *true*; otherwise, it is *false*.

A <simple condition> that immediately contains a <negated subcondition> is *true* if the <condition> immediately contained in the <negated subcondition> is *false*; otherwise, it is *false*.

A <simple condition> that immediately contains a <relation condition> is *true* if the <relation condition> is *true*; otherwise, it is *false*.

4. The first (second) operand of a <relation condition> is the value of the first (second) <operand> in that <relation condition>.
5. The <relation> of a <relation condition> specifies a comparison to be made between the two operands of the <relation condition>. The comparisons specified by the alternative <relation>s are as follows:

<relation>	Comparison
<	less than
<=	less than or equal
=	equal
>=	greater than or equal
>	greater than
<>	not equal

6. If neither operand of a <relation condition> is an array, then the <relation condition> is *true* if the specified comparison of the two operands is *true*; otherwise, the <relation condition> is *false*. If one of the operands of a <relation condition> is an array, then the <relation condition> is *true* if the specified comparison of every value in the array sequence with the other operand is *true*; otherwise, the <relation condition> is *false*.
7. Case:
 - a. If operands are of exact numeric or approximate numeric type, then the comparison is performed according to their numeric values.
 - b. If the operands are of character string type, then:
 - 1) If the lengths of the two operands are not equal, then the shorter operand is considered to be extended on the right with space characters.
 - 2) The comparison is performed from left to right, comparing the individual characters of the operands. If all such individual comparisons are equal, then the two operands are equal; otherwise, the result of the comparison of the operands is defined to be the result of the leftmost non-equal comparison of the individual characters.

- 3) The comparison of individual characters is implementor-defined.
8. In general rules, the terms "less than", "greater than", "equal", and "not equal" refer to the corresponding relational operators.

5.2 <operand>

Function

Specify a value.

Format

<operand> ::=
 <component identifier>
 | <component view identifier>
 | <parameter identifier>
 | <literal>

Syntax Rules

1. Case:
 - a. If an <operand> is contained in a <schema>, then it shall not be a <parameter identifier> or a <component view identifier>.
 - b. If an <operand> is contained in a <module>, then it shall not be a <component identifier>.
2. The data type of an <operand> is the data type of the <component identifier>, <component view identifier>, <parameter identifier>, or <literal>.

General Rules

1. The value of an <operand> is the data item or array referenced by the <component identifier>, <component view identifier>, or <parameter identifier>, or the value of the <literal>.

[ISO page 20]

This publication is a Draft for Development and is therefore not to be regarded as a British Standard. See page 2.

5.3 <identifier>

Function

Specify names for parameters in a <procedure> or objects in the database.

Format

<identifier> ::=
 <regular identifier> | <escape identifier>

<regular identifier> ::=
 <upper case letter>
 [{{<underscore>} <letter or digit>}...]

<underscore> ::= _

<letter or digit> ::=
 <upper case letter> | <digit>

<letter> ::=
 <upper case letter> | <lower case letter>

<upper case 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

<lower case 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

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

<escape identifier> ::=
 '<escape identifier character representation>...'

<escape identifier character representation> ::=
 <escape identifier character>
 | <apostrophe representation>

<escape identifier character> ::=
 See Syntax Rule 3.

<apostrophe representation> ::= "

Syntax Rules

1. A <regular identifier> shall contain at most 18 characters. An <escape identifier> shall contain at most 18 <escape identifier character representation>s.
2. A <regular identifier> shall not be the same sequence of characters as a <key word> (see Clause 5.8, "Comments, Spaces, and Key Words" on page 29).
3. An <escape identifier character> is any character in the implementor-defined character set other than the apostrophe character (').
4. Each <apostrophe representation> in an <escape identifier> represents a single apostrophe character.
5. If an <escape identifier> (apart from the enclosing apostrophes) conforms to the format of a <regular identifier>, and if it is not a <key word>, then it is equivalent to the <regular identifier> formed by removing the enclosing apostrophes.

General Rules

None.

[ISO page 22]

This publication is a Draft for Development and is therefore not to be regarded as a British Standard. See page 2.

5.4 <literal>

Function

Specify a literal value.

