A Relational Schema Description and Manipulation Facility

YUKSEL UCKAN
Systems Analysis Department, Miami University, Oxford, OH 45056, USA

This paper presents the schema description facility of DDS, an experimental relational DBMS designed and partially implemented by the present investigator. The proposed schema description language and the DDS/DBMS software which supports it are primarily designed to satisfy requirements of the database administrator (DBA). The language includes a comprehensive schema description capability for a relational database environment, and can also be used in the generation of the DDS/DBMS software. It is a block-structured, English-like, user-friendly general-purpose schema description language. The architecture and logical structure of DDS/DBMS as well as the DDS system-generation sequence are also explored in the paper.

1. INTRODUCTION

Following the introduction of the relational model of data by Codd,[1] the last decade has witnessed a heavy concentration of research and development on relational database languages and database management systems. As a result, a large number of data languages became available among which may be cited Alpha, Quel, Ingress and Damas,[2,3,4,5] which are based on the first-order predicate calculus, MacAims and IS/1[6,7] of the relational algebra origin, Square, Sequel and System R[8,9,10] which are set-oriented, and Query-by-Example,[11] which resembles Sequel, but has a two-dimensional syntax.

These efforts were quite successful towards the goal of developing reasonably comprehensive, easy to use, non-procedural and relationally complete[3] database query languages. Most also included storage operations which can be used for database manipulation or maintenance purposes. However, we observe an almost complete neglect of schema description languages in recent research. This is partly because the query formulation aspects of database systems are intellectually more challenging and stimulating, and partly because the database administrator (DBA) who would most likely be the person to need a schema description language, can take care of his problems more easily compared to the average database user due to his more technical background. Nevertheless, the need for a simple and effective relational schema description and manipulation language exists.

This paper presents results of a research effort striving to fill this gap. It concentrates on the schema description and manipulation facility of a relational database management system (DDS: *D*ynamic *D*ata Base *S*ystem) which is proposed, designed and partially implemented by the present investigator. DDS is an experimental DBMS and a data language based on the relational model of data. The designed data language has a linear syntax and is block-structured, English-like and user-friendly. It comprises a data definition (DDL), a data manipulation (DML), and a query formulation (QFL) language subset. This paper is one of the products of a continuing research effort related to the design and implementation of DDS/DBMS.

The proposed data language is stand-alone, and it can be used in producing highly readable data definition, database manipulation and query formulation programs. The language design is based on the most elementary database concepts and can be easily learned and used by those with little or no mathematical or technical background. All three language subsets include features which can satisfy almost every need of a database user.

This paper discusses DDS/DDL and the relevant features of the DDS/DBMS design. DDS/DDL is designed primarily to satisfy the requirements of DBA. It can be used in one-shot or step-by-step definition of database relations, and conceptual schema updates which include expanding a relation, changing structure and/or key attributes of a relation, deleting a relation, and changing type, field length or security code of an attribute existing in a relation. In addition, DDS/DDL statements can be formulated to specify a new relation by joining two existing database relations, to prescribe structural similarity of a transient relation to a permanent database relation, and to render permanent a transient relation. While such operations are permitted, the integrity of the database is carefully maintained by DDS/DBMS. Finally, DBA can utilise DDS/DDL in defining 10 major DDS/DBMS system control structures which are designed as relations. In other words, DDS/DBMS has a self-definition capability, and DBA can take advantage of this feature in the system-generation phase for DDS/DBMS.

Section 2 of this paper describes the basic features of DDS/DDL, whose syntax is presented in Appendix A. It also contains examples which demonstrate schema definition and manipulation capabilities of the proposed language. In Sections 3 and 4 we present the highlights of the DDS/DBMS design and major data structures incorporated into the design. Also in Section 4, we discuss the basic design assumptions and restrictions and interactions among the DBMS software and the language components. Section 5 is devoted to an exploration of the self-definition capability of DDS/DBMS and the problem of system generation. Sections 3, 4 and 5 together provide a clear and concise algorithmic description of the relevant features of the proposed DBMS design.

2. DDS SCHEMA DESCRIPTION LANGUAGE

DDS/DDL includes six language constructs which can be used in describing schema definition and manipulation operations in terms of English-like, easy-to-follow DDS program blocks. Syntax descriptions of DDS/DDL constructs are presented in Appendix A. Their functions are summarized below:

DEFINE RELATION: assigns a name to a relation and defines its tuple structure and key attributes. In case key attributes are not specified, the first attribute appearing in the tuple structure is, by default, the key attribute. A somewhat different form of this block, namely, DEFINE RELATIONS/USING, defines a new relation by joining two existing database relations over a common attribute and projecting the join over a set of attributes specified in the tuple structure clause. As the join relation is permanent by intent, in order to preserve database integrity, DDS/DBMS will delete both the definitions and the contents of two source relations.

DEFINE ATTRIBUTE: specifies type (either alphanumeric or numeric), field length, and optionally, security code of each attribute in a relation. A DEFINE ATTRIBUTE block must contain attribute definitions for relations which have already been structurally described through a DEFINE RELATION block. All attribute definitions for a relation may be accomplished through a single DEFINE ATTRIBUTE block, or DBA may choose to use several DEFINE ATTRIBUTE blocks at different times. DDS/DBMS accepts step-by-step attribute definitions; however, it should be noted that unless all domains of a relation are defined, the software will not permit any other database operation on that relation.