Format

```
<literal> ::=
    <character string literal>
  | <numeric literal>
```

```
<character string literal> ::=
    "<character representation>..."
```

```
<character representation> ::=
    <non-quote character>
  | <quote representation>
```

```
<non-quote character> ::=
    See Syntax Rule 1.
```

```
<quote representation> ::=
    ""
```

```
<numeric literal> ::=
    <exact numeric literal>
  | <approximate numeric literal>
```

```
<exact numeric literal> ::=
    [+|-] { <unsigned integer>[.<unsigned integer>]
           | <unsigned integer>.
           | .<unsigned integer>}
```

```
<approximate numeric literal> ::=
    <mantissa>E<exponent>
```

```
<mantissa> ::= <exact numeric literal>
```

```
<exponent> ::= <signed integer>
```

```
<signed integer> ::= [+|-] <unsigned integer>
```

```
<unsigned integer> ::=
    <digit>...
```

Syntax Rules

1. A <non-quote character> is any character in the implementor-defined character set other than the double quote mark character (").
2. The data type of a <character string literal> is character string. The length of a <character string literal> is the number of <character representation>s that it contains. Each <quote representation> in a <character string literal> represents a single quotation mark character in both the value and the length of the <character string literal>.
3. An <exact numeric literal> without a decimal point (.) has an implied decimal point following the last <digit>.
4. The data type of an <exact numeric literal> is exact numeric. The precision of an <exact numeric literal> is the number of <digit>s that it contains. The scale of an <exact numeric literal> is the number of <digit>s to the right of the decimal point.
5. The data type of an <approximate numeric literal> is approximate numeric. The precision of an <approximate numeric literal> is the precision of its <mantissa>.

General Rules

1. The value of a <character string literal> is the sequence of characters that it contains.
2. The numeric value of an <unsigned integer>, <signed integer>, or <exact numeric literal> is derived from the normal mathematical interpretation of signed positional decimal notation.
3. The numeric value of an <approximate numeric literal> is the product of the exact numeric value represented by the <mantissa> with the number obtained by raising the number 10 to the power represented by the <exponent>.

[ISO page 24]

This publication is a Draft for Development and is therefore not to be regarded as a British Standard. See page 2.

5.5 <data type>

Function

Specify a data type.

Format

```

<data type> ::=
    <character string type>
    | <exact numeric type>
    | <approximate numeric type>

<character string type> ::=
    CHARACTER [<length>]

<exact numeric type> ::=
    FIXED <precision> [<scale>]
    | NUMERIC <precision> [<scale>]
    | INTEGER

<approximate numeric type> ::=
    FLOAT <precision>
    | REAL
    | DOUBLE PRECISION

<length> ::= <unsigned integer>

<precision> ::= <unsigned integer>

<scale> ::= <signed integer>
  
```

Syntax Rules

1. The value of an <unsigned integer> that is a <length> or a <precision> shall be greater than 0.
2. If a <length> is omitted, then it is assumed to be 1. If a <scale> is omitted, then it is assumed to be 0.

General Rules

1. A <data type> specifies a class of data item values.
2. A data item of <data type> CHARACTER has a character string value with length equal to the value of the specified <length>.
3. A data item of <data type> FIXED has an exact numeric value with scale equal to the value of the specified <scale>, and with precision greater than or equal to the value of the specified <precision>.

[ISO page 25]

4. A data item of <data type> NUMERIC has an exact numeric value with precision and scale equal to the value of the specified <precision> and <scale> respectively. If the precision is P and the scale is S, then the absolute value of any non-zero NUMERIC data item shall be greater than or equal to $1E-S$ and less than or equal to $1E(P-S)-1E-S$.
5. A data item of <data type> INTEGER has an exact numeric value with a scale of 0 and with implementor-defined precision.
6. A data item of <data type> FLOAT has an approximate numeric value with precision equal to or greater than the value of the specified <precision>
7. A data item of <data type> REAL has an approximate numeric value with implementor-defined precision.
8. A data item of <data type> DOUBLE PRECISION has an approximate numeric value with implementor-defined precision that is greater than the implementor-defined precision of REAL.

[ISO page 26]

This publication is a Draft for Development and is therefore not to be regarded as a British Standard. See page 2.

Documents

Document Sharing Hub

5.6 <occurs clause>

Function

Define an array by specifying its <extents>.

Format

<occurs clause> ::=
OCCURS <extents>

<extents> ::= <unsigned integer>...

Syntax Rules

1. The value of each <unsigned integer> in the <extents> shall be greater than 0.

General Rules

1. An <occurs clause> specifies an array of data items. The product of the values of the <unsigned integer>s in the <extents> specifies the number of data items contained in each occurrence of the array.

[ISO page 27]

5.7 <subscripts>

Function

Specify a data item of an array.

Format

<subscripts> ::=
 (<operand>...)