REDEFINE RELATION: is a block designed to change the name of a relation, its tuple structure (change existing structure, add new attributes, or delete attributes) and its key attributes. It also permits deletion of an entire relation. As the result of this statement, both the schema and the content of the relation will be updated.

REDEFINE ATTRIBUTE: can be used to change name, type, field length and/or security code of any attribute which has already been defined. It results in schema update, and in case field length of an attribute is compressed, also in content update of the relevant relations.

SIMILAR: is a statement needed to specify structural similarity of a transient relation to a permanent database relation.

PERMANENT: this statement transforms a transient relation to a permanent relation. Both SIMILAR and PERMANENT statements are basically of DDL type; however, they are needed and used in certain DML operations, such as LOAD, ADD, COPY and COMBINE.[12, 14]

From a program structure point of view, a DDS/DDL program block has no syntactical constraint; it may contain any number of the basic six language constructs discussed above. This is because the DDS language processor has been designed to pre-process a given program block as a whole and go through all necessary structural modifications on the basis of the semantic relationships inherent in the subschema under consideration. For example:

```
BEGIN: DEFINE ATTRIBUTE:
  a ALPHANUM (6);
  b NUMERIC (5)
        END.
        DEFINE RELATION:  r;
          TUPLE STRUCTURE:  a,b;
          KEY:  a
        END.
END.
```

is seemingly incorrect as it defines attributes of r before its structure is prescribed. However, DDS/DBMS will interchange the two blocks in the program before it attempts to process it. Therefore, DDS/DDL is completely non-procedural from the user's viewpoint.

Table 1 contains some examples of schema definition and manipulation programs written using the proposed language. All basic features of DDS/DDL are exemplified in Table 1. The simplicity and self-documentation aspects of the language should be noted. It should also be noted that a DDS/DDL program not only changes the existing database structure, but it will also cause DDS/DBMS to approriately update certain system control structures. The major system control structures are discussed in Section 4 of this paper.

3. DDS/DBMS SYSTEM ARCHITECTURE

Fig. 1 shows the system architecture for DDS/DBMS. The system consists of five principal components:

(*a*) DDS data language program,
(*b*) DDS/DBMS software,
(*c*) ten system control structures,
(*d*) permanent (corporate) database,
(*e*) transient database.

Fig. 1 also shows the interrelationship or communication paths among these system components. DDS/DBMS software is a collection of programs written in a high-level implementation language; a partial implementation exists in PL/1. A complete implementation in FORTRAN is presently under way. Fig. 1 indicates only functionally significant software modules. During the actual implementation, a larger number of program modules is emerging. It should be noted that Fig. 1 does not attempt to indicate the interrelationships among the DBMS software modules.

The permanent database, as well as the transient one, is relational. The transient database contains copies of database relations, or some horizontal or vertical subsets of such relations. It is embedded in the design in order to allow for certain DML operations while preserving database integrity.

In addition to some auxiliary data structures which are implemented in the DDS/DBMS software, ten additional data structures are required by the design. These data structures are referred to as system control structures. They are conceived as relations and are actually part of the DDS database. There is a complex set of interrelationships between the DDS system control structures and the DDS/DBMS software modules. The next section discusses the system control structures and these interrelationships, thereby shedding more light on the overall logic of DDS/DBMS.

4. DDS SYSTEM CONTROL STRUCTURES

Table 2 shows ten DDS system control structures, their tuple structures and basic functions. The underlined attributes in the tuple structures are the key attributes.

SYS 01 contains authorised user definitions and user authorisation codes as assigned by DBA. A user with authorisation code of 1 is permitted access to all database relations and attributes; the user is also authorised to specify any DDS data language operation. On the other hand, a user whose SYS 01 authorisation code is, say *x*,

Table 1. DDS/DDL program examples

DDS/DDL program	Meaning
1. BEGIN: USER-ID = 55555, USER-NAME = SMITH. DEFINE RELATION: r; TUPLE STRUCTURE: a, b, c; RELATION: t; TUPLE STRUCTURE: a, d, e; KEY: a, d END. END.	Defines a relation r with three attributes a, b, and c, where a is key; also, defines a relation t with three attributes a, d, and e, where a‖ d is the key.
2. BEGIN: BEGIN USER-ID = 55555, USER-NAME = SMITH. DEFINE ATTRIBUTE: a ALPHANUM (6) b NUMERIC (5) 2; c NUMERIC (4) 1; d NUMERIC (2) END. END.	Defines domains for attributes a, b, c of r, and attributes a, d of t. Now, r is completely defined and is user-accessible; t, however, is not.
3. BEGIN: USER-ID = 66666, USER-NAME = GEORGE. DEFINE ATTRIBUTE: e ALPHANUM (10) 2 END. END.	Completes domain definition for relation t.
4. BEGIN: BEGIN: USER-ID = 66666, USER-NAME = GEORGE. DEFINE RELATION: tt USING r AND t; TUPLE STRUCTURE: a, d, b, e; KEY: a, d END. END.	Defines a new relation tt by joining r and t over the common domain a, and projecting it over a, d, b, e with a and d as keys. r and t will no longer exist in the database.
5. BEGIN: USER-ID = 55555, USER-NAME = SMITH. REDEFINE RELATION: r as rx; TUPLE STRUCTURE: a, b, c, f END. END.	Changes the name of relation r to rx; also, changes its tuple structure from (a, b, c) to (a, b, c, f) where f is a new attribute to be defined before rx can be used. r will no longer exist in the database.
6. BEGIN: USER-ID = 55555, USER-NAME = SMITH. REDEFINE RELATION: rx AS; KEY: b END. END.	Changes the key of relation rx from a to b, provided b is an attribute with the property of unique identification.
7. BEGIN: USER-ID = 77777, USER-NAME = JOHN. REDEFINE ATTRIBUTE: a AS ax (8) END. END.	Changes the name of attribute a to ax; also, changes its field length from 6 (see Ex. 2) to 8.
8. BEGIN: USER-ID = 55555, USER-NAME = SMITH. RELATION tx IS SIMILAR TO t. RELATION tx IS PERMANENT. END.	Defines a relation tx which is similar to t; then copies t to tx, makes tx permanent and t transient.

will be restricted to certain database operations, and a subschema defined by attributes with a security code, y, satisfying $y \geqslant x$ (see SYS 04 format in Table 2). Thus, together with SYS 04, SYS 01 is used in maintaining database security.

DDS/DBMS software will produce a log entry for each DDS data language program run, be it successful, in error or blocked by the software. These log entries are accumulated in SYS 02 and can be used by DBA in extracting user profiles, evaluating user requirements and observing security and authorisation violations.

SYS 03 contains diagnostic error messages for syntax and/or semantic errors. It is used by the DDS software in producing and displaying error diagnostics which aid the user in program debugging.

SYS 04, SYS 05, SYS 06 and SYS 07 are relations needed to define the database schema. SYS 04 includes tuple structure and key attribute specifications for all permanent database relations and all system control structures. SYS 04 tuples also contain domain specifications for all attributes. This approach, coupled with a one-character-per-byte character string specification for all database tuples, permits dynamic schema definition for an arbitrary relation. It eliminates the necessity of

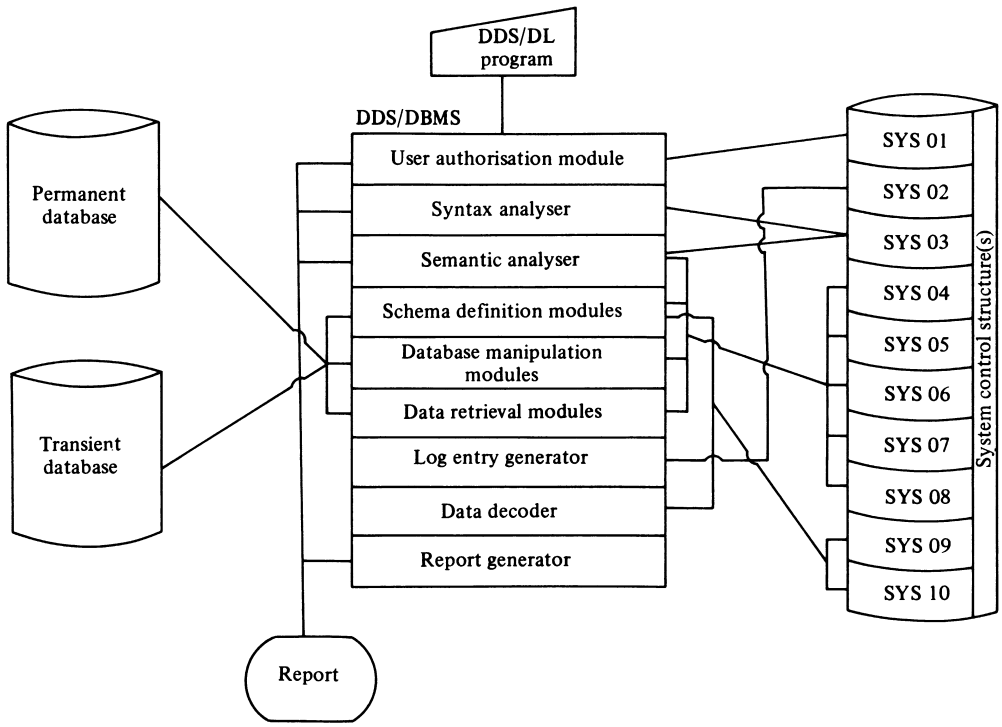

Figure 1. DDS/DBMS system architecture.

Table 2. DDS/DBMS system control structures

Name	Format	Function
SYS 01: User identification relation	(*user-id*,user-name, authorization-code)	Authorization control (system security)
SYS 02: System log relation	(*user-id*,*run-date*,*run-time*, statement-keyword,run-result relation-used-1,relation-used-2,relation-used-3)	Usage statistics, system security
SYS 03: Error diagnostics relation	(*diagnostic-message-code*, diagnostic-message)	User diagnostics aid
SYS 04: Schema definition relation	(*relation-name*,*attribute-name*, attribute-seq-no,seq-no-in-key, attribute-type,field-length, attribute-security-code)	schema definition, subschema generation, system security
SYS 05: Attribute-relation relation	(*attribute-name*,relation-name-1 relation-name-2,relation-name-3, relation-name-4)	Schema definition (attribute-name to relation-name conversion)
SYS 06: Schema definition control relation	(*relation-name*, degree of relation, definition-string)	Schema definition
SYS 07: Similar relation table	(*relation-name*, similar-relation-name-1,similar-relation-name-2)	Schema definition (database integrity)
SYS 05: Access path definition relation	(*attribute-name*,*attribute-value*, path-type, access-path, inversion-type,frequency-of-use, recent-use-date)	Direct conditional retrieval
SYS 09 Attribute-coding scheme table	(*attribute-name*,coding-scheme)	Data decoding (report generation)
SYS 10: Data decoding relation	(*coding-scheme*,*attribute-value*, open-description)	Data decoding (report generation)

changing format specifications in the DDS/DBMS software whenever a relation is to be defined, deleted or updated, and of recompiling the entire software. It should be noted that the DDS software accesses SYS 04 for complete schema description before it can access any database relation. Only the schema definition for SYS 04 need be software embedded.