Syntax Rules

1. The *contextual* <occurs clause> of a <subscripts> is specified in the Syntax Rules of the production symbol that immediately contains the <subscripts>.
2. The number of <operand>s shall be equal to the number of <unsigned integer>s in the <extents> of the contextual <occurs clause>.
3. Case:
 - a. If an <operand> is a <literal>, then it shall be an <unsigned integer> whose value is positive and is not greater than the value of the corresponding <unsigned integer> of the <extents> of the contextual <occurs clause>.
 - b. If an <operand> is not a <literal>, then its data type shall be exact numeric with a scale of 0.

General Rules

1. Let d be the number of <operand>s in the <subscripts>.
2. For i from 1 to d , let S_i be the value of the i -th <operand> in the <subscripts>.
3. For i from 1 to d , let E_i be the value of the i -th <unsigned integer> in the <extents> of the contextual <occurs clause>.
4. For i from 1 to d , if S_i is less than 1 or S_i is greater than E_i , then raise exception *array reference: subscript out of bounds*.
5. Let M_d be 1. For i from 1 to $(d-1)$, let M_i be the product of the terms E_j , for j from $(i+1)$ to d .
6. Let J be the sum of the terms $M_i(S_i-1)$, for i from 1 to d .
7. The value of the <subscripts> is $J+1$.

[ISO page 28]

This publication is a Draft for Development and is therefore not to be regarded as a British Standard. See page 2.

Documents

Document Sharing Hub

5.8 Comments, Spaces, and Key Words

Function

Specify lexical units.

Format

<separator> ::=
 { <comment> | <space> | <newline> }...

<comment> ::=
 (*[<character>...]*)

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

<special character> ::=
 See Syntax Rule 1.

<space> ::=
 space character

<newline> ::=
 implementor-defined end-of-line indicator

<word> ::=
 <key word> | <identifier> | <literal>

<key word> ::=
 ABSOLUTE | ALL | AND
 | AS | ASCENDING | AUTOMATIC
 | CASCADE | CHARACTER | CHECK
 | COBOL | COMMIT | CONNECT | CONTAINS
 | DEFAULT | DESCENDING | DISCONNECT
 | DOUBLE | DUPLICATES
 | EMPTY | ERASE | EXCLUSIVE
 | FIND | FINISH | FIRST | FIXED
 | FOR | FORTRAN | FLOAT | FROM | FULL
 | GET
 | IN | INSERTION | INTEGER | ITEM
 | KEY
 | LANGUAGE | LAST
 | MANDATORY | MANUAL | MEMBER | MODIFY | MODULE
 | NEXT | NOT | NULL | NULLIFY | NUMERIC
 | OCCURS | OF | OPTIONAL | OR | ORDER | OWNER
 | PARTIAL | PASCAL | PLI
 | PRECISION | PRIOR | PROCEDURE | PROHIBITED | PROTECTED
 | READY | REAL | RECONNECT | RECORD | RELATIVE | RENAMED
 | RETAIN | RETENTION | RETRIEVE | ROLLBACK

| SCHEMA | SESSION | SET | SHARED | SORTED
 | STATUS | STORE | STRUCTURAL | SUBSCHEMA | SYSTEM
 | TEST | TO | TYPE
 | UNIQUE | UPDATE
 | WHERE | WITH

Syntax Rules

1. A <special character> is any character in the implementor-defined character set other than a <digit> or a <letter>. If the implementor-defined end-of-line indicator (<newline>) is a character, then it is also excluded from <special character>.
2. A terminal production of <schema>, <subschema>, or <module> consists of a sequence of <word>s and symbols optionally separated by <separator>s. A <word> shall not be immediately followed by another <word> without an intervening <separator>.
3. The string "<character>..." within a <comment> shall not contain the substring "*)".

General Rules

None.

6. Schema Definition Language

6.1 <schema>

Function

Define the logical structure of a database.

Format

<schema> ::=
 <schema name clause>
 [{ <record type> | <set type>}...]

Syntax Rules

None.

General Rules

None.

6.2 <schema name clause>

Function

Name a <schema>.

Format

<schema name clause> ::=
 SCHEMA <schema name>

<schema name> ::= <identifier>

Syntax Rules

1. The <schema name> shall be different from the <schema name> of any other <schema> in the same environment. The concept of environment is implementor-defined.

General Rules

1. A <schema name clause> defines the <identifier> to be a <schema name> that designates the containing <schema>.

6.3 <record type>

Function

Define a <record type>.

Format

```
<record type> ::=  
    <record name clause>  
    [ { <record uniqueness clause>  
        | <component type>  
        | <record check clause>}...]
```

Syntax Rules

None.

General Rules

None.

6.4 <record name clause>

Function

Name a <record type>.

Format

<record name clause> ::=
 RECORD <record name>

<record name> ::= <identifier>

Syntax Rules

1. The <record name> shall be different from the <record name> of any other <record name clause> in the containing <schema>.