SYS 05 permits extraction of a unique relation name given an attribute name, provided the attribute is non-key. This is possible due to the fact that the DDS database is

integrated, and is required because certain DDS/QFL statements allow user reference to relations, indirectly through attribute names. Content generation for SYS 05 can be algorithmically defined on the basis of SYS 04 content.

SYS 06 indicates whether a database relation is partially or completely defined. DDS/DDL permits step-by-step definition of a relation, and hence SYS 06 is included in the design. The software firstly accesses SYS 06 and, unless the relation specified is completely defined, prohibits access to the relation itself.

SYS 07 is used to specify transient database relations and to define their schema by associating them with permanent database relations.

SYS 08 is the system control structure which is designed to store access paths created by the software whenever a user specifies in a run a conditional retrieval operation using DDS/QFL or certain DML operations involving conditions. The access paths are retrieved and used in directly accessing tuples of database relations in subsequent DDS/QFL runs. This greatly enhances the retrieval capabilities of the system. DDS/DBMS maintains and, if necessary, deletes SYS 08 tuples throughout the lifespan of the system. This data struture and its behavioral characteristics are currently under further investigation.

SYS 09 and SYS 10 are used for data decoding purposes, following retrieval and before report generation, in case so requested by a DDS/QFL program, and if attributes retrieved have been encoded for data compaction.

As was mentioned before, to process a DDS data language program, the DDS software accesses and updates most of the DDS system control structures. In view of the functional specifications for DDS system control structures, the required operations on these structures for a given DDS program element can be prescribed. Table 3 relates DDS/DDL statements to DDS system control structures and indicates the required operations necessary for processing. For example, to DEFINE a RELATION, the DDS/DBMS software executes the following algorithm:

1. Access SYS 04 and fetch schema specification tuples for SYS 01, SYS 02, SYS 03, SYS 05 and SYS 06.
2. Using SYS 01 schema, access SYS 01 for user authorization control.
3. Analyse DDS/DDL program. If a syntax error exists, using SYS 03 schema, access SYS 03, fetch error message; display message and terminate.
4. Extract relation name and attribute names from the DDS/DDL program. Using SYS 06 schema, access SYS 06; if there exists a SYS 06 tuple for the relation, access SYS 03 to retrieve the appropriate error message, display it and terminate. If not, generate a SYS 06 tuple for the relation and add it to SYS 06.
5. Generate SYS 04 tuples for the relation and its attributes.
6. Access SYS 04 and add these tuples to SYS 04.
7. Generate SYS 05 tuples for each attribute of the relation.
8. Using SYS 05 schema, access SYS 05 and either add these tuples to SYS 05 or, if there already exist tuples in SYS 05 for a given attribute, update corresponding SYS 05 tuples.
9. Generate a log entry for the operation.
10. Using SYS 02 schema, access SYS 02 and add this tuple to it.
11. Terminate.

Hence, each row of Table 3 is a summary of the requirements of the indicated DDL operation on the system control structures, as defined by the corresponding software logic.

Following is a list of system requirements as implied by the above discussion:

(*a*) Schema definitions for all DDS system control structures must exist in SYS 04.

Table 3. DDS/DDL statements vs. DDS system control structures

DDS statement	SYS 01	02	03	04	05	06	07	08	09	10	Database relations
DEFINE RELATION	R	A	R*	R A	R, A or U	R A	—	—	—	—	—
DEFINE RELATION/ USING	R	A	R*	R A D	R U D*	R D	—	R D	—	—	R W D
DEFINE ATTRIBUTE	R	A	R*	R U	R	R U	—	—	—	—	—
REDEFINE RELATION/ DELETED	R	A	R*	R D	R, D or U	R D	R D*	R D*	—	—	R D
REDEFINE RELATION/ RELATION NAME	R	A	R*	R U	R U	R U	R U*	—	—	—	—
REDEFINE RELATION/ TUPLE STRUCTURE	R	A	R*	R U	R*, A or U	R U*	R D*	R D*	—	—	R U
REDEFINE RELATION/ KEY	R	A	R*	R U	—	R	R D*	R D*	—	—	R W
REDEFINE ATTRIBUTE/ ATTRIBUTE NAME	R	A	R*	R U	R U	R	—	R U*	R U*	—	—
REDEFINE ATTRIBUTE/ ATTRIBUTE TYPE	R	A	R*	R U	R*	R	—	—	—	—	—
REDEFINE ATTRIBUTE/ FIELD LENGTH	R	A	R*	R U	—	R	—	R* D	R D*	R* D	R U
REDEFINE ATTRIBUTE/ ATTR. SEC. CODE	R	A	R*	R U	—	R	—	—	—	—	—
SIMILAR	R	A	R*	R	—	R	R, A or U	—	—	—	—
PERMANENT	R	A	R*	R U	R U	R U	R U	—	—	—	—

Notation. R, read; W, write; *, conditional; A, add tuple; D, delete tuple; U, update tuple.

Table 4. Data base operations on DDS database relations

Relation	Definition	Initial loading	Tuple addition	Tuple deletion	Tuple update	Retrieval/ schema display
SYS 01	DDL[r]	DML[r]	DML[r]	DML[r]	DML[r]	QFL[r]
SYS 02	DDL[r]	Not required	DDL[a] DML[a] QFL[a]	DML[r]	Not permitted	QFL[r]
SYS 03	DDL[r]	DML[r]	DML[r]	DML[r]	DML[r]	QFL[r]
SYS 04	DDS Software	DDL[a]	DDL[a]	DDL[a]	DDL[a]	QFL[r] (all users)
SYS 05	DDL[r]	DDL[a]	DDL[a]	DDL[a]	DDL[a]	QFL[r]
SYS 06	DDL[r]	DDL[a]	DDL[a]	DDL[a]	DDL[a]	Not permitted
SYS 07	DDL[r]	Not required	DDL[a] DML[r]	DDL[a] DML[r]	DDL[a] DML[r]	QFL[r]
SYS 08	DDL[r]	Not required	DML[a] QFL[a]	DDL[a] DML[a] QFL[a]	DDL[a] DML[a]	Not permitted
SYS 09	DDL[r]	DML[r]	DML[r]	DML[r] DDL[a]	DML[r]	QFL[r]
SYS 10	DDL[r]	DML[r]	DML[r]	DML[r] DDL[a]	DML[r]	QFL[r]
USER DATABASE RELATION	DDL[r] (all users)	DML[r] (all users)	DML[r] (all users)	DML[r] (all users)	DML[r] (all users)	QFL[r] (all users)

Notation. DDL[r], DML[r], QFL[r], appropriate sublanguage statement used for the indicated relation; DDL[a], DML[a], QFL[a], appropriate sublanguage statement used for any database relation.

(*b*) SYS 04 schema is software-embedded; however, its definition must also be included in SYS 04, as SYS 04 is the primary system structure for the entire database schema.

(*c*) SYS 01, SYS 03, SYS 09 and SYS 10 must be loaded and updated by DBA. The system is designed such that DBA can use DDS/DML statements (LOAD, ADD, DELETE and UPDATE) for this purpose.

(*d*) SYS 02, SYS 04, SYS 05, SYS 06, SYS 07 and SYS 08 are automatically loaded and updated by the DDS/DMBS software as DDS data language programs are being processed. Table 3 details this requirement for all DDL operations.

In addition to these requirements, the following assumptions are needed in order to simplify or rationalise the design.

(*a*) The following operations on DDS system control structures are categorically prohibited:

- DDS/DDL operation: DEFINE RELATIONS/ USING, REDEFINE RELATION, REDEFINE ATTRIBUTE, PERMANENT.
- DDS/DML operations: COPY, COMBINE, SORT, INVERT.

(b) The following operations on certain DDS system control struuctures are prohibited:

- LOAD, DELETE, ADD and UPDATE for SYS 04, SYS 05, SYS 06 and SYS 08.
- LOAD for SYS 02 and SYS 07.
- Retrieval and display of tuples from SYS 06 and SYS 08.

(*c*) The following DDS/DDL operations are permitted for all DDS system control structures: DEFINE RELATION, DEFINE ATTRIBUTE and SIMILAR.

(*d*) Retrieval from SYS 04 using DDS/QFL is permitted to all authorised database users. All other permissible operations are intended for DBA.

Table 4 summarises the above requirements and assumptions and for each system control structure and database relation indicates the permissible operations. It also displays the DDS operations applied to an arbitrary relation which results in loading or updating the contents of the DDS system control structures.

5. SYSTEM GENERATION: DEFINING DDS USING DDS/DDL

Clearly, the third assumption cited above implies that, while generating the system, following a bootstrap routine, it is possible to make use of a subset of the proposed schema description language. This is a useful and interesting design characteristic. It saves DBA the time and the effort to write a special-purpose software package whose main function is to define schema for all system control structures and to load those which are required to act as read-only files (i.e. SYS 01, SYS 03, SYS 09 and SYS 10).

Schema description for a new relation requires DEFINE RELATION and DEFINE ATTRIBUTE blocks in a DDS/DDL program. An examination of Table 3 (first and third rows) reveals that it is necessary to have schema descriptions for SYS 01, SYS 02, SYS 03, SYS 05 and SYS 06 existing in SYS 04, and SYS 01, SYS 03 and SYS 06 at least partially loaded, before these language constructs can be used for an arbitrary relation. In other words, the 'starter' (or the bootstrap routine) for this recursive system characteristic has to handle definition and manipulation operations for the first six system control structures. Once this is accomplished, all

other system control structures and database relations can be defined and loaded directly through DDS/DDL and DDS /DML programs, respectively.

A bootstrap routine can be easily written and included in the DDS/DBMS software. However, as this routine will be used only once during system generation, it must be kept as simple and concise as possible. The bootstrap routine is considerably simplified if the following assumptions are made.

(*a*) User authorization is bypassed in the bootstrap routine.