General Rules

1. A <record name clause> defines the <identifier> to be a <record name> that designates the containing <record type>.

[ISO page 34]

This publication is a Draft for Development and is therefore not to be regarded as a British Standard. See page 2.

6.5 <record uniqueness clause>

Function

Specify a uniqueness constraint for occurrences of a <record type>.

Format

<record uniqueness clause> ::=
 UNIQUE <component identifier>...

Syntax Rules

1. If the <component name> of a <component identifier> designates an array, then that <component identifier> shall contain <subscripts>.
2. A given <component identifier> shall be contained at most once in a single <record uniqueness clause>.
3. The contextual <record type> of each <component identifier> in a <record uniqueness clause> is the containing <record type>.

General Rules

1. When the <component identifier>s of a <record uniqueness clause> are referenced for a record occurrence, the contextual <database key> for those <component identifier>s is the <database key> of that record occurrence.
2. A <record uniqueness clause> is violated if the database contains two occurrences of the containing <record type> in which the values of each of the components designated by the <component identifier>s in the <record uniqueness clause> in the first record occurrence are *equal* to the values of the corresponding components in the second record occurrence.

6.6 <component type>

Function

Define a <component type>.

Format

<component type> ::=
 <component name clause>
 <data type>
 [<occurs clause>]
 [<default clause>]

Syntax Rules

None.

General Rules

None.

[ISO page 36]

This publication is a Draft for Development and is therefore not to be regarded as a British Standard. See page 2.

6.7 <component name clause>

Function

Name a <component type>.

Format

<component name clause> ::= ITEM <component name>

<component name> ::= <identifier>

Syntax Rules

1. The <component name> shall be different from the <component name> of any other <component name clause> in the containing <record type>.

General Rules

1. A <component name clause> defines the <identifier> to be a <component name> that designates the containing <component type>.

6.8 <default clause>

Function

Specify a default value for a <component type>.

Format

```
<default clause> ::=
    DEFAULT <literal>
```

Syntax Rules

1. The subject <data type> of a <default clause> is the <data type> of the containing <component type>.
2. Case:
 - a. If the subject <data type> defines character string values, then the <literal> shall be a <character string literal>. The length of the <character string literal> shall not be greater than the value of the <length> of the subject <data type>.
 - b. If the subject <data type> defines exact numeric values, then the <literal> shall be an <exact numeric literal>. The number of significant digits to the right of the decimal point shall not be greater than the <scale> of the subject <data type>. The number of significant digits to the left of the decimal point shall not be greater than P-S, where P and S are respectively the <precision> and <scale> of the subject <data type>.
 - c. If the subject <data type> defines approximate numeric values, then the <literal> shall be an <approximate numeric literal> or an <exact numeric literal>. The number of significant digits shall not be greater than the <precision> of the subject <data type>.

General Rules

1. When an occurrence of the containing <record type> is stored by a <store statement>, if the corresponding <record view> does not contain a <component view> whose <component view name> designates the containing <component type> and a <default clause> is not specified, then the database control system shall report an error and shall not change the database.
2. When an occurrence of the containing <record type> is stored by a <store statement>, if the corresponding <record view> does not contain a <component view> whose <component view name> designates the containing <component type> and a <default clause> is specified, then:

Case:

- a. If the <literal> is an <exact numeric literal>, then the value of each data item contained in the occurrence of the containing <component type> is the exact numeric value of the <literal>.

[ISO page 38]

This publication is a Draft for Development and is therefore not to be regarded as a British Standard. See page 2.

- b. If the <literal> is an <approximate numeric literal>, then the value of each data item contained in the occurrence of the containing <component type> is the approximate numeric value of the <literal>.
- c. If the <literal> is a <character string literal> and the length of the <literal> is equal to the length of the subject <data type>, then the value of each data item contained in the occurrence of the containing <component type> is the <literal>.
- d. If the <literal> is a <character string literal> and the length of the <literal> is less than the length of the subject <data type>, then the value of each data item contained in the occurrence of the containing <component type> is the <character string literal>, extended on the right to the length of the subject component with space characters.

6.9 <record check clause>

Function

Specify a validity condition for occurrences of a <record type>.

Format

<record check clause> ::=
CHECK <condition>

Syntax Rules

1. The contextual <record type> of each <component identifier> in the <condition> is the containing <record type>.

General Rules

1. When the <condition> of a <record check clause> is evaluated for a record occurrence, the contextual <database key> for any <component identifier> in the <condition> is the <database key> of that record occurrence.
2. A <record check clause> is violated if the database contains a record occurrence of the containing <record type> for which the <condition> is *false*.