(*b*) No error diagnostics will be produced during its execution. To ensure that the SYS 03 content is not needed, the first four DDL programs of the system generation sequence must be 'canned' DDS/DDL programs containing no errors.

(*c*) A log entry will not be produced for the first program of the system-generation sequence.

Under these assumptions the bootstrap routine has to perform these functions.

(*a*) Read schema descriptions for SYS 01–SYS 06 expressed in terms of DEFINE RELATION and DEFINE ATTRIBUTE blocks in the first program of the system-generation sequence.

(*b*) Generate SYS 04 tuples for SYS 01–SYS 06 and write them on to SYS 04.

(*c*) Generate SYS 05 tuples for the SYS 04 content and write them on to SYS 05, using its schema from SYS 04.

(*d*) Generate SYS 06 tuples for the SYS 04 content and write them on to SYS 06. SYS 06 schema is already in SYS 04 and can be used for this step.

(*e*) Generate a single fixed-content SYS 01 tuple (00001, DBA, 1) and using SYS 01 schema available in SYS 04, write this tuple on to SYS 01.

The bootstrap routine has been incorporated into the DDS/DBMS software. The DDS system-generation sequence consisting of six DDS programs is based on this routine. The system-generation sequence is shown in Appendix B. Note that it also includes some DDS/DML runs of LOAD and DELETE type. LOAD and DELETE operations on the DDS system control structures require only a subset of the schema needed by DEFINE RELATION and DEFINE ATTRIBUTE blocks.[12] Therefore they can be legitimately requested after the first program of the generation sequence, as by then the bootstrap routine will have generated the required schema.

Steps B.1 and B.5 of Appendix B are DDS/DDL programs with domain specifications for all attributes in the system control structures. It should be noted that some of these domain specifications are the origins for certain rules which in turn define DDS/DBMS. For example, the domain specification for the attribute 'relation-name' of SYS 04, SYS 05, SYS 06 and SYS 07, relation-name ALPHANUM (10), causes the following rule to emerge:

RULE: A relation-name in DDS is a character string of length 1–10.

There are quite a few such system rules which take their origin not from the proposed design, but from the utilised system generation sequence and the bootstrap routine incorporated into the software. It is DBA's responsibility to decide upon the domain specifications for the system control structures and keep the users informed about the consequent system rules.

6. SUMMARY

This paper has presented a general-purpose schema definition and manipulation facility for a relational database environment, consisting of a data definition language, DDS/DDL, and an accompanying database management system, DDS/DBMS. It is designed exclusively for the database administrator, and its capabilities comprehensively cover all requirements of DBA. DDS/DDL is one of the three sublanguages in the proposed data language, which also includes a database manipulation sublanguage and a query formulation sublanguage.

DDS/DDL is a block-structured, user-friendly and English-like language with linear syntax. It produces easy-to-read, non-procedural data definition programs to be accepted, compiled and processed by DDS/DBMS. DDS/DDL language constructs can be used for any database relation, including ten basic DDS system control structures which are part of DDS/DBMS and are designed as relations themselves. Therefore, it is possible to generate DDS/DBMS using its own data definition sublanguage.

The proposed schema definition facility has an implementation in which PL/1 is the implementation language. An extended and improved version of the design is currently being implemented in FORTRAN. In this implementation all database relations are organised using FORTRAN's relative (direct) file organization technique, together with index tables to permit keyed retrieval from relations.

REFERENCES

1. E. F. Codd, A relational model of data for large shared data banks. *Communications of ACM* **13** (6), 377–387 (1970).
2. E. F. Codd, A data base sublanguage founded on the relational calculus. *Proceedings of ACM SIGFIDET Workshop on Data Description, Access, and Control*, 35–68 (1971).
3. E. F. Codd, Relational completeness of data base sublanguages, *Courant Computer Science Symposia*, **6**: *Data Base Systems*. Prentice-Hall, Englewood Cliffs, New Jersey (1972).
4. J. B. Rothnie, Jr, An approach to implementing a relational data management system. *Proceedings of ACM SIGMOD Workshop on Data Description, Access, and Control* (1974).
5. G. D. Held, M. R. Stonebraker and E. Wong, INGRES – a relational data base system. *Proceedings of the National Computer Conference* **44**, 409–416 (1975).
6. R. C. Goldstein and A. J. Strand, The MacAIMS data management system. *Proceedings of ACM SIGFIDET Workshop on Data Description and Access* (1970).
7. M. G. Notley, *The Peterlee IS/1 System*. IBM (UK) Scientific Center Report UKSC-0018 (1972).
8. R. F. Boyce *et al.*, Specifying queries as relational expressions: the Square data sublanguage. *Communications of ACM* **18** (11), 621–628 (1975).
9. D. D. Chamberlin and R. F. Boyce, Sequel: a structured English query language. *Proceedings of ACM SIGMOD Workshop in Data Description, Access, and Control* (1974).
10. M. M. Ashtrahan *et al.*, System R: a relational approach

to data base management. *ACM Transactions on Data Base Systems* **1** (2), 97–137 (1976).
11. M. M. Zloof, Query by example. *Proceedings of the National Computer Conference* **44** (1975).
12. Y. Uckan, *Design of a Relational Data Language and A Data Base Management System.* Technical Report, Department of Computer Engineering, Middle East Technical University, Ankara (1980).
13. Y. Uckan, Design of a relational data base system, *Proceedings of the Fourteenth Annual Pittsburgh Conference on Modeling and Simulation* **14** (3), 959–964 (1983).
14. Y. Uckan and W. D. Haseman, An optimal access path handling strategy in a data base environment (working paper).

APPENDIX A: DDS DATA DEFINITION SUBLANGUAGE SYNTAX

In the following syntax definitions, the CBL-notation and its conventions are used.

A.1. define-relation-block:: = *DEFINE RELATION*: relation-name-1

{;TUPLE *STRUCTURE*: attribute-name-list-1 / *USING* relation-name-2 *AND* relation-name-3[;TUPLE *STRUCTURE*: attribute-name-list-1]}
[;*KEY*: attribute-name-list-2]
[;RELATION: relation-name-4
{;TUPLE *STRUCTURE*: attribute-name-list-3 / *USING* relation-name-5 *AND* relation-name-6[;TUPLE *STRUCTURE*: attribute-name-list-3}
[;*KEY*: attribute-name-list-4]]... *END*.

A.2. define-attribute-block:: = *DEFINE ATTRIBUTE*:

attribute-name-1 {*ALPHANUM* / *NUMERIC*} (field-length) [attribute-security-code-1]

[;attribute-name-2{*ALPHANUM* / *NUMERIC*} (field-length-2) [attribute-security-code-2]]... *END*.

A.3. redefine-relation-block:: = *REDEFINE RELATION*: relation-name-1

AS { *DELETED* / relation-name-2 [;TUPLE *STRUCTURE*: attribute-name-list-1] [;*KEY*: attribute-name-list-2] / [relation-name-2] ;TUPLE *STRUCTURE*: attribute-name-list-1[;*KEY*: attribute-name-list-2] / [relation-name-2] [;TUPLE *STRUCTURE*: attribute-name-list-1];*KEY*: attribute-name-list-2 }
[;*RELATION*: relation-name-3
AS { *DELETED* / relation-name-4 [;TUPLE *STRUCTURE*: attribute-name-list-3] [;*KEY*: attribute-name-list-4] / [relation-name-4];TUPLE *STRUCTURE*: attribute-name-list-3[;*KEY*: attribute-name-list-4] / [relation-name-4] [;TUPLE *STRUCTURE*: attribute-name-list-3];*KEY*: attribute-name-list-4 }]... *END*.

A.4. redefine-attribute-block:: = *REDEFINE ATTRIBUTE*:

attribute-name-1 [*OF* RELATION relation-name-1] *AS*

{ attribute-name-2 [{*ALPHANUM* / *NUMERIC*}] [(field-length-1)] [attribute-security-code-1]
[attribute-name-2] {*ALPHANUM* / *NUMERIC*} [(field-length-1)] [attribute-security-code-1]
[attribute-name-2] [{*ALPHANUM* / *NUMERIC*}] (field-length-1) [attribute-security-code-1]
[attribute-name-2] [{*ALPAHNUM* / *NUMERIC*}] [(field-length-1)] attribute-security-code-1 }

[;attribute-name-3 [*OF* RELATION relation-name-2] *AS*

{ attribute-name-4 [{*ALPHANUM* / *NUMERIC*}] [(field-length-2)] [attribute-security-code-2]
[attribute-name-4] {*ALPHANUM* / *NUMERIC*} [(field-length-2)] [attribute-security-code-2]
[attribute-name-4] [{*ALPHANUM* / *NUMERIC*}] (field-length-2) [attribute-security-code-2]
[attribute-name-4] [{*ALPHANUM* / *NUMERIC*}] [(field-length-2)] attribute-security-code-2 }]...*END*.

A.5. similar-specifications:: = *RELATIONS* relation-name-list

{*IS* / *ARE*} *SIMILAR TO* RELATION relation-name.

A.6. permanent-specification:: = *RELATIONS* relation-name-list

{*IS* / *ARE*} PERMANENT.

A.7. relation-name-list: = relation-name-1[,relation-name-2]...
A.8. attribute-name-list:: = attribute-name-1[,attribute-name-2]...
A.9. field-length:: = integer
A.10. attribute-security-code:: = integer

APPENDIX B: DDS SYSTEM GENERATION SEQUENCE

B.1. Define relations and all attributes of relations SYS 01–SYS 06.

```
BEGIN:
  USER-ID = 00000; USER-NAME = SYSTEM.
    DEFINE RELATION: SYS 01;
      TUPLE STRUCTURE: user-id, user-name,
      authorisation-code;
        RELATION: SYS 02;
      TUPLE STRUCTURE: user-id, run-date,
      run-time, statement-keyword, run-result, relation-
        used-1, relation-used-2, relation-used-3;
      KEY: user-id, run-date, run-time;
        . . . . . .
      END.
      DEFINE ATTRIBUTE:
        user-id NUMERIC (5) 1;
        attribute-name ALPHANUM (20);
        relation-name ALPHANUM (10);
        . . . . . .
      END.
    END.
```

B.2. Load SYS 01.

```
  BEGIN:
    USER-ID = 00001; USER-NAME = DBA.
    RELATION S1 IS SIMILAR TO SYS 01.
      CONTENT OF RELATION S 1:
        00002, JOHNSON, 1;
        . . . . . .
    END.
    LOAD RELATION S1 TO SYS 01.
  END.
```

B.3. Delete 00001, DBA from SYS 01.

```
  BEGIN:
    USER-ID = 00002, USER-NAME = JOHNSON.
    DELETE TUPLE WITH KEY 00001 IN RELATION
    SYS 01.
  END.
```

B.4. Load SYS 03.

```
  BEGIN:
    USER-ID = 00002, USER-NAME = JOHNSON.
    RELATION S3 IS SIMILAR TO SYS 03.
    CONTENT OF RELATION S3:
      . . . . . .;
      . . . . . .
    END.
    LOAD RELATION S3 to SYS 03.
  END.
```

B.5. Define SYS 07–SYS 10. (Similar to step 1, except USER-ID and USER-NAME must correspond to an authorised user which exists in SYS 01 due to step 2.)

B.6. Load SYS 09 and SYS 10. (Similar to step 4.)

Mathematical Models of File Growth

C. H. C. LEUNG AND K. WOLFENDEN
Department of Computer Science, University College London, Gower Street, London WC1E 6BT

The number of records in a file system is often recognised as a key determinant of efficiency. For example, the performance of sequential processing is $O(N(t))$ and that of tree search is $O(\log N(t))$, where $N(t)$ is the number of records in the file at time t. In this paper, the growth behaviour of files is studied in terms of quite general record insertion and deletion characteristics, and the performance evolution of some of the common systems is analysed. The growth data of an actual system are compared with the model results and reasonable agreement is observed.

1. INTRODUCTION

The efficiency of a file system is often related to its size. As the size of a file grows, its efficiency deteriorates, since the amount of time required to locate an item in a larger file tends to be longer. As one of the chief requirements of file systems is to enable record insertion and deletion to be carried out in a simple and routine manner, the variation in file efficiency over time is therefore to be expected and such variation is especially pronounced for volatile systems. For many of the common access techniques, their performance is directly related to the number of records present. For example, in sequential processing, the average number of accesses incurred in locating a given record varies approximately[7] as $N/2$, where N is the number of records in the file. In the case of random processing, the average access distance separating two randomly located records varies approximately[14] as $N/3$. In these cases, each record is responsible for a certain amount of contribution to the overall performance penalty – e.g. in sequential processing, this is one-half of an access. Thus the record insertion and deletion mechanism, which governs the file size, has a direct bearing on time-dependent file behaviour. Now insertions and deletions are random events occurring in time and often follow definite statistical patterns. They are dependent on factors such as the nature of the file system, its usage pattern, and the business activities to which the file relates. For instance, in order processing applications, the insertion and deletion of order records – which is related to order placement and delivery – depend on the seasonality of the products, the buoyancy of the market, the availability of stock, and the speed with which the orders are processed. These underlying mechanisms will consequently shape the overall statistical structure to which the insertion and deletion processes conform. The growth pattern of a file system is therefore not unlike that of a software system[2] in that it is related to the behaviour of the underlying application the characteristics of which are governed by the wider laws of, for example, business and economic systems.

Growth and deterioration of file systems have been previously considered in Refs. 1, 3–5, 8–11, 13, 15 and 17–19. In Refs. 15, 17 and 19 abstract deterioration patterns are adopted as basic assumptions without explicit reference to the underlying record insertion and deletion patterns for the purpose of formulating suitable reorganisation procedures. In Refs. 1, 3, 4, 8–11, 13 and 18 the impact of record insertions and deletions on performance deterioration is explicitly taken into account, and in these studies the number of records present in the system is generally recognised to be a key determinant of efficiency. They are focused either on the analysis of particular file structures[1, 3, 4, 8, 10, 13, 18] or the determination of optimal reorganisation strategy,[1, 9, 11] and the patterns of record insertion and deletion are invoked solely for the purpose of supporting an overall evaluation. The characteristics of record insertions and deletions do not form the focus of these studies and they take on only a supporting role there; accordingly assumptions concerning them, although not always unreasonable, are sometimes restrictive and are mostly adopted for tractability or convenience. In Refs. 1 and 3 insertions are given in terms of the actual number of records, and the statistical pattern of insertion over time is not considered. In Refs. 13 and 18, the insertion pattern over time is assumed to follow a homogeneous Poisson process and the deletion process for individual records is likewise taken to be Poisson with the same rate. Ref. 8 adopts a slightly more general deletion pattern by allowing the deletion rate to be different from the insertion rate, both of which remain to be homogeneous Poisson processes. Ref. 4 relaxes the Poisson deletion assumption by allowing the lifetime of individual records to be generally distributed but retaining the homogeneous Poisson insertion assumption. In Refs 9 and 11, the insertion of records is assumed to follow a renewal process in which the inter-insertion interval of records is assumed to be independent, identically distributed; the statistical distribution of record lifetime is not considered and deletion is only permitted at file reorganisation. In Ref. 10, the insertion process is taken to be a non-homogeneous Poisson process, but again the deletion mechanism is not considered. A particularly noteworthy study which specifically aims to model the time-dependent performance of files is Ref. 5, where stochastic diffusion models are employed to approximate growth behaviour. However, the dynamic pattern of record insertion and deletion over time there is not explicity represented and the study primarily focuses on file degradation after a given number of transactions have been entered.

The need for studying the impact of dynamic record insertions and deletions on file evolution has been pointed out in Ref. 1, but as yet there is no systematic study which specifically analyses the growth behaviour of general file systems over time. It is the aim of the present paper to provide such a study. The model presented here is quite general and includes all the above characteristics